Deep Learning Using Tiny Domain-Specific Datasets with Sparse Labels

PhD Thesis

Thomas James Smith *MSci*

Computer Vision Lab

Computer Science

University Of Nottingham

Prof Michel Valstar, Dr Mercedes Torres Torres,

Dr Don Sharkey, Prof John Crowe

October 2016 to March 2021

N.B. : *I suffer from dyslexia, and I would greatly appreciate that my spelling, grammar and sentence structure are not judged negatively.*

Abstract

Machine learning is an ever-expanding field of research, and recently deep learning has been the architecture of choice. However, traditional deep learning methodologies require substantial amounts of data to train their networks. This requirement for large data means that there are large numbers of real-world problems that cannot utilise the power of these deep learning networks due to a lack of data. Being able to use deep learning architectures with tiny domain-specific datasets would allow sectors such as healthcare to use deep learning as an aid in training and potentially in real time procedures.

In this thesis, deep learning using tiny domain-specific datasets with sparse labels is achieved on two machine learning problems: semantic segmentation and action recognition. This is accomplished by utilising semi-unsupervised learning to train a convolutional neural network (CNN) to predict superpixels for an image, using a novel structural representation named Multi-channel Connected Graphs (MCGs). These deep-learned superpixels are then used in an end-to-end network consisting of two hourglass modules, with each specialising in a separate task; 1) deep learned superpixels, 2) semantic segmentation. For multi-class semantic segmentation, a variation on transfer learning is used. Action recognition with tiny amounts of training data is obtained by drastically reducing the input feature from full HD resolution down to $32 \times 32 \times 2$, with each cell consisting of the majority class in the first channel, and the secondary class in the second channel. This input is then used in a recurrent neural network consisting of multiple CNNs and bidirectional long short-term memory layers.

Firstly, to my principle supervisor, Michel. I would like to say a big thank you for all of your support and guidance throughout my PhD If you had never suggested that I pursue a PhD after completing your machine learning module and third year dissertation during my MSci degree, I would have let this opportunity pass me by. I am truly grateful for all of your help, and I am looking forward to continuing to work with you in the future.

To my additional supervisors Don, John and Mercedes I would like to say thank you for all of your support and enthusiasm.

To my wife Jess, I would like to thank you for all of your support throughout my PhD You have always been there for me, through the highs and lows. You have believed in me from the start and reassured me when I was down. Thank you for all of the help with proof reading my work, and finding those pesky missing "s"'s at the end of *word*. Thank you for putting up with the long days of writing and coding. I love you always.

A final thank you to my parents and my sister. Especially to my parents for always believing in me, and pushing me to do my best. Thank you for always being very excited whenever I give you something of mine to read. I'm sorry it's generally complicated and boring. Of course I cannot forget to thank Chip and Taco, who have given the me best motivational dog cuddles anyone could ask for.

Contents

Chapter 1

Introduction

1.1 Motivation

Artificial Intelligence (AI) has been used for a long time to help humans solve complex problems with the aid of computer systems. Machine learning is one key field of AI that uses observations of the world around us to teach a computer to perform specific, well-defined tasks. With vast amounts of data being generated daily across the world, researchers have an abundance of observations to solve increasingly complex tasks such as object detection. Deep learning is a particularly successful machine learning technique that takes these observations and passes them though multiple layers of artificial neural networks. Complex architectures are regularly proposed, and research into ever-better ways to optimise the vast number of weights in such architectures generates models that have much higher accuracy than other machine learning techniques, including shallow neural networks (artificial neural networks with a single hidden layer). However, to train these deep neural networks, a large number of observations (i.e. data points, or instances) are needed to train a model that can generalise well to new and unseen data. A good question is exactly how much data would be required to reach a sufficiently high accuracy for a particular task in a particular domain? It is generally considered that the number of observations should be a minimum value of in the thousands for a good representation of the problem.

There are many real world problems that could fall into the supervised learning category, in the sense that a well-defined task can be described, but not all problems are suitable for generating sufficient numbers of ground truth annotations. One example of this is the class of clinical image analysis problems. Collecting large sets of annotated data in a clinical imaging domain is generally not practical, as it would require considerable time, money, skill, and effort. To begin with, patient data is difficult to gather because in our(western) society patients have to opt into the data collection procedure and not everyone is willing to share sensitive data. The clinical data collected then needs to be annotated by someone who understands the problem, which is often a highly trained healthcare professional who would rarely have the spare time and motivation to annotate substantial amounts of data. When they are motivated, their capacity for annotation is generally low, and their time expensive.

Another example of a dataset that would benefit from being able to train with small amounts of data is rapid prototyping of novel atomic systems that use computer vision. An atomic system is a process that is able to be fully automated by a machine. The time and monetary value for these annotations could potentially outweigh the value of the rapid prototyping. For example, collecting real world data with the prototype system is not achievable due to regulations. Thus the only way to collect the data is manually by person, and for a small team it would take a infeasible amount of time to collect a large dataset.

Therefore, if deep neural networks could be trained with much less data, this could mean that professionals would be able to invest a reasonable amount of time to annotate these datasets. This ability to use tiny, domain-specific datasets would open up an extensive number of real world problems that could be solved or made easier by using deep learning for assistance.

There are three main categories that the problem of learning with tiny datasets can be broken down into. Each of the categories requires different learning techniques to deal with them. Firstly, supervised learning is used for problems where the predictions can be compared against known labels. Examples of supervised learning are classification or regression. Supervised learning requires each observation to have a label belonging to it; these are known as ground truth labels. These labels are used to train the architecture's parameters to correctly predict the ground truths. Secondly, unsupervised learning is used for problems where ground truth labels are not provided and thus it is necessary to find out relationships between the observations within the dataset. This uses methods such as clustering. The third category is reinforcement learning, this is used when there is an agent operating in an environment, such as a robot vacuum cleaner in an office. Reinforcement learning uses a reward-based system to teach the agent what to do by analysing its actions in the environment in real time.

In this work I wanted to find out: to what degree is it possible to train deep learning architectures on tiny domain-specific image datasets whilst achieving useful predictions, and what are the key factors that determine success?

As there are many fields of research in which deep learning architectures can be applied, and it would be impossible to test all scenarios, this thesis will focus on two computer vision problems: semantic segmentation and action recognition. These two problems occur in many different sectors in the real world, from industry applications to clinical procedures. To answer the two questions, two different datasets will be used:

gLitter and Newborn Resus. gLitter is a segmentation dataset consisting of various types of litter found in the city parks scattered on different floor types. This dataset poses the challenge of being able to detect complex object boundaries with tiny amounts of data. The Newborn Resus dataset consists of paediatric staff performing resuscitation on newborn babies. The resuscitation procedure is comprised of multiple actions that must be followed. This dataset poses the challenge of having a tiny number of occurrences per action due to the duration required for collecting and annotating the data. This is because the dataset consists of sensitive imagery, and a clinical professional is required to annotate the data.

Traditionally, semantic segmentation datasets with large amounts of data are labelled with sub-optimal annotations, where a loss of annotation quality is accepted as a trade-off for speeding up the process of annotation. These generally consist of rough polygon outlines of an object. Amazingly, when using vast amounts of data with a large number of occurrences for each class in the dataset, the networks trained on these systems can generalise away from the sub-optimal segmentation masks, closer to the true pixel-level segmentation. However, for a network that is trained on a small amount of data, this generalisation is unachievable because the network massively over-fits to the training data. This means that the incorrect segmentation masks are learnt and thus the network performs poorly on unseen test data. To reduce confusion during training, the use of a close to pixel-perfect mask is preferred. In this thesis I explore the effect of annotation accuracy in relation to the size of the dataset. From the study shown in chapter 6.3, it can be concluded that for small domain-specific datasets, the effect of lower accuracy annotations is more pronounced than the effect of the amount of training data on the performance of a model's accuracy. The greater the degradation of the near pixel-perfect annotations, the lower the overall accuracy is, regardless of the amount of training data. Additionally, it was also concluded that increasing the amount of training data does improve the accuracy of the model, but for high levels of degradation much greater numbers of training data would be required to achieved similar results to the better quality annotations. This is an outcome of practical importance to those creating small datasets - it means that for small datasets the annotations should be of the highest possible quality.

To achieve pixel-perfect annotation, the annotations must be done at the pixel level using a polygon tool. A good tool for this is Photoshop, because its built-in tools have been heavily refined. However, the annotation time is proportional to the number of pixels in an image, therefore an image of resolution 1080p would take many times longer to annotate than a 480p image. Annotation times for complex objects/scenes can take multiple hours to fully annotate, per image.

Near-pixel-perfect annotations can be achieved much faster than pixel-perfect annotations because they can make use of over-clustering algorithms. With manually-tuned parameters for the clustering technique, the annotations can be performed at a similar speed on a wide variety of resolutions. These near pixel-perfect annotations tend to follow clusters of pixels, meaning that objects with similar colours and textures next to each other can be clustered together and need to be manually corrected via another annotation tool, such as a polygon drawing tool. Near-pixel-perfect annotations only take a fraction of the time that pixel-perfect annotations take, with complex objects/scenes taking several minutes rather than hours. These annotations are closer to pixel-perfect than the rough polygons are, meaning there is less confusion in the labels for the machine learning networks to overcome.

In this thesis, near pixel-perfect annotations are used for both the gLitter and Newborn Resus datasets. To create annotations, a novel image annotation tool is created. This image annotation tool leverages the inherent property of superpixel, to not cross object boundaries, to speed up human annotation whilst following the boundaries of objects. The use of a brush tool that fills the selected superpixel with the current class colour, and the fill tool allow for fast annotations. For large objects the user can quickly select the boundary superpixels using the brush, then fill the centre using the fill tool. For objects that are smaller that the current superpixel size, the superpixel algorithm can be re-run at a higher target number to allow for smaller superpixels to be created, or the use of a polygon tool can be used in conjunction with the super-pixel tool.

Another potential solution to creating deep-learned machine vision algorithms that generalise well after training on tiny datasets is the creation of a common backbone or low-level image encoder suitable for domain-agnostic semantic segmentation.

Having a small dataset means that there is the probability of having insufficient data to train a network, resulting in a network that is unable to learn the features/patterns of the data that would allow it to generalise well for use with unseen data. A method to combat this is to specify extra input features that would help the network to find these patterns in the data. In this thesis, I explore the novel idea of using learned superpixels to add the extra input feature of structural relationships between clusters. Each cluster informs the network which pixels have similar properties: shape, colour, and texture. However, during initial experiments it was found that to include superpixels as a feature for a convolutional neural network (CNN), a new representation of superpixels was required.

To represent the relationship in regards to superpixels, for each pixel to its neighbours, a novel representation was developed - Multi-channel Connected Graphs (MCGs). This representation allows for handcrafted superpixels to be represented in a form which a CNN can interpret. Additionally, this leads to the ability to truly deep learn superpixels for the first time. A deep-learned superpixel has the advantage of being able to be trained in an unsupervised manner on an extensive and varied dataset, meaning that the network is able to create a generalised superpixel representation that can also be used in an end-to-end network for the specific tiny dataset domain. On the other hand, a handcrafted superpixel algorithm would require manual tuning of its parameters (number of superpixels, compactness, and sigma) for each instance in the dataset.

Artificially creating training samples is another possibility for increasing the number of samples in a domain-specific dataset so that it could be used to train deep-learned superpixels. There are several techniques to do this; the use of 3D rendered scenes, staged videos, or the newly popular generative adversarial network (GAN) could be used. Each of these methodologies have their flaws for this thesis. Firstly, 3D rendering is not suitable as this would require expertise that cannot be considered to be readily available. Staging is a process to create additional training data using artificial scenarios, such as using a training doll to perform resuscitation upon instead of a human. An inherent disadvantage of using large volumes of staged data is that a model trained on staged videos may not be able to generalise to real scenarios. In addition, staging videos is a time consuming process, thus with regards to the Newborn Resus dataset this would take valuable time away from the paediatric staff.

GANs generate novel images based on those included in a given dataset, however the quality of these images at this point in the thesis were not of high enough quality, and even modern GANs would need considerable amounts of data themselves to train their models, more than what constitutes a tiny dataset, thus making them unsuitable in this instance. Additionally, with the low quality of images produced, artefacts would be introduced into the deep-learned superpixels.

Therefore, using a different dataset with broad variation in what is shown in each video was the chosen course of action for this thesis. This enables a large CNN network to use an unsupervised approach to training.

The popular stacked hourglass network was used as a base network architecture for semantic segmentation. The network architecture used in this thesis consisted of two hourglass modules, feeding from the first to the second in a end-to-end manner, with each being pre-trained for specific tasks. Both hourglass modules use a single residual module at each step in the hourglass. The first of the two hourglass modules was pre-trained to predict the deep-learned superpixel, the second module was trained for semantic segmentation.

A common problem with deep learning using small datasets is having a poor representation of the data one would like the machine to learn. Too many features in the representation would result in the network overfitting due to the curse of dimensionality. The curse of dimensionality describes the problem that arises when there is a much greater volume of space, caused by the high dimensional representation of the data, compared to the available data being represented. Too few features mean that there would not be enough information for the network to learn all of the outcomes. This can be because there is not enough variation in the given features to distinguish between examples.

For the action recognition problem with tiny domain-specific datasets, the clinical dataset Newborn Resus is used. The Newborn Resus dataset consists of 22 videos, 19 actions and 23 semantic object classes. With such small amounts of data, the representation of the videos was first reduced to its semantic segments, then scaled down to $(32 \times 32 \times 2)$ from 1080p. Using a small network consisting of two convolutional layers and three bi-directional LSTM layers resulted in a micro (weighted) average F1 score of 0.5013 on 7 classes. The 'Airway manoeuvre' (I) action has the best detection with an F1 score of 0.6277, recall score of 0.8284, and precision score of 0.5053. There are four classes[1] which fail to learn due to lack of instances in the dataset, with a resulting F1 score of zero.

Based on the findings from this thesis, when training a small semantic segmentation network with a tiny domain specific dataset, I would recommend using a transfer learning technique to train a single class at a time with an adaptive unknown class, starting with the largest most frequent classes first. In regards to action detection I would highly recommend having a balanced dataset with at least 30 instances per action, but preferably a minimum of 50 instances. Additionally, for action detection when using a small dataset, using a sparse representation of the input works effectively. For instance, a sparse representation of an frame could be using the most prominent semantic segmentation class when down-sampling. If more data is required than a single class, the number of channels can be increased by adding the next most prominent semantic segmentation class per area. This is preferred to doubling the resolution which results in quadruple the number of input features.

For both action recognition and semantic segmentation, the balance of classes is a huge factor in regards to performance (as demonstrated in Chapters 7 and 6). Less frequent classes are often ignored by the network during training as they add little weight to the calculation of the loss functions. To improve results I would recommend either removing these classes from the list of classes to detect, or increasing the number of occurrences in the dataset to equalise the distribution.

If inference speed is a priority, I would highly recommend reducing the resolution of the end-to-end network, or using newer techniques of aggressive down-sampling to reduce the amount of parameters in the network.

[1]Dried with a towel (A), Attachment of pulse oximeter plus Pulse oximeter adjusted (E+F), Inflation breaths given incorrect ratio plus Provide five inflation breaths lasting three seconds plus Ventilation breaths given incorrect ratio plus Ventilation breaths given for 30s and 1s each (J+K+L+M), Incubation attempt(P)

1.2 Thesis Contributions

This thesis has produced the following contributions:

1. Novel graph-based representation for relationships between pixels called Multi-channel Connected Graphs (MCG)

2. End-to-end semantic segmentation with deep-learned superpixels for tiny domain-specific datasets

3. Action recognition predictions using low dimensional representation of the input video for tiny clinical datasets

4. Novel technique for full scene semantic segmentation annotation that incorporates the use of deep-learned cooperative learning for tiny clinical specific datasets

5. Image Annotation Tool V.2 (IAT) is an open source software used for annotating images and videos for full scene semantic segmentation by making use of superpixels to aid annotation.

Below is a list of outputs:

1. Paper 1: Clinical Scene Segmentation with Tiny Datasets - ICCV 2019 - The 2nd International Workshop on Computer Vision for Physiological Measurement (CVPM)

2. Open source software: Image Annotation Tool V.2

3. Paper 2: In preparation - Journal paper - Clinical Action Recognition with Tiny Datasets

4. Paper 3: In preparation - Paper - Effects of Ground Truth Annotation Quality on Training Deep Learning Models with Tiny Dataset.

Chapter 2

Background Theory

2.1 Computer Vision

In this section I will provide formal definitions of computer vision theory terms that are used frequently throughout this thesis. Notation introduced here will be used throughout.

2.1.1 Segmentation

It is commonly accepted that humans have five basic senses: sight, hearing, taste, smell, and touch. In computer vision the main goal is to replicate our understanding of the seen world. To do this the computer needs a visual representation of the world. The most common representation used comes in the form of digital imagery, referred to as rasterized images. Raster images are made up of a grid of pixels, where a pixel represents both colour and intensity of light at a particular location on the image. An image is said to have a higher resolution the more pixels it has. This means that more information can be stored in it.

However, individual pixels hold very little information about the real world other than the colour and intensity at a specific point. Therefore, several techniques have been developed to look at multiple pixels at once to see if there is any additional information that can be found.

An example of this is local binary patterns (LBP)[44, 103]. LBPs use a (3×3) grid around a central pixel and the central pixel value is used as a threshold. This threshold is then used to see which of the eight neighbouring pixels are greater than or less than the central pixel. If a pixel has a greater value than the threshold, it is labelled '1', otherwise it is labelled '0'. Then, reading in a clockwise motion starting in the upper left position, the ones and zeros are read to create a binary number. This binary number is converted into a decimal number and assigned in the output image at the coordinates of the current pixel, Figure 2.1. This is repeated for every pixel in the image. An eight-bit binary number can have a value of zero to 255, 256 different values. However, using uniform patterns and removing binary numbers which switch between one and zero multiple times, the number of values is reduced to 59. The purpose of LBPs is to identify texture and edges in an image.

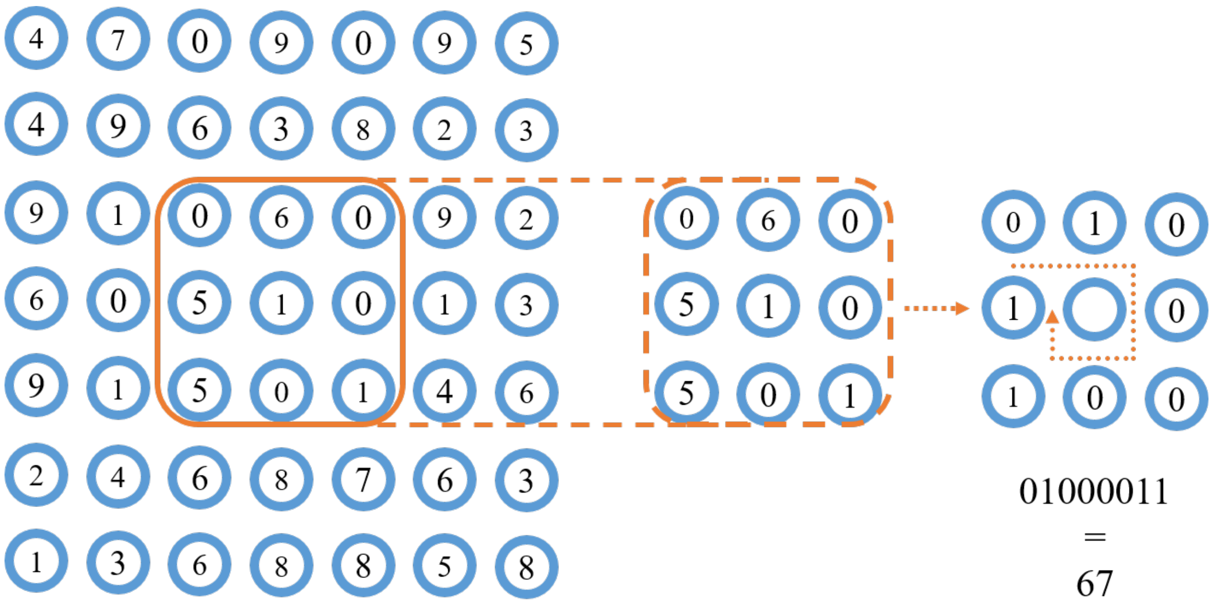


Figure 2.1: This figure shows an example calculation for LBPs using random numbers between zero and nine in a grid of (7×7). The central pixel is used as an example and the corresponding LBP value is 67.

Being able to extract texture from an image is very useful, but on its own does not offer much more information about what is happening in an image. Being able to segment an image into multiple regions can give more information about what the image represents.

Forsyth and Ponce [34] describe segmentation as an image representation problem where the outcome must be both simultaneously compact and expressive whilst being able to provide a summary of an image. These summaries could either be from single pixels or groups of pixels by their colour or texture. Segmentation has also been referred to as grouping, perceptual organisation and fitting.

When a human performs segmentation they often follow these nine rules according to the Gestalt school of psychologists[34]. Tokens in psychology refer to objects in a scene, however in computer vision, tokens would represent pixels or clusters of pixels.

1. Proximity: Tokens that are nearby tend to be grouped.
2. Similarity: Similar tokens tend to be grouped together.
3. Common fate: Tokens that have coherent motion tend to be grouped together.
4. Common region: Tokens that lie inside the same closed region tend to be grouped together.

5. Parallelism: Parallel curves or tokens tend to be grouped together.
6. Closure: Tokens or curves that tend to lead to closed curves tend to be grouped together.
7. Symmetry: Curves that lead to symmetric groups are grouped together.
8. Continuity: Tokens that lead to continuous (as in joining up nicely, rather than in the formal mathematical sense) curves tend to be grouped.
9. Familiar configuration: Tokens that, when grouped, lead to a familiar object tend to be grouped together.

The Gestalt rules are said to be key to how the human vision system works, thus many algorithms developed for image segmentation have taken these rules as a starting point to build upon.

Common examples of segmentation uses are background subtraction, shot boundary detection, and interactive segmentation. Background subtraction is a common pre-processing step for some algorithms, and consists of keeping only the objects of interest and removing any distracting backgrounds. The more stable the background is, the easier it is to remove the background from an image. A use of this could be to track objects through a video. With a stable background this would be trivial, but if it is not stable then the use of Gestalt rule 'common fate' can be used to check the motion of pixels. These pixels can then be grouped together for both the object and again for the background. Then the background segment can be subtracted from the image.

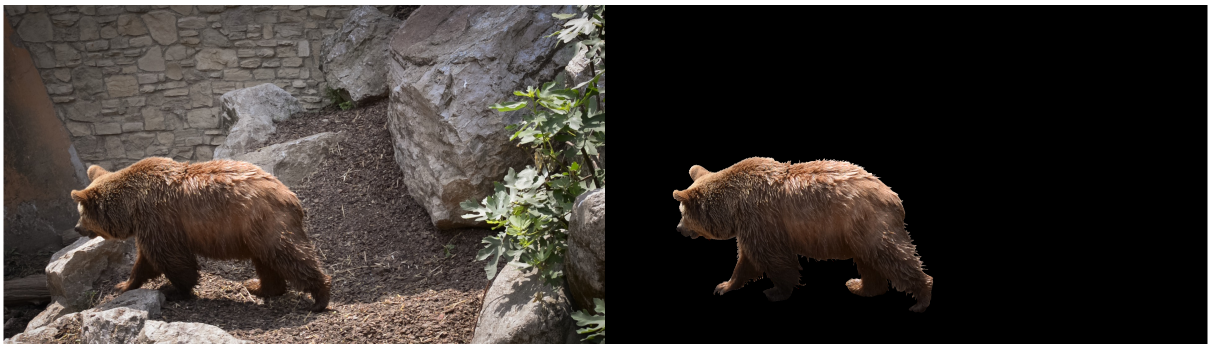


Figure 2.2: This figure shows an example of segmentation of a bear from its surroundings. This image has been adapted from the DAVIS dataset [83].

Shot boundary detection is used when analysing video to determine when there is a substantial change between frames, such as a switch in camera view or a transition to a new scene altogether. This allows for long sequences of videos to be broken up into shorter videos that can be expected to consist of similar objects. This is a relatively simple task of finding the difference between two adjacent frames, and if the difference is above a given threshold, then a cut in the video is made. When comparing the frames there are several methodologies that can be used. These include: calculating the difference between all pixels in the frame, however this is slow; comparing histograms of each frame; block comparison, where the frames are split into grids and histograms are calculated for each cell of the grid, then compared; or calculating edge maps of the two frames and computing the difference between the two masks created.

Interactive segmentation is a form of segmentation where a user gives the algorithm some input such as a bounding box to help the algorithm focus on a specific area of interest. This reduces the computation needed as the algorithms are generally working with much smaller input resolutions.

2.1.2 Superpixels and Over-Segmentation

One method of creating segmentation is to over-segment the image into smaller segments of similar pixels. These over-segmentation methods of combining multiple pixels to create regions of similarities combine colour, location and texture into a single cluster of pixels dubbed 'superpixels'. Clustering of pixels like this dates back to the late 1980's, where Mester and Franke [73] produced similar outcomes to the morphology of superpixels. In 1997 an adaptation of the watershed algorithm was used to create improved clustering of pixels [71].

The watershed algorithm [87] is based on the geological meaning of watershed, which describes how water will run off certain parts of the land, and pool in others, depending on the lay of the terrain. This represents a theory in which something starts at a high point, and follows slopes down to the lowest achievable point. Regarding watershed superpixels, this is done pragmatically by computing the gradients of the image to be segmented to produce a gradient map. Zeros in this map are considered to be extreme values, and are given unique IDs. Each pixel in the image follows the path with the steepest downwards gradient on the gradient map until it terminates at a zero point. The starting pixel is then assigned to the ID of the pixel it ends at. This process is repeated for all pixels in the image. Visualising this process, especially simultaneously for all pixels, resembles the water flooding the image and pooling in local minimums, hence the name 'watershed'. Whilst following the gradients, if a pixel could go via multiple paths, then the shortest path is used to calculate the ID.

In 2010, Achanta et al. [1] created the SLIC (Simple Linear Iterative Clustering) algorithm which is still a very popular superpixel algorithm today [2] as it provides good over-segmentation of images. SLIC works by clustering pixels in relation to their colour and spatial properties to create almost uniform superpixels that try to closely follow boundaries of objects. An example of this can be seen in Figure 2.3.

When representing an image in a computer, there are different colour spaces that can be used. A colour space is a way to represent physical light as digital information. The most common colour space is RGB (red, green, blue) colour space. The most simple colour space is grey-scale, which only stores the intensity of light at a particular pixel. For example, an image taken on a one megapixel camera would have dimensions of $(1280 \times 720 \times 1)$ for a grey-scale image. Whereas for an RGB image, the representation would use three channels, one for each colour, and would have the dimensions of $(1280 \times 720 \times 3)$.

SLIC uses another colour space called CIELAB, which was designed by the International Commission on Illumination in 1976. CIELAB is sometimes abbreviated to LAB. In the LAB colour space, l represents the lightness of the colour with zero being black, and one being white. Different implementations of CIELAB use different ranges for a and b, but all versions use this format for the range $-x$ to $+x$. a represents the green and red colours, where $-a$ represents the green intensity, and $+a$ represents the red intensity. b represents the yellow and blue colours, where $-b$ represents the blue intensity, and $+b$ represents the yellow intensity.

SLIC uses the three channels from the LAB colour space and the x y coordinates of pixels to create a five-dimensional space in which to cluster pixels. The distance measure used in the algorithm enforces the structure of the superpixels. Achanta, Shaji, Smith, Lucchi, Fua, and Süsstrunk [1] introduced a novel distance measure to create superpixels, in which a target number of superpixels (K) are taken as an input for an image. Whilst creating the superpixels, the algorithm tries to enforce the area (S) of the superpixels to be N/K pixels, where N is the total number of pixels in the image. If the superpixels were even in size, the superpixels would form a grid with the centres being at grid intervals:

$$S = \sqrt{\frac{N}{K}} \tag{2.1}$$

The superpixel centres are represented by:

$$Ck = [l_k, a_k, b_k, x_k, y_k] \quad k = 1 : K \tag{2.2}$$

As the spatial distance of each cluster is $2S$, it is assumed that all pixels belonging to the cluster should lie in a $(2S \times 2)S$ area in the x y dimensions around the central pixels in the grid. When using Euclidean distances, the straight line distance between two points, in the five dimensions $[l, a, b, x, y]$ the x y, the proportion of the distance calculation greatly outweighs the colour distance when the pixels in the image are further than the maximum colour distance.

To combat this, the distance measure D_s was designed:

$$d_{lab} = \sqrt{2(l_k - l_i) + 2(a_k - a_i) + 2(b_k - b_i)} \tag{2.3}$$

$$d_{xy} = \sqrt{2(x_k - x_i) + 2(y_k - y_i)} \tag{2.4}$$

$$D_s = d_{lab} + \frac{m}{S} \times d_{xy} \tag{2.5}$$

In equation (2.5) the term m is used to control the compactness of the superpixels. The larger m is, the more spatial proximity of pixels is prioritised, causing the clusters to be more compact. m can be set between 1 and 20, with the default setting being 10. $\frac{m}{S}$ term multiplied by the d_{xy} normalises the x y dimensions by the grid intervals.

The algorithm is initialised with the cluster centres described above, then these starting points are moved to the seed location. To calculate the seed location, a (3×3) grid is taken around the starting point and the lowest gradient position is chosen. This reduces the probability of the starting point starting on an edge. To calculate the gradients the following equation is used:

$$G(x, y) = ||I(x + 1, y)I(x1, y)||^2 + ||I(x, y + 1)I(x, y1)||^2 \tag{2.6}$$

$||\quad||^2$ is the L2 norm. L2 norm is the euclidean distance from the origin. $I(x, y)$ returns the LAB vector at the pixel location (x, y).

From the new seed location, pixels are assigned to the closest cluster using the D_s equation. Then the new centre point is computed by finding the average LABXY vector of the superpixel. These steps are repeated until there are no updates, or until a set number of iterations has passed. At the end of this process any adjacent small clusters with identical labels to larger neighbours are added into these neighbouring larger clusters. Equations 2.1-2.5 are taken from Achanta et al. [1] paper, SLIC Superpixels.

A variant of superpixels was the Superpixels Extracted via Energy-Driven Sampling (SEEDS [100]) method reported in 2012. In this method, the problem was defined as a boundary-optimisation algorithm. Here, the first iteration of superpixels is created by spacing them out uniformly over the image like a chequerboard. Then the pixels at the edges of the superpixels can switched to a neighbouring superpixel where they are more similar, and this is repeated until the boundaries stabilise. To account for longer distance movements, SEEDS creates a hierarchy of blocks of blocks. These blocks are groups of pixels of either (2×2) pixels or (3×3) pixels. Iteratively the blocks are grouped together until the size of the block is approximately the number of superpixels. In their work to produce superpixels they proposed an iterative block and pixel update using hill-climbing optimisation.

The creation of super-regions is becoming more popular, with methods arising such as Superpixels from MUlti-scale ReFinement of Super-regions (SMURFS) by Luengo et al. [70]. A super-region is a large collection of pixels and generally not a full object therefore super-regions are also classed as an over-segmenting method. They achieve this by iteratively using their graph-based split and merge scheme to yield superpixels.

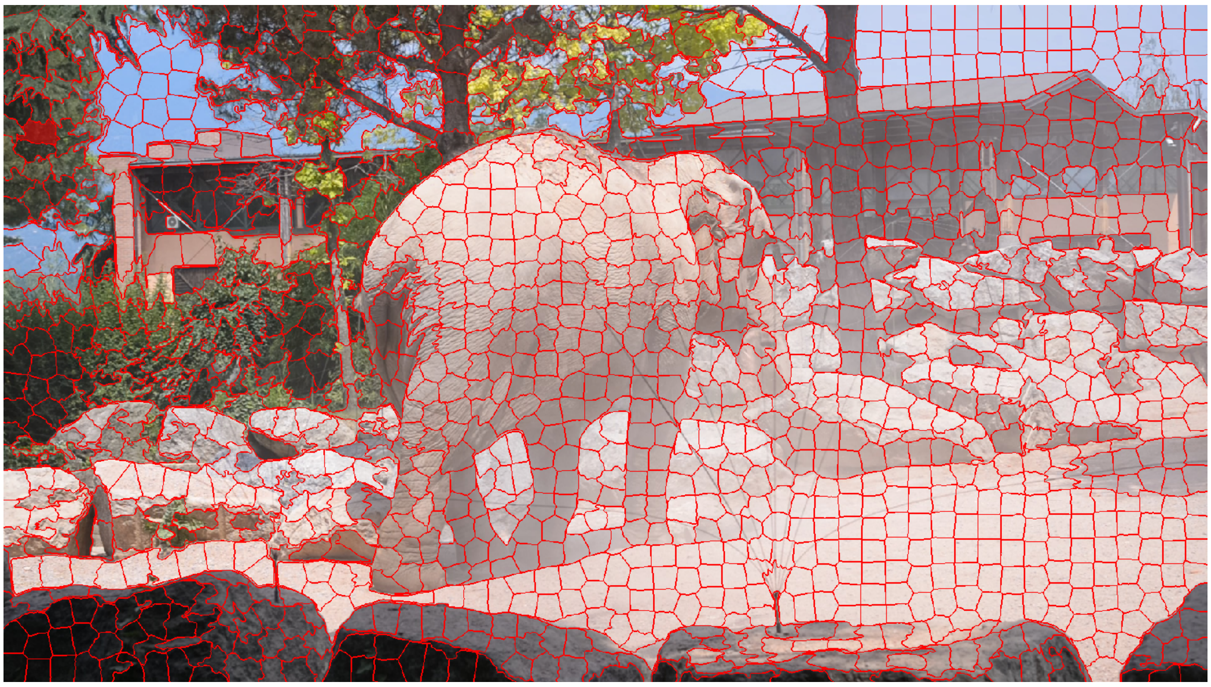


Figure 2.3: This figure shows an example of the SLIC algorithm performed on a sample from the DAVIS dataset [83]. This implementation is in Python using the scikit-image package with a target number of superpixels equal to 1500, a sigma value of 2 and a compactness value of 10. These parameters allow the algorithm to follow the structure of large objects such as the elephant and the rocks, but are less precise with smaller objects such as the foliage.

Then, the split scheme is applied over the pixel grid to separate large super-regions into superpixels. Next, the merge step is applied where superpixels are grouped together into more accurate super-regions.

Semantic segmentation

Semantic Segmentation is the process of segmenting a scene into *meaningful* segments. Each segment must belong to a single class that exists in a dictionary of possible objects. Multiple smaller segments may be needed to be joined together to form a single meaningful segment. An example of this is a car, as a car is made up of several parts such as its body, windows and wheels.

This thesis will use a neonatal dataset named Newborn Resus as a use-case for semantic segmentation. An example of semantic segmentation on this real world dataset can be seen in Figure 2.4, where each class in the dataset consists of smaller objects encapsulated together as a single class. Semantic segmentation of images and videos is continually growing in popularity as it has many uses from foreground/background detection [24] to brain tumour detection [50].

One of the most popular learned segmentation methods is the Fully Convolutional Network (FCN) developed by Long et al. [67] in 2015. In their work, an end-to-end network is used to produce pixel-by-pixel predictions. This produces both segmentations and classification. This is achieved by their skip architecture which makes use of both the deep coarse semantic information and the shallower appearance information.

Another variation on the learned segmentation is the Learned Watershed (LW) method by Wolf et al. [104]. This method expands on work such as that by Bai and Urtasun [4] by going a step further and building an end-to-end CNN that produces segmentations instead of the original energy maps used. LW keeps the original watershed algorithm unchanged by starting with seeds and iteratively adding the best pixels for each seed from their queues, whilst prioritising closer pixels.

Semantic segmentation has been approached by using handcrafted superpixel methods and combining superpixels to form super-regions/segments, and deep learning approaches such as going from RGB to semantic segmentation. However, deep-learned superpixels have not been used for semantic segmentation before. Making use of deep-learned superpixels would allow end-to-end learning. In Chapters 6 & 7 it is shown to be effective.

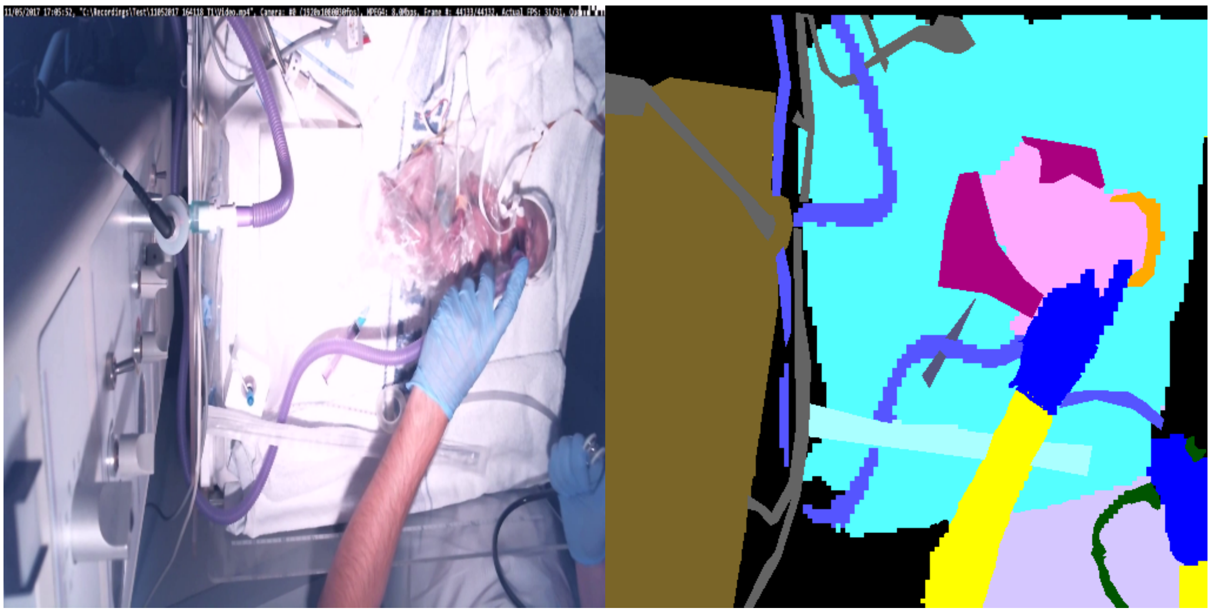


Figure 2.4: This figure shows an example of semantic segmentation on a sample from the Newborn Resus dataset. Unlike segmentation, each separate segment has a semantic meaning, it belongs to a particular class. For example the pink represents the baby, while blue represents the gloves.

2.2 Machine Learning

In this section I will provide formal definitions of machine learning theory terms that are used frequently throughout this thesis. Notation introduced here will be used throughout.

2.2.1 Overview

Machine learning is a subtopic of artificial intelligence, in which the machine learns from experiences, which we call observations, akin to a human, rather than being explicitly programmed on how to deal with a certain situation. Instead of having specific conditions to deal with the input information, a model is used. A model has a structure, and a list of parameters that can be changed. The machine changes the values of these parameters to reduce the error of predicting the labels, ideally on unseen data. The parameters in many machine learning models are called weights.

Machine learning can be used to solve multiple types of problems, but each type of problem requires a different approach to be used. Firstly, the problem must be formulated into one of the four following methods: supervised learning, unsupervised learning, semi-supervised learning, or reinforcement learning. Each method requires different data representations.

A collection of data is referred to as a dataset, and consists of features to describe each data point (instance). By examining these features, the machine is able to find patterns in the data. Some machine learning methods require the data to be labelled with either discrete or continuous labels. When labelling the dataset, it is recommended to label the data with the most accurate method possible, as, at least in first principle, the machine can only be as accurate as the data it was trained on[1]. If there is a large amount of uncertainty in the labelled data, the machine will not be able to find the relationships between features and their labels.

Once a machine learning method is chosen, the output of the method must be determined as either discrete or continuous, because each type of data requires different learning techniques. For discrete data, classification methods are used, whereas continuous data requires regression. Classification works by finding a separating hyperplane that separates the data points belonging to different classes by their features. Regression works by finding a hyperplane that best fits the training data, and that in effect represents the unknown function that relates the features to the continuously valued labels.

Supervised learning requires each data instance in the dataset to have at least one label, so that the machine knows to learn the relationship between these labels and the features of the dataset. Unsupervised learning is the opposite to supervised learning, where no ground truth data is given to the machine. Unsupervised methods can be used for data mining purposes. Instead of predicting labels, the machine is to find hidden structures and trends in the data and report these back. These trends could then be used as features to train a supervised method. Semi-supervised learning is somewhere in-between supervised and unsupervised learning. A small amount of labelled data is given to the machine, along with a much larger amount of unlabelled data, and the goal is

[1]It has been observed that Deep Learning Methods can make predictions with lower error than the label noise. However, the full reasons behind this are not fully understood yet.

to make use of the unlabelled in such a way that the classification accuracy is better than when only the labelled data is used. This method is used when it is costly to label data. Having access to a small amount of labelled data allows these methods to vastly improve the training accuracy of the model. Reinforcement learning is a methodology in which the machine is allowed to interact with an environment through given actions. These actions will have different consequences depending on the situation that the machine is currently in. The aim of reinforcement learning is to maximise the rewards from the actions taken, and minimise any penalties. Often a trial and error approach is used here to find out the best possible weight for the machine's model.

An example of classification is to imagine a dataset that consists of images of cats and dogs, and the desired outcome is for the machine to be able to automatically predict if a cat or dog is present in the image. Using a supervised method, the dataset must first be labelled by a human, with either 'cat' or 'dog' for each image in the dataset. The classifier would then cluster the images by their features, for example the pixel values of the image, and predict the classes. Whereas for an unsupervised method, the dataset would not be labelled, and the machine would have to determine itself how many classes there are, and which images fall into these classes.

An example of supervised regression is to imagine a dataset that consists of properties, with the desired outcome to be for the machine to be able to predict the price of the house pictured when valuing it. The dataset would consist of instances with features describing the house, such as: number of bedrooms, number of reception rooms, the garden size, and location, and labelled with the price the house was sold for. The machine finds the best hyper-plane to fit to the data. For any new instances given to the model, it looks where the instance would fall on this plane and thus predicts the house price.

Training a model requires the machine to either minimise an error (loss/cost) function or maximise a fitness (reward/profit) function. The most common error function is the mean squared error (MSE) function, equation 2.7. MSE is used for regression problems as it forces predictions that have a greater difference to the ground truth to have a greater error than predictions that are close to the ground truth. The most common classification loss function is the hinge loss, equation 2.8. The hinge loss is a maximum margin loss function, this means that the line that divides the data into classes is as far away from the data points as possible whilst keeping the data as separable as possible. The margin comes from $(1 - z) \times T$, where z is the distance from the line, and T is the true label. A standard misclassification loss function, where each misclassification is worth one, does not take into consideration how wrong a result is, whereas the hinge loss takes the distance from the line as the error. Also the margin is used to give instances that are close to the line an error.

$$MSE = \frac{1}{n}\sum_{i-1}^{n}(Y_i - \hat{Y_i})^2 \tag{2.7}$$

$$HingeLoss = max\{0, 1 - Y - \hat{z}\} \tag{2.8}$$

When training a model, the machine is fed the entire dataset and for each data instance, the error is calculated. The model's parameters can then be updated by a function of the error value and some bias. Updating the weight should improve the model for future predictions. The machine is fed the dataset multiple times until the error no longer changes, this is when it is said to have converged. Each time the machine is shown the entirety of a dataset,this is referred to as completing an epoch. When it has converged, we can then say how many epochs it took to converge. Whilst predicting, other performance metrics can be calculated, such as accuracy. Accuracy can be compared against other methods to see which method performs best.

However, when evaluating a model, it is bad practice to evaluate the model on the same data it has been trained on as it may have learnt the features of the training data very well, and therefore will have an almost perfect accuracy. This is referred to as over-fitting to the data. If the same model is then tested on unseen data and performs poorly, it is said to have poor generalisation. Generalisation is the main goal of a model so it can be used on unseen data with good reliability. To combat problems with this, there are several techniques available. The most fundamental of these techniques is to split the dataset into three partitions: training, validation, and testing.

These three splits of data allow the algorithm to update the weight to prevent over-fitting to training data. When splitting the data like this, an epoch now refers to the machine sampling all of the training and validation data. Each epoch is now split into two phases, training and validation. During the training phase, the machine makes predictions on the training sample and calculates its error, then updates its weights in the model. Once the model has seen all of the training data for a given epoch, it is then shown the validation data. Whilst predicting on the validation data, the model is not allowed to update its weights. A common strategy is to tune the hyper-parameters of the model during the validation phase for the next epoch, such as the structure of the model. Additionally, the error is calculated for the validation phase, and this error is used to determine when the model has converged. After the model has converged, the test data is fed through the model. The test data is used to simulate future unseen data, and is used only on the final model, thus no alteration to the model weights are made during this phase. The performance metrics, such as accuracy, of the model on the test data are used to compare different models and evaluated to see how generalisable the model is.

2.2.2 Linear Regression

One of the most simple machine learning techniques is linear regression. Linear regression is the process of finding the linear relationship that most closely represents the training data, or finding the relationship between the dependant variable (measured output) and the independent variable (controlled input). When only one independent variable is used, this is referred to as simple linear regression. An example of this is measuring the value of a house (dependant variable) in relation to the number of bedrooms and reception rooms (independent variable), Figure 2.5. Using the hypothesis $h_w(x) = w_1x + w_0$, where w_i are the intrinsic parameters to model the training data, t, the parameters are found by minimising an error function for all data points i in t. Most commonly the least squares approach (L2 norm), equation 2.10, is used over the absolute error (L1 norm), equation 2.9, because the absolute error is not differential. A differential function is a parabolic which means that it converges faster, because the error is larger between steps.

$$J(w_0, w_1) = \min_{w_0 w_1} \frac{1}{n} \sum_i |h_w(x_i) - t_i| \tag{2.9}$$

$$J(w_0, w_1) = \min_{w_0 w_1} \frac{1}{2n} \sum_{i=i}^{n} (h_w(x_i) - t_i)^2 \tag{2.10}$$

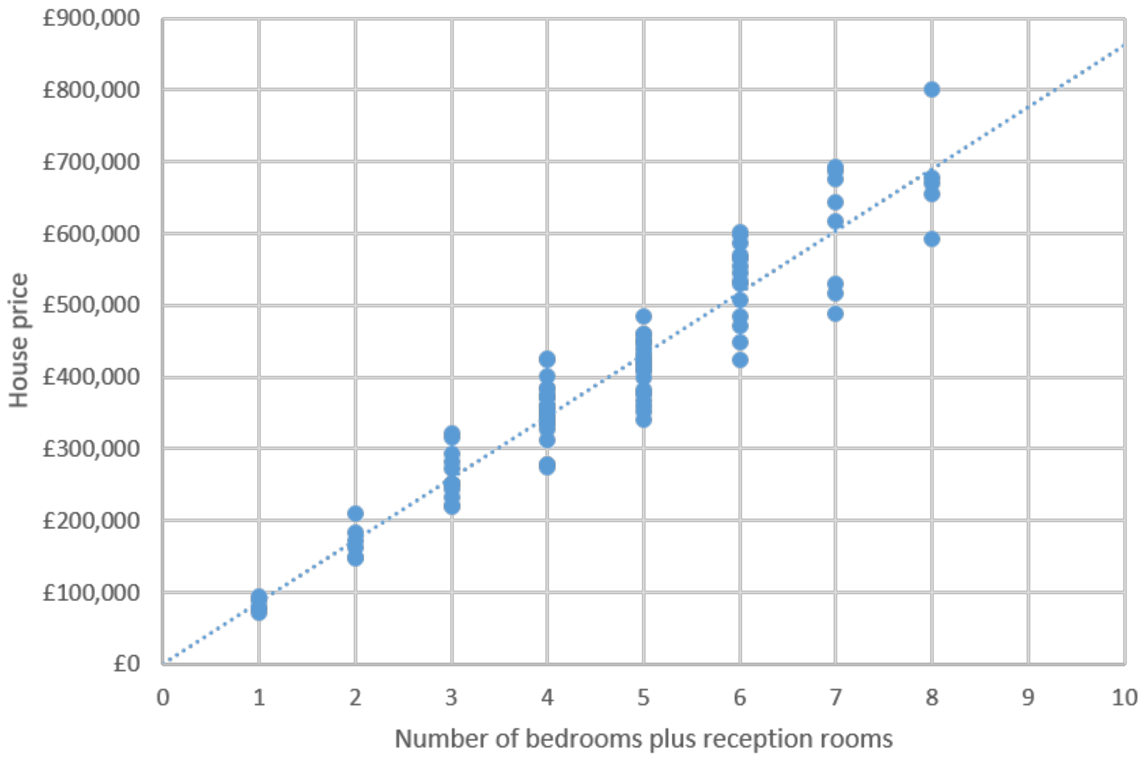


Figure 2.5: This figure shows a toy dataset of house price compared against the number of bedrooms plus reception rooms for a given area.

When there is more than one independent variable, this is referred to as multiple linear regression. However, when the desired outcome is to predict multiple dependent variables, generally using multiple independent variables, this is referred to as multivariate linear regression. The simplest form of this is to take the summation of features and their weight terms, equation 2.11. If a more complex model is needed to represent the data, a polynomial function can be used, 2.12.

$$h(x, w) = w_0 + w_1x_1 + ... + w_jx_j + ... + w_nx_n \tag{2.11}$$

$$h(x, w) = w_0 + w_1x_1 + w_2x_1^2 + ... + w_kx_1^k + ... + w_{k*d}x_d^k \tag{2.12}$$

Each type of linear regression uses a cost function to evaluate how well the current parameters fit the data, but it is also necessary to know how to change these parameters and know when to stop. Here, the gradient descent algorithm is used. The gradient descent algorithm works by starting with some initial values for w_i, then changing these values and calculating the gradient between the two errors that are given from $J(w_i)$. Each w_i is updated simultaneously by updating each by the same error. To calculate the amount to change the weights by, the value of the gradient is multiplied by a predefined learning rate. The learning rate decides how big of a step to take when change the weights. If the learning rate is too high, then the loss can bounce around the minimum or possibly diverge. If the learning rate is too low, then the update may take a long time to converge (gradient is equal to zero).

There are two key problems that gradient descent faces. These are called local minimums and plateaus. Local minimums are where the gradient descent converges with all directions from the converged point having worse errors, but it is not the global minimum for the data. Plateaus are when the gradient descent converges and is either faced with all directions either being worse or the same as the current position, but again, not the global minimum. To combat both of these, multiple runs of the algorithm are carried out, but with differing random initial values for the weights. This means during training the model will have different parameter values, therefore gradient descent will converge in different locations. The best performing run is taken as the final model.

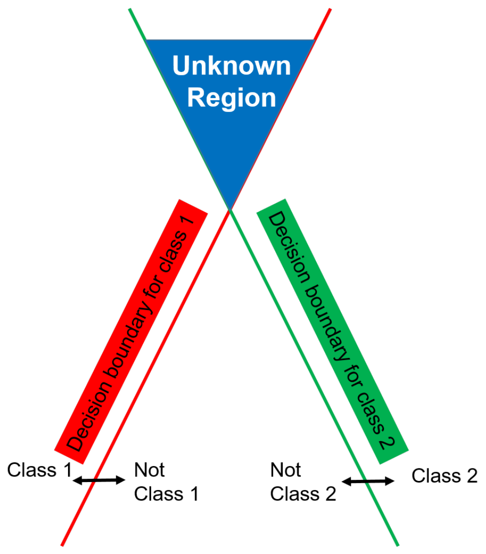
2.2.3 Linear Classification

Linear classifiers generally work by classifying an input vector x into one of K discrete, disjointed classes. Therefore the input space can be divided into decision regions where the lines that separate these regions are called decision boundaries. These decision boundaries are defined as a linear function of x. Datasets whose classes can be separated by a linear decision boundaries are said to be linearly separable. If a dataset is separable but not by a linear decision boundary then it is said to be non-linear. If a dataset is unable to be separated then it is said to be inseparable. The data does not just have to be separable by a line in a 2D space, it can be separable by a plane in 3D or a hyper-plane in more than the three dimensions.

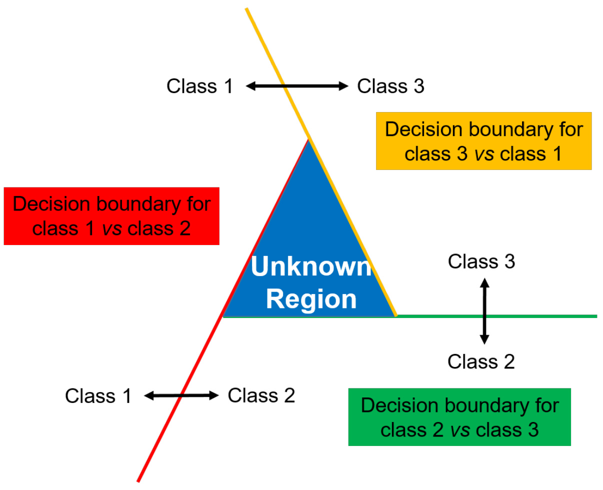
The most simple case of classification is binary classification. This is where the data can be said to belong to one of two sets, either the positive or negative set. This binary classification is represented by $t \in 0, 1$, where zero equals the negative class, and one equals the positive class. For a discrete multi-class representation, $t \in 0, 1, ..., N$ where N is the number of classes. When training these are represented as vectors containing zeros and ones. An example with 10 classes where the data is labelled as class 6, the vector would be $t = [0, 0, 0, 0, 0, 1, 0, 0, 0, 0]$.

The hypothesis used for linear regression can be converted for classification by applying a non-linear activation function, f, to obtain linear decision boundaries, $hw(x) = f(w1x + w0)$. Discriminate functions are functions that take a linear function and produce a decision boundary. For example, given the function $y(x)$, if $y(x) > 0$ equal the positive class, and $y(x) \leq 0$ equal the negative class, the decision boundary would be $h(x) = 0$.

For multi-class linear classification we simply combine multiple binary linear classifiers. However, there is a problem when doing this, and that is how to combine the classifiers. Firstly there is the one-vs-all method, where each classifier predicts either C_k or $notC_k$. This leads to $N - 1$ classifiers, and some areas of the decision space are left ambiguous, Figure 2.6. The other method is to perform one-vs-one classification between classes, this produces $\frac{N(N-1)}{2}$ classifiers. A majority vote is then used to classify the input. The one-vs-one method again leaves some areas of ambiguity, Figure 2.6.



(A)


Class 1
Class 3
Decision boundary for class 3 *vs* class 1
Decision boundary for class 1 *vs* class 2
Unknown Region
Class 3
Class 2
Class 1
Class 2
Decision boundary for class 2 *vs* class 3

(B)

Figure 2.6: This figure shows linear ambiguity for multi-class classification using binary linear classifiers. Figure (A) shows the region in feature space where the one-vs-all, and figure (B) shows the ambiguous region for one-vs-one.

This ambiguity can be avoided by forming a single k-class classifier that consists of N linear functions, $y_k(x) = w_{k1}x + w_{k0}$, then assigning the input point x to C_k if $y_k > y_j(x)$ for all $j \neq k$. The decision boundaries are then where $y_k(x) = y_j(x)$.

2.2.4 Support Vector Machines

Support vector machines (SVMs) are a type of kernel method. A kernel method tries to map a non-linearly separable input to a higher dimensionality, with the hope to find a space where the data is now linearly separable, Figure 2.7, and can be back-projected to the original dimensionality. However, these kernel methods do not actually map to the input data to this higher dimensionality, but instead return the distance between all elements in this new space. This is referred to as the kernel trick.

SVMs are max-margin classifiers that use a kernel to determine the decision boundaries. When data is linearly separable there are often multiple solutions that separate the data, but not all solutions are optimal. A max-margin classifier tries to find the decision boundary that is as far away from data points as possible whist still correctly classifying, allowing for the greatest generalisation. For a two class problem, equation 2.13 defines the classification, where $\theta(x)$ is the feature space, and the labels are $t \in \{-1, 1\}$. Where $y(x) \geq 0$ then $t = 1$ else $t = -1$.

To find the maximum margin, the distance of each point to a decision boundary must be calculated. The output of the classification function can be used to calculate the points. For a decision boundary, the equation 2.13 is used, and for a hyperplane, the equation is 2.14. Assuming all points are correctly classified, the distance becomes 2.15. This translates to finding the values for b and w which maximise this distance, 2.16.

$$y(x) = w^T\theta(x) + b \tag{2.13}$$

$$\frac{|y(x)}{||w||} \tag{2.14}$$

$$\frac{t_n y(x)}{||w||} = \frac{t_n(w^T\theta(x_n) + b)}{||w||} \tag{2.15}$$

$$\arg\max_{w,b}\left\{\frac{1}{||w||}\min_n[t_n(w^T\theta(x_n) + b)]\right\} \tag{2.16}$$

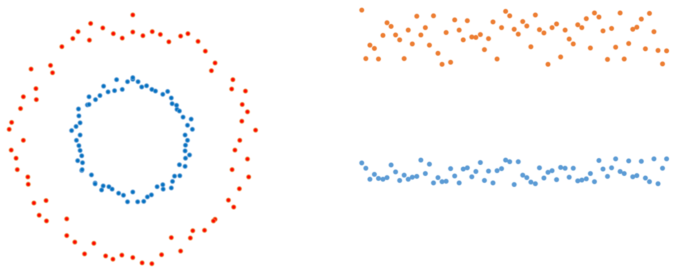
It is possible to hand-craft kernels for each specific problem, however this is a very hard task. Instead one of the standard kernels are used: the linear kernel (equation 2.17), the polynomial kernel (equation 2.18) or the gaussian radial basis function kernel (equation 2.19).

$$k(x, x') = x^T x' \tag{2.17}$$

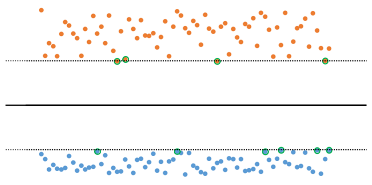
$$k(x, x') = (x^T x' + c)^M \tag{2.18}$$

$$k(x, x') = exp\left(\frac{-||x - x'||^2}{2\sigma^2}\right) \tag{2.19}$$

SVMs are sparse methods, meaning that they only take on as little of the training points as is needed to form a good representation of the structure of the data. These are referred to as the support vectors. Each support vector is used to define where the decision boundary lies, the fewer the better, and can be seen in Figure 2.7.



(A) (B)



(C)

Figure 2.7: (A) shows an example of toy data where the two classes are not linearly separable. (B) shows the data from (A) projected into a higher dimensionality via a kernel. (C) shows the max margin for this data and its support vectors. The solid line represents the decision boundary, the dotted lines represent the margins, and the highlighted points represent the support vectors.

Not all data is linearly separable, so SVMs introduce a slack variable called c which allows some data points to be misclassified or lie inside of the margins. Any point that lies within the margins gets added to the support vectors. The slack variable can range between zero and infinity. If the slack is set to infinity, this enforces a hard margin. A hard margin is where no points can be misclassified or lie inside the margins. This produces a model that will over-fit to the data. The decision boundary for a hard margin would snake between data points to try and separate them into the different classes, or create small islands around each point, making the model not generalisable. At the other extreme setting, the slack variable close to, or even, zero creates a soft margin. A soft margin does not penalise misclassification, meaning that effectively all data points can be included as support vectors and misclassification is not accounted for.

SVMs are inherently binary classifiers, and suffer from the same issue of ambiguity that linear classifiers have when performing multi-class classification by combining multiple classifiers in either a one-vs-all methodology or one-vs-one methodology. There is the additional problem that multiple binary classifiers have when using the one-vs-all methodology, which is that there is a class imbalance, because the positive class is generally much smaller than the combined negative class. Having unbalanced data for training any machine learning model means that the model may become biased to predicting one class over another as this will produce the lowest error. For example if there is a dataset of 100 images, in which 95 of them are of dogs and the other five contain cats, the model might learn to predict only dogs as this results in a 95% accuracy and low error rate, rather than trying to correctly classify the cats, which may lead to some dogs being misclassified too, increasing the error.

2.2.5 Artificial Neural Networks

Artificial neural networks (ANNs), unlike other methods such as SVMs, have a fixed number of basis functions. ANNs use parametric versions of the basis function that allow these parameter values to be updated during training. For many applications, an ANN model is more compact than an equivalent SVM model which has the same generalisation performance. This means that the ANN model will be faster to evaluate, however this comes at a price. The price is that the model is no longer convex (multiple minimums). Due to the evaluation speed being much greater, it is often advantageous in real world scenarios to spend more computing power on training the model, than when the model is being used on new data. The most successful ANN is the feed-forward neural network (multilayer perceptron), which consists of multiple layers of logistic regression models, rather than multiple perceptrons.

ANNs take inspiration from the biological neural networks inside brains. A neural network is made up of many neurons that are connected together, and, if activated by their input, send an output. This is modelled by having layers of neurons called nodes and each node in a layer is connected to all nodes in the following layers by synapses. The most simple ANN has three layers: an input layer, a hidden layer, and an output layer. The input layer has as many nodes as input features. The hidden layer's number of nodes is a hyper-parameter that can be optimised during training. Hidden nodes are referred to as this because the parameters to the functions in these node are unknown and trained. Finally, the output layer consists of the same number of nodes as classes to predict. Each node from each layer is connected to every node in the next layer. The output of the node is assigned a weight per synapse and these weights are also updated during training. Each node in the hidden layer consists of a summed input of all of the inputs to the node, an activation function, and an output, Figure 2.8.

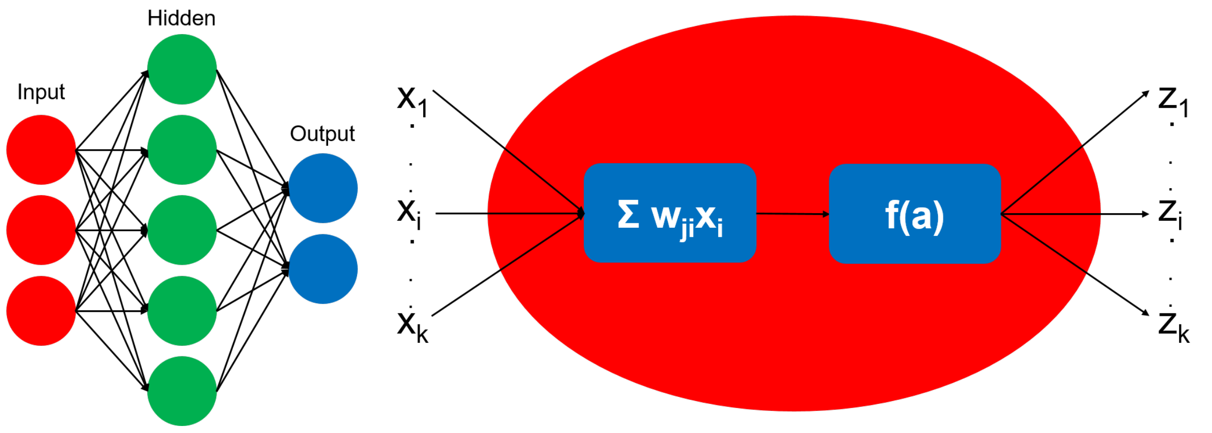


Figure 2.8: On the left a neural network with three layers is depicted. It has an input layer, a hidden layer, and an output layer. The input layer has three nodes for inputting data into the network. There is a single hidden layer consisting of five nodes, and the output layer has two nodes. On the right is the representation of a node. Each node takes in all inputs from the previous layer, applies a weighted sum, then applies an activation function to create the input for the next layers.

$$\sigma\left(\begin{bmatrix} w_{0,0} & \dots & w_{0,n} \\ \vdots & \ddots & \vdots \\ w_{k,0} & \dots & w_{k,n} \end{bmatrix}\begin{bmatrix} a_0^{(0)} \\ \vdots \\ a_n^{(0)} \end{bmatrix} + \begin{bmatrix} b_0 \\ \vdots \\ b_n \end{bmatrix}\right) \tag{2.20}$$

$$a^{(1)} = \sigma(Wa^{(0)} + b) \tag{2.21}$$

There are several activation functions that can be used for σ, such as stepped sigmoid, or tanh, but the most common activation function is the rectified linear unit (ReLU). ReLU works by clamping values between zero and positive infinity, thus anything less than zero returns zeros. For regression and binary classification there is only one output node, whereas multi-class classification has a node per class. For each of the three cases, a different output activation function is used. Regression uses the identity activation function, where the output is equal to the weighted input of this final node. Binary classification generally uses the stepped sigmoid function, which activates when over a given threshold. Soft-max function is used for multi-class classification, and works by normalising the output vector into a probability distribution. This is calculated with the equation 2.22, where the output is now between zero and one, and all these values add up to one, meaning the class prediction with the highest value is the overall prediction.

$$y(k) = \frac{exp(a_k)}{\sum_j exp(a_j)} \tag{2.22}$$

A common dataset used for training a primary classification neural network is the modified National Institute of Standards and Technology (MNIST) [21], which consists of grey-scale images of hand written digits at a resolution of (28×28) pixels. This dataset has 10 classes, $[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]$. Consider a multilayer perceptron with an input of 784, two hidden layers with 5 nodes each, and 10 nodes for the output layer, Figure 2.9. To train this network, each data point in the dataset is fed forward through the network, and each will get a prediction vector, eg $[0.1, 0.6, 0.2, 0.3, 0.4, 0.3, 0.3, 0.6, 0.2, 0.5]$. The error of each prediction is then calculated and summed together. For example the MSE could be used to calculate the error. The equation for this is 2.7. For a network to train, it must minimise the selected cost function, MSE in this case, by updating the parameters of the network's model. During training, only the intrinsic parameters; weights and biases, are updated. To update these parameters we minimise the gradient of the error function, and then backpropagate the error through the network. Backpropagation is the technique

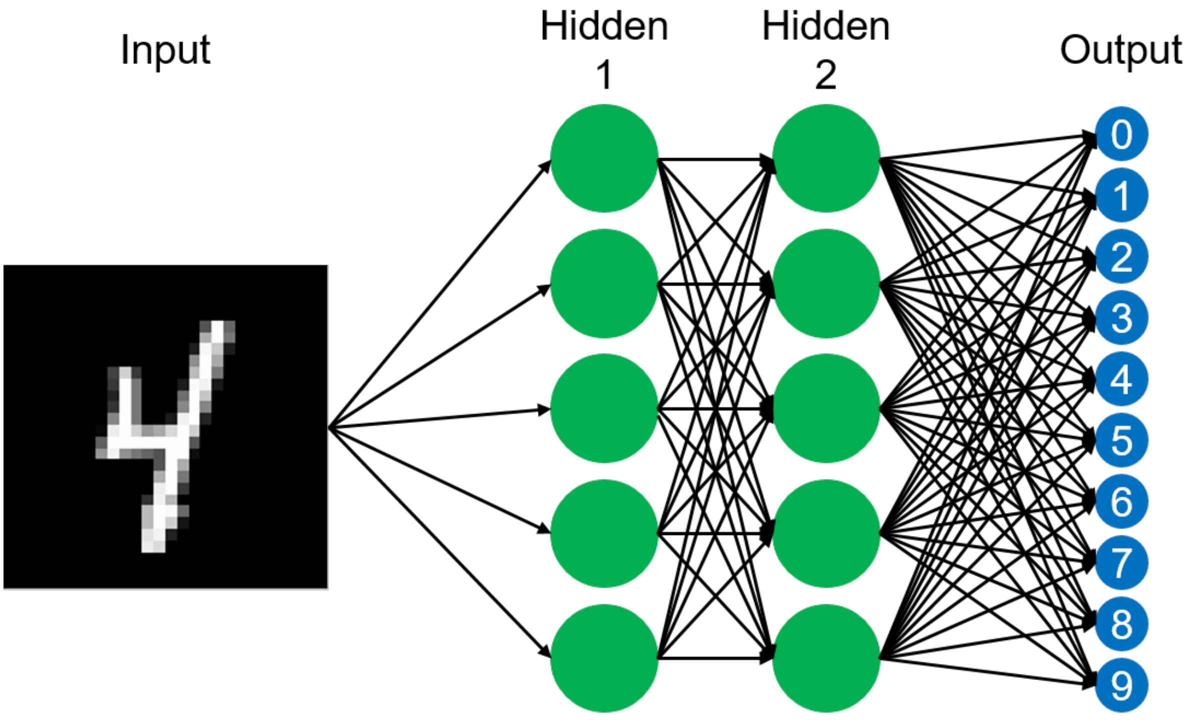


Figure 2.9: This figure shows an example ANN structure for tackling the MNIST dataset. It consists of an input of 784 nodes, two hidden layers with five nodes each, and an output of 10 nodes. There are a total of 3995 weights $(784 \times 5 + 5 \times 5 + 5 \times 10)$, and 20 biases $(5 + 5 + 10)$, giving a total of 4015 trainable parameters.

of cascading the error from the output layer back through each layer of the network to update the intrinsic parameters. To calculate the amount each parameter must be updated, a gradient descent algorithm is used. The gradient descent algorithm calculates the derivatives of the cost function with respect to the intrinsic parameters. Then, small steps are repeatedly taken towards the steepest descent until the error function converges. The values returned by the gradient descent function inform the network of which direction to optimise their weights towards, and how much to change the values by. The steeper the slope of the gradient, the greater the change to the weights is.

The gradient descent returns a vector of gradients, where each value corresponds to each output node. This vector is applied to each weight and bias connected to the output layer to minimise the error. Rather than updating the network with each input and its error, all of the data is fed through the network and an average error, and thus average gradient, is calculated. The average gradient informs the previous weights and biases of how far away it is from being on average correct. Therefore nodes which correspond to a certain class will be updated more aggressively when far away from the correct answer than nodes that are not helpful to the class. This is then repeated for the previous layers to update the weights and biases and enforce which nodes in that layer are most important per class.

For this to work well, large amounts of training data are needed, but unfortunately doing true gradient descent with large amounts of data is very costly. This is because it takes a long time to compute all of the gradients and then to update all of the weights and biases. Often stochastic gradient descent is used instead. This stochastic gradient descent differs to gradient descent by working in mini batches. The training data is shuffled each epoch and split into small batches, where the number of instances in a batch is predefined. Once the network has gone through a mini batch, it performs backpropagation. This method does not give the most optimal step down the gradient of the cost function, but it does give a close approximation at a considerable speed boost.

A key problem that gradient base learning suffers from is called the vanishing gradient. This occurs when the gradient that is being backpropagated becomes minuscule in earlier layers, meaning that small changes are made to the parameters in these layers. This equates to large changes in the early layers having little to no influence upon the output. The vanishing gradient commonly occurs when using activation functions to convert the input into a non-linear, smaller output space. If the activation function converts the input into a range between zero and one, such as the sigmoid function, then larger inputs are mapped to a region where the distance between them is very small. Thus when calculating the gradients between them, they also become very small. The ReLU activation function solves this issue by not squashing the input into a small region, but it does discard any negative numbers, thus allowing for a more sensible gradient to be computed.

2.2.6 Probabilistic Graphical Models

Probabilistic graphical models are graph structures for representing the probability of data occurrence. The edges of the graphs represent the probabilistic relationships between two nodes, and the nodes represent random variables or groups of variables. Nodes can be either a single variable, or a group of variables. In the book Pattern Recognition and Machine Learning by Bishop [9], in Chapter 8 he states that there are three useful properties of probabilistic graphical models, with the primary factor being that probabilistic graphical models are an easy way to visualise probabilistic models.

There are two types of probabilistic graphical models: a directed graph, and an un-directed graph. A directed graph is also known as a Bayesian network. Bayesian networks have directions to the links in the graphs illustrated by arrows. The links show the relationship between the random variables. Additionally, Bayesian networks are acyclic, meaning that there are no routes that loop back to a previous node. Un-directed graphs are known as Markov Random Fields (MRFs). These graphs have no direction to their edges and represent the soft connections between variables. MRFs can be cyclic.

A simple example of a Bayesian network is to take the joint probability of $p(a, b, c)$ and to apply the product rule to get $p(a, b, c) = p(c|a, b)p(b|a)p(a)$. This can be represented in a probabilistic graphical model, as seen in Figure 2.10.A. A more complex network of $p(x_1)p(x_2)p(x_3)p(x_4)p(x_5|x_1)p(x_6|x_1, x_2, x_3)p(x_7|x_4)p(p_8|x_6, x_7)$ can be seen in Figure 2.10.B, which illustrates the point made by Bishop [9] about these graphs making visualisation of the model easier. These models can be used in inference in two ways: firstly, calculating the joint probability of nodes given an assignment to the variables, and secondly, they can be used to infer a task's probability, given a subset of the variables.

Figure 2.10.B depicts a Markov blanket for X_6 which is in the yellow region. It consists of the parents of the node, the children of a node, all co-parents of the children. The Markov blanket represents the set of nodes that isolate a node from the rest of the graph, and describes the relationship of the node to the rest of the network. Therefore the Markov blanket illustrates which nodes either depend on a given node, or the nodes that directly contribute to the state of a given node.

MRFs must satisfy the Markov property: that the process is a memory-less, stochastic (random) process. This means that the future states of the system must not rely on past states, only the present state. A key property of MRFs is that if there exists a set of nodes that can separate a given node from another node, then these two nodes are conditionally independent. Consider three sets: A, B, and C. A is said to be conditionally independent of B if there are no edges connecting A and B, and all routes joining A to B go through C. Therefore if C was removed from the network, there would be no connections from A to B. This can be written $A \perp\!\!\!\perp B|C$. For a MRF, the Markov blanket is the smallest set of nodes that separate the given node from all other nodes in a graph. For example,

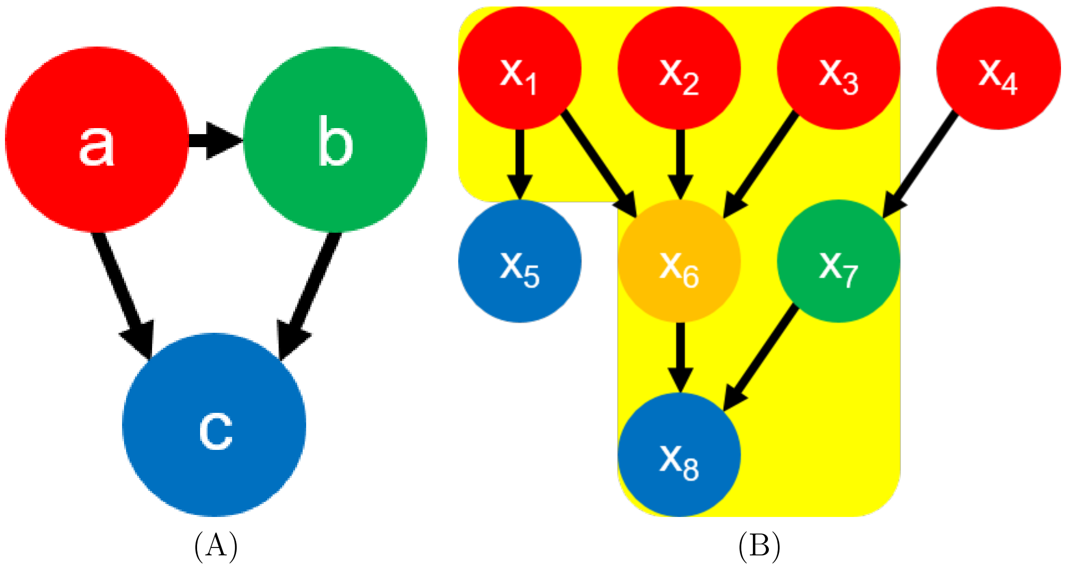


Figure 2.10: Figure (A) shows a simple probabilistic graphical model for the join probability of $p(a, b, c)$, and Figure (B) shows a more complex $p(x_1)p(x_2)p(x_3)p(x_4)p(x_5|x_1)p(x_6|x_1, x_2, x_3)p(x_7|x_4)p(p_8|x_6, x_7)$ with a Markov blanket in yellow.

for a single pixel in an image, the Markov blanket for a MRF representation of this image would be the pixel's four neighbours. A common use for MRFs is de-noising of images as prior knowledge of the strength of noise can be captured by the MRF. Take two arrays, one y for the noisy image, and another x for an unknown, noise-free image. We know that x_i, where i in the index in the arrays, and y_i have the same intensity that the probability should be high. Similarly for x_i and x_j, where x_j is a neighbour of x_i, the probability of these pixels being the same intensity should be high. Solving for these constraints generates an image where noisy pixels are removed if they do not match their neighbouring pixels.

Markov chains are similar to Bayesian networks, however Markov chains allow for cyclic networks. A Markov chain describes the probability of moving from one state to another. For example, if there are two states A and B, it is possible to move between these two states, and you are currently in state A. There are two possible transitions, moving from A to B with probability $p(A_B)$, and staying in state A with probability $p(A_A)$. If you move to state B there are two more possibilities, $p(B_B)$ and $p(B_A)$. This allows the calculation of the possibility of moving between these states over time.

Hidden Markov models (HMMs) are used to represent a process that uses Markov chains with unobservable states. These hidden states assume there is another process that is dependent on the output current model. The goal of a HMM is to learn the current model by observing the other process. HMMs also assume that the other process does not rely on the current process. Hidden Markov random field is a variation on HMM where instead of using Markov chains, MRFs are used. Conditional random fields (CRFs) are a special case of MRF where the previous (posterior) information for each variable in the MRF are given by a dataset and the probabilities are computed including the posterior. Alternatively, HMM are unlike MRFs because the prior information is explicit for these types of models [10].

2.3 Deep Learning

2.3.1 Overview

Deep learning is defined as an ANN with greater than one hidden layer between the input and output layers. Deep networks allow more abstract features to be learnt in the deeper layers than a single hidden layer ANN, which effectively maps feature to output. A major problem that deep learning faces is a lack of data. Since the models have more layers, this means they also have more intrinsic parameters. When there are more intrinsic parameters than training samples, the model has the potential to over-fit to the training data. Zhang et al. [107] describe a theorem in which an ANN with two layers with ReLU activation function and $2n + d$ weights is capable of representing any function for a given dataset of size n with dimensionality d. However, having more layers and thus more parameters allows for more complex functions to be represented with fewer parameters than are needed for the function. A common method to deal with this overfitting is to add regularisation to the network.

There are many forms of regulations to reduce overfitting. The most common variants are: L2 norm, early stopping, data augmentation, and dropout. L2 norm is a regulation term that is added to the error function of the network. This is shown in equation 2.23. L2 norm is sometimes referred to as weighted decay, as it forces the weights in the network to tend towards zero if λ is larger.

$$j(w_0, w_1) = h(w_0, w_1) + \frac{\lambda}{2m}||w||_2^2 \tag{2.23}$$

$$||w||_2^2 = \sum_{j=1}^{n_x} w_j^2 \tag{2.24}$$

Early stopping is a technique where the error is plotted per epoch for the validation set whilst training the network. When the gradient of the cost functions for the validation set starts to increase, generally after it has reached its optimal step, the network is then forced to stop learning and the parameters at the best epoch are used for the final model. This is then tested on the test set. This prevents the network from continuing to get better results on the training set, whilst not improving the generalisation of the network on the validation set.

Another way to combat this is to gather more training data. However in practice this is not always feasible, both in regards to time and cost. Thus data augmentation can be used. This allows more data to be created from the existing training set by performing trivial transformations. The most common transformation used for image based problems are: flipping across the vertical axis, cropping round the subject, zooming the image, and rotations of the image. Each of these transformations are only done by small percentages so as to not lose important information from the input. Data augmentation does result in more training examples, but it does not solve the issue of variety in the dataset which can only truly be solved by using new data.

Dropout is a technique applied to individual layers of the network. It allows some of the nodes in the network to be randomly deactivated for a given epoch. The randomness is defined by the threshold which is set per layer. Generally, a higher rate of dropout is used for layers with more connections, and lower or no dropout is used for layers with fewer connections. The input layer very rarely has dropout applied to it because it is best to provide as much information as possible to the network. This deactivation of the nodes forces the network to learn how to deal with missing information, by not relying on specific activations of features in the previous layer, but instead relying more heavily on a wider range of node activations.

So far each node in a layer of the network has been fully connected to every node in the following layer. These are called fully connected layers. However, these are not the only types of layers that can be used in a deep neural network. In the MNIST example, the image was flattened into a vector of 784 input features, and fed through two fully connected layers. A network does not have to only take vectors as inputs; it can also take a matrices, but a new type of layer must be used.

2.3.2 Convolutional Neural Networks

There are many types of layers that can be used in a deep network, with each layer being used to learn different types of features from the training data. When working with images, the most common layer is the convolutional layer, because it can process inputs with a grid-like topology. Convolutional neural networks (CNNs) date back to the late 1980's when LeCun et al. [60] created the CNN to identify handwritten postcodes.

Convolutional layers apply the K filter to an image using the sliding window technique, Figure 2.11. Each filter is like a tiny neural network that is applied to the whole image without updating the weights. All of the filters in the same convolutional layer have the same size and stride. The stride is how many pixels the sliding windows move by each step. If the stride is equal to one, the filter output will have roughly the same height and width as the input but with K dimensions / feature maps, whereas if the stride is two, the output is roughly half.

These are only rough estimates due to the initialisation of the filters. The filter starts so its edges meet the input edges in the upper left corner. The filter does not go outside of the input image. The filter then slides across the image to the opposite side. If, when moving across, the filter goes outside the image, it skips these pixels and moves onto the next row. This type of convolution is called a valid convolutional layer. If it is necessary to include the edge pixels, a technique called padding can be used. Padding is where the image is expanded by adding zeros around the edges of the image so the kernel can visit every pixel. Adding zeros means that the kernels are not activated for these pixels.

Each filter is referred to as a kernel. The weight values in the kernel are multiplied by the input values. The summed output is then passed to an activation function, and this output is stored in the corresponding location in the output feature maps. The weights in the kernel are what are updated during the backpropagation. In turn, this changes the output feature maps to something more meaningful to the network.

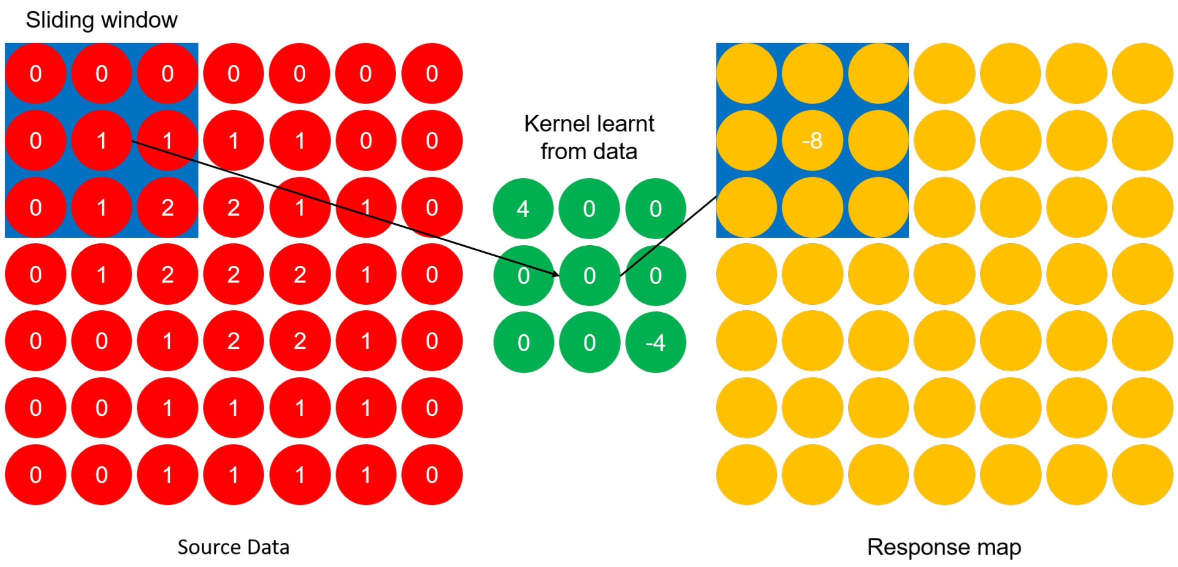


Figure 2.11: In this figure the blue box represents the sliding window of the CNN. Each pixel in the sliding window from the data source has the learnt kernel applied to the window of data, and the output of the kernel is then stored in the response map for the central pixel of the window. If there are multiple kernels, multiple response maps are created. If there are multiple channels for in-source data, the kernels are applied to all channels simultaneously.

When working with CNNs, the number of weights quickly increases with the addition of more convolutional layers. To reduce the number of feature activation, pooling layers are used. Pooling layers work by applying a single filter with a stride greater than one to each feature map. The most common type of pooling is the max pooling layer, which usually has a stride of two and a filter size of (2×2). Max pooling takes the largest value in the filter and this becomes the output for the current feature map at that location. Max pooling with this size and stride reduces the input height and width by half, Figure 2.12.

Other examples of pooling layers are average pooling, which is taking the average of the values the filter region, and L2 pooling, which is finding the square root of the sum of the squares in the filter region.

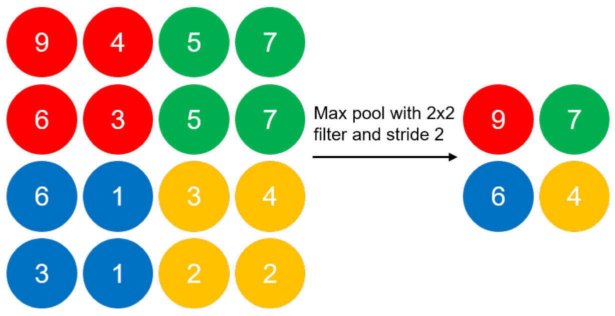


Figure 2.12: This figure shows a small example of max pooling with a (2×2) filter and stride of 2. For each window in the max pooling, the data has been colour coded. The max number in each window is then stored in a response which is half the size of the input source.

2.3.3 Recurrent Neural Networks

Thus far the methodologies discussed have been focused on non-sequential data, or in other words independent and identically distributed (i.i.d) data. This assumption means that there is no correlation between two points, only that they belong to the same distribution. Sequential datasets generally consist of time-dependent data, where the following sample is partially or fully dependent on the sample(s) before it. Non-temporal sequence data are in the form of series of data, such as sentences written in English, the base pairs in DNA, or sequences of actions in a planning problem.

With regards to deep learning, recurrent neural networks (RNN) are the most common method of dealing with sequence data. These types of networks use layers which allow for memory to be added into the network, therefore allowing sequences in the data to be learnt. RNNs use a directed graph approach, similar to that described in the Bayesian networks Chapter 2.2.6. The edges of the network represent the sequence order. RNNs contain a cyclic property called the hidden state, in which the previous data is passed back to the node, Figure 2.13. The hidden state allows the RNN to remember sequence data of varying lengths, in turn allowing for previous data to affect the next prediction. The cyclic hidden state can be unfolded such that each part in the sequence can be given its own layer. The input to an RNN is combined with the hidden state from the previous layer and fed through a tanh function to force the output to be between negative one and one. RNNs struggle with the vanishing gradient problem because the longer the input is, the less influence the earlier time steps have on the final prediction. In turn, during backpropagation the gradient becomes too small to make meaningful updates to the earlier layers. Imagine each next time step the new input gets is weighted 50 percent,

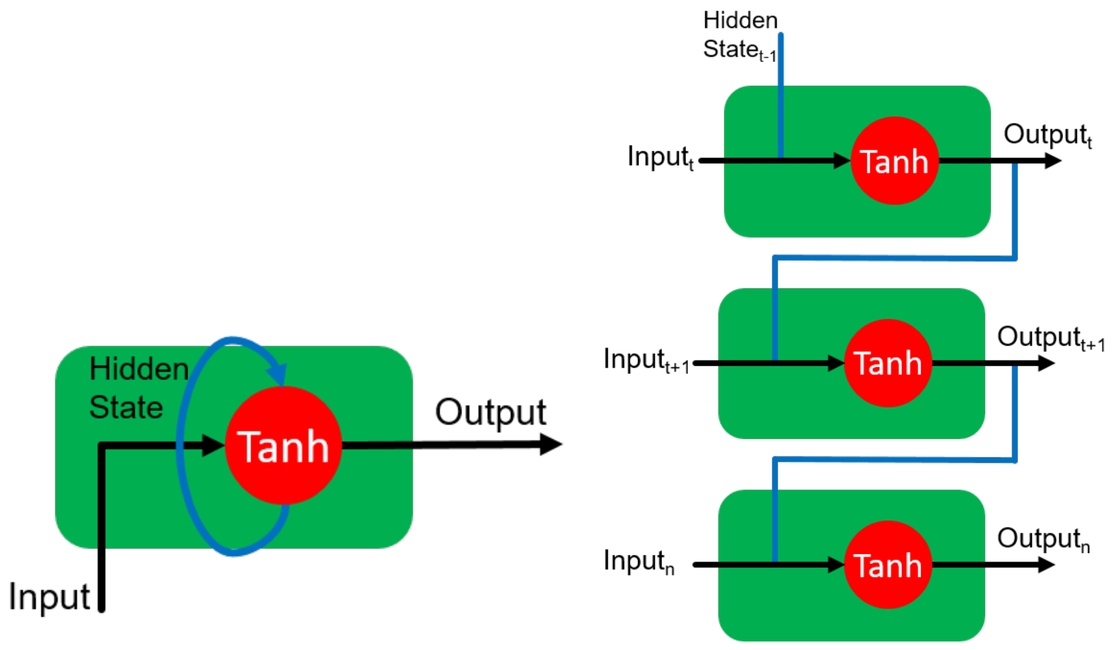


Figure 2.13: A representation of a RNN node is depicted on the left. This takes an input and has a looping hidden state which it uses to remember past inputs. On the right is an unfolded version of the RNN, where each recursive state is given its own layer.

and the combined previous layers is weighted 50 percent. After a long enough period the initial layers become so small that they no longer have an effect on the final prediction. Similarly, when the gradient is back-propagated, the proportion that effects the previous layer decreases until it vanishes. This is a huge problem for RNNs because if there is vital information in the initial part of the sequence it could be ignored and therefore give incorrect predictions.

To combat the vanishing gradient problem, the Long Short-Term Memory (LSTM) network [48] was derived. LSTMs use a gated system in which they can decide which information is important to the prediction and what can be forgotten. In addition to the hidden state, LSTMs also have a cell state which is passed forward to the next node. There are three gates in an LSTM: an input gate, a forget gate, and an output gate. Firstly the input and hidden state are combined together $(h + x)$. This combined state is fed through the forget gate, which consists of a sigmoid function multiplied by the previous cell state. A copy of $(h + x)$ is fed into the input gate, which consists of two paths. The first path contains a sigmoid function, and the second, a tanh function. The tanh function is used to regularise the input, and the sigmoid decides what to forget from the tanh output by multiplying them together. This is then added to the current cell state. A copy of the input and hidden state $(h + x)$ are fed into the output gate, where a sigmoid function is used to decided what to pass on to the next node. The cell state is given two paths. The first goes to the next node, and the second goes through a tanh function which is multiplied by the output of the output gate. This becomes the hidden

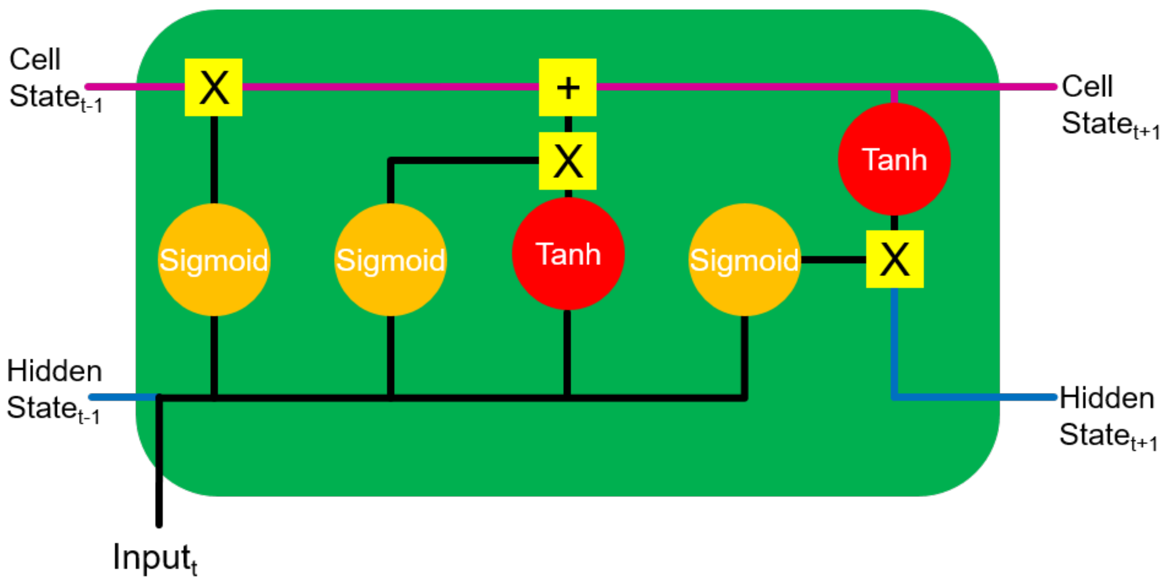


Figure 2.14: This figure illustrates the inner workings of an LSTM node. The node has three inputs: a cell state, a hidden state, and an input of data. These are fed through various gates containing sigmoid and tanh functions. The sigmoid functions control which data is to be remembered and the tanh functions regularise the information stored.

state for the next node, and is illustrated in Figure 2.14.

The output from either an LSTM or an RNN must go through a feed forward layer to make the final prediction. The error from this can then be back-propagated through the recurrent networks to update the weights and biases for each of the functions.

A variant of the RNN is the bidirectional RNN (BRNN), which was invented by Schuster and Paliwal [88] in 1997. The BRNN allows the data to not only flow forwards but also backwards. Figure 2.15 shows a mock example of the BRNN structure. This new flow of data allows the network to have a greater amount of data available to it, meaning both past and present data can be used to train the network. Therefore networks that use this type of data flow will generally have a better understanding of context. The forward and backwards flows are not connected to each other's inputs. This type of data flow through an RNN was then adopted to create a bidirectional LSTM (BLSTM) by Graves and Schmidhuber [39]. Here, they used two LSTM architectures, one for the forward data flow and the other for the backwards data flow. BLSTMs have proven to be very capable of learning context in a variety of different scenarios, including: speech recognition [40], text recognition [84], and brain-controlled robotic arms [53].

2.3.4 Loss/Cost Functions

Throughout this chapter loss functions, sometimes referred to as cost functions or error functions, have been mentioned, without going into much detail of what they exactly are. Loss functions are used to compute how correct a prediction is compared to its ground

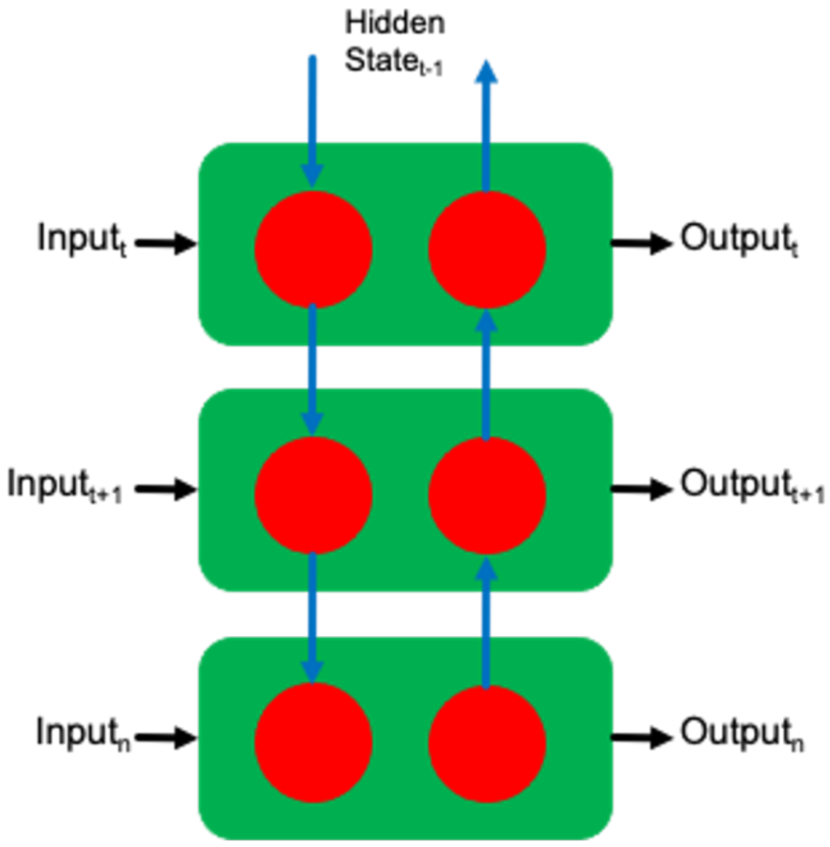


Figure 2.15: This figure shows the data flow through a BRNN.

truth. This error value is then used in backpropagation to update the weights of the networks. There is no master loss function that works best for all situations, in concordance with the ‘no free lunch’ theorem. In particular, loss functions are task-dependent. At the very least, regression and classification use their own types of loss functions.

The most common loss function that is used for regression problems is mean squared error (MSE). MSE is calculated by taking the sum of the prediction (x) value minus the ground truth (y) value, squared, then divided by the number of samples, equation 2.25. Other common regression loss functions are: mean absolute percentage error (MAPE), mean squared logarithmic error (MSLE), and Huber loss. MAPE computes the error as a percentage by minusing the ground truth from the prediction, and dividing by the ground truth value. This make MAPE robust against outliers. MSLE uses a logarithmic scale, meaning that it will penalise underestimating more than overestimating, thus again making it robust to outliers. Huber loss is used when the solution being less sensitive to outliers is needed. The Huber function is quadratic with small values, but linear with large values. This allows for this function to be less sensitive to outliers.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(x-y)^2 \tag{2.25}$$

For classification, the most basic loss function is binary cross-entropy (BCE). BCE is used for single class predictions, also known as binary classification. BCE, equation 2.26, measures the probability prediction between zero and one, and gives a higher loss the further away from the ground truth the prediction is. If the prediction is 100% correct then this loss would be zero. For multi-class problems, categorical cross-entropy can be used. This calculates the separate loss between class predictions and their ground truths, then sums these results, equation 2.27.

$$BCE = -(ylog(x) + (1-y)log(1-x)) \tag{2.26}$$

$$categoricalcross - entropy = -\sum_{c=1}^{M} y_{o,c}log(x_{o,c}) \tag{2.27}$$

Jaccard loss [6] is a loss function specifically designed to work with segmentation problems. In machine learning, it is often the case that the loss function is not the same metric as the metric used to measure the performance of the network. For segmentation, the intersection over union of the ground truth and prediction is often used. These metrics often differ because the performance metrics are generally not differentiable. If a function is not differentiable, then it cannot be used with gradient descent algorithms. Thus Berman and Blaschko [6] proposed an estimation of the Jaccard index as a loss function, such that the same metric used to evaluate can be used to train the network.

For unbalanced multi-class datasets, where the number of instances per class are not equal, it has become common practice to use weighted loss functions. Weighted loss functions introduce a new parameter per class to existing loss functions, which the calculated loss per class is multiplied by. This weighted loss is calculated using the distribution of instances in the dataset, or can be tuned as a hyper-parameter.

2.3.5 Optimiser Functions

Optimiser functions are variations of gradient descent which minimise the objective function of the parameterised network model by updating the parameters in the model in a direction opposite to that of the calculated gradient. Optimiser functions use the learning rate to determine how large of a step to take. Too large of a learning rate would result in the update overshooting, whereas too little of a learning rate would result in the network taking forever to converge. Learning rate is a hyper-parameter that can be adjusted for each network. An increasingly popular method is to use a learning rate decay, where the learning rate decreases the more epochs the network completes, getting closer to convergence. This decreases the likelihood of overshooting a lower local minimum, whilst being able to pass earlier higher local minima.

Batch gradient descent is the simplest version of gradient descent. The gradient descent algorithm is performed on the whole dataset to compute a single update to the network. This is very slow and memory expensive, thus it is not generally used. Batch gradient descent does guarantee to find the local minimum for a convex function and a local minimum for non-convex functions. The opposite to batch gradient descent is stochastic gradient descent (SGD). SGD performs an update to the network's parameters for every instance in the dataset. This means it is much more memory-efficient and generally faster to compute, however SGD generates more fluctuations, causing overshooting. This can be a useful side effect as it allows the network to jump out of the local minima, but this can cause it to jump around the global minimum multiple times before converging. Mini-batch gradient descent combines the best of batch gradient descent and SGD by computing the update over several instances, generally between 50 to 256 instances, depending on size of network. This mini-batch methodology reduces the instability of SGD by averaging out the errors and increases the speed at which the batching process can take place, since less data is needed to be stored and computed at once.

To improve the gradient descent methods further by speeding up convergence, optimiser functions are used such as: Adagrad [27], RMSProp[94], and Adam[55]. Adagrad optimises the gradient descent algorithm by adapting the learning rate to the parameter. Parameters that are frequently occurring have smaller updates, whereas less frequent parameters have larger updates. This adaptive gradient descent allows large sparse datasets to be used, where the number of instance per class is small. RMSProp is an unpublished work by Tieleman and Hinton [94], which was described during one of their lectures in a Coursera class. RMSprop stands for root mean-squared propagation, and is used to adapt the learning rate for each parameter. This is achieved by taking the average magnitude of the current and several previous gradients for each parameter, then updating according to the calculated average gradient. Adam uses an adaptive moment estimation which works similarly to RMSprop, but instead when the current gradient levels tends toward zero, it gains more weight, causing the updates to decrease faster.

2.3.6 Residual Neural Networks

Residual neural networks (ResNet)[46] use skip connections to allow data to bypass specific layers. This allows data to skip, in general, one to three layers in the network. The main motivation for using ResNets in a network is to reduce the vanishing gradient problem. The layers skipped are assigned the function $F(x)$, where x is the input into the first layer in f. The skip layer allows x to be passed into f and bypassed at the same time. The result of $f(x)$ is then summed with x, and this result is passed into the following layers. To address the vanishing gradient, the network can learn which skipped layers are useful to the network and which are not. If the optimal solution is to use just x rather than $f(x) + x$, then the network can set the weights of the layers in $f(x)$ to zero which

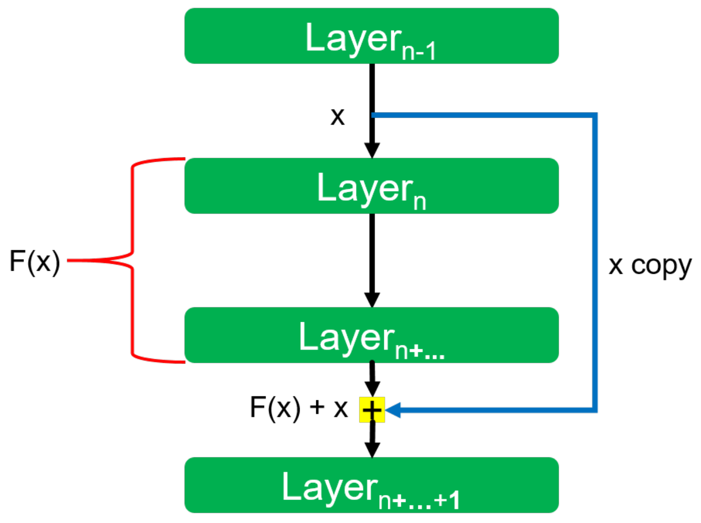


Figure 2.16: This figure shows the structure of a ResNet. The layers that are skipped are represented by $f(x)$ and the input to the ResNet layer is represented by x. The output of f(x) is summed with a copy of x before being passed to the next layer.

means that $f(x) + x$ becomes an identity map. This technique is easier for the network to compute rather than computing $f(x)$ to be an identity map. This is especially true with multiple skipped layers. With the weights set to zero, the gradient will not be adjusted when passing through $f(x)$, meaning earlier layers receiver a larger update. Figure 2.16 shows the structure of a ResNet skip layer.

Autoencoders and Hourglass network

Autoencoders [56] are comprised of two blocks of layers. The first block is the encoder block, followed by the decoder block. The encoder block reduces the input dimensionality down to a smaller, encoded version of the input. This is referred to as bottle-necking. The decoder block takes the encoded input and tries to recreate the original input from the reduced information. The decoder loss is measured in regards to how closely matched the reconstruction is to the input. Autoencoders work very well for de-noising data. This is the case because when the input is compressed down into its encoded version, there is information loss, and this is generally the noise in the data. When decoding, the decoder has no knowledge of the noise, so it is not included. A side effect of using an autoencoder is that it has built-in anomaly detection. When the trained model is used on new data and the new data does not fit the trend of the training data, the decoder will generate an output dissimilar to the input. Therefore the loss for the decoder will be high, and can be used to spot anomalies.

Hourglass networks [77] are a special case of autoencoders that add skip layers from ResNets into the network. Using skip layers allows for deeper autoencoders, reducing the representation of the input down to an absolute minimum. Then the data is scaled back up in the decoder. At each reduction in dimensionality, a skip layer is added from the encoder block to the decoder block at the same resolution. Stacked hourglass networks, multiple adapted autoencoders piped one after the other, allow the network to capture data at different scales across the entire input. The name hourglass comes from the shape the autoencoders represent when drawing their network structure.

2.4 Resuscitation

In this section the processes of Cardiopulmonary resuscitation (CPR), with an emphasis on CPR for newborn babies are discussed. The process described here will be reintroduced in Chapter 8 as the use-case for the action recognition with tiny domain-specific datasets.

CPR is the procedure used to revive someone from unconsciousness or apparent death caused by a physiological disorder, *e.g.* the patient has stopped breathing, or their heart has stopped beating. The currently recommended CPR procedure by the National Health Service [76] for adults is:

1. Place the heel of your hand on the centre of the person's chest, then place the other hand on top and press down by 5 to 6cm (2 to 2.5 inches) at a steady rate of 100 to 120 compressions a minute.

2. After every 30 chest compressions, give 2 rescue breaths.

3. Tilt the casualty's head gently and lift the chin up with 2 fingers. Pinch the person's nose. Seal your mouth over their mouth, and blow steadily and firmly into their mouth for about 1 second. Check that their chest rises. Give 2 rescue breaths.

4. Continue with cycles of 30 chest compressions and 2 rescue breaths until they begin to recover or emergency help arrives.

Postnatal Resuscitation is a specific subtype of resuscitation for newborn babies. This procedure differs greatly to the adult as a newborn baby's body is too weak to sustain chest compressions. Additionally their respiratory and cardiovascular systems are too fragile for the pressure from another human blowing into them. Thus an alternate procedure is needed [82]. Unlike the adult procedure, this requires additional equipment for different scenarios.

When the infant arrives, they are transferred to the bed where they are wiped down with a towel, and simultaneously their heart rate is checked via a stethoscope. If the infant is either not breathing or crying, a good colour tone, or born before term, then the procedure must begin. If the infant is not warm, then they must be wrapped in a polythene bag to help warm them up to prevent hypothermia. If the infant is not breathing or crying, then they must be stimulated via rubbing and firm squeezing of the limbs. Meanwhile, the head is tilted back and away from their body, to stretch the neck and open the airways. The head is then supported in this position. If the airways are blocked, then they must be cleared using a suction tube and a laryngoscope.

If the heart rate is below 100 beats per minute, they are gasping for air, or are experiencing apnea (no longer breathing), then an electrocardiogram (ECG) and SpO_2 monitor (pulse oximeter) can be attached along with the use of a positive pressure ventilation (PPV) to give assisted breathing. If they are not breathing, then five inflation breaths lasting exactly three seconds should be given. After that, ventilation breaths can be given for 30 seconds at one second intervals.

If the infant does not meet these requirements, but has laboured breathing or has persistent cyanosis (blueish tint to skin), then a continuous positive airway pressure can be used and then the infant can go to post-resuscitation care.

If after 60 seconds of the procedure beginning and receiving assisted breathing, the infant's heart rate is above 100 beats per minute, then they can be sent to post-resuscitation care. However, if this is not the case then adequate ventilation should be used, and endotracheal tube insertion (intubation) can be considered. If intubation is attempted, a surfactant can be used to allow the alveoli to remain open. This makes gas exchange easier. Once intubation is complete, the PPV can be attached to the endotracheal tube.

If the infant's heart rate drops below 60 beats per minute, then chest compressions coordinated with the PPV can start. The ratio of compressions to breaths is 3:1 and the superimposed thumb position (thumbs overlapping each other) is used to administer compressions to the lower one third of the sternum. If the heart rate does not increase, then an IV containing epinephrine can be administered, and chest compressions are continued. Once the infant's heart rate is back above 60 beats per minute, assisted breathing can be resumed. Finally, once the infant's heart rate is above 100 beats per minute, they can be taken to post-resuscitation care.

If the procedure is not followed correctly, severe damage can be caused to the newborn, such as burst blood vessels or blood clots, which can lead to brain damage, or worse death, of the infant.

According to Perlman et al. [82], approximately 85% of infants born at term will not need any assistance with breathing within 10 to 30 seconds of being alive, and an additional 10% will respond during the drying process. This leaves about 5% of term babies that require assisted breathing via PPV (3%), incubation (0.1%) or chest compressions. Even though the percentage of babies needing CPR is small, the large number of births each year means this is still a significant number. Having staff trained to highest possible standards would decrease the number of complications or deaths arising from incorrect procedural issues.

Chapter 3

Related Work

In this chapter, work related to this thesis will be discussed. The following topics will be covered: superpixels in deep learning, deep-learned semantic segmentation, deep learning with small domain-specific datasets, and annotation software.

3.1 Superpixels in Deep Learning

In this section the use of superpixels in deep learning networks is discussed. Superpixels have become a staple in semantic segmentation as their inherent edge detection is useful for defining object boundaries.

In 2018, Zhao et al. [109] proved that using SLIC superpixels [1] with CRF upon the predictions from a fully convolutional networks (FCN) [89] can clean up the boundary of the predictions. Figure 3.1 illustrates the pipeline used in Zhao et al. [109] architecture. To verify their results they used the following two datasets: PASCAL VOC 2012 Dataset [30], and Cityscapes [20][19]. For both datasets, they beat the state-of-the-art networks at the time with a mean Jaccard index over all classes in the datasets of 74.5% for PASCAL, and 65.4% for Cityscapes. Comparing the mean Jaccard indices to a baseline network using a plain FCN with eight pixel strides resulted in an increase from 62.7% for PASCAL, which is an 11.8% absolute increase, and for the Cityscapes dataset an increase of 9.3%, from 56.1%. The closest competitors at the time were DeepLab V2 [17] for Cityscapes, and Deep Parsing Network (DPN) [66] for PASCAL.

Kwak et al. [58] in 2017 used superpixels in their network by creating a new pooling layer, which they dubbed Superpixel-Pooling (SP) layer. Like a traditional pooling layer, they aggregate features together, but unlike traditional pooling that uses a rectangular windows to aggregate the feature in, they use the shape of a superpixel. The output of the SP layer is a $(N \times K)$ matrix, where N is the number of superpixels and K is the number of channels in the feature map. The SP layer was developed to solve the issue of segmentation classification where the output of the network generated a low resolution

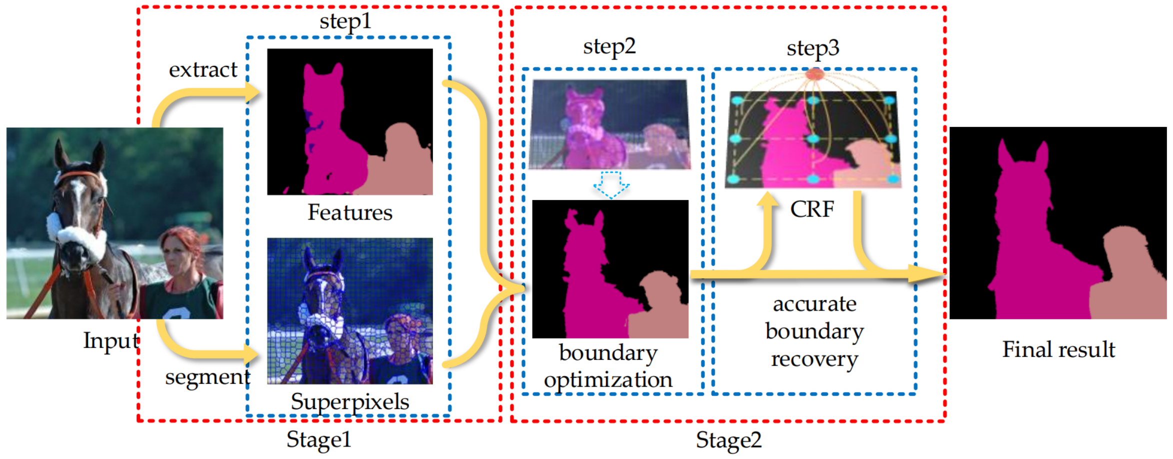


Figure 3.1: This figure shows the data flow chart proposed by Zhao et al. [109] . Image taken from [109].

prediction with poor shape encapsulation. The effects of the Superpixel-Pooling can be seen in Figure 3.2. Their results showed mean accuracies of 50.2% and 46.9% on the validation and test set of the PASCAL 2012 dataset, respectively.

Superpixels are often used for feature selection for training a network. Xiong et al. [105] used SLIC superpixels in their work with plant segmentation to generate candidate examples to train their network. These candidates are automatically labelled to either be from the confirmed background or a plant candidate, using an offline CNN. They also use another type of superpixels in their work, Entropy rate superpixels (ERS)[65]. ERS are generated using a random walk though graph representation of the input image, where each node represents a pixel and the edges between adjacent nodes specify the similarity between the two nodes. The entropy rate of the random walk is used to create compact homogeneous superpixels. In Xiong et al. [105]'s work, the authors used ERS to optimise their segmentation results. They claimed to have attained better results than other segmentation algorithms used in their field of research, such as HSeg[93], i2 hysteresis thresholding [26] and jointSeg [68], with an F1 score of 0.7673.

More recent work in 2019 by Verelst et al. [101] reported on the creation of a new superpixel method that combines the handcrafted algorithm of SLIC with deep-learned features of an image to compute superpixels. The authors did this by extending the space in which the SLIC algorithm clusters on from just the LAB colour space plus x and y coordinates, to 81 features including the original five. The new features came from intermediate layer outputs (layers half-way down the feed-forward artificial neural network), which capture a deep representation of texture, gradients and edges. Using these deep representations allows the algorithm to better fit object boundaries compared to SLIC. Without these extra features, SLIC sometimes expands the segmentation boundary beyond the actual object boundaries if the object and background have similar texture

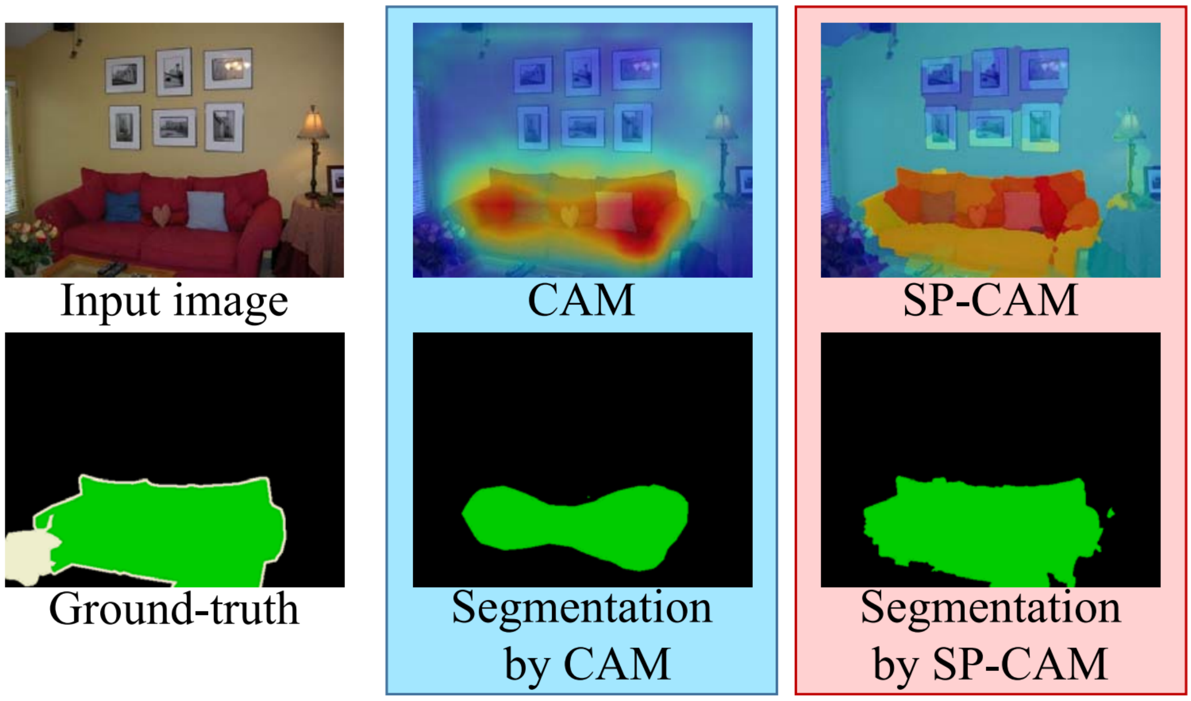


Figure 3.2: This figure shows how superpixels improve the segmentation of the target object in the work by Kwak et al. [58]. Image taken from [58].

and colour. The extra deep features allow the algorithm to distinguish between the two areas. This is achieved by using semantic segmentation ground truth masks as a means to correct the boundaries during training. Verelst et al. [101] claimed that with their three-layered regression network, their new superpixels were of higher quality and could be fine-tuned for specific domains.

These discussed works use superpixels with deep learning, however none of these works learn the structural representation of superpixels. This is the novel idea proposed in this thesis, that superpixels could be learned by a CNN. Having superpixels that are created using a deep network rather than a handcrafted algorithm allows these deep-learned superpixels to be truly fine-tuned during backpropagation. In turn this means that the superpixels would be able to be fine-tuned for a specific domain, and as shown in this thesis, that is beneficial to be used with tiny domain-specific datasets. The adapted SLIC superpixels which make use of deep-learned features, by Verelst et al. [101], have the possibility to work well for tiny domain-specific dataset because the deep-learned image features could potentially encapsulate vital information from the dataset to better segment the objects. However, when being used with tiny amounts of data, the major problem that this methodology may face could be that features learnt by the network may not be generalisable, meaning these superpixels do not better represent the data. In a worst-case scenario the non-generalised features could be detrimental to the creation of the superpixels.

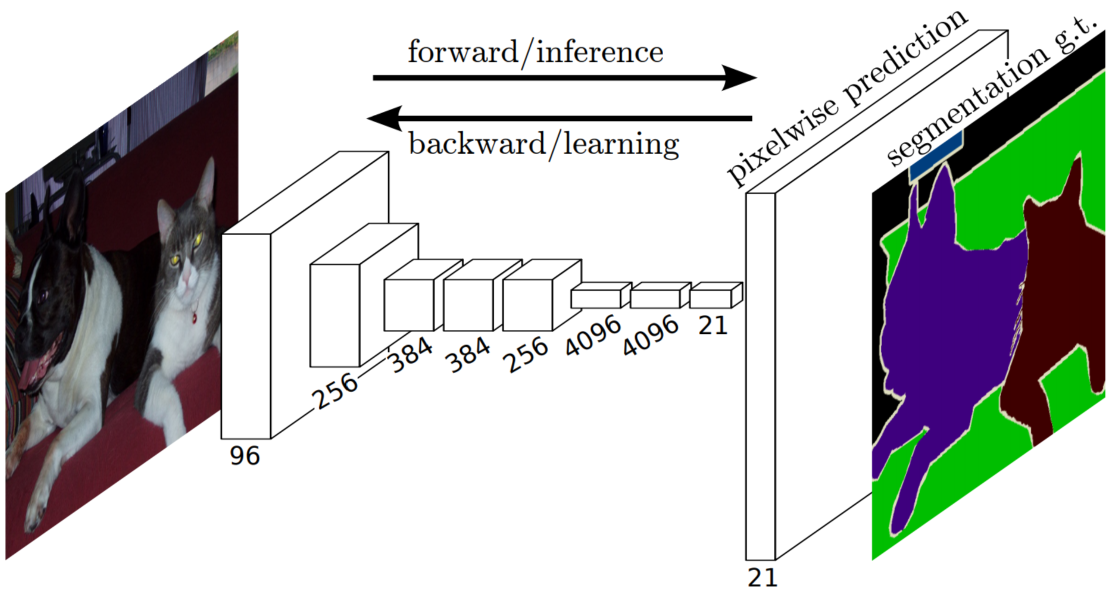


Figure 3.3: This figure shows the FCN architecture proposed by Long et al. [67]. Image taken from [67].

3.2 Deep-Learned Semantic Segmentation

In this section the use of deep-learned semantic segmentation networks are discussed. Semantic segmentation is a useful tool for many different applications, ranging from identifying brain tumours in MRI scans [75], to aiding self driving cars [99].

One of the most popular learnt segmentation methods is the Fully Convolutional Networks (FCN) developed by Long et al. [67][89] in 2015. In their work they use an end-to-end network to produce pixel-by-pixel predictions. FCN are networks that consists of solely convolutional layers, pooling layers and up-sampling layers. They do not make use of dense (fully connected) layers. To achieve pixel-wise predictions a (1×1) convolutional kernel can be used instead of a dense layer.

This produces both segmentations and classification. This is achieved by their skip architecture, which makes use of both the deep coarse semantic information and the shallower appearance information. The residual data is added to the later layers by element-wise additions. At the time of publishing they were achieving state-of-the-art results on VOC2011 and VOC2012 [30]. The FCN architecture can be seen in Figure 3.3. They use multiple convolutional layers followed by some pooling to reduce the dimensionality down to $(10 \times 10 \times 21)$. This is then up-scaled using strides of first 32, then 16, then 8, using an additional convolutional layer between each up-scaling. At each scale the predictions become more precise, at the increased cost of computing time due to the larger network, however despite being more complex the FCN-8 network is said to run at approximately 175ms at inference time.

There are a group of residual networks referred to as encoder-decoder networks. These form an hourglass shape by using convolutional layers followed by pooling layers in the encoder, which shrinks the resolution. Then the decoder uses a technique to increase the resolution back close to the input size of the hourglass. There are several different techniques that can be used to increase the resolution in the decoder network, such as up-pooling, up-sampling, and deconvolutions.

U-net [85] is an hourglass network that was originally designed for segmentation on biomedical images. The Ronneberger et al. [85] network uses a repeated pattern of two convolutional layers with rectified linear unit activation function, followed by (2×2) max pooling with stride 2 for down-sampling in the encoder. In their decoder, they use an up-convolution (up-sampling), which first halves the number of feature channels and then adds the residual feature map as new layers. This is then followed by two convolutional layers. To achieved full image segmentation they use a sliding window with even dimensions. In their work they achieve high intersection over unions results (Jaccard index) on the ISBI cell tracking challenge 2015, with an average increase of 9.03% on the dataset and 31.49% on another in the challenge.

SegNet [3] is an hourglass network that uses up-pooling as its means of increasing the resolution of the data in the decoder part of the network. The up-pooling differs to the previously discussed up-sampling in that instead of using residual feature maps, SegNet uses stored pooling indices. During max-pooling, traditionally the location of the max value is lost, but by creating a mask of the locations of the max values per pooling operation, the up-pooling layer is able to correctly infer the locations of the maximum values. This produces a sparse feature map. To create dense feature maps from these sparse feature maps, they use two to three convolutional layers. Use of a indices mask compared to the full feature map in U-net reduces the amount of memory needed for the network, as less is data is required. At the time of publishing, their network out-performed the current state-of-the-art on the CamVid dataset[13] when training with over 80K iterations, whilst reducing the average computation time and memory.

Deconvolutional layers are like an inverse convolutional layer and are also known as transpose convolutional layers. The kernels for deconvolutional layers can be thought of as a drawing stencil that increases the magnification of the input. Unlike interpolations, the kernels are learnt to produce outputs that are as close to the original input from the deep representation of the input. Deconvolutional layers can not only be trained to predict the input image, but also a segmentation mask [78]. Noh et al. [78] outperformed FCN-8s on VOC2012 [30] with a mean increase of 10.3% for several classes.

The hourglass network structure in the stacked hourglass paper by Newell et al. [77] uses residual modules rather than the traditional convolution-only layers. This network was designed to predict heat maps for human pose estimation. Each residual module is comprised of multiple convolutional layers with a skip layer that spans three convolutions. For the up-sampling they use nearest neighbour up-sampling [96]. Their hourglass network is symmetrical, meaning the encoder and decoder have the same number of steps up or down, therefore giving a pixel-to-pixel prediction of the input to the hourglass.

In 2019, Fu et al. [35] proposed a stacked deconvolutional network (SDN), which is a hybrid of the stacked hourglass network and the deconvolutional network. Their network uses dense blocks from DenseNet [49] as the front-end. Dense blocks consist of multiple convolutional layers back-to-back, with the input of the following convolution being the concatenation of the previous output plus the input. At the end of a dense block a (1×1) kernel is used in the convolutional layer to reduce the feature maps and preserve the spatial resolution. Like the RNN component of stacked hourglasses, they pass information from earlier layers via skip layers. Where their network differs to a stacked hourglass, is that the skip layers start in the DenseNet and are passed into the deconvolutional layers. At the time of publishing, their work out-performed the state-of-the-art methods on: VOC2012 [30], VOC2012+COCO [64], CamVid [13], and GATECH [51]. Respectively, SDN performed 0.9%, 0.9%, 2.2%, and 2.4% better than the state-of-the-art for these datasets.

In this thesis, the work of Newell et al. [77] is extended from human pose estimation to semantic segmentation by pixel-to-pixel predictions. In its current state, their network generates heat maps at a lower resolution than the original input to the network. Thus for pixel-to-pixel segmentation, an additional up-scaling layer would be needed. As well as this, it would be necessary to change the target of the network from Gaussian maps of key landmarks to N binary masks, one for each class. FCNs produce boundary predictions that are not crisp, but in this thesis the inclusion of superpixels will be used to combat this by making use of their inherently superior boundary detection. Having poor separation between objects becomes detrimental when there are small objects amongst large objects because these smaller objects can easily be misclassified. In medical datasets there are often small objects, such as a syringe, next to large objects, such as hands. If is the boundary is poor then the syringe could be undetected, which for certain applications would be a negative outcome, like when using a syringe is a critical step in a procedure. The following two modern architectures could be used instead of the hourglass network: PSPNet [108], and DeepLabV3 [18]. The exact backbone architecture is not the focus of research in this thesis. Instead, the techniques to aid learning with small datasets which can be used with any deep CNN, are the focus. Both PSPNet and DeepLabV3 perform semantic segmentation. PSPNet uses a pyramid style architecture where the feature maps are pooled into multiple resolution then combined back together using up-sampling and concatenation. DeepLabV3 improves upon their earlier work [16],[17] by removing the post-processing dense conditional random fields, and adding atrous convolution. Atrous convolution applies the kernel with a differing number of pixels between the used pixels, dubbed “rate”. When the rate is one, the kernel is applied to neighbouring pixel. Increasing the rate affects the network by giving it a wider field of vision, which allows object encoding at multiple scales.

3.3 Deep Learning with Small Domain-Specific Datasets

Traditionally, deep learning requires large amounts of data to train a network with adequate generalisation performance. However, there are many real world applications for which acquiring what is normally considered to be sufficient data to train a network is not feasible, either in terms of the number of examples that can reasonably be collected in the first place, or the extent to which it is feasible to generate large amounts of ground truth data.

The problems caused by having less data to train a network on can be broken down into three categories: poor generalisation, class imbalance, and poor optimisation.

Common practice to reduce poor generalisation is to use data augmentation. Data augmentation applies predefined transformations to the input data with random values, between set ranges for each transformation. Data augmentation is not applied to the test set as this would give different results for comparison. The more epochs that are run effectively increases the number training samples the network has seen. Another option is to increase the number of samples per epoch by having multiple instances of the same input but with different augmentations applied. There are four types of transformations that can be applied to image inputs: flipping the image either horizontally or vertically, cropping and/or zooming the image, increasing or decreasing the brightness, sharpness and contrast of the image, or rotating the image. All transformations must also be applied to the ground truth data to keep them correct.

Another method to increase the number of training samples is to use artificially generated data. Original methods for artificial data generation in regards to computer vision would use computer-rendered images. More recently, generative adversarial networks (GANs) [37] have become popular for generating new data. GANs use a two component architecture called the generator and the discriminator. The two components compete against each other during training, with the generator creating fake images and the discriminator determining which are real and which are fake. Both networks try to minimise their loss functions throughout training until the losses plateau and the generator is no longer improving.

A third method for dealing with poor generalisation is to use semi-supervised learning methods such as active learning, or cooperative learning. Active learning is a concept where the algorithm asks the user to label specific data points in a similar way to how a child asks what new things are called. As only the requested data points are labelled, there are fewer labels in the dictionary for the learner to learn than classic labelling, thus speeding up the process of learning. Additionally, the learner only asks for the next most informative data points, therefore there is less redundancy [111]. Cooperative learning is where both the computer and user annotate unlabelled data [22]. Initially the user would label the first few frames in a dataset, then the computer would annotate the rest of the dataset. As the computer will not always get it right, each label is given a confidence value which is calculated using a pre-existing classifier. If the confidence is lower than a given threshold, the user is prompted to correct it. Any corrections made are then propagated through the network to improve the labels to improve labelling speed further.

Class imbalance refers to the number of samples in each class in the dataset. Data balancing is done because unbalanced datasets cause an unbalanced penalisation during propagation. This is due to the inherent weighting of the larger classes over the smaller classes, meaning that small classes do not get learnt as they contribute to the loss function less and optimising the larger classes generates a greater gain. There are two main strategies to compensate class imbalance. Strategy one consists of either up-sampling the classes with fewer instances than the more populated classes, or down-sampling the more populated classes so that they have the same number of samples as the smaller classes. Up-sampling, sometimes referred to as over-sampling, re-samples the same data point more than once during each epoch to increase the number of samples for specific classes, and is often used when there is a small total number of instances in a dataset. Down-sampling, sometimes referred to as under-sampling, takes a subset of the larger classes to decrease the number of instances, and is used when there is a large dataset with many thousands of samples. The second strategy for combatting data imbalance is to adopt the loss function for weighted classes, where the inverse weighting of the class ratio is used to weight the classes during the loss calculations.

Poor optimisation of a network is caused when a network is trained on a small dataset and the network does not find an optimum solution. This can be seen by the loss failing to decrease over time, or by the presence of very sporadic spikes in the loss values. There are several methods of tackling this such as: problem reductions, transfer learning, and using methodologies that learn with less data.

Problem reduction is the process of converting a problem from a high number of features to a lower, more manageable number of features. The more features a problem has, the more data is needed to train a network, and a larger network is generally needed to be able to represent and encapsulate all of the features. A traditional machine learning method is principal component analysis (PCA) [81], which maps high dimensional features to a lower dimensionality, therefore reducing the number of input features and increasing the variance between the remaining features. The co-variance matrix (correlation) between features and eigenvectors are calculated. The eigenvectors with the largest eigenvalues can be used to represent the data, because they encapsulate enough of the information to reconstruct the majority of the variance of the data. Other methods for image data such as scaling down the resolution of the input is a good method of reducing the number of features for CNN networks.

Transfer learning is the act of using the weights from a network trained on another dataset as the initial weights for the network. Using a pre-trained network from an online resource in turn limits the user to keeping the same network structure. Layers can be removed and added to customise to the problem, but the main architecture remains the same. Another option is to design an network architecture and pre-train this network on a larger, more varied dataset. This has the drawback of having to train the network multiple times with different data which can be time-consuming. Transfer learning is a technique that can be used with all varieties of deep learning, from large datasets to small domain-specific datasets. Transfer learning works best when used with datasets which have similar classes, or similar object structures. To use transfer learning with a new dataset that uses a different sized input dimensionality, the head of the network must be removed and updated to fit the new input, whilst keeping the trained weights. Similarly the output layer can be replaced to fit the new output dimensionality. Another method that can be used with transfer learning is to train iteratively, starting with a single class and adding another class with each iteration, until all classes are learnt by the network. This is a good strategy for small datasets, as it allows the network to focus on an individual class at a time. This can improve performance on less frequent classes.

Two examples methodologies for learning with less data are one-shot learning, and zero-shot learning. One-shot learning [31] tries to mimic how humans can see a single instance of an occurrence and then recall it at a later date to classify something as the same as this single example again. This is a hard task for the previously described networks and their loss functions. To combat this, either the loss function needs to be changed, or the representation of the problem needs to change. Siamese neural networks (SNN) [12] are a type of one-shot learning where the network consists of two identical network structures and the resultant feature maps can be compared to determine if two inputs are of the same type. The loss function for SNNs is used to optimise the radius of the classification area of the clusters. During evaluation of the model, the input data is passed to one path of the network, and repeatedly for each class in the dataset, a sample is given to be passed to the other path of the network. The SNN calculates the distance between the two output feature maps, and if they are close in hyperspace then they are classified as the same class.

Zero-shot learning [59][80] uses large datasets in which the classes are similar to the problem classes, such as ImageNet [86], to pre-train a network but without including the target classes in the training. The pre-trained network's final classification layers are removed to expose the feature maps, then the network is ran over the dataset to create a mapping to feature space. The unseen classes are also passed through the network, and the feature space is then clustered using a clustering algorithm, such as K-nearest neighbours (KNN). The resultant KNN clusters can be used for classifying new data.

Recent work by Mercedes Torres Torres *et al.* [97][98] used a small domain-specific dataset to perform gestational age estimation using deep learning. They made use of

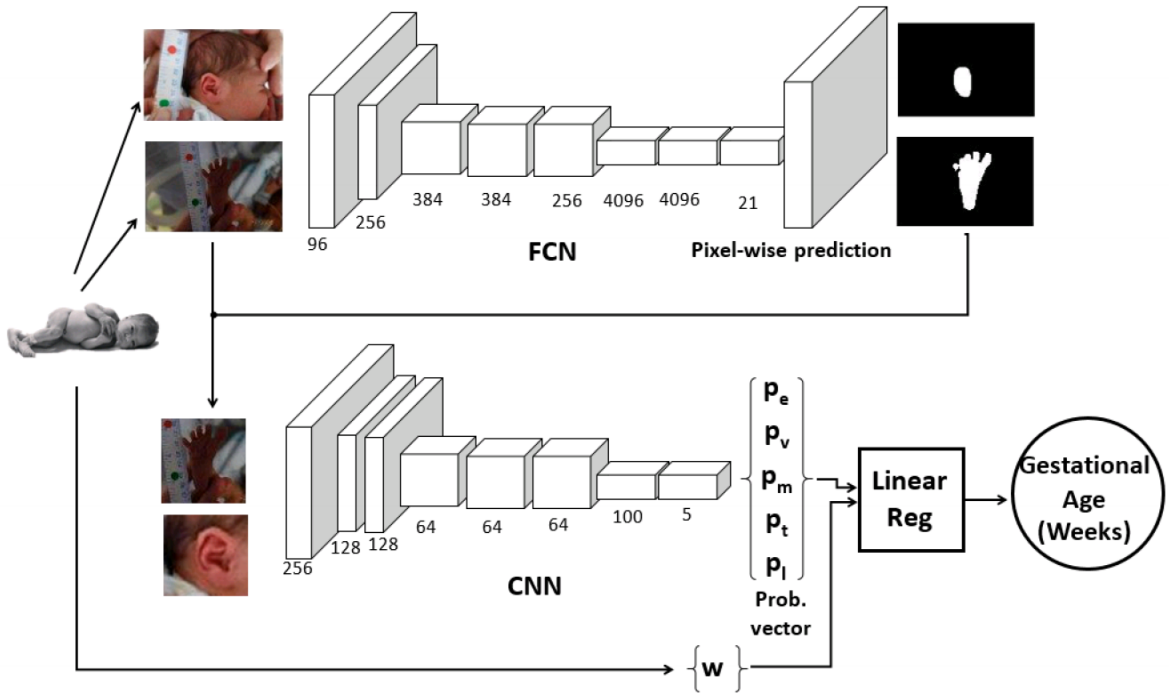


Figure 3.4: This figure shows the pipeline proposed by Torres et al. [97] for gestational age estimation from three different images. This figure was taken from [97].

several FCN networks to predict probability vectors for different parts of the newborn's body, face, foot, and ear. The vectors were then combined, plus the baby's weight, to feed through a regression network in order to predict the infant's gestational age. This pipeline can be seen in Figure 3.4. This was done using only 130 babies with between two and ten photos per body part taken. The results were calculated both upon the Jaccard index of segmentation for each of the classes, and the regression predictions for the baby's age. In their work they achieved better segmentations than the ground truth data, with smoother and closer matches to the actual object than the human annotations. Moreover they achieved an RMSE of 1.14 with a standard error of 0.88 (6.16 days), which at the time were state-of-the-art results.

In 2020, a paper was published by Meinich-Bache et al. [72], Activity Recognition from Newborn Resuscitation Videos, tackling a similar problem to the Newborn Resus dataset that is used in this thesis, demonstrating that this is still an active field of research. In their work, they used a 2D CNN to perform object detection via bounding boxes and object tracking to follow the objects over time. Due to low frame rates, they used frame interpolation, then calculated the optical flow between frames. The RGB image and optical flow images were collected over several time intervals and fed into separate 3D CNNs, where the output feature maps were combined together before making predictions on whether an action is active or not. The two systems were separate networks and the information was piped from one to another. The dataset used in this works was the

Laerdal Newborn Resuscitation Monitor(LNRM) dataset[102]. In Meinich-Bache et al. [72]'s work, only 96 of the 481 videos were randomly chosen, with six actions. These were: uncovering the baby, stimulation, ventilation, suctions, attaching the ECG, and removing the ECG. This was done to reduce the number of videos to annotate. In their results they claimed that the system performed well for ventilation, with a recall value of almost 90%, and suction with a recall value of about 60%. On the other hand, the system did not detect uncovering or stimulation of the baby.

Meinich-Bache et al. [72] system relies on bounding box detection to find the objects in the scene, and with such a small dataset this could lead to the problems such as confusion of what the object actually is. This can be seen with how the blanket is not detected when the infant is covered and uncovered. Using semantic segmentation would allow for a better understanding of what is happening in the scene. Additionally, this would reduce the dimensionality of the data. Unlike Meinich-Bache et al. [72], a 3D CNN was not used in this thesis, instead LSTMs were used because these have proven a superior approach to learning time-dependant information. The major advantage of LSTMs over 3D CNNs is their ability to process longer temporal sequences. With longer sequences 3D CNNs require significantly more memory to be able to process the sequence.

Additionally this thesis utilises multiple methodologies presented in this section to combat the use of small datasets. Semi-supervised learning was employed to help annotate the Newborn Resus dataset. Transfer learning was used during training of the semantic segmentation with deep-learned superpixels. To learn action recognition, dimensionality reduction was applied to convert from RGB images at 1080p resolution down to $(32 \times 32 \times 2)$ input to the action recognition network. LSTM layers work best when there are fewer features for them to find patterns in. When using larger inputs to an LSTM, the training time become very long and in some cases unusable. Additionally, having a complex input, such as raw RGB values, increases training time. This is because the LSTM has to effectively decipher the raw image as well as learn the temporal sequence. Reducing the dimensionality of the input frame by first taking a semantic segmentation representation of the frame simplifies the problem. Then, reducing the size of the input by taking the N most frequent classes per region, in this case $N = 2$, can drastically reduce the complexity and training time.

3.4 Deep Learned Action Recognition

In this section the use of different temporal architecture for deep learning action recognition are discussed. Action recognition can be used for a variety of applications including body motion recognition, sports activity classification, and human-object interaction. There are many large datasets available for training non domain-specific action recognition such as: HMDB-51 [57], UCF-101 [91], ActivityNet-200 [14] and Kinetics [54].

Inception 3D (I3D) was developed by Carreira and Zisserman [15] in 2017 and achieved state-of-the-art performance on UCF-101, 98.0% accuracy. I3D uses a two-stream inflated 3D convolutional network version of the popular image classification network Inceptionv1[52]. The term "inflated" means that the 2D convolutional layers of Inceptionv1 are altered to 3D convolutional layers, meaning the network becomes a very deep neural network. This inflation to 3D convolutions is attained by converting the squared filters to cubed filters, and duplicating their weight over the third axis (time). "Two-stream" networks consist of two temporal input streams of data. For I3D, the two streams are optical flow and RGB.

The residual neural network ResNet [45], and inflated versions of ResNet [33], are common backbone architectures used for training deep recurrent neural networks such as Temporal Pyramid Networks (TPNs) [106] and SMART [38].

TPNs for Action Recognition [106] use a structure similar to PSPNet [108], where the temporal input is pooled into multiple resolutions. The different pools are then fed through five modules: backbone, spatial modulation, temporal modulation, information flow, and final prediction. This pipeline can be seen in Figure 3.5. The backbone module uses inflated ResNet as the backbone network. The predicted features are then passed into the spacial modulation, where the features are down-sampled and aligned to semantics. The output of the spacial modulation is passed to the temporal modulation, where the features are down-sampled to adjust the tempo, with differing tempos per level. Information flow takes the output of the temporal modulation and enriches the level-wise representation by aggregating the features. Finally the predication module concatenates the output of the information flow module along the channel dimension and then predictions are made using a fully connected layer. TPN achieves a top-1 accuracy of 78.9% and a top-5 accuracy of 93.9% on the Kinetics-400 [54] dataset. Top-1 accuracy is when the first prediction is the correct prediction, whereas top-5 accuracy is when the correct prediction is within the five most confident predictions.

SMART Frame Selection for Action Recognition was developed by Gowda et al. [38] in 2020 and aims to reduce the computation cost of action recognition by selecting only "good" frames. By reducing the number of frames, not only do they claim that the computational time will decrease due to less data, but also that the harder-to-classify frames can be discarded, thus improving the final accuracy of the predictions. SMART is designed to be used with existing network architectures as the backbone for action recognition. SMART uses a two stream method where the first stream uses a multi-layer perceptron that evaluates each frame individually and computes a score δ_i. The second stream makes use of LSTM layers with pairs of frames as input, then uses an attention relational network to obtain a score, γ_i, for the pair of frames. The two scores are combined together, and the frames are ranked. A budget is set which dictates the number of frames, n, required. Then the best n frames are then used in the backbone action recognition network. In

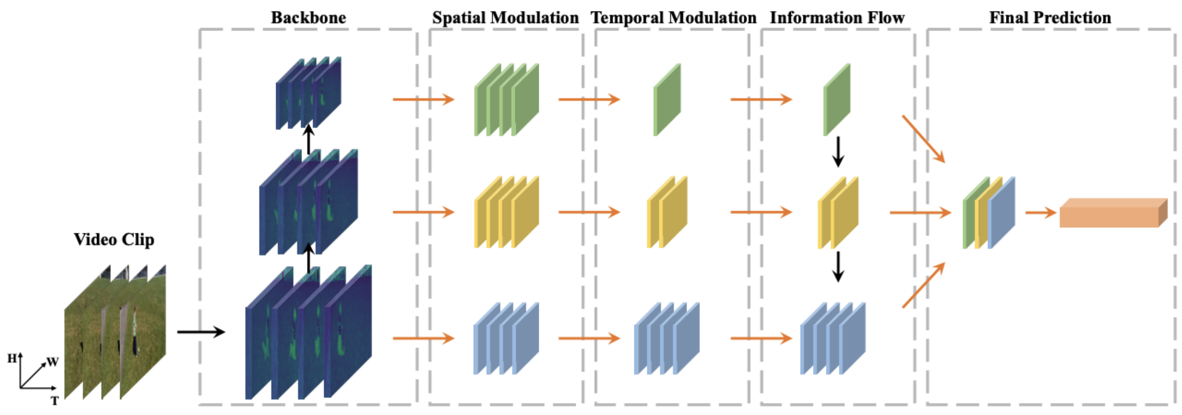


Figure 3.5: This figure shows the TPN pipeline proposed by Yang et al. [106] for action recognition. This figure was taken from [106].

Gowda et al. [38] paper they show that for all backbone networks tested, the addition of SMART improved the accuracy. Three notable architectures they tested SMART with were: I3D [15] , STM-ResNet [32] and ISTPAN [25]. They were tested on the HMDB-51 [57] and UCF-101 [91] datasets with up to a relative increase of 1.4% in accuracy.

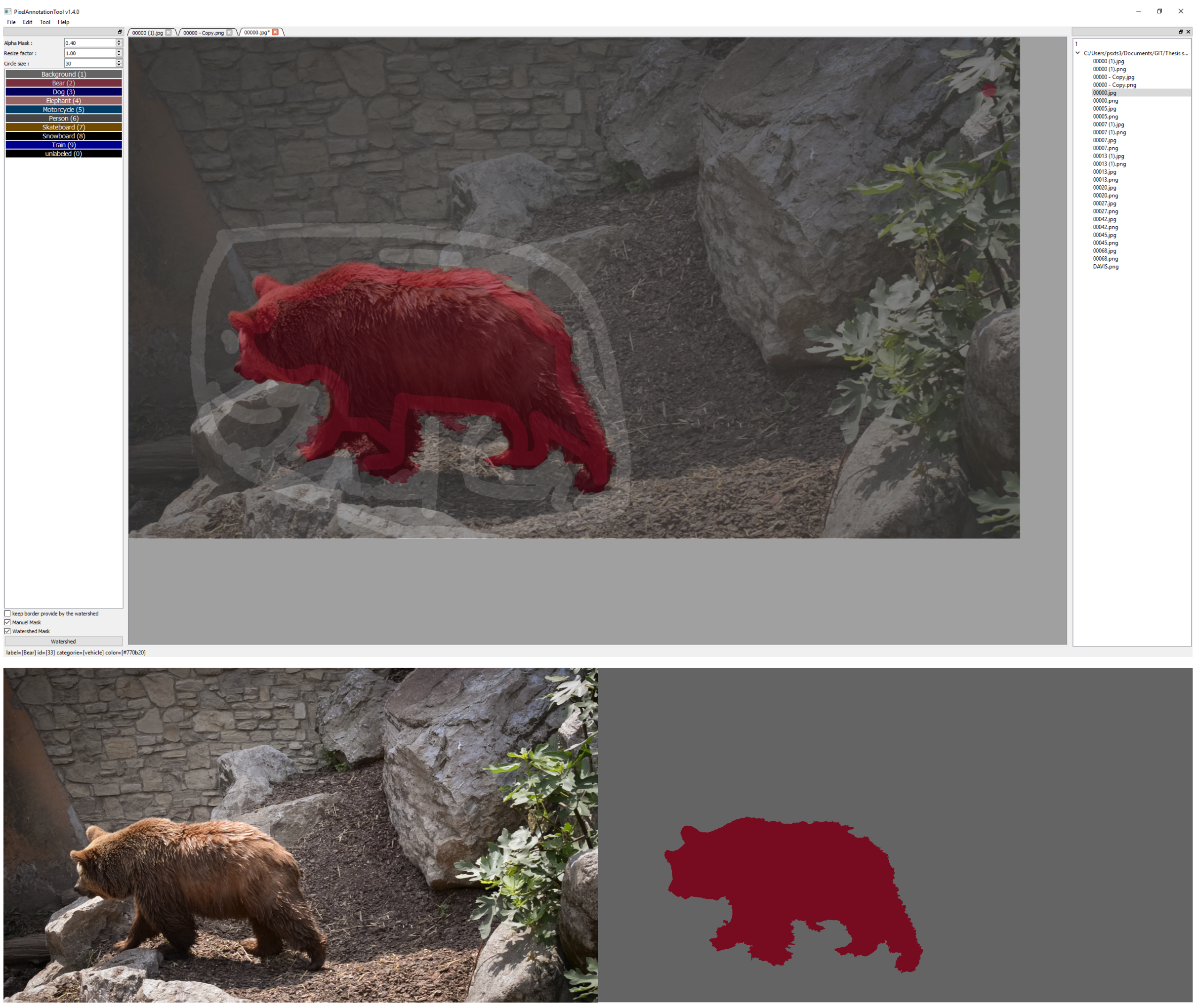
The papers in this section all follow the trend of being trained on very large dataset, and commonly taking multiple weeks to converge. To reduce complexity of the network and therefore reduce overfitting on the tiny sparse dataset used in this thesis, Newborn Resus, a small network consisting of a small number of convolutional layers and bidirectional LSTM layers will be used. Using a smaller network also allows for architecture searching to be performed, where the number of convolutional and LSTM layers can be altered, and the input to the network can be varied to find the best performing architecture.

3.5 Annotation Software

In this section I will describe the most relevant software for annotation of semantic segments in images and videos, as well as annotation of actions occurring in videos.

3.5.1 Semantic Segmentation Annotation

Pixel Annotation Tool by Bréhéret [11] makes use of the watershed algorithm to segment the image into different regions defined via brush strokes by the user. Watershed is the algorithmic equivalent of the geographical watershed where water pools in different locations depending on the topology of the terrain. Bréhéret [11] use the OpenCV implementation of the marked watershed algorithm, which was invented by Meyer [74]. The pixel annotation tool is a good tool for annotating images with large objects because the user can do quick strokes to get the base segmentation, and then fine-tune the



Alpha Mask : 0.40
Resize factor : 1.00
Circle size : 30

☐ keep border provide by the watershed
☑ Manuel Mask
☑ Watershed Mask

label=[Bear] id=[33] categorie=[vehicle] color=[#770b20]

Figure 3.6: This figure shows an example of the PixelAnnotationTool on sample frame from the DAVIS dataset.

boundaries with additional strokes. However, the tool is not well optimised for annotating small items in a scene, because the brushes can be too large, and small details can be lost.

Lu et al. [69] proposed a new method called Coherent Parametric Contours (CPC). This method works by explicitly modelling bézier curves for the boundaries between segments. These curves are initialised by the user's annotation. Again, the user has sparsely annotated the first frame with strokes on the foreground and background. The initial curves are then propagated through the video by a spatio-temporal optimisation algorithm. As the bézier curves are parametrised, this gives the users full control over the shape of the curves, allowing for fine tuning. The user can manually edit the curve in later frames, allowing areas that are very similar in terms of colour and motion to be correctly added to the foreground. Contrastingly, other stroke-based methods may still not be able to identify the difference between the foreground and background segments. To try to reduce tracking loss, a rigidity function is applied to all terminal control points. Terminal points are snapped to strong edges in an image, whereas control points are surmised to be

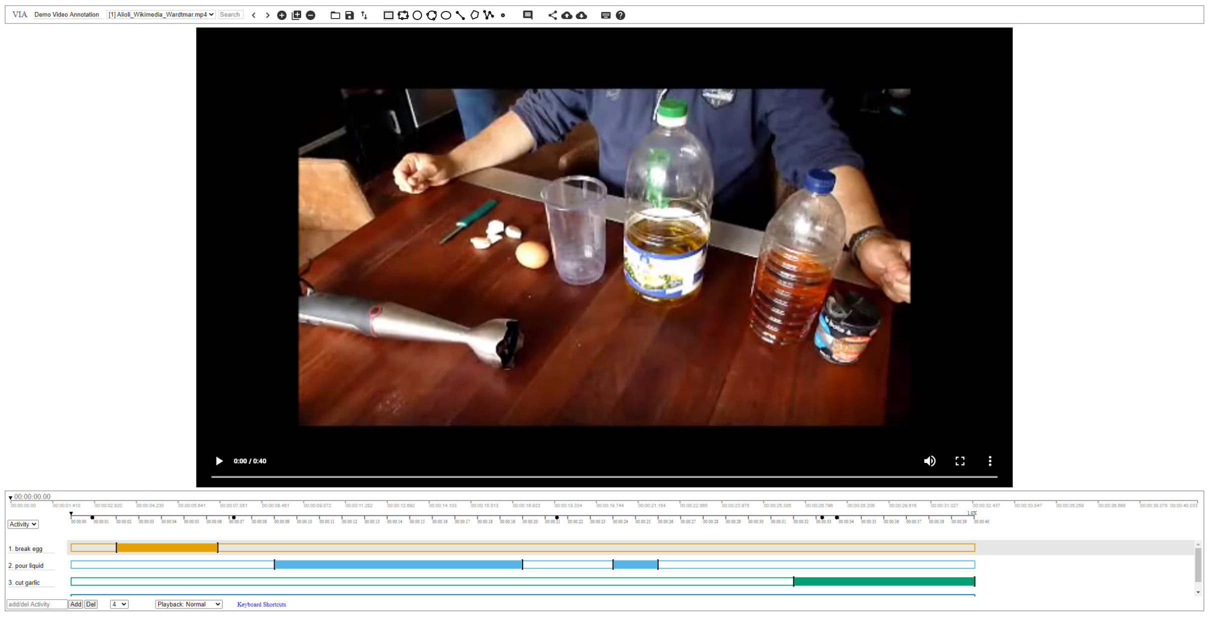


Figure 3.7: This figure shows an example of the VIA video annotation tool Dutta and Zisserman [28][29].

in the correct position. Additional intermediate points are needed between control points to force the curve to fit to a boundary in the image.

3.5.2 Action Recognition Annotation

VGG Image Annotator (VIA) was developed in 2019 by Dutta and Zisserman [28][29], and is an image, video and audio annotation tool. This tool is unlike the other tools as it is a web application that runs in a web browser, and is less than 1.5Mb in size, making it easy to distribute. The video annotations consist of tiers of activities, and for each activity segments can be created to block out temporal regions where the activity is occurring. An example of this can be seen in Figure 3.7. VIA exports the data in the common CSV (comma separated value) format, meaning it is easy to process the data.

For this thesis, the annotation tools mentioned in this section do not meet the requirements for the datasets needing to be annotated. Firstly the Pixel Annotation Tool was not available at the time of annotation, and additionally the difficulty to annotate small objects in a scene meant that it would not be applicable for the Newborn Resus dataset. VGG's VIA tool would have been a good contender for annotation of the temporal data in the Newborn Resus dataset if it was available at the time of annotating. The tools used to label the ground truth data in this thesis were an in-house image annotation tool for semantic segmentation annotation, and the non verbal annotation tool (NoVA) for temporal action labels for the Newborn Resus dataset. These two tools are described in detail in Chapter 5, as well as my improvements to these pieces of software.

Chapter 4

Datasets

In this chapter, the three major datasets used throughout this research will be discussed. These are: gLitter, Densely Annotated Video Segmentation (DAVIS), and Newborn Resus. Newborn Resus and gLitter are two privately collected domain datasets. Newborn Resus is an action recognition dataset, whereas gLitter is a object detection and segmentation dataset. These are two domains that were explored with the use of deep learning and small domain-specific datasets. DAVIS is a publicly available challenge dataset for foreground segmentation. Both the gLitter and Newborn Resus datasets were used in a supervised manner, whereas the DAVIS dataset was used to pre-train the deep-learned superpixel subnetwork in an unsupervised manner.

In this thesis there are several terms used to describe different types of annotation for segmentations. Polygon annotations are created by connecting points together to form a polygon. A polygon can be saved as either a list of coordinates of the points that it consists of, or converted to a mask by rasterization. If there are multiple segments these can be identified by using unique IDs or unique colours. When using unique IDs, an ID map is created where each pixel is assigned to a specific class. Pixel-perfect annotations are annotations where the segmentations are accurate down to the pixel level. Pixel-perfect annotation comes in both ID maps and colour segmentation images. Pixel-perfect annotations are the gold standard as they are 100% correct, but the major disadvantage to pixel-perfect annotations is the time taken to annotate the data. Near pixel-perfect annotations is a term coined in this thesis to mean annotations that are not pixel-perfect but are much closer than rough outlines of objects. Near pixel-perfect annotations have less false positives than polygons being used for rough outlines. The imperfect boundaries between segments is where the majority of the annotation time is saved. These near pixel-perfect annotations are stored in either IDs maps or colour images.

4.1 gLitter

In May of 2017 Prof Michel Valstar and colleagues proposed a Kickstarter project named gLitter. The aim of this project was to produce a fleet of litter-picking robots. The fleet would be used in areas such as parks and estates to combat the increasing levels of litter being dropped there. The fleet would be comprised of two main types of bots: drones, for finding and identifying litter around the designated areas, and rovers, for collecting and disposing of the litter, Figure 4.1.

The drones would be given a working area in which they would be allowed to operate. This area would be defined using GPS coordinates and the drones would periodically patrol the area looking for litter that had been dropped on the floor. To identify the litter on the floor, the drones would be equipped with RGB cameras. The video feeds from the camera would be fed through a litter detection network to classify if an item is litter or not. Once a piece of litter had been identified, the approximate coordinates and reference image would be sent to a central control unit (CCU) to be added to a list of known litter pieces that needed to be collected.

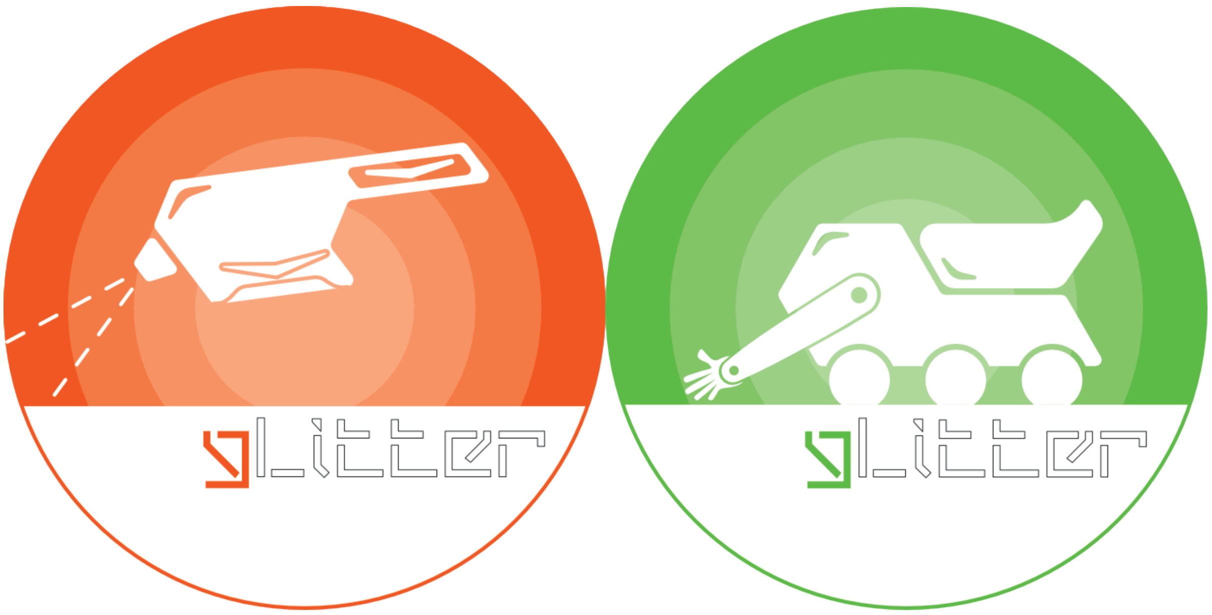


Figure 4.1: This figure show mock sketches of the drone and rover for the gLitter project.

After the drones had finished their patrols, the CCU would calculate optimum routes for the rovers to go collect the litter. Like the drones, the rovers would also be equipped with an RGB camera for detecting litter. The rover would make its way to the approximate GPS coordinates, then begin a scavenging routine, where the rover moves around the target location looking for the litter. Once found, the litter would be collected and an update sent to the CCU. When either the rover had completed its route, or the rover's personal bin had filled, it would return to its station to either charge or to empty its bin, before continuing with its tasks.

To test the feasibility of the project, the first problem that was addressed was the detection of litter in a frame. This required a custom dataset to be collected. This dataset would be collected by the backers of the Kickstarter project using their mobile phones. The backer would use a custom app to record short videos of litter. The videos would start with the litter out of frame, and the user a few meters away. The user would then walk towards the litter with their phone at arms length, recording the ground until the litter comes into frame. Once in frame they would centre the litter on the screen and after a few seconds, stop the recording. This video would then be uploaded to a sever hosting the collected dataset.

Prof Valstar and the other founders invited me to join the project as it would have fit nicely into my work, as the litter detection system would require a custom built object detection system that needed to be initially trained on a small domain-specific dataset. Additionally, this would have been a good stand-in for the Newborn Resus dataset that had been delayed and was going to take many months to become available. I accepted the invitation to participate in the Kickstater project.

Before the backers signed up to the project, Prof Valstar and I gathered approximately 300 videos of varying types of litter in a local park over several visits. These videos consisted of differing light conditions, varying backgrounds, and objects of litter with a wide variety of shapes, sizes and colours. To annotate the dataset, the Image Annotation Tool described in Chapter 5 was used to achieve near pixel-perfect object segmentation. In this dataset, a single master class was created named *litter*, which encapsulated all types of litter. In Figure 4.2 12 samples can be seen from the labelled dataset.

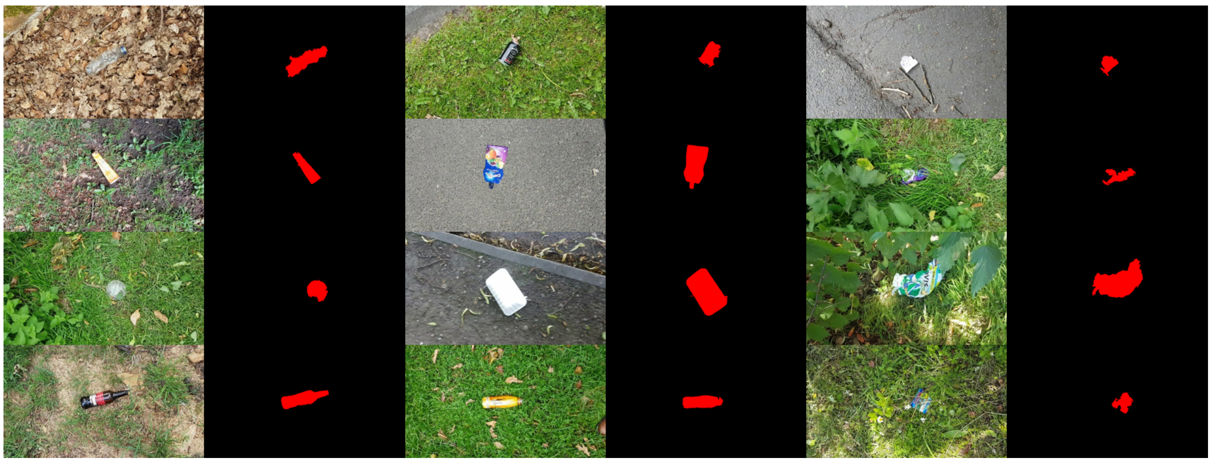


Figure 4.2: This figure shows 12 example frames from the gLitter dataset and their corresponding ground truth segmentation masks.

The dataset was then further expanded to 550 videos, including a small sample of negative videos. The negative videos consisted of no litter in the full video, whereas the positive videos had to contain at least one sample of litter, and the video had to end with the main piece of litter being in the centre of the frame.

In this thesis, the gLitter dataset is used as an example use case for training a segmentation network with small amounts of data and complex object boundaries.

4.2 Newborn Resus

The second domain this thesis focuses on is action recognition, and for this a clinical dataset was used. This dataset was collected by Dr Don Sharkey at the Queens Medical Center (QMC) in Nottingham, United Kingdom, and consists of video of neonatal resuscitation. The data was collected along side a study being performed by Henry et al. [47]. The data collection process started in 2016 and took two years to complete. The data collection overran by a few months due to limited numbers of parents signing up to volunteer their videos. In the data collection programme there are two routes a baby can be delivered by either; at term via a caesarean section after a normal pregnancy, or the babies are born by any route, such as natural pregnancy. When the second route is taken, babies are at a higher likelihood of needing either stabilisation or resuscitation.

In total, 70 videos of newborn babies were collected, and all babies in this dataset were in good health after the videos. Not all of the babies in the dataset required stabilisation or resuscitation. Once the dataset had been collected, Dr Don Sharkey had an honours year medical student label the dataset as part of their thesis from September 2018 to December 2018. The full dataset was then given to me at the end of February 2019.

Figure 4.3 shows an example Resuscitaire machine used for the collection of the Newborn Resus dataset. The camera was connected to the light arm above the cradle, allowing for a top down view of the cradle. The camera used for the collection of data was the Logitech C615, with the resolution set to 720p. The following additional devices were used to record the heart rate of the infants: electrocardiogram (ECG), pulse oximeter (PO) and fhPPG device.

The cameras used to collect these videos did not have real-time automatic exposure correction, thus some videos were deemed unusable for deep learning with tiny datasets. This was because there was either too much variability, or too much information loss due to severe under- or over-exposure. In regards to exposure, the exposure change is generally due to a lighting change in the room. This could be due to the recording being started with something obstructing the camera lens so no light can get in, this leads to an over-exposed video. In contrast, when a light is turned off to allow the baby to rest without glaring lights, this has the opposite effect and causes the videos to become under-exposed. As the exposures have gone to the extremes, this can not be corrected for and the lost information is not recoverable.

The Newborn Resus dataset was collected to both help review the performance of paediatric staff after the procedure, and to be used as a training aid for future members of staff. Currently, the paediatric staff rely on memory alone to evaluate the performance

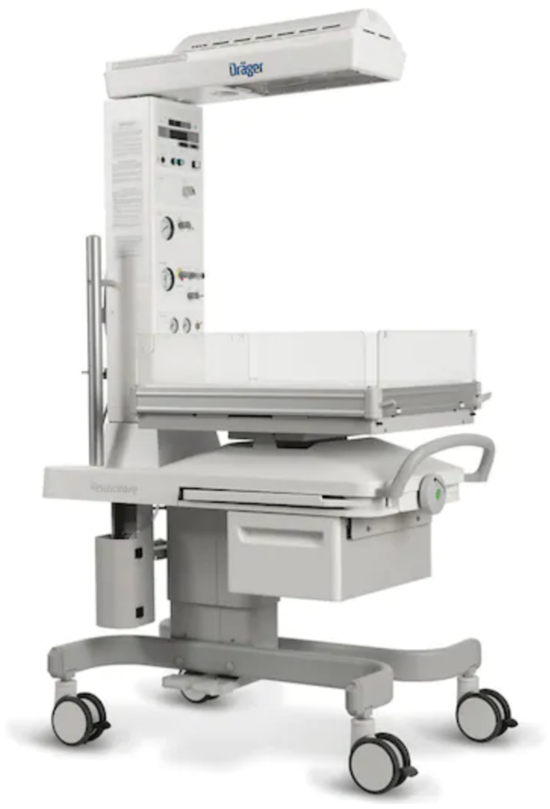


Figure 4.3: This figure shows an example Resuscitaire machine [23] used during the study by Henry et al. [47].

of the resuscitation after the procedure. This method can be unreliable because the staff are likely to have been in a high-pressure situation and may not be able to perfectly recall what happened and when. During their training, the staff are taught to follow a very strict procedure but due to the pressure of the situation they may not always follow every step in the procedure. Having a system that could help hone their skills faster would be very valuable to the field as it would mean staff would be trained faster and to a higher level.

Recording the procedure would be a big step forward for the reviewing process, as there would be video evidence to show exactly what happened at what time. However, these videos can be quite long as they are started when the mother goes into the second stage of labour, where the baby's head has moved out of the uterus and into the vagina. From this point to the baby being born can take anywhere from thirty minutes to an hour or longer. Next the baby is taken to the resuscitation bed, this is where the reviewing process begins. The paediatricians first have to find the correct time in the video that the baby arrives at the bed. Then they must find at which time points the various steps of the procedure occur. Finding all of these steps in the videos takes up valuable time from trained paediatricians. Thus an automated system would save them many hours in which they could be caring for other patients.

This dataset was labelled with 19 actions, Table 4.1, by the medical student at five

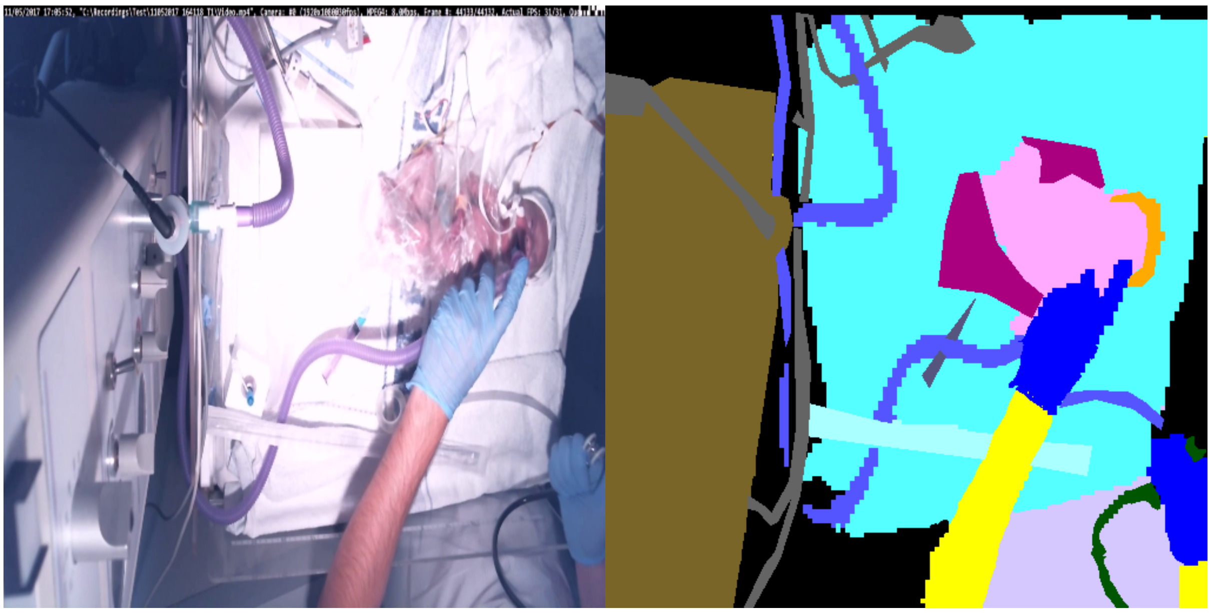


Figure 4.4: This figure shows an example frame and its semantic segmentation label for the Newborn Resus dataset.

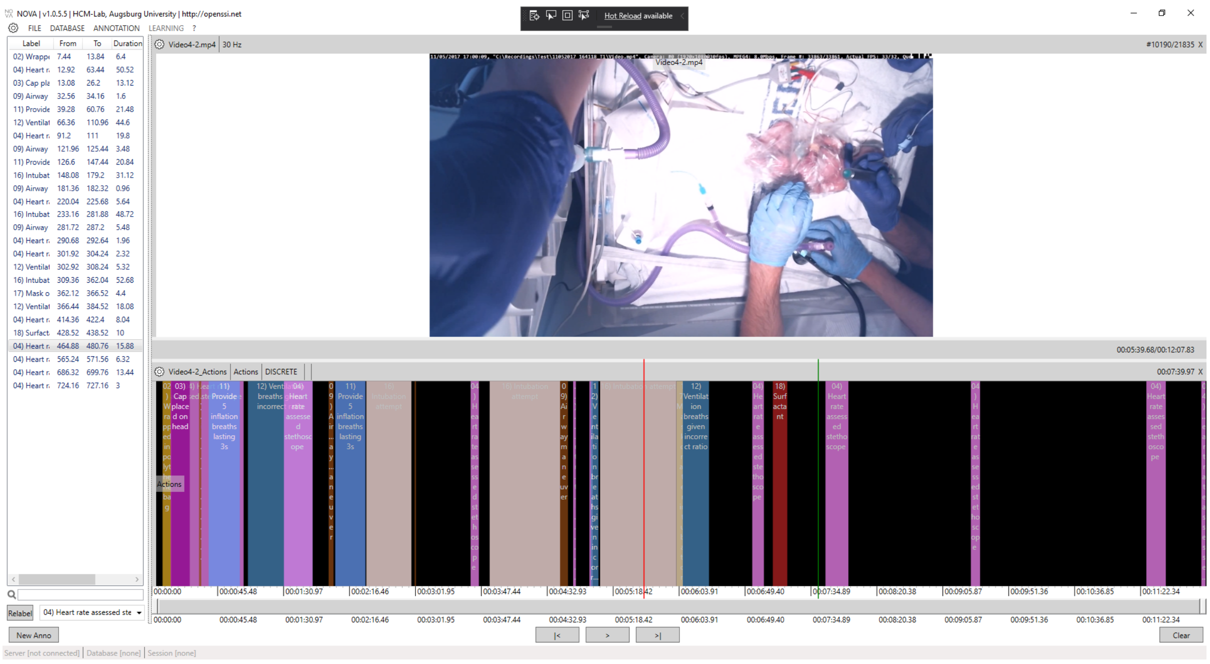
second intervals, meaning that it would need to be relabelled with real-time labels if it were to be used in an action recognition network. Due to the video being split into five second windows, the start and stop points of the actions are not precise enough for a network to learn the intricacies of the actions with such small amounts of data. In regards to this thesis, the Newborn Resus dataset was labelled with 23 semantic segmentation classes including an unknown class (shown in Table 4.2), and 19 actions (shown in Table 4.1). The unknown class represents all parts of the frame that do not belong to one of the other 22 classes. The software used to annotate these labels was the Image Annotation Tool (IAT) for semantic segmentation and NoVA for the actions. These two pieces of software are described in Chapter 5. Examples of the annotations for each labels can be seen in Figures 4.4 and 4.5, respectively.

Semantic Segmentation Classes					
⬛	Unknown	🟦	Gloves	🟦	Bed
🟪	Baby	🟦	Pipes	🟩	Stethoscope
🟨	Arms	🟧	Hat	🟫	Machines
🟥	Syringe	🟩	Blue Towel	🟩	Scissors
🟨	Electric Patches	🟪	Mobile	🟪	Plastic Bag
🟦	Packaging	🟦	Umbilical Cord Clamp	🟩	Pink Jacket
⬛	Wires	🟪	Name tag	🟪	Umbilical Cord
🟪	Clothing	⬜	Airway Opener		

Table 4.2: The 23 semantic segmentation classes for the Newborn Resus dataset and the colour used to identify them.

Class ID	Description
A	Dried with towel
B	Wrapped in polythene bag
C	Cap placed on head
D	Heart rate assessed stethoscope
E	Attachment of pulse oximeter
F	Pule oximeter adjusted
G	Place ECG on chest
H	ECG adjusted
I	Airway manoeuvre
J	Inflation breaths given incorrect ratio
K	Provide five inflation breaths lasting three seconds
L	Ventilation breaths given incorrect ratio
M	Ventilation breaths given for 30s and 1s each
N	Stimulation the infant
O	Cleared away suctioning
P	Incubation attempt
Q	Attach mask on after incubation
R	Administer suffoctant
S	Insert NGT (feeding tube)

Table 4.1: List of all actions provided in the Newborn Resus dataset.


NOVA | v1.0.5.5 | HCM-Lab, Augsburg University | http://openssi.net
FILE
DATABASE
ANNOTATION
LEARNING
?
Video4-2.mp4 | 30 Hz
#10190/21835
Hot Reload available
00:05:39.68/00:12:07.83
Video4-2_Actions | Actions | DISCRETE
00:07:39.97
Actions
Relabel
04) Heart rate assessed ste
New Anno
Clear
Server [not connected] | Database [none] | Session [none]

Label	From	To	Duration
02) Wrappe	7.44	13.84	6.4
04) Heart r.	12.92	63.44	50.52
03) Cap pla	13.08	26.2	13.12
09) Airway	32.56	34.16	1.6
11) Provide	39.28	60.76	21.48
12) Ventilat	66.36	110.96	44.6
04) Heart r.	91.2	111	19.8
09) Airway	121.96	125.44	3.48
11) Provide	126.6	147.44	20.84
16) Intubat	148.08	179.2	31.12
09) Airway	181.36	182.32	0.96
04) Heart r.	220.04	225.68	5.64
16) Intubat	233.16	281.88	48.72
09) Airway	281.72	287.2	5.48
04) Heart r.	290.68	292.64	1.96
04) Heart r.	301.92	304.24	2.32
12) Ventilat	302.92	308.24	5.32
16) Intubat	309.36	362.04	52.68
17) Mask o	362.12	366.52	4.4
12) Ventilat	366.44	384.52	18.08
04) Heart r.	414.36	422.4	8.04
18) Surfact	428.52	438.52	10
04) Heart r.	464.88	480.76	15.88
04) Heart r.	565.24	571.56	6.32
04) Heart r.	686.32	699.76	13.44
04) Heart r.	724.16	727.16	3

Figure 4.5: This figure shows an example video and its actions labelled for the Newborn Resus dataset.

4.3 Densely Annotated Video Segmentation (DAVIS)

As part of the object detection network, the novel technique of incorporating deep-learned superpixels was used, as described in Chapter 6. To train the deep-learned superpixel network in an unsupervised manner, a sufficiently large dataset was needed. This dataset did not need to be semantically similar to the target domain, in this case they were litter detection and resuscitation image analysis. The network needed to have sufficient variability in the samples that a wide variety of superpixel shapes and sizes could be captured.

Additionally, a dataset with good quality videos and annotation would be preferable as this removes the introduction any superpixel ambiguity being trained 'into' the superpixel model by poor quality images. Because the two domains that this work uses consist of video data, the selected dataset for training the learned superpixel model can be reasonably expected to result in the best possible model if it was also comprised of video data. Otherwise any temporal artefacts that are inherent in videos, such as motion blur, could be missing.

Several datasets were considered for use in the semantic segmentation research: Video Segmentation Benchmark 100 (VSB100) [36, 92], Discontinuity-Aware Video Object Cutout[110], SegTrackv2 Dataset[62], Efficient Hierarchical Graph-Based Video Segmentation[41], and DAVIS [63, 78, 83].

When training a deep network with small datasets, the quality of the target annotations is important, as shown in Chapter 6.3. However, with sufficiently large datasets the networks are able to generalise and learn closer to true segmentation of objects.

The VSB100 dataset consists of 100 pixel-perfect video segmentations, however on average, each video is only about five frames long. Therefore it was considered to be too small a dataset for the purpose it would have been used for. Discontinuity-Aware Video Object Cutout consists of only 21 videos, each lasting a few seconds with pixel-perfect annotation, however consists of only 10 classes. The SegTrackv2 dataset consists of 14 videos and 24 classes. Some classes are multiples of the same object in a scene, an example of this is the penguins video where the front six penguins are annotated but the ones in the background are not. Each frame has a pixel-perfect annotation. Finally, the Efficient Hierarchical Graph-Based Video Segmentation dataset consists of 15 videos that are pixel-level annotated.

The DAVIS dataset is a dataset used to create challenges where video segmentation techniques are tested. DAVIS' focus is foreground segmentation, but contains multiple different classes. DAVIS started in 2016 with 50 high definition videos (1080p), with each video containing a different class. The following year it was increased to 120 classes, with 90 classes used for training and 30 for validation. The ground truth segmentations consist of pixel-perfect segmentation of the foreground object or objects. Samples from the DAVIS dataset can be seen in Figure 4.6.

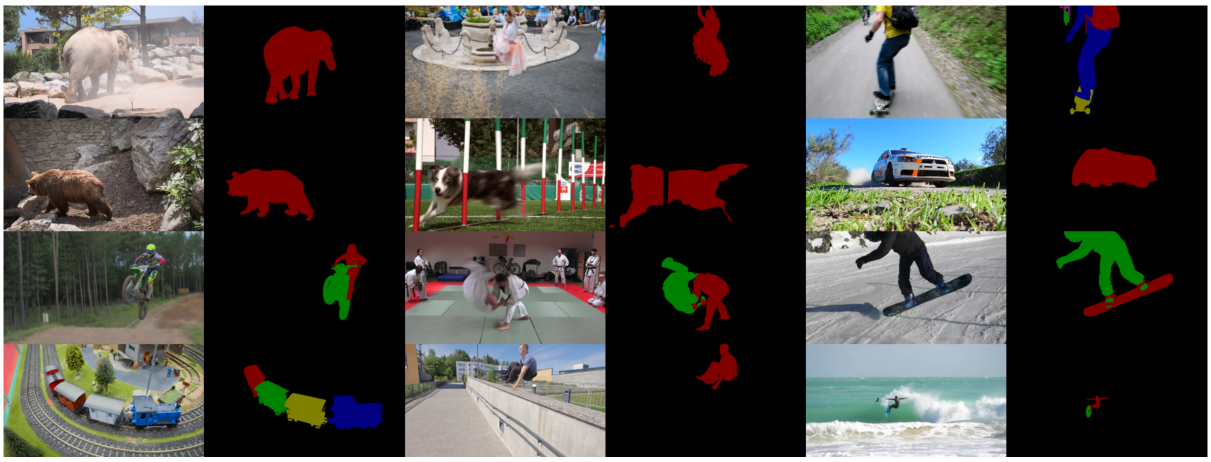


Figure 4.6: This figure shows 12 example frames from the DAVIS dataset and their corresponding pixel-perfect ground truth segmentation masks.

The DAVIS dataset was chosen over other similar datasets for two main reasons. Firstly, it has superior segmentation annotation. Most commonly available datasets do not use pixel-perfect segmentation, and often lean towards rough outlines or bounding box segmentation (object detection) for annotation. However, if a vague ground truth is used, this means that the network can only ever be as accurate as these poor ground truths that are passed to it. Secondly, DAVIS includes several common challenges of segmentation: appearance change, occlusion and motion blur.

Chapter 5

Annotation Tools

This chapter discusses the creation and evolution of the annotation tool, Image Annotation Tool (IAT), that was developed to label the ground truth data for semantic segmentation of the Newborn Resus and gLitter datasets. Additionally discussed is the adaptation of the NOn Verbal Annotation (NoVA) tool which was originally going to be used for labelling the datasets.

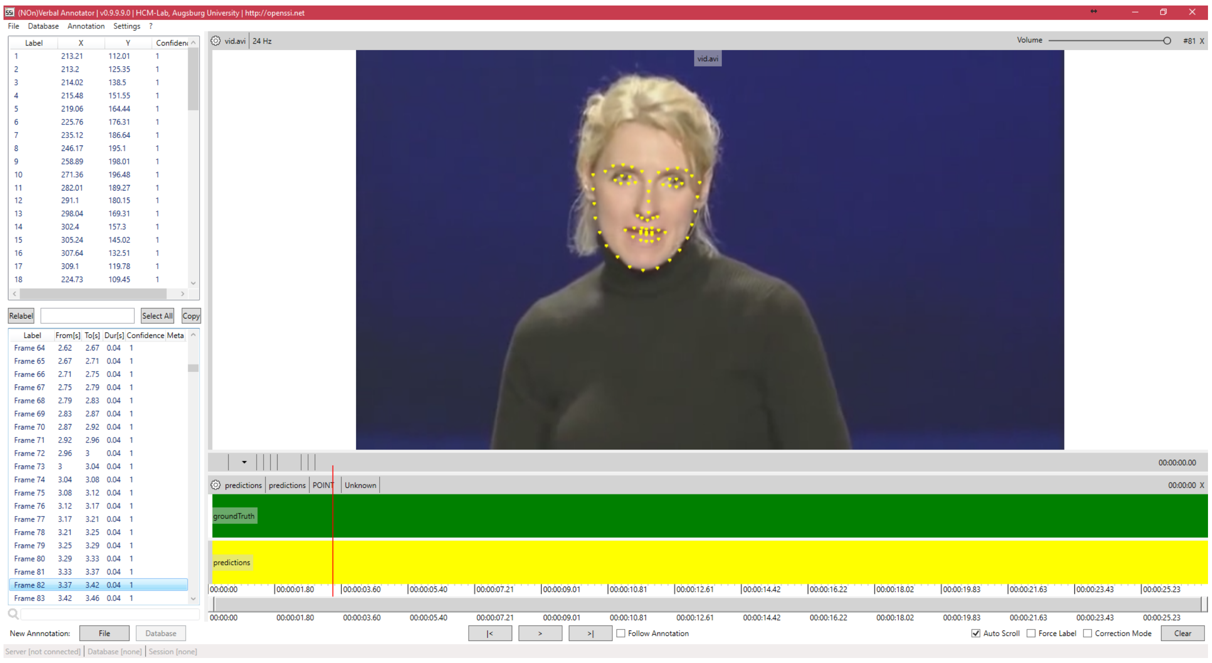
5.1 NOn Verbal Annotation (NoVA) Tool

NoVA is an annotation tool developed by Baur et al. [5] in 2015 to perform 1D temporal annotations on temporal data, which are most commonly used for video or audio data. These annotations can be separated into three sub classes: continuous, discrete, and free. Continuous annotations are used to plot a variable over time that exists between an upper and lower limit. Examples of this are the heartbeat of a person, which is recorded at every time frame, or valence and arousal measures for emotions, where both valence and arousal can be split into 1D continuous data rather than combined as 2D information. Discrete annotations are used to label information that occurs over a time, with a set start and finish time. These annotations are made up of a predefined dictionary of annotations. An example of this is facial expression. It is widely accepted that there are six basic emotions, one of which is happiness, often detected with the smile action. This action has a start time, when the person begins the smile, and an end time, when the smile has finished. Another example is labelling actions taking place in a scene from when the action starts to when the action is complete. Finally, free annotations are similar to discrete annotations, however they do not have a predefined dictionary. The main use for this is speech transcription, where each spoken word can be labelled from the moment the word is started to when the word is finished. NoVA uses a tier system for its annotations. Each tier consists of a single annotation type and its parameters are defined when creating a tier, *e.g.* the created dictionary of classes for discrete annotations. This allows both multiple and different annotations to be created for the same input data in a single project. For example, both speech and emotions can be recorded at once.

One of the main problems with labelling data is the man-hours it takes to label a full dataset, as these datasets often consist of hundreds of thousands of data samples, either in terms of individual images or frames in a video. NoVA makes use of several techniques to reduce the time spent labelling the dataset. The three main techniques used by NoVA are: active learning, cooperative learning and collaborative annotations. Active learning is a concept where the algorithm asks the user to label specific data points in a similar way to how a child asks what new things are called. As only the requested data points are labelled, there are fewer labels in the dictionary for the learner to learn than classic labelling, thus speeding up learning. Additionally, the learner only asks for the next most informative data points, therefore there is less redundancy [111]. Cooperative learning is where both the system and user annotate unlabelled data [22]. Initially the user would label the first few frames in a dataset, then the system would annotate the rest of the dataset. As the computer will not always get it right, each label is given a confidence value which is calculated using a pre-existing classifier. If the confidence is lower than a given threshold then the user is prompted to correct it. Any corrections made are then propagated through the network to improve the labels. Collaborative annotation is a distributive technique which means that several users on different machines, even in different geological locations, can annotate the same dataset simultaneously. This allows the dataset to be partitioned between users to increase the speed of labelling the full dataset. Using collaborative annotations along with active and cooperative learning techniques, allows the request from the system to be addressed by any one of users and thus reduces the time the system has to wait for human intervention.

NoVA works well with 1D data but has no way of representing 2D annotations that would exist directly on a video. Therefore, I will be adding this functionality to NoVA. These 2D annotations can be represented as geometric annotations which are drawn on an overlay of the input video. Geometric annotations can be split into three categories: points, polygons, and acyclic graphs. Being able to represent these three categories greatly widens the use cases for this tool.

Firstly, point annotations can be used for data such as facial landmarks. Facial landmarks are identifying points on a persons face which are consistent across all faces, such as the corners of the eyes, the tip of the nose, and the corners of the lips. This can be seen in Figure 5.1. Each point consists of an x & y coordinate on the image, and generally will be part of a larger set of points. Thus the point annotations allow the user to pre-define the number of points per tier. Polygons can be used to represent segmentation problems, from object detection and localisation, to full-scene semantic segmentation. Polygons are a form of cyclic-directed graphs, with only the first and last nodes connecting to create the cycle. All other nodes in the graphs have only two connections, input and output. To represent a polygon, a tuple list of points(p) and lines(l) is used. Each l_n is connected from its paired p_n to $p_{n_{+1}}$, and for the final l_n is connected from p_n to p_1. The area encompassed by the polygon is then assigned to a label. Acyclic graphs are a subset of graphs where there are no cycles. They are bi-directional and each node can have multiple connections. The shape of acyclic graphs is defined when the tier is created. Common uses for this are the full skeletal annotations for pose estimation, or skeletal annotations of the hand for hand gesture recognition. Each joint in the skeletal representation is used as a node in the acyclic graph.


(NOn)Verbal Annotator | v0.9.9.9.0 | HCM-Lab, Augsburg University | http://openssi.net
File Database Annotation Settings ?
vid.avi 24 Hz
Volume
#81
predictions predictions POINT Unknown
groundTruth
predictions
Relabel Select All Copy
New Annotation: File Database
Follow Annotation
Auto Scroll Force Label Correction Mode Clear

Label	X	Y	Confidence
1	213.21	112.01	1
2	213.2	125.35	1
3	214.02	138.5	1
4	215.48	151.55	1
5	219.06	164.44	1
6	225.76	176.31	1
7	235.12	186.64	1
8	246.17	195.1	1
9	258.89	198.01	1
10	271.36	196.48	1
11	282.01	189.27	1
12	291.1	180.15	1
13	298.04	169.31	1
14	302.4	157.3	1
15	305.24	145.02	1
16	307.64	132.51	1
17	309.1	119.78	1
18	224.73	109.45	1

Label	From[s]	To[s]	Dur[s]	Confidence	Meta
Frame 64	2.62	2.67	0.04	1	
Frame 65	2.67	2.71	0.04	1	
Frame 66	2.71	2.75	0.04	1	
Frame 67	2.75	2.79	0.04	1	
Frame 68	2.79	2.83	0.04	1	
Frame 69	2.83	2.87	0.04	1	
Frame 70	2.87	2.92	0.04	1	
Frame 71	2.92	2.96	0.04	1	
Frame 72	2.96	3	0.04	1	
Frame 73	3	3.04	0.04	1	
Frame 74	3.04	3.08	0.04	1	
Frame 75	3.08	3.12	0.04	1	
Frame 76	3.12	3.17	0.04	1	
Frame 77	3.17	3.21	0.04	1	
Frame 78	3.21	3.25	0.04	1	
Frame 79	3.25	3.29	0.04	1	
Frame 80	3.29	3.33	0.04	1	
Frame 81	3.33	3.37	0.04	1	
Frame 82	3.37	3.42	0.04	1	
Frame 83	3.42	3.46	0.04	1	

Server [not connected] | Database [none] | Session [none]

Figure 5.1: Here, predictions of a facial landmark detector displayed in NoVA are shown. The points are drawn directly on the video frame. These are drawn in real-time so they can be altered manually if it is necessary to do so. To do this, firstly the frame is selected, along with the point/s to be moved. Then to move the point/s, the user right clicks and drags.

The previously mentioned tier system is advantageous for all three geometric annotations as it allows for multiple objects to be traced at once. For example, when performing facial landmark detection, there may be multiple subjects in the frame and a separate tier for each specific subject could be used, or when using polygons for semantic segmentation, each object would be allocated to its own tier.

Being able to create the ground truth data for training a network is a good use of the tool, but it can also be used to display the predictions over the input data. Because the multiple tier system can load both the predictions and ground truths simultaneously, it is possible to visually inspect the performance of the predictions.

Geometric annotations are used to annotate temporal data from a video sequence. As there are only small changes between frames of a video, especially with high frame rates, the annotation can be copied from frame to frame, then altered to fit the new frame. This greatly speeds up the annotations process because the copied annotations should be relatively close to the where they need to be for the next frame.

5.2 Image Annotation Tool

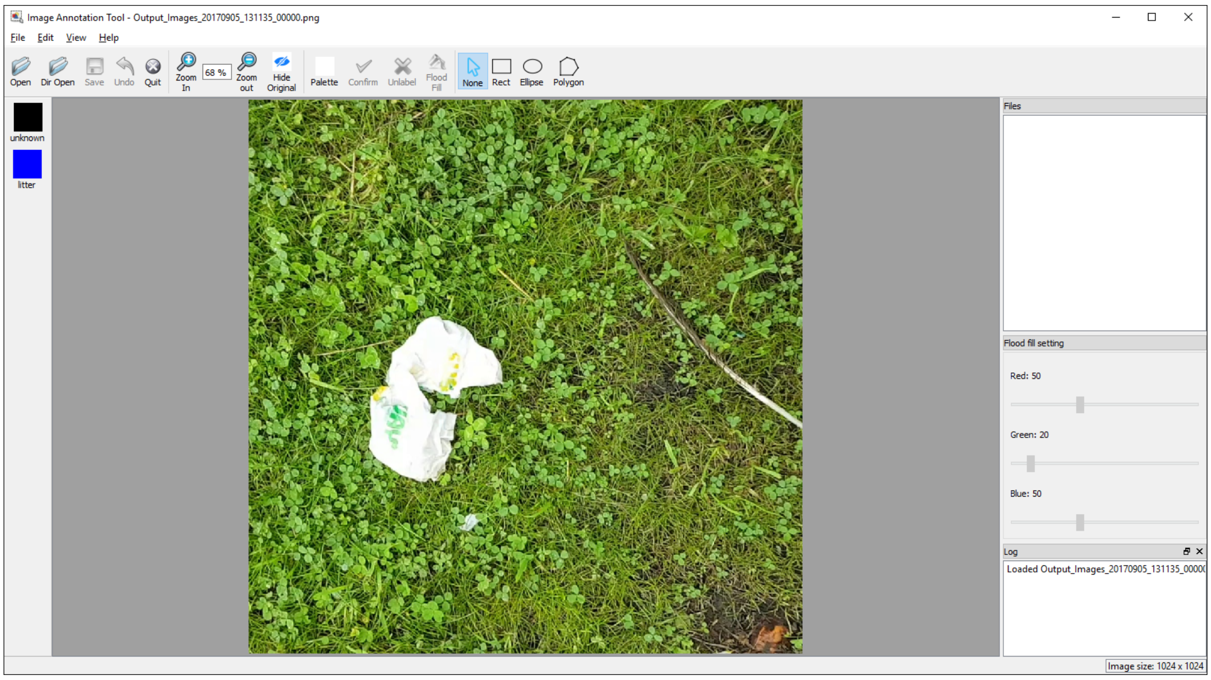
Pixel-perfect semantic segmentation annotation is a difficult and time-consuming task. There have been several methods implemented to decrease the time it takes to complete these annotations. A common method is to use a pre-trained network to perform predictions on the unlabelled data and then to make manual adjustments to this. This is a good method for data that exists in the training set of the software, but when faced with domain-specific datasets that are completely unrelated to the training set, the tool does not work. Another common method is to use polygons to label the data, which can be very fast when the ground truth data does not need to be pixel-perfect, as the polygons can be rough outlines of the object. However, when working with small datasets, the more precise the ground truth annotations, the better the predictions will be, since there may not be sufficient data for the model to learn past the imperfections in the annotations. In Chapter 6 this has been proven for small domain-specific datasets.

Semantic segmentation is classic segmentation with the addition of giving each segment a meaningful label. The use of an over-segmenting method could be used to decrease the speed of labelling. By using an over-segmenter such as superpixels to first segment an image into small semantically-meaningless clusters, and at the same time fine-tuning the size and shape of the superpixels so they do not cross object boundaries, means that the dimensionality of the image is drastically reduced from pixel level to superpixel level. These superpixels can then be combined together to create the full segments for each object in the frame.

NoVA underwent a complete refactor and the geometric annotation functionality was no longer supported, therefore extending NoVA's functionality to include superpixel annotation was no longer feasible.

I was made aware of an in-house piece of software that an undergraduate student had started as a summer internship in the Computer Vision Lab, which could be extended. Image Annotation Tool (IAT) [90] was first created by Jingxiao Ma under the supervision of Dr Michael Pound and Dr Tony Pridmore. In its initial state, the IAT had very limited functionality as it was designed for rough labelling plant datasets. The original user interface can be seen in Figure 5.2.

IAT's initial functionality consists of opening a single image or directory of images, and labelling individual images using basic geometric shapes. The three shapes that IAT uses are the rectangle (Figure 5.3), ellipse (Figure 5.4), and polygon (Figure 5.5). The masks are then saved into a new directory that is created in the same location as the file,



Flood fill setting

Red: 50

Green: 20

Blue: 50

Log

Loaded Output_Images_20170905_131135_0000

Image size: 1024 x 1024

Figure 5.2: IAT original user interface using a sample from the gLitter dataset.

using the file format portable network graphics (.png).

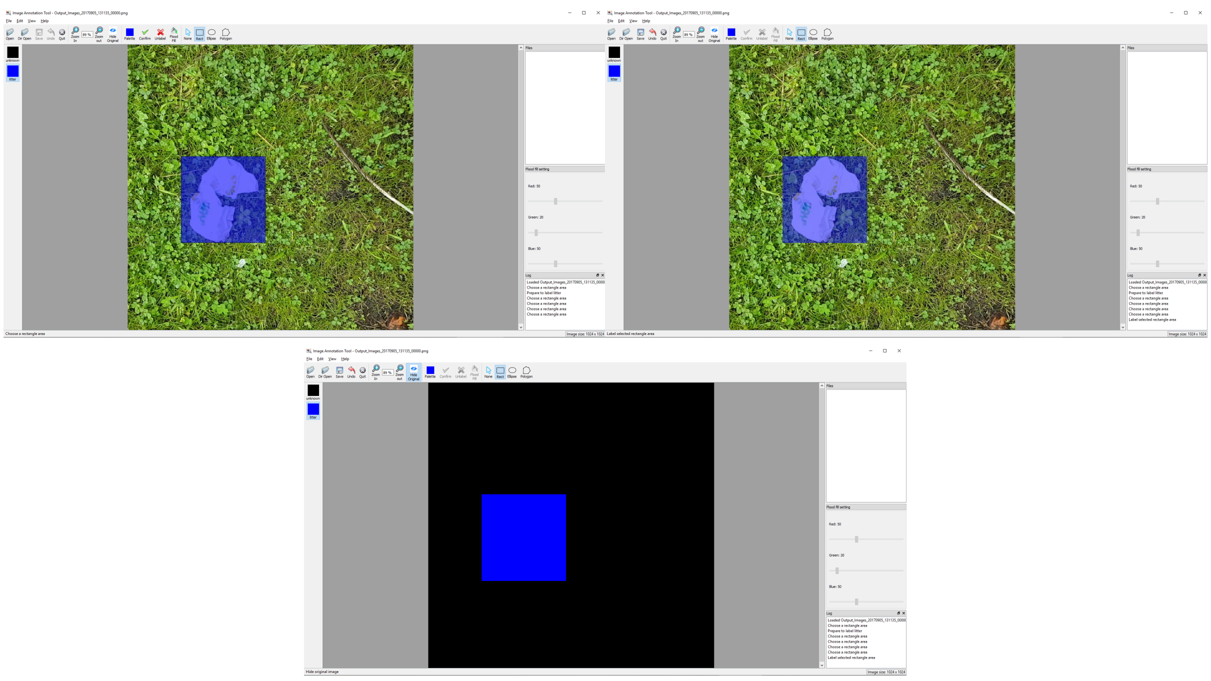
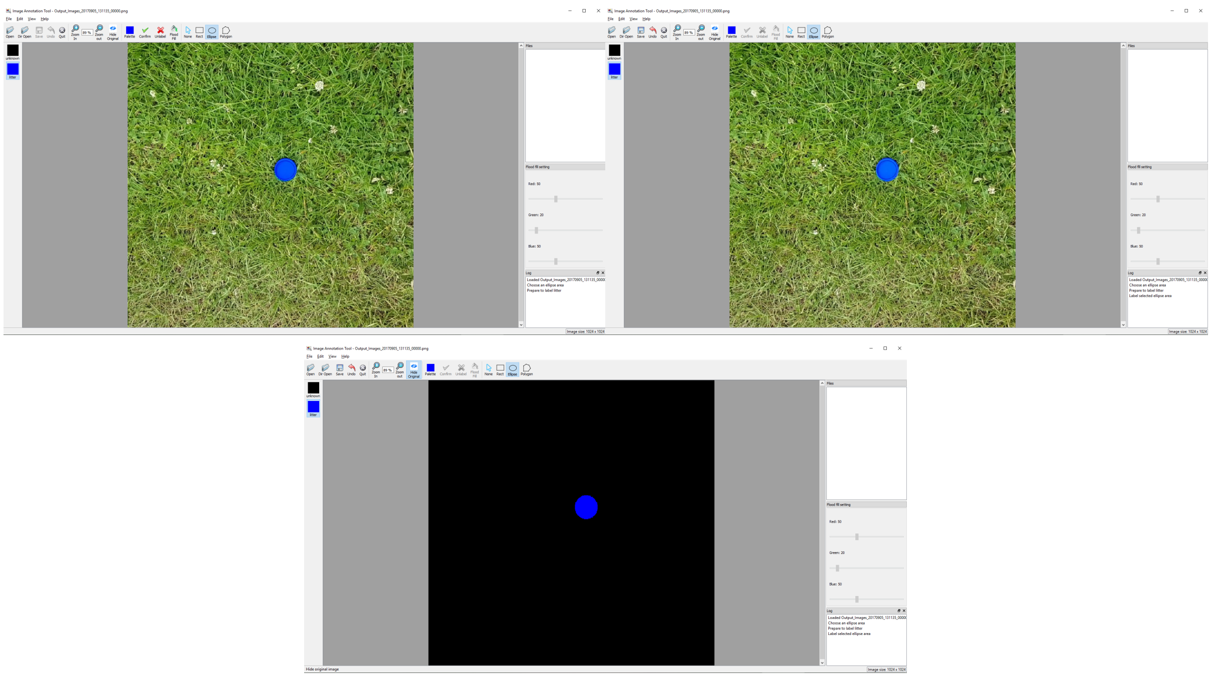


Figure 5.3: Using a sample from the gLitter dataset, the rectangle annotation tool is shown in its three stages of use. First is highlighting the object. Next is confirming the selections, and finally is the mask that is produced.

The rectangle tool works by the user clicking and dragging to select a rectangular area for the label. The initial click is where the primary corner of the rectangle is placed, the secondary corner is the diagonally opposite corner, and is placed when the user releases the mouse click. When the mouse button is released, a temporary outline is created for the label. The user then has to manually confirm the selection and label colour. This tool is useful for giving annotating an approximate area that an object exists in. A functionality that could be added to this in the future, would be to create a list of top left coordinates and bottom right coordinates. This would allow for bounding box annotations to be easily be created with the addition of the masks.

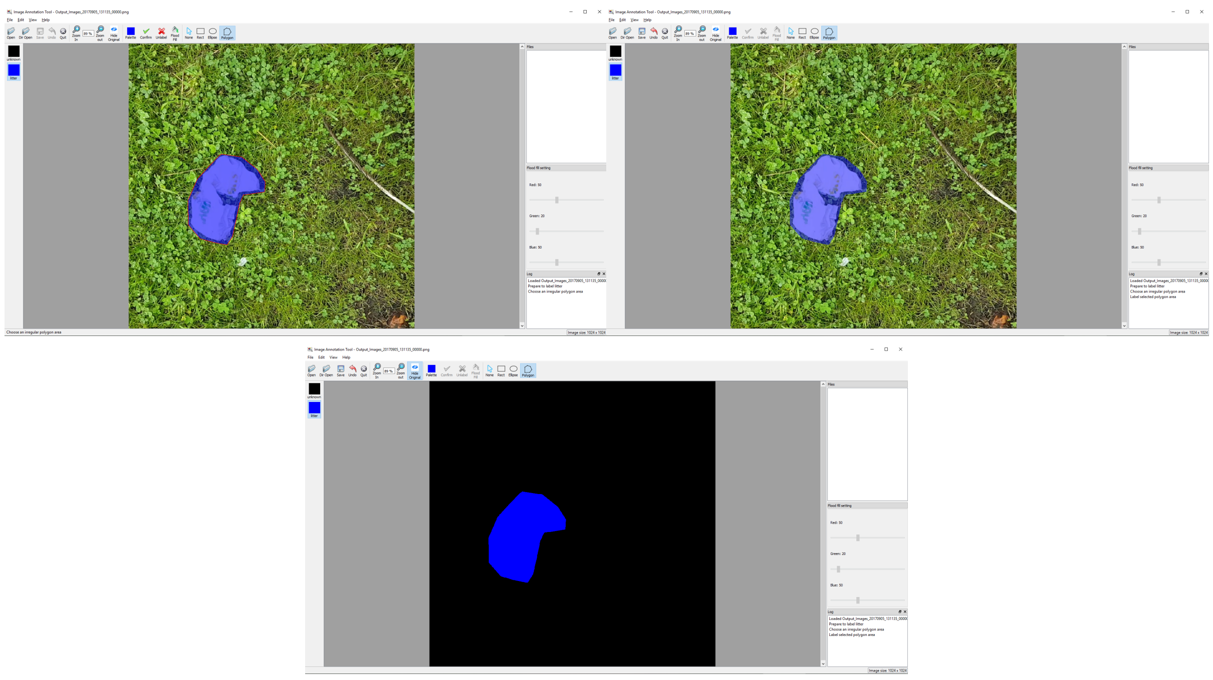


```
Loaded Output_Images_20170905_131135_0000
Choose an ellipse area
Prepare to label litter
Label selected ellipse area
```

Figure 5.4: Using a sample from the gLitter dataset, the ellipse annotation tool is shown in its three stages of use. Firstly is highlighting the object. Next is confirming the selections, and finally is the mask that is produced.

The ellipse tool works similarly to the rectangle tool. To start the annotation, the user first clicks and drags from the top left corner of the object that they want to label. However, instead of the first click being connected to the label, it creates a rectangle, and the mid points between each of the four corners are four points on the ellipse. These points are then connected together to form the ellipse label. The user must confirm the selection and colour to complete the annotation. This tool works best for annotating circular or elliptical objects. In Figure 5.4, the ellipsis tool has been used to annotate the bottle lid. This tool is much easier and faster for round objects. A user experience improvement that would be beneficial for both the rectangle and ellipse tools would be the ability to move and resize the selection after the user has released the left mouse button. This would allow greater fine-tuning of these annotations.

The polygon tool works by left clicking to place points around the object. There is a red dashed line drawn between the points to show where the boundary of the annotation


Flood fill setting
Red: 50
Green: 20
Blue: 50

Log
Loaded Output_Images_20170905_131135_0000
Prepare to label litter
Choose an irregular polygon area
Label selected polygon area

Image size: 1024 x 1024

Figure 5.5: Using a sample from the gLitter dataset, the polygon annotation tool is shown in its three stages of use. Firstly is highlighting the object. Next is confirming the selections, and finally is the mask that is produced.

will lie. Once the object has been fully outlined, to place the final point and enclose the loop, the user double clicks. If the user is happy with the colour and selection they can confirm the annotation. This tool works well for coarse annotations, where the precision of the annotation is not as important, such as labelling large datasets with semantic classes. A consistent usability flaw is the dashed red outline used. This is particularly a problem for colour blind users when being used with certain data such as the gLitter dataset. The current industry standard for this is to use a marching ants approach where the dashes alternate between black and white. This has been added to the future improvements for the program.

To create the labels that appear in the left panel, a “label.txt” file has to be manually created in the file directory of the images to be labelled. This can be seen in Figure 5.2. The format of this file is “red value, green value, blue value, name”. In the example in Figure 5.2 the label file content consists of two classes: unknown (0, 0, 0), and litter (0, 0, 255).

Although the IAT program did function, there were some flaws present which distracted from a good user experience. The worst of these flaws was the program hanging when loading files. This was due to the program running in a single thread, so when a file was loading, the user interface (UI) completely froze. This was easily fixed by separating the UI into its own thread and having other tasks in their own threads. Another issue was the reliability of the double click with the polygon tool. When double clicking, the tool did not always select the correct position. To mitigate any movement due to double clicking, the finishing step of the polygon tool was altered to be a right click instead. A key feature that was missing from the program was the ability to undo and redo previous actions. This meant that to undo something, the user would have to manually fill in the label they had just created, and if it overlapped another annotation this would have to be manually redone. This could take several minutes depending on how complicated the annotations were, and a simple one click solution would be more efficient.

5.2.1 Superpixels

When annotating objects with complicated boundaries, it was very slow to achieve good segmentations with the polygon tool. Additionally there was no relationship between the current finished annotations and the next annotations the user was creating, thus it was easy to go over another annotation in error whilst creating a new annotation.

Phase one of the adaptation of the IAT was to implement a superpixel algorithm to first segment the image, then select these superpixels to create super-regions. Superpixels with the inherent object boundary detection allow near perfect segmentation without having to manually select around each pixel. Additionally, because the location of each superpixel is known, there is no possibility of overwriting another superpixel when adding a superpixel. The superpixel algorithm of choice was the simple linear iterative clustering (SLIC) [1].

The SLIC algorithm was chosen for its fast performance whilst still having good superpixel creation. Additionally, SLIC was to be used for the deep learning superpixel work, thus it was decided to keep this variable consistent throughout.

For the first implementation of superpixels into IAT, a fixed value for sigma and compactness for SLIC was used. The user was able to choose the target number of superpixels before running the algorithm. Once the algorithm was run, the ID map was saved into a csv (comma-separated values) file, and at the same time an overlay of boundaries between superpixels appeared over the image. ID maps are a matrix of the same width and height as the input image, with each cell representing which superpixel the corresponding pixel is assigned to. These ID maps generally use numeric IDs, starting at zero. The user could then select the Add/Remove superpixel tool to start to select the relevant superpixels for the object that they were labelling.

In the current version of the program any annotations made with the superpixel selection overwrite any manual annotation already made. A future feature that may be useful is the ability to add a superpixel but only to the section that exists in the unknown class. The current workflow is to use the superpixel annotation method first before using any of the more basic annotation methods.

When only labelling a few objects in the frame, the technique of manually selecting every superpixel was sufficient, but it did not provide a good user experience when transferring to larger objects or full scene labelling. The first improvement to this was the click and drag functionality. This works like a paint brush in that whichever superpixel the cursor passes over whilst the mouse button is held down, is added to the selection. Again this works well for medium objects but is very time consuming for large objects or scene labelling because the user has to manually fill in the interior of an object.

The solution was to create a custom fill function that worked with the superpixels. This allowed users to select the outline of an object then fill in the central superpixel with a single click whilst holding the correct hotkey, 'Ctrl'. This can be seen in Figure 5.6.

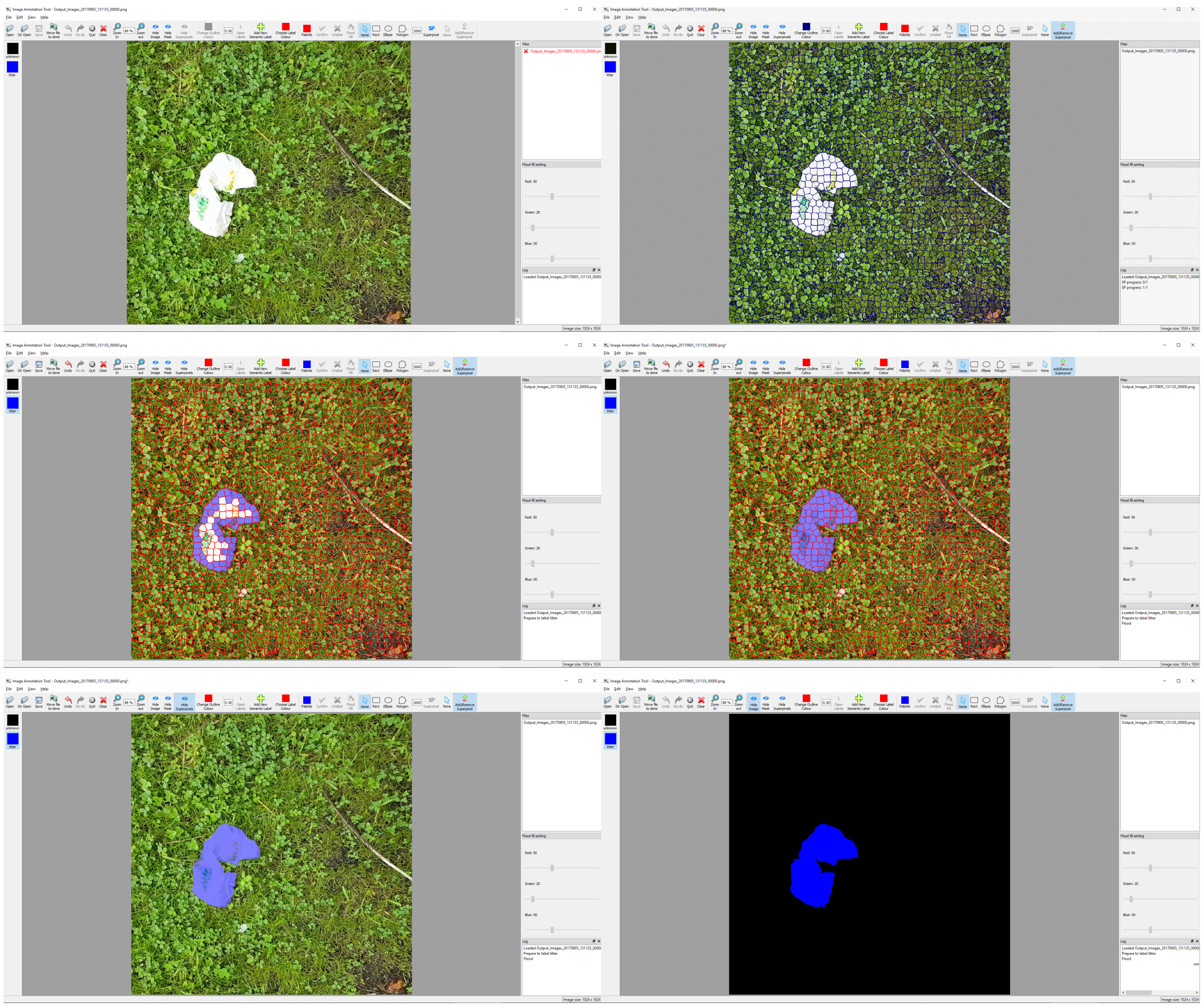


Figure 5.6: The left image in the first row shows the IAT before superpixels have been created. The right image on the first row shows the superpixel outline in an overlay over the input image with a target of 2000 superpixels. The central row shows the litter being outlined by selecting the superpixels on the perimeter of the object. The right image on this row shows the filled region. The final row shows the label mask as an overlay on the left, with superpixel overlay turned off, and the label mask alone on the right.

The ability to create a label file and add labels from the program was missing. A feature was added whereby the user could choose the colour they would like the label to be, and then save it to the file. With this method of adding labels, there was no quick way to select an already used label to label another object of the same class after changing labels. This would be tricky when there are many labels being used, such as when performing full scene segmentation. It was possible to use the left hand window with all of the labels, but to speed up annotations, a second single click with hotkey, 'Alt', was added. This allowed the user to select a label that was already in the output image.

The two domains used within the thesis both use videos/temporal image sequences, thus the functionality to convert a video into an image sequence was added. In addition to this, the functionality to perform the superpixel algorithm on each frame of the video whilst unpacking it from a video was introduced. The video and superpixel functionality were separated into their own separate threads to decrease the time taken to process a video. However, as the superpixel algorithm runs slower than the unpacking of the video, a queue system was implemented to allow all of the frames to be processed without having to wait for other parts of the program.

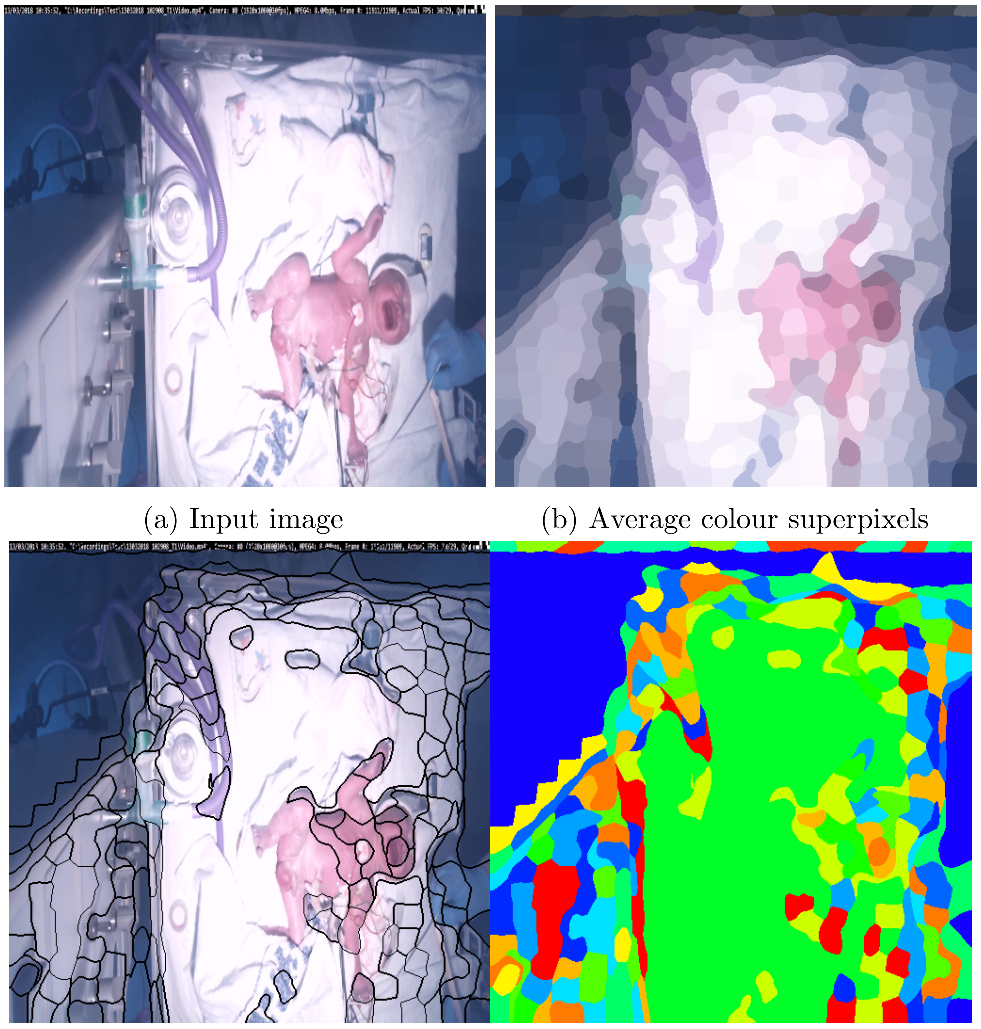
To give the user more freedom with the SLIC algorithm, two additional text boxes were added for the user to specify the exact parameters they want to use on a particular frame or image sequence. The most practical way to use the superpixel system is to find the best superpixel parameters for an image or small collection of images, before applying the algorithm to the entire dataset and beginning annotations.

5.2.2 Superpixel Clustering

To further decrease annotation times, a novel approach to labelling the data was derived which consisted of clustering the superpixels using low level machine learning such as K-means. Initially a handcrafted superpixel algorithm was used to over segment each frame in the Newborn Resus dataset. The averaged colour superpixel images were used to train a clustering algorithm. Next, the clustering algorithm model was applied to a random sample of 50 images from the Newborn Resus dataset, and the cluster that occurred most frequently was given a semantic class. The 50 images each then had a prediction mask which could be used as training data for an hourglass network. The trained hourglass model was used to predict on the 50 more random images from the dataset. The already labelled parts of the image were blanked out and the superpixel algorithm was applied again. The new, most frequent cluster was classified and the masks were added to the training of the hourglass model. This was repeated until the entire dataset had full scene labels. Finally, the users had to validate the segmentation masks to ensure that all classes were detected, and if not, they had to manually add them in.

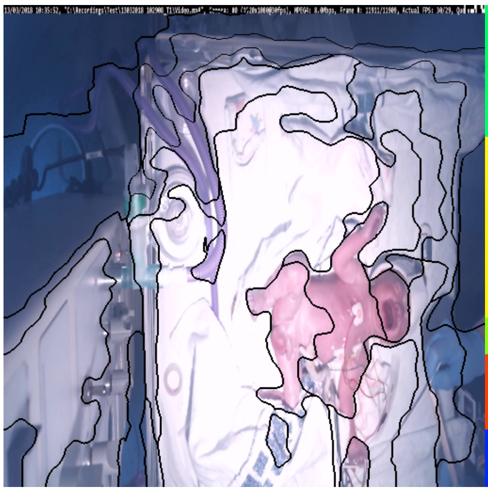
This novel labelling approach would hypothetically speed up annotating full scene semantic segmentation markedly. The first task would be to find a suitable clustering algorithm. Four low-level clustering algorithms were tested, but none of the algorithms performed well enough to make this worthy of the time invested to create the system. Thus it was decided to postpone this feature for future work. The four algorithms tested were: K-means, hierarchical clustering, region adjacency graphs, and mean shift.

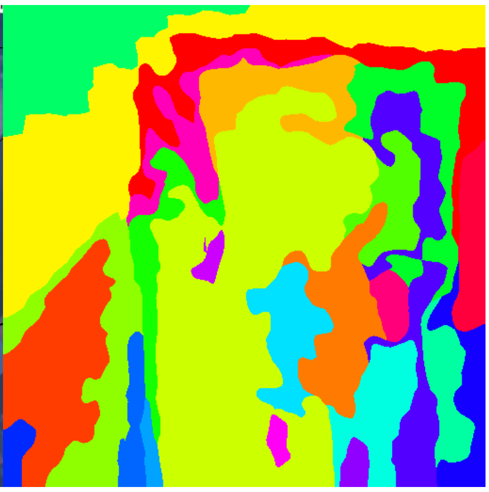
The dataset used to test the clustering techniques was the Newborn Resus dataset. The SLIC algorithm was used to create the superpixels for the clustering and the average colour was assigned to each superpixel, Figure 5.7.a,b. Firstly, the K-means algorithm was tried with a target of 5, 10, 25, and 75 clusters. The K-means algorithm was trained with 500 images rather than the full dataset to test that the theory worked. K-means works by clustering data points by their spatiality in a feature space. Close-together features are assigned to the same cluster. The target number of clusters enforces how large each cluster must be in the feature space. The performance for all K-means variants was poor in that the clusters created were not useful, due to either not being large enough, or jumping object boundaries.

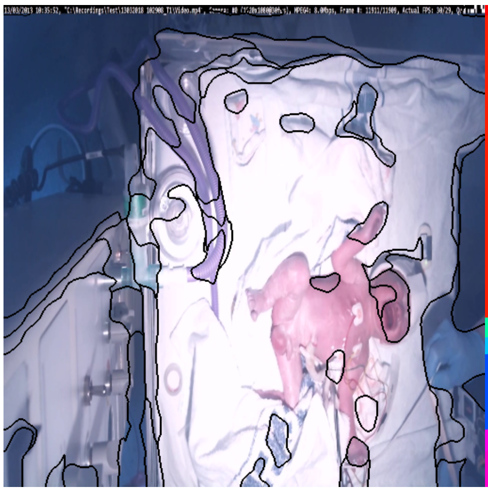


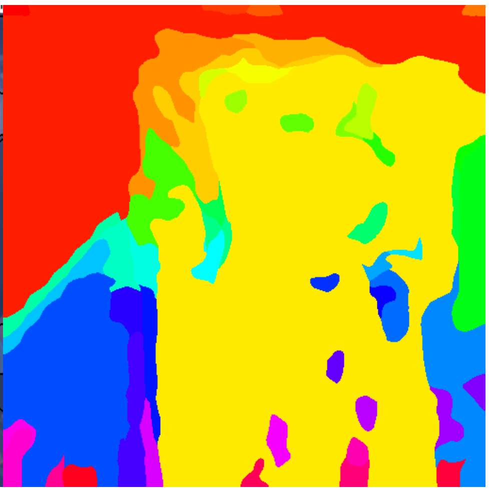
(a) Input image (b) Average colour superpixels

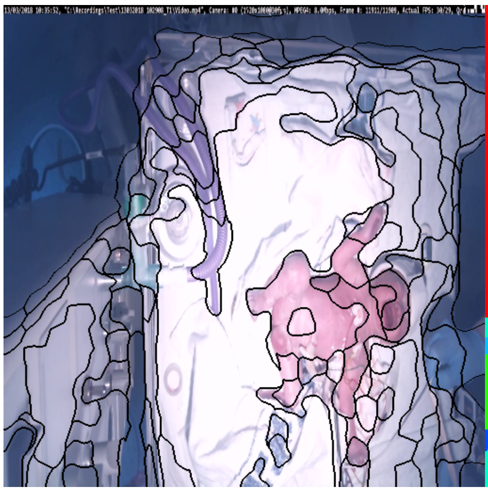
(c) K-means (d) K-mean regions


(e) Hierarchical clustering


(f) Hierarchical clustering regions


(g) RAG


(h) RAG regions


(i) Mean shift

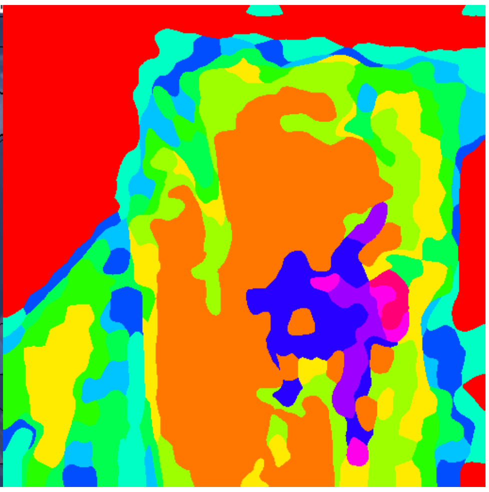

(j) Mean shift regions

Figure 5.7: This figure shows the different clustering algorithms tested on the input image (a). The colours in the region images are to distinguish between regions and do not represent classes. To determine the performance of each algorithm, 50 sample images were inspected manually to determine how well the new clusters followed the objects in the scene. Hierarchical clustering (e) demonstrates the best clustering around the infant by encapsulating in the least segments and with little clustering of unrelated object with regards to the infant. This is closely followed by mean shift (i). K-means clustering (c) is unusable due to the the regions crossing major object boundaries, for example how the infant is included in the same cluster as the bed. The RAG algorithm (g) merges too many superpixels into ambiguous regions.

The next algorithm tried was hierarchical clustering, Figure 5.7.c. Hierarchical clustering is a clustering algorithm which starts from the bottom up and works in levels. Each level clusters similar features together. In the initial level there are as many clusters as data points. With each increasing level the number of clusters decreases and similar features are grouped together. The algorithm stops when the target number of clusters is reached. Again this method performed poorly when using a target of five clusters, but well at 25 clusters. The main drawback to this method was the speed of execution, as it would take too long to train and deploy a usable model for a dataset, especially as this was only trained on the subset of 500 images.

Region adjacency graphs (RAGs), Figure 5.7.g, unlike the previous two methods, work on one frame at a time. RAGs are graphs where the connections between regions are given a weight for how similar the two connecting regions are. Then the RAGs can be evaluated to combine regions together that are below the given threshold limit. When applied to the superpixels, these regions are the individual superpixels, and the combined regions become the clusters. This method did not work well as finding a suitable threshold for the combining of regions differed greatly depending on the frame that was fed into the program. It was difficult to find the threshold for the Newborn Resus dataset, with most thresholds leading to poor clustering.

The final clustering algorithm tested was the mean shift algorithm, Figure 5.7.i. Unlike K-means, mean shift is not provided with a target number of clusters, instead random initialisation points in the feature space are used as starting points for the centroids. The more starting centroids, the better the end clustering will be. A common technique is to take random samples from the dataset as the starting centroids. At each step the centroids use a bandwidth, a radius around the centroid, and calculate the mean position of all data points that are in the bandwidth. The centroid's positions are then updated until they converge. If there are any overlapping centroids, the duplicates are removed. The bandwidth plays a key role in the final centroid locations. With a too-large bandwidth, fewer clusters are found. In an extreme case this could be a single cluster. On the other hand, a too-small bandwidth can lead to each data point having its own cluster. Initially the automatic features were used for the mean shift algorithm, however this produced only two clusters which were unusable, thus the bandwidth was set to 25 and this generated much better clusters. However this method, like previous methods, proved to be too time-consuming for real-time annotations.

To reduce the input feature space of the clustering algorithms, only unique RGB colour values were used. This did increase the speed of training the clustering algorithms, but not enough for training on more than 500 images. Thus other techniques would be needed to be explored to make this method feasible. A possible improvement would be to experiment clustering of different colour spaces such as HSV or LAB. Another experiment that could be tested would be resizing the inputs to a smaller dimension, resulting in different superpixels. A combination of both these improvements may make this feasible, otherwise other clustering techniques would need to be tested.

To conclude, the superpixel clustering algorithm was not used to annotate the Newborn Resus dataset. This was because the performance, with regards to inference speed and clusters produced by the algorithms and techniques tested, did not increase the annotation speed any more than using the superpixels alone. Having a system that further reduces the annotation time for full scene segmentations would be especially valuable for domain specific datasets that do not have much success with pre-trained semantic segmentation networks because their classes have never been seen by their networks before.

Using the superpixel annotation method without clustering drastically improves the annotation speed whilst achieving near pixel-perfect annotations. When the best parameters for the superpixel algorithm are found for a given frame/video, the superpixels closely match the boundaries of the objects in the scene. This allows for the near pixel-perfect annotations to be produced. Achieving similar results by manual annotation with a polygon tool would take several times longer, as this requires manually finding the boundaries between objects by zooming in until individual pixels can be seen. Having these superpixels clustered together would, in theory, further reduce the time during annotations. Figure 5.8 illustrates the updated UI for IAT and shows an example using K-means trained on 500 images. The suggested cluster in this example is completely unusable as several classes have been grouped together, including the infant. Other clusters may be used alongside the superpixel annotations to speed up annotations, however this would be infeasible for the entire dataset, since training the model on the entire dataset would either take large amounts of time or be impractical due to memory limitations.

5.2.3 Usage and Performance

The IAT has been used for both the gLitter and the Newborn Resus datasets. The gLitter dataset consists of 550 videos, and the last frame from each video is taken to create an image dataset of equivalent size. The last frame in each video contains the litter in the centre of the image. The pieces of litter can greatly differ in regards to size, shape, and texture, making it hard to find the best set of parameters for the SLIC algorithm for each instance in the dataset.

Using the updated version of the IAT with the superpixels drastically improved the annotation speed. When using the polygon tool to annotate a complex object it can take over 15 minutes per image to achieve near pixel-perfect annotation. Whereas because the SLIC algorithm closely follow objects boundaries, when there is good contrast the annotation time can be reduced down to two minutes - a speed-up of approximately 7×. Tuning the SLIC parameters took about five to ten minutes for the first image, then for consecutive images that are similar, it could either make use of the previous parameters

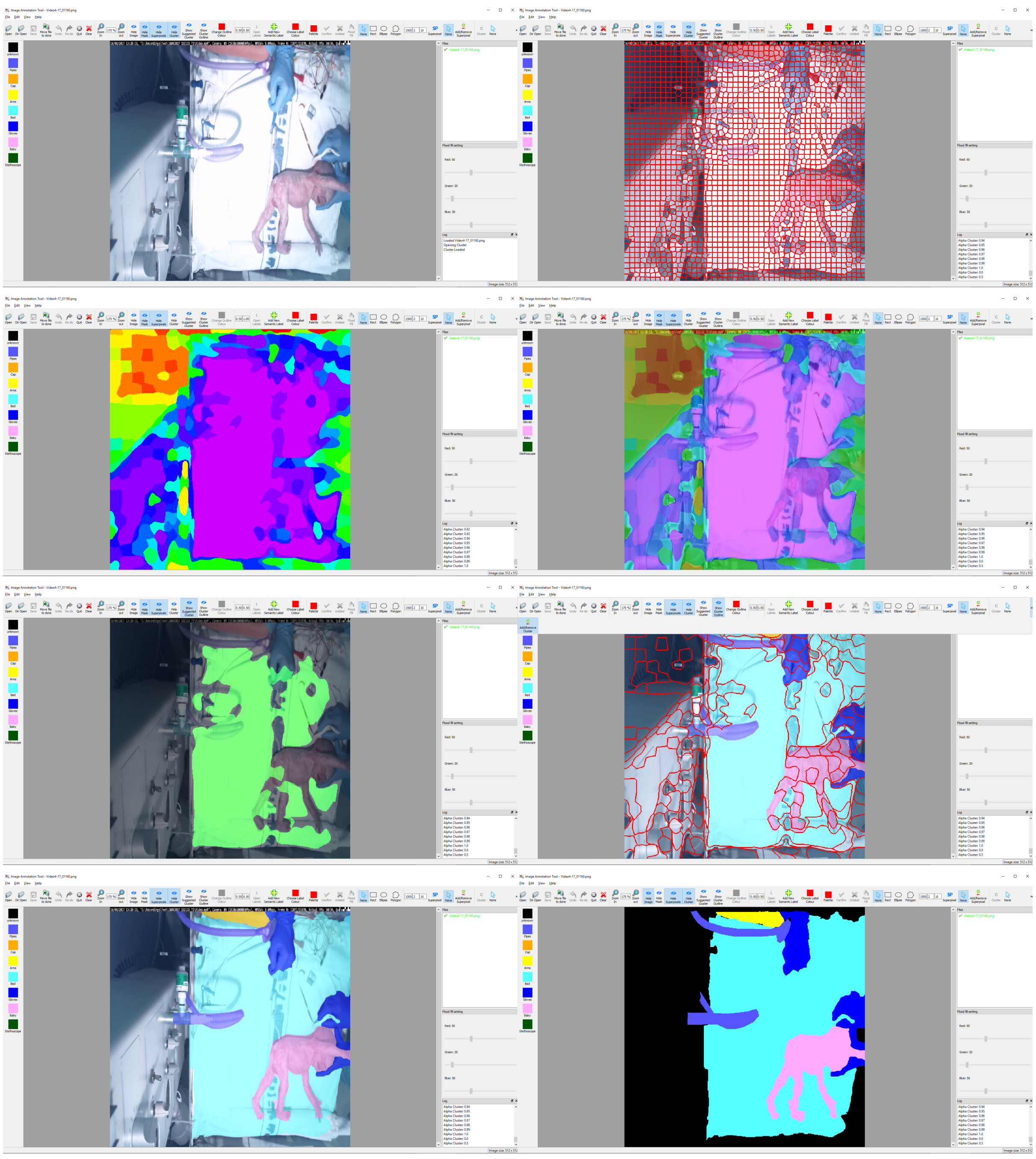


Figure 5.8: From left to right, top to bottom, the first image shows the updated user interface with the addition of new buttons and features. The second image shows the superpixel algorithm applied on a sample frame from the Newborn Resus dataset. The third and fourth images show an example of the clustered superpixels using the K-means algorithm. Image five shows the suggested largest cluster of superpixels. Image six shows the cluster outlines on a mask over the frame and annotation mask. The last row shows the final annotation mask both as an overlay and separately.

which takes only 15 seconds to run the algorithm, or a slight fine-tuning may be needed, which approximately takes one to two minutes. For simple objects such as regular shapes which can be easily followed with the polygon tool, these took approximately three minutes to annotate. Using the superpixel method of annotation took about 35 seconds, including the time to run the superpixel algorithm.

When comparing these annotation times, the superpixel method was much faster than the polygon tool. In these cases, superpixel annotations were created using the superpixel tool described in Chapter 5.2.1, and polygon annotations were created using the tool described in Chapter 5.2. Both of these tools produced colour segmentation maps. To calculate the time saved using the superpixel assisted annotations, the following four assumptions were compared:

1. All objects are complicated shapes and have similar scenes.
2. All objects are simple shapes and have similar scenes.
3. All objects are complicated shapes and have large variety of scenes.
4. All objects are simple shapes and have large variety of scenes.

The assumptions were applied to the gLitter dataset with 550 instances and the results can be seen in Table 5.1. In the comparison, the upper bounds are taken as an approximation. This illustrates how the two annotation methods differ at the extremes. For single, one-off annotations where assumption 2 or 4 was used, the polygon tool was much faster, saving a time of 7 minutes and 35 seconds. However, if the object was complex, the superpixel-assisted annotations were approximately 3 minutes faster. Furthermore, when the superpixel-assisted annotation was used on a dataset of 550 instances, such as the gLitter dataset, it was possible to save up to 119 hours, from 137.5 hours using the polygon tool, to 18.5 hours with the superpixel assistance. The only case where it may not be beneficial to use the superpixel tool is when there are a large variety of scenes, because this causes the tuning time to increase, when the objects to be labelled are simple.

In a real-world scenario, a standard size dataset would consists of a variety of all of these assumptions, as there would be a mix of similar and dissimilar scenes, with varying styles of objects. A combination of the polygon and superpixel assistance would be the recommended method for the IAT. This becomes more obvious when annotating full scene segmentation because there is no single superpixel parameter set that will correctly segment every object in the scene. Thus using them in conjunction would result in the best annotations. Small domain-specific datasets such as the Newborn Resus dataset use fixed views of a scene, so each instance in the dataset can use very similar parameters for the SLIC algorithm. The time-consuming part of annotating a database such as the Newborn Resus, is the number of classes in the scene and their interactions with each other. For example, the pipe from the ventilation machine is a similar colour to some of the gloves that are worn by the paediatric staff.

Annotation Type	Approximate Duration for Assumption			
	1	2	3	4
Polygon Single frame	15_m	3_m	15_m	3_m
Superpixel Assisted Single frame	12_m $(10_m + 2_m)$	10_m35_s $(10_m + 35_s)$	12_m $(10_m + 2_m)$	10_m35_s $(10_m + 35_s)$
Polygon gLitter Dataset	137.5_h	27.5_h	137.5_h	27.5_h
Superpixel Assisted gLitter Dataset	18.5_h	5.5_h	110_h	88.2_h
Improvement Single frame	-3_m	$+7_m35_s$	-3_m	$+7_m35s$
Improvement gLitter Dataset	-119_h	-22_h	-27.5_h	$+60.7_h$

Table 5.1: The upper bounds of each annotation have been taken to illustrate the improvements on a worst-case scenario. A standard dataset is generally a mix of these assumptions; large and small objects with some variations in scenes.

Chapter 6

Deep-Learned Superpixels

6.1 Deep Learning with Superpixels

This chapter presents a novel method of detecting superpixels by using deep learning to recognise them. From the review of the semantic segmentation literature in Chapter 3, it emerged that regardless of the domain, the current state-of-the-art method was still not pixel-perfect when it comes to the location of the boundaries between objects in a scene. This can be seen in Figure 6.1, where the street signs, the leg of the rider and the bottom part of the sheep are not correctly segmented by automatic segmentation methods. This type of error in detecting segmentation boundaries with deep learning image analysis has been addressed with some degree of success by including superpixels as a feature to the network (e.g. [8] and [42]).

Bhatti et al. [8] used superpixels in conjunction with optical flow and saliency maps to achieve an F1 score of 0.568 on the SegTrack v2 dataset [62], and Gu et al. [42] made use of superpixels in their foreground extraction pipeline, beating the state-of-the-art techniques at the time on four out of six videos for the SegTrack v2 dataset. It is becoming more common to use superpixels in conjunction with deep learning techniques for image analysis. However, these techniques use handcrafted algorithms to create the superpixels, which cannot be optimised in an end-to-end system. I hypothesise that it would be desirable for a superpixel technique to be deep-learned itself, thereby creating the opportunity to end-to-end learn segmentation with superpixels. Such a network could benefit from pre-training of the superpixel part on large, unsupervised datasets, and then fine-tuning on a domain-specific supervised dataset. The solution presented in this chapter makes it possible for the first time to *learn* the superpixels, which overcomes these shortcomings.

During a preliminary review of the literature discussing the use of superpixels to aid semantic segmentation, it was revealed that superpixels had never been directly learned

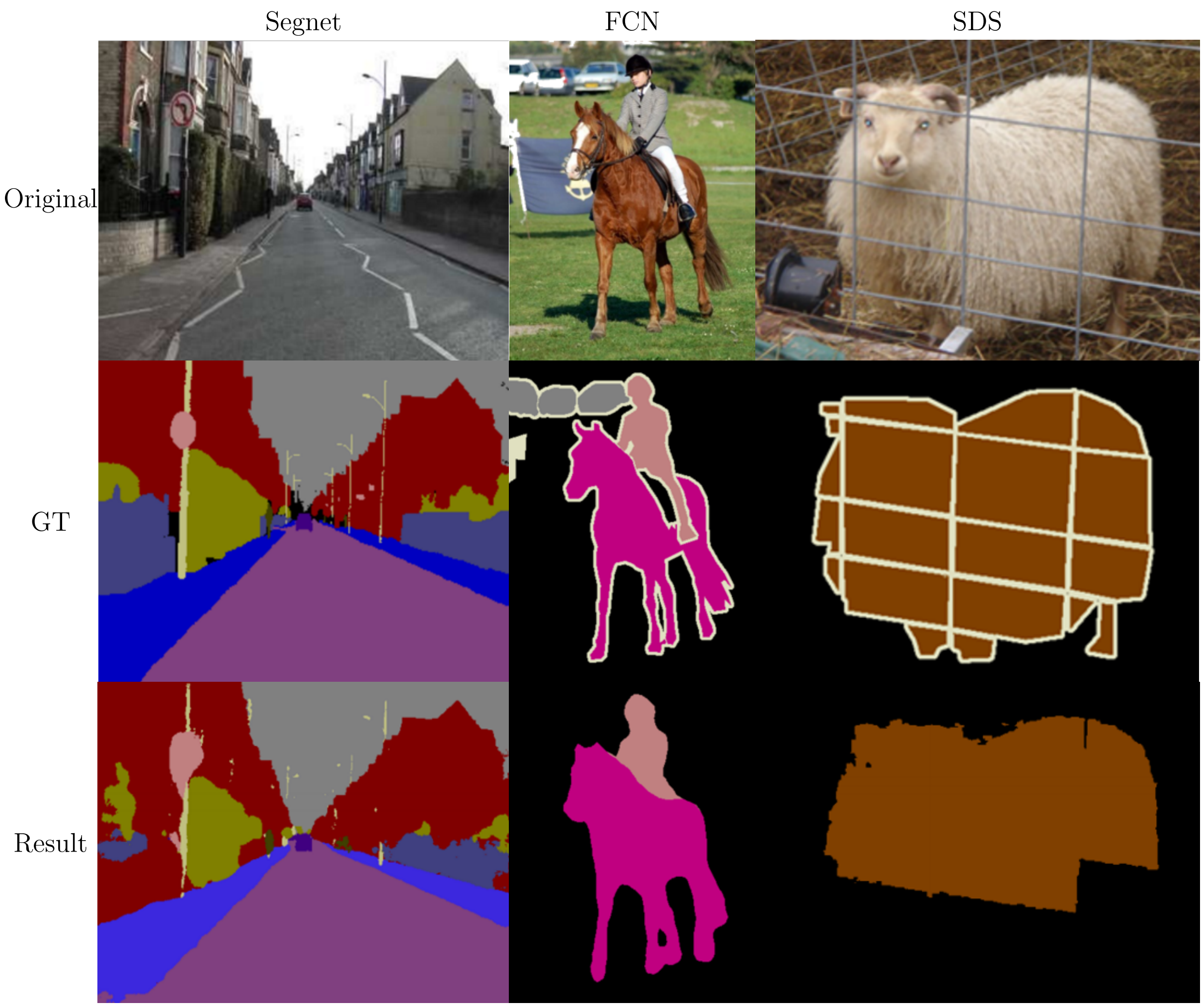


Figure 6.1: Three examples from three different networks, illustrating how the boundaries between objects are not yet perfect. From left to right, Segnet [3], FCN (8s) [89], Simultaneous detection and segmentation (SDS)[43]. From top to bottom are the input images, ground truths, and predictions. All three examples show poor boundary detection; firstly, the road sign in the Segnet clearly illustrates this problem; next, the riders leg is completely missed by the FCN; and finally SDS is missing large chunks of information in the lower sections between the fencing.

using a neural network and used as an over-segmentation method. With this novel idea, a system would be needed that allows the creation of superpixels to be incorporated into an end-to-end network. To develop this system, a new representation is needed so that a CNN architecture could understand the information created by the handcrafted superpixel algorithms. Having the ability to *learn* the superpixels allows the superpixel characteristics to be more than just defined by a handcrafted algorithm and instead morph to the dataset that it is being used on during end-to-end learning. But before one can fine-tune on a small domain-specific dataset, first a pre-trained superpixel network needs to be created on a large dataset, ideally without requiring manual supervision (i.e. manual annotation). In this thesis this was achieved by using a combination of settings for handcrafted algorithm techniques as the ground truths to train the CNN on.

To create a deep-learned superpixel predictor that could be used in a wide range of applications, the neural network structure used to encapsulate this needed to be complex enough to embed the features of the superpixels. Complex network architectures generally consist of large amounts of parameters. This in turn required a sufficiently large dataset to train the network. However, for many domain-specific segmentation tasks, such large datasets are often not available. A solution to this was to use unsupervised learning on a dataset with a lot of variation. This variation in the content of the dataset would improve the generalisation of a model.

The dataset chosen for training the networks was the DAVIS [83] dataset. This dataset was chosen because it has a large variation of scenes, ranging from sheep in a field to cars driving around a race track. This wide variety should have allowed the network to create a more generalised superpixel detection model.

Superpixel algorithms produce identity maps where each superpixel is given a unique ID and every pixel in an image is assigned to one and only one superpixel ID. Turning these identity maps into a pixel-to-pixel CNN target representation means that the superpixels must be transformed into a set of labels that can be learned with a CNN architecture. However, it was discovered that using this representation to solve deep learning superpixels was not as straightforward as first assumed.

The first methodology experimented with to learn the superpixels was use of the identity maps created by the handcrafted algorithm as the ground truths for the network. An example of this can be seen in Figure 6.2, where the SLIC algorithm was applied to a frame from the DAVIS dataset. When training a network to learn these identity maps using the MSE loss function, the loss values were in the ranges of 10^5, whilst seeming to converge extremely quickly, at less than five epochs. The results were completely unusable, as shown in Figure 6.3. This was because the position of each superpixel could differ greatly between images, as the identity maps were effectively stating where a particular superpixel class is located, 'class 93'. Each image would have a different representation of the 'class 93' superpixel, with different colour, texture, shape and location, which would

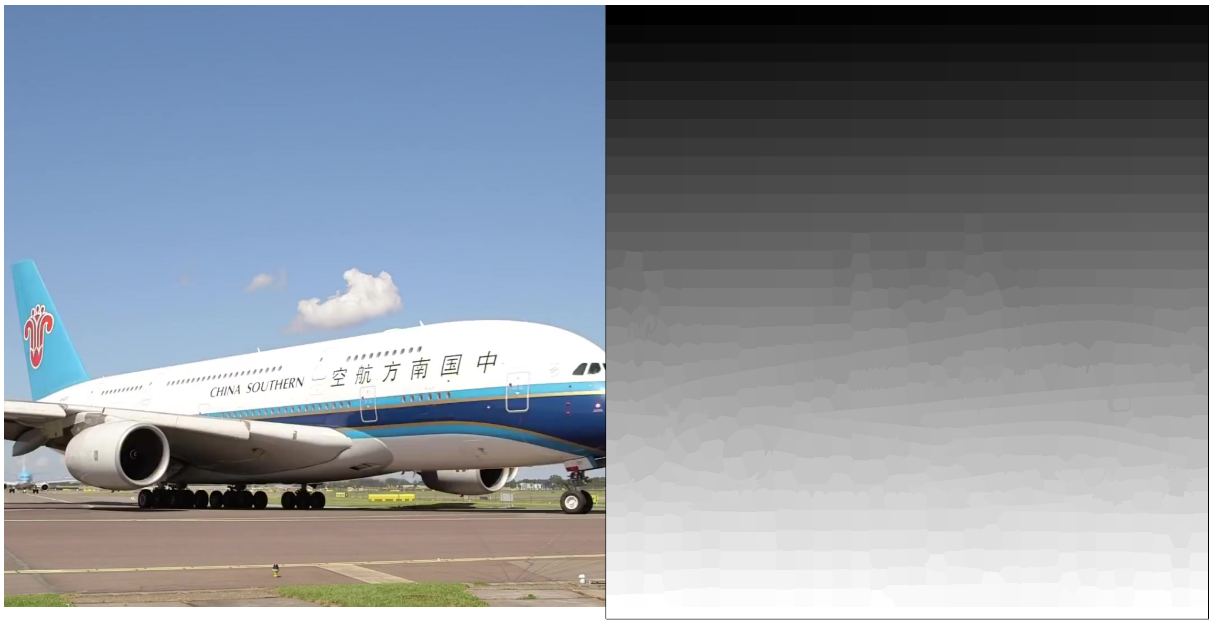


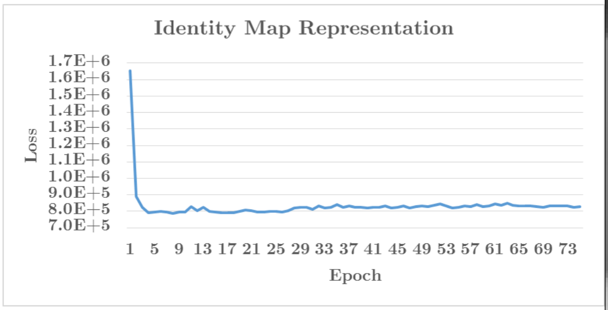
Figure 6.2: This figure shows the original image and the corresponding identity map, using the SLIC algorithm to create the identity map. The IDs have been normalised between 0 and 255 for visualisation.

confuse a network. Thus making and learning to predict identity maps was an ill-posed solution for this problem.

The chosen network was the highly successful hourglass network architecture introduced by Newell et al. [77], Figure 6.4. This network was chosen as it produced good results for human pose estimation using heat maps, and thus it was hypothesised that the network could be adapted for learning superpixels. The input to the network was an RGB image with dimensionality $(64 \times 64 \times 3)$ and a ground truth target of (64×64).

The next methodology tested was the one-vs-all methodology where a sliding window was used, and for each window the pixels in the superpixel were labelled as the positive class and all other pixels were labelled as the negative class, Figure 6.5. A window size of (64×64) was used and varying step sizes were experimented with. A major constraint to this method was the number of times this would need to be run over an image; to get full coverage, all pixels must be used as the target pixel. This scales linearly with the number of pixels and each operation uses an $(n \times n)$ mask, thus producing an $n-1 \times n-1$ oversampling rate (each pixel is used many times as input, rather than just once). If a small network was used, it would be much harder for a network to encapsulate the superpixel behaviour as there would be too much variation. Thus larger step sizes (strides) for the sliding window were experimented with to reduce the number of pixels visited. This proved to be an ill-posed solution as there was too much information lost when combining the negative classes together. This led to the network being unable to identify any superpixels.

The third methodology to learn the superpixels was to define the task where each



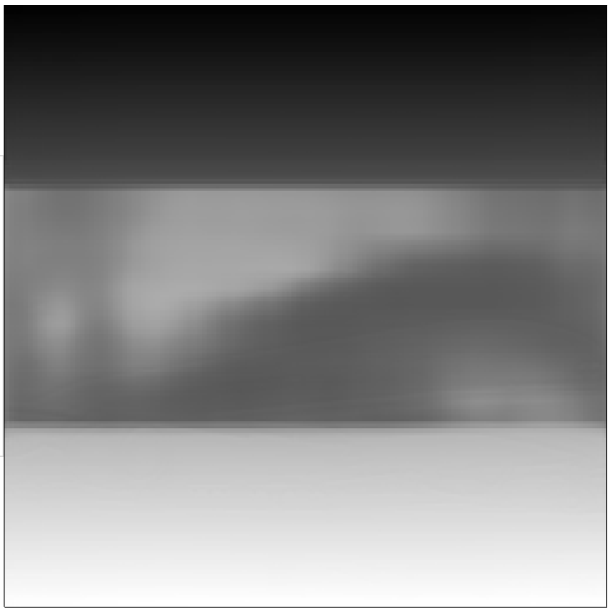


Figure 6.3: This figure shows the network's prediction of the images in Figure 6.2. It shows that the output had three distinct bands of predicted labels. This pattern was present in all predictions. Additionally, the central band did not resemble the input image.

superpixel is a unique class. This is effectively assigning each superpixel to its own output channel in a network. The CNN would be performing multi-class classification on the input image, rather than trying to recreate the identity map. This methodology followed the same ill-posed solution as the first methodology, but instead with multiple masks to predict rather than a single identity map. Additionally, this would require an output resolution of the same height and width of the input, with x number of channels where x is the number of superpixels. This methodology would need a lot of GPU memory to run due to the number of intermediate maps needed to be trained in the network. This in turn limits the number of superpixels allowed to be found in an image, which would lead to poor fitting of the superpixels. Finally, this would also mean that a model would need to be trained for each number of target superpixels and thus any software using this would also need to package all of the models. This methodology was not feasible or desirable with today's technology.

6.2 Multi-Channel Connected Graphs

To resolve the ambiguity of the superpixel unique identities, a new representation was required. The new representation had to be able to encapsulate the same information identity maps, and be easily transferable between the two representations. To this end, this thesis presents Multi-channel Connected Graphs (MCGs), a novel representation that allows the efficient and accurate training of deep-learned superpixel networks.

Inspired by Markov blankets, MCGs represent the relationship between neighbouring

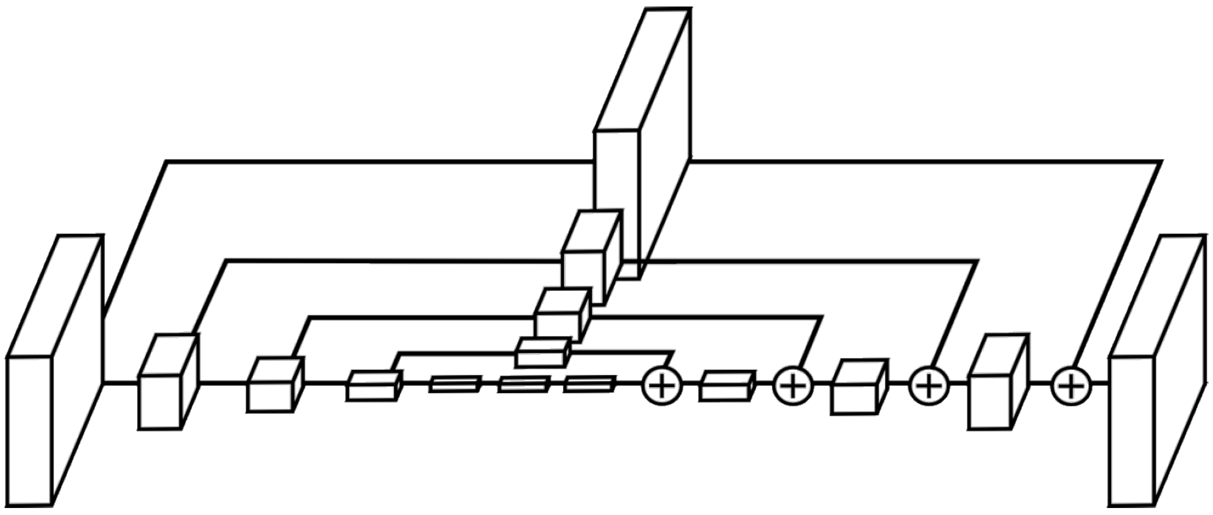


Figure 6.4: Illustration of the hourglass shape from Newell et al. [77] network. At each layer, there was a convolution applied, followed by max pooling to reduce the dimensionality. However, before the pooling layers were applied, a copy of the layer was kept for use in the up-sampling method. Once it reached the minimum dimensionality, the network started to grow again by use of up-sampling until the input dimensionality was achieved.

pixels. MCGs take the form of a two dimensional image with multiple channels. There are b channels, one for each direction in which the pixels' relationships are described. These descriptions state whether a pixel is in the same superpixel as its neighbour. The natural number for b would be four, but eight neighbours or more esoteric relations like triangular directions are also possible.

Using superpixels as an example use case, each pixel in the algorithm's output mask carries the information of whether it belongs to the same superpixels as any of its b neighbours. This means that every pixel in the two-dimensional data space carries connectivity information for the b relative directions. This representation allows the CNN to apply kernels to this and get meaningful results. Thus the problem has been converted to multi-channel connectivity maps, one indicating connectivity in each direction (up, right, down, left).

The transformation from a superpixel mask to a MCG representation is simple. First, there must be b channels of the same width and height as the superpixel mask. These channels must be initialised to zero. Next, the superpixel mask must be iterated over pixel by pixel, whilst checking if the current pixel is in the same superpixel as its neighbours. For each of the pixel's b neighbours, the respective masks are updated by setting the cell at the current pixel's coordinates. This will be set to a one if the two neighbours are in the same superpixel, otherwise this is left at zero. For the edge cases where it would be indexing out of the superpixel mask, there are two options. Edge cells would be set to zero if there is no evidence that they are connected to the same theoretical superpixel. They would be set to one if it is assumed that the superpixels expand past the frame.

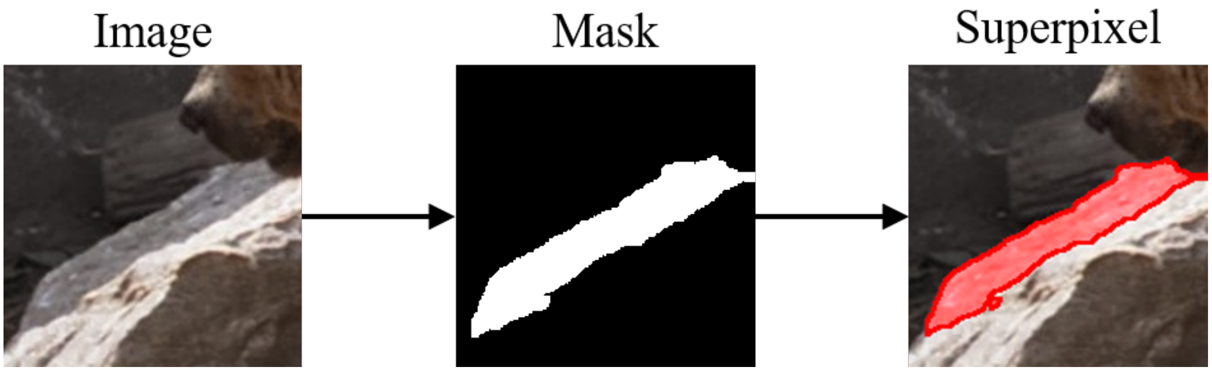


Figure 6.5: This figure shows the original image patch from a sample frame from the DAVIS dataset, the central corresponding positive superpixel, and the superpixel overlaid. The superpixel in this image is the shadow on the rock.

In this thesis, the first assumption was used. However, if working with video data, the second assumption may be more fitting.

In Figure 6.6 a small example is shown of how to represent a superpixel mask in the MCG format, but instead of using numbers as the unique identifiers for each superpixel, different colours and connecting lines are used.

The MCG methodology could easily be expanded with a larger neighbourhood, such as one containing eight neighbours, to include the diagonal neighbours, or even 24 to cover neighbours of neighbours. However, expanding the number of channels could cause issues with GPU memory space depending on the resolution of the network.

Using the MCG methodology to represent the superpixels in conjunction with the sliding window was no longer an ill-posed task because each pixel was now represented by the connectivity it had with its neighbours, rather than pixels being assigned to a specific superpixel 'class'. Therefore a network should have been able to learn this representation of superpixels. Firstly, the hourglass network needed to be adjusted to give four output channels, one for each MCG. Next, the SLIC algorithm was pre-applied to each image in the dataset and a sliding window of (64×64) was used on both the MCGs and the RGB images, Figure 6.7. The RGB patches were then fed into the network with the corresponding MCGs as ground truths. Initially a step size of 64 was used to move the sliding windows. When applying the model to a test image, firstly the full resolution MCGs were stitched together from the output of the model. Then the reverse-creation process could be performed to make a superpixel identity map. The identity maps produced were unusable as there were holes between the superpixels, meaning during the reconstruction process, individual superpixels bled into other superpixels and had unusable maps. This can be seen in Figure 6.8.

To convert MCGs back to superpixel ID maps, the output must first be initialised to an array of the dimensions of the input MCGs, with the array's values set to -1. Then first cell is then set to the 0 ID. The output is iterated over, and for every cell in the

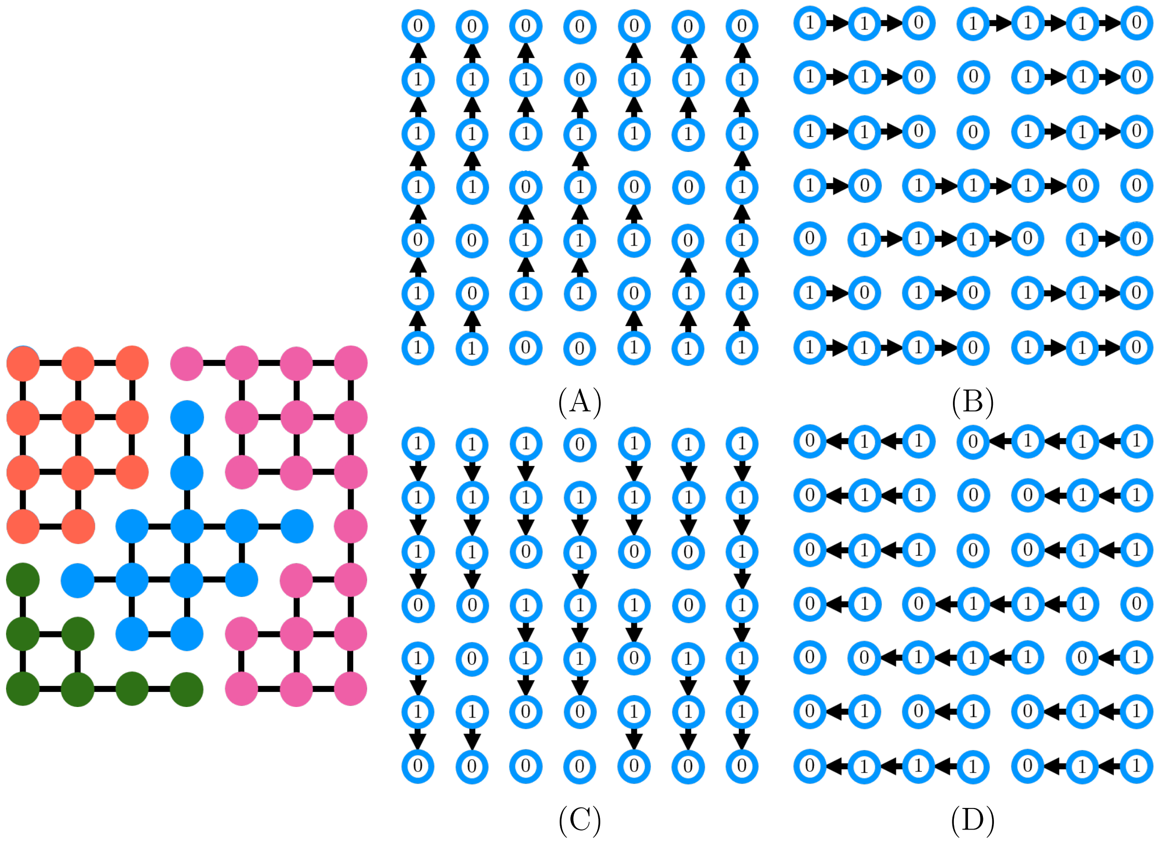


Figure 6.6: The MCG representation. Left shows an example of a handcrafted superpixel mask, showing the connections for each of the pixels and their four immediate neighbours. Next to this are the four MCGs (A:up, B:right, C:down, D:left) for this example. Each individual graph is made up of zeros and ones. Each pixel in the same superpixel as its neighbour, in regards to the respective direction, is set to one or zero.

output, the corresponding MCG map's cells are checked in the order: right, down, up, left. The neighbouring cells are then updated with the same ID as the current output cell or the next free ID. If a cell has already been updated but there is a conflict, that is, the IDs should match but do not, then the cell with the most confidence from its priors is taken. Python code for this can be found in Appendix A.

One possibility for the bleeding problem was that the gaps in superpixel boundaries were caused by low probabilities for a boundary pixel. To test this hypothesis, the values for the threshold that control the binarization of the MCGs were experimented with. The MCG masks returned real numbers ($\mathbb{R}$), between 0.0 and 1.0. To binarize the MGCs, a threshold was needed for rounding the pixels to the nearest whole number. Several thresholds were automatically tested between 0.0 and 1.0, along with the Otsu method [79]. The Otsu method works by minimising intra-class intensity variance, or equivalently, by maximising inter-class variance. Results showed that no value for the threshold would create binary MCGs that could be reformed into superpixels. Upon manual inspection of the MCGs it could be clearly seen that the network had abstracted away from superpixel, to what looked like super-regions, Figure 6.9. This abstraction to super-regions is likely

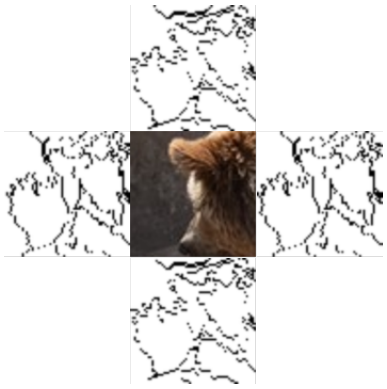


Figure 6.7: This figure is an example of the MCG representation of the SLIC superpixels for a sliding window. The frame used is from the DAVIS dataset.

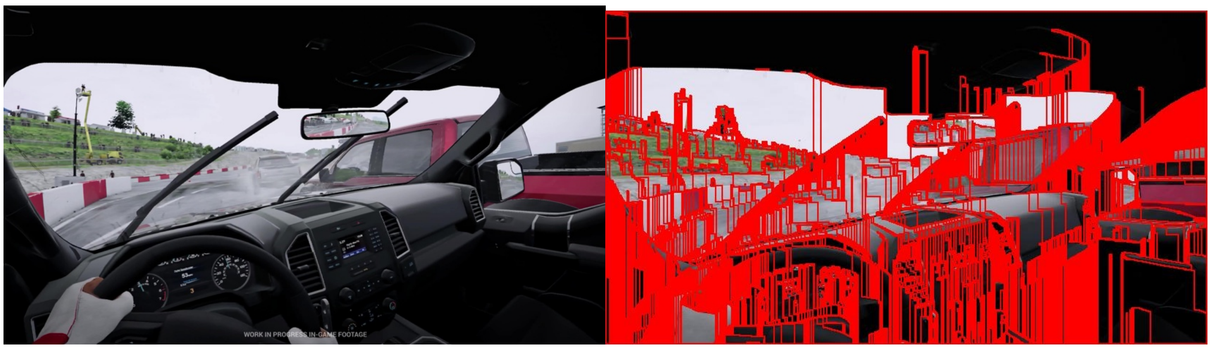


Figure 6.8: This figure shows the reconstruction of the deep-learned superpixels using the sliding window technique.

because the network has learnt to remove the boundaries between similar superpixels with regards to colour and texture. A faint bounding box where each of the sliding windows were was also noted. This was caused by the network not being able to see what is outside of a window, so it assumes that it is the edge of the image. This can be negated by overlapping the sliding windows slightly and averaging the results. Several strides were experimented with, but all stride lengths tested did not eliminate the boxes. The stride sizes tested were 0.25× box width, 0.5× box width, and 10 pixels less than box width.

To completely remove the need for sliding windows and to solve the stitching issue, the resolution of the input to the network was increased to $(1024 \times 1024 \times 3)$. This allowed the input of an RGB image at a higher resolution to reduce information loss when down-scaling the images and superpixel masks. An up-scaling layer before the output of the network was added to bring the resolution back to $(1024 \times 1024 \times x)$, where $x = 4$ for the superpixel MCG.

The adapted hourglass network consists of three blocks. First is the pre-processing block, which feeds into the hourglass block, which finally feeds into the post-processing block. The adapted hourglass network uses the same residual module in the original architecture by Newell et al. [77]. This residual module consists of three convolutional

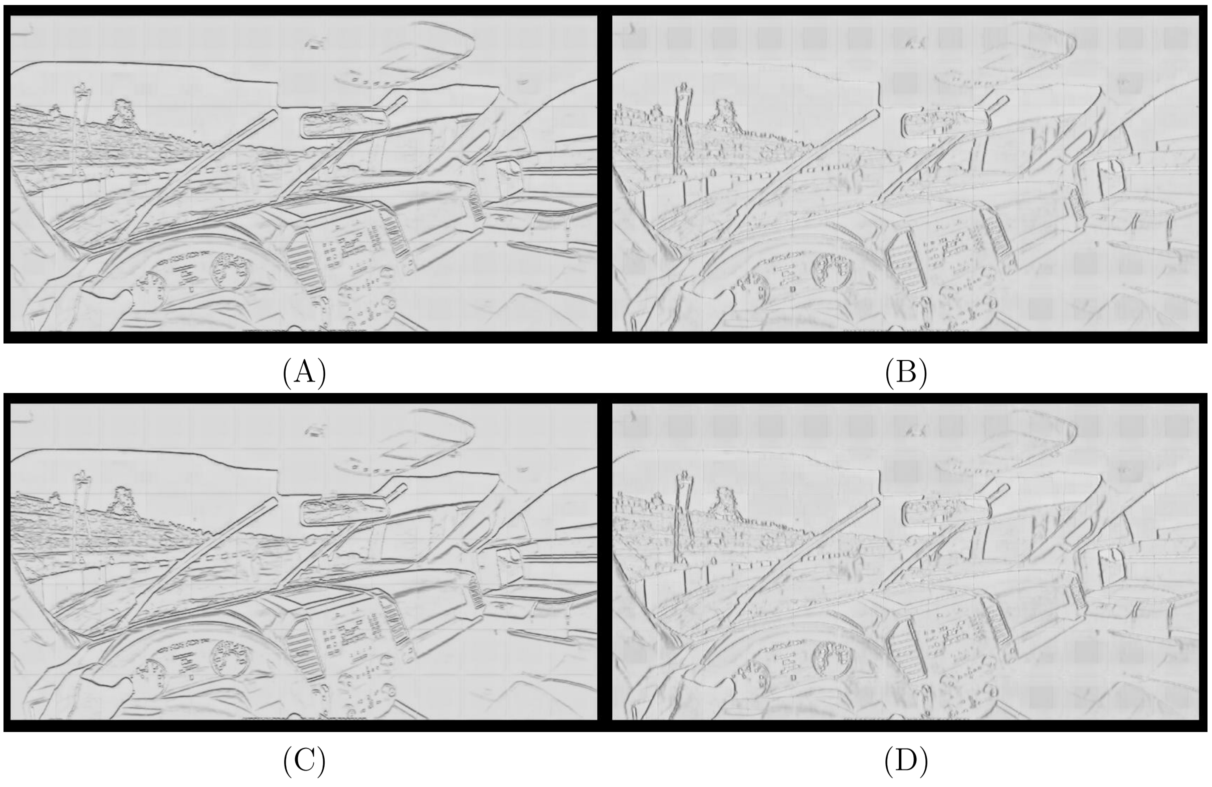


Figure 6.9: The stitched-together MCG applied to a frame from the DAVIS dataset for each direction: up, right, down, and left, represented by A, B, C, and D, respectively. In these images, white represents one in the MCG, where the pixels are in the same superpixels. Black represents zero, where there is a boundary between superpixels.

layers, with all three layers using the RelU activation function. The first convolutional layer uses a (1×1) kernel and 128 features. The second convolutional layer uses a (3×3) kernel and 128 features. The final convolution uses a (1×1) kernel with 256 features. The output of the third convolutional layer is fed into a max pool layer with stride two.

The pre-processing block takes in an RGB image of $(1024 \times 1024 \times 3)$ and first feeds it through a convolutional layer with a (7×7) kernel, stride step size of two, and 64 features, resulting in an output tensor size of $(512 \times 512 \times 64)$. This output is then fed through a residual module followed by a max pool layer with stride two. It is finally fed through a further two residual modules. The output of the pre-processing block is then fed into the hourglass module with an input resolution of $(256 \times 256 \times 256)$. The hourglass has a depth of four, meaning that there are four residual modules in the decoder and encoder parts of the hourglass module. The lowest resolution that the hourglass reduces down to is $(16 \times 16 \times 256)$. The encoder side of the hourglass uses max pooling to reduce the resolution, and the decoder uses nearest neighbour up-sampling. The post-processing block consists of two more up-sampling layers and a fully connected dense layer to predict the superpixels. The loss function used to train this model is the mean squared error loss function in conjunction with RMSprop optimiser. The learning rate used to train the model is $2.5e^{-4}$ and is reduced every 30 epochs by a factor of 10, that is, the learning rate is multiplied by 0.1.

To deal with the ambiguity of exact superpixel placement and size, the target number of superpixels was varied. The network was trained with multiple superpixel ground truth masks per input image. In principle, this could also be done using a combination of superpixel techniques, but in this work a single handcrafted algorithm was used, SLIC.

For each of the frames that are used from the DAVIS dataset, the SLIC algorithm ran multiple times with varying numbers of target superpixels, ranging from 500 to 5000 superpixels in steps of 500. The other SLIC parameters were set to $\sigma = 1$, $compactness = 30$. In total, the new dataset consisted of 4000 instances with a three to one ratio for training and validation.

This created an unsupervised pre-trained network that could be used to predict superpixels in the MCG format. When inspecting the output from this network, these too had abstracted away from superpixels to super-regions. However, like the previous method, the outputs when binarized did not reform to make superpixels/super-regions when using the inverse method that creates MCGs. When the results from this network were manually inspected, the holes in the boundaries of regions were too ambiguous in regards to both the size of the holes and also where they appeared. The predictions no longer followed the ground truth data. Therefore it was decided to keep the predictions as the deep-learned superpixel MCGs. This was because the deep-learned superpixels were always intended to be part of a larger end-to-end network structure in predicting semantic segmentation. The predicted boundaries in the MCGs follow the structures in the images. This was a promising sign that they would be useful in aiding the boundary detection in semantic segmentation. This is shown in Chapter 6.4.2, where these deep-learned superpixels aided in supervised semantic segmentation, and examples can be seen in Tables 6.2.

6.3 Ground Truth Degradation

In this section, the effects that the quality and quantity of ground truth data has on the accuracy of a network when using a small domain-specific dataset are discussed.

6.3.1 Methodology

With large datasets it is very common to use non-perfect segmentation masks, because these types of masks do not take as much time to hand-annotate than pixel-perfect, or near pixel-perfect, annotations. Due to the large number of these sub-optimal annotations, deep neural networks are able to generalise and learn the correct segmentations for the objects or classes. When using a small or tiny dataset, there are not enough instances for the network to generalise and not overfit to the training data.

In this section, the near pixel-perfect hand annotations of the gLitter dataset were used to train a baseline model. To create the less accurate masks, the original hand annotations were first converted to polygons. The polygons consisted of a list of points that defined the perimeter of the mask. To reduce the accuracy of the mask, a subset of points was created by stepping through the list of points. The greater the step size, the smaller the number of points in the new polygon mask were, and thus a less accurate mask was created. Figure 6.10 shows example ground truth segmentation masks for the four datasets, including the baseline. The four datasets are dubbed '0', '2', '4', and '6', where the number represents the scale of degradation applied. For instance, '0' had no degradation applied, '2' uses $2n$ step size for the degradation algorithm, and so on. n is a tunable parameter which is optimised for a given dataset; the larger n is, the more points are skipped per step. A large n value is used when the boundaries of the perimeter are noisy.

In addition to degradation of the masks, the number of instances used to train the network were varied. The full training set for the gLitter dataset consisted of 423 instances. To determine the effect of the number of instances on the training of the network, five different percentages of training data were used: 100%, 50%, 35%, 25%, and 10%, (423, 212, 106, 74, 43 instances respectively).

6.3.2 Experimental Results

Figure 6.11.A reflects the general consensus that the more training data that can be provided to a network, the better it performs for all scales of ground truth annotation degradation. Additionally in Figure 6.11.A it can be seen that the larger the scale of degradation, the lower the average accuracy is. For a small dataset, less than 500 training instances, it can be concluded that there is a trend in which more data would result in better performance of a network, even with high levels of degradation. However, it can be seen that to achieve good performance with high levels of degradation, much larger amounts of data would be required. Figure 6.11.B shows simultaneously the effects that the level of degradation and the number of training instances have upon average accuracy. The upper left of the surface in Figure 6.11.B had the highest accuracy with good annotations and the most amount of data. Then at the other extreme is the surface showing poor accuracy for smaller amounts of training data and poor annotations.

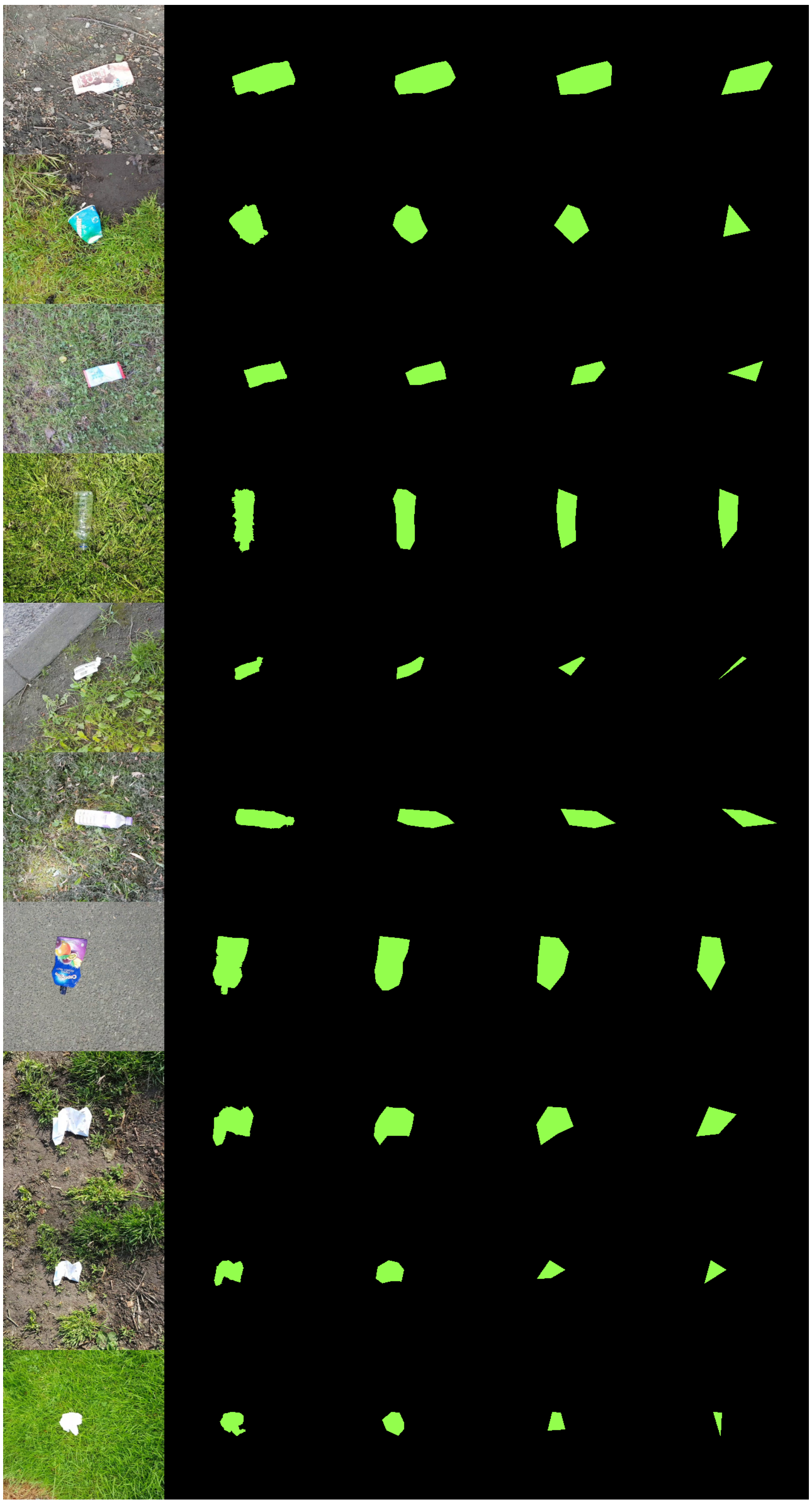


Figure 6.10: From left to right, this figure shows: the RGB images, hand-annotated ground truths, two-step degradation, four-step degradation, and six-step degradation.

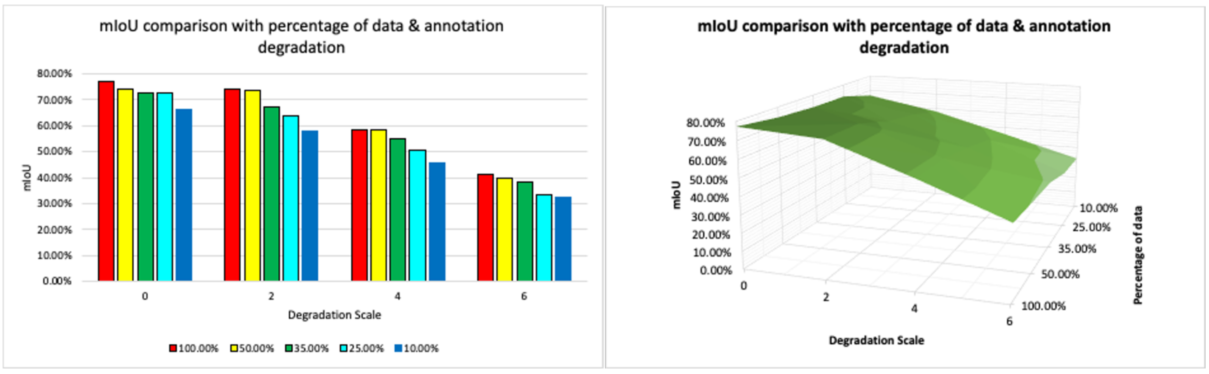


Figure 6.11: Effect of number of training samples and the accuracy of ground truth annotations have on the average accuracy of a network. Figure B illustrates Figure A as a surface, where it can be easily seen that there is an almost linear trend between training data, lower quality annotations, and the average accuracy.

6.4 Semantic Segmentation with Deep Superpixels

6.4.1 Methodology

Semantic segmentation is a well-researched field but often uses, and indeed requires, large datasets for training. When using a small domain-specific dataset, the performance decreases drastically, as shown in Chapter 7. In this section the gLitter dataset is used, achieving an accuracy of 81.6% using an hourglass network with RGB plus deep-learned superpixels represented by a MCG as input to the network. The gLitter dataset is relatively simple, consisting of 300 image samples of litter of various types taken on multiple different backgrounds. This variation means that there are large variations in colour, size and shape between instances. For this dataset, a combined class dubbed "super-class" was used to identification all litter types.

When using a small dataset with a super-class that contains large variations between sub-classes, the identification of the boundaries for an object would often be fuzzy and incorrect. The addition of superpixels partially solved this issue because the inherent properties of superpixels enforce the superpixel structure to follow colour and texture boundaries. In Table 6.2 a reduction in fuzzy boundaries between predictions in column A (RGB pipeline) and column E (RGB plus deep-learned superpixels pipeline) can be seen. However, when comparing column A to column C (RGB plus SLIC superpixels pipeline) the improvement is less noticeable than that of column E.

To first test that deep-learned superpixels using the MCG representation can work for semantic segmentation, the gLitter dataset was used. This dataset had been fully annotated using the Image Annotation Tool (IAT) with the superpixel method. This dataset is more simplistic than the Newborn Resus dataset in regards to classes and annotation time because it only consists of a single super-class, which is litter. This dataset took approximately two to three minutes per frame to annotate, giving a maximum total annotation time of 27.5 hours for a full manual annotation.

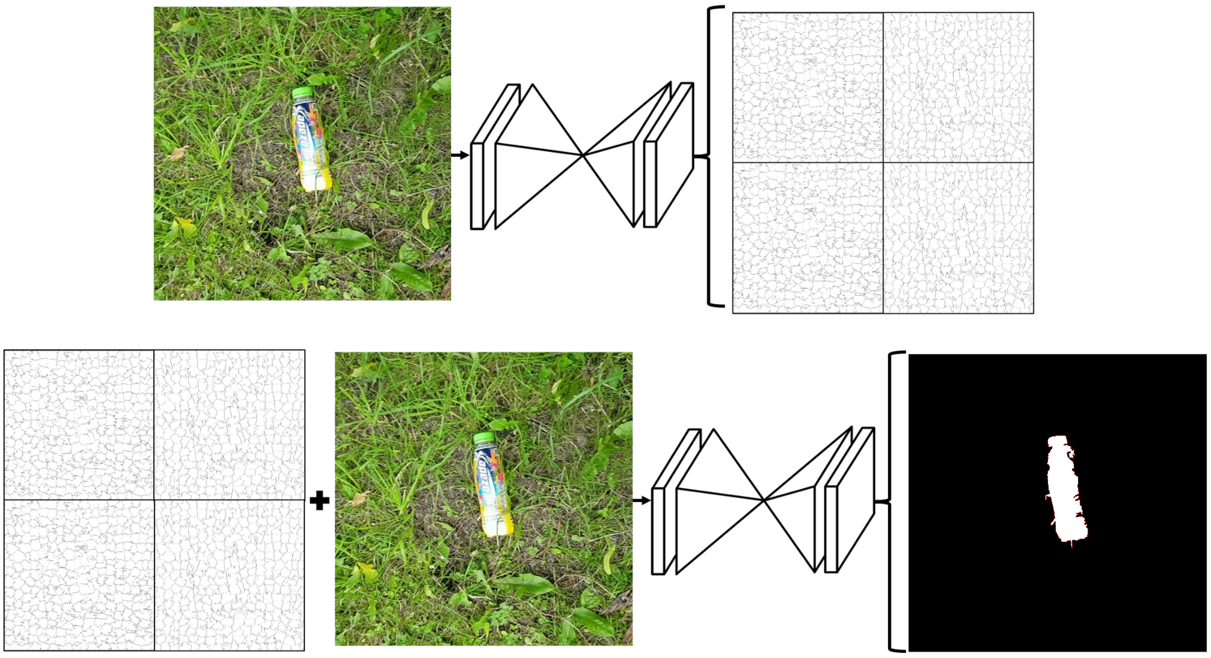


Figure 6.12: This figure shows the network structure for the fifth pipeline. It takes in an RGB image and first passes it through the superpixel generator network and predicts the four channel MCG representation of the superpixels. Then the original RGB image plus the deep-learned superpixels represented via MCG are passed through the second network to predict the super-class, litter.

To test the hypothesis that the addition of deep-learned superpixels would be beneficial to semantic segmentation, an experiment comprising of four pipelines plus a baseline was performed. Each pipeline uses the adapted hourglass network to learn the litter super-class. The network structure now consisted of two separate hourglass networks, with the first being pre-trained on the DAVIS dataset for deep-learned superpixels, and the second network being trained from scratch for semantic segmentation. This structure can be seen in Figure 6.12. The second network used the inputs described by a pipeline and predicted the ground truth semantic segmentation masks. To compare the pipelines, the accuracy of the network was used. The accuracy is calculated by taking the intersection over union of the prediction and ground truth masks. In addition to accuracy, entropy of the prediction was also used. Entropy describes the coherence of the results. In this scenario, entropy can be used to explain how noisy a network's predictions are. The more noisy a prediction is, the higher the entropy score becomes, thus a lower entropy score is desired. In addition to a low entropy score, a prediction value closer to one demonstrates a more confident prediction for that class. Entropy is calculated by first finding the normalised histograms of the image, p, then summing the product of the p and the log base two of p, as seen in Equation 6.1.

$$entropy = -sum(p. * log2(p)) \tag{6.1}$$

The five segmentation pipelines were:

(A) RGB - Going directly from an input of RGB to perform semantic segmentation (Baseline).

(B) MCG(SLIC) - Handcrafted superpixel pipeline represented by MCGs to perform semantic segmentation.

(C) MCG(SLIC)+RGB - Combining the RGB and MCG handcrafted superpixels together to perform semantic segmentation.

(D) MCG(DL) - Deep-learned superpixels represented by MCGs to perform semantic segmentation.

(E) MCG(DL)+RGB - Combining the RGB and MCG deep-learned superpixels together to perform semantic segmentation.

In the following chapter (Chapter 7), the pipelines are re-evaluated on both the gLitter and Newborn Resus datasets for use in an end-to-end pipeline with a reduced input size of (512×512) to reduce the amount of video memory needed for training the network. The Newborn Resus dataset has been used as a multi-class dataset for semantic segmentation. This dataset consists of 23 classes including an unknown class, and has manually annotated ground truth semantic segmentation masks for the 23 classes using the IAT. There were a total of 50 instances of hand annotation, with these instances split into three sets: training (20 instances), validation (10 instances), testing (20 instances).

6.4.2 Experimental Results

The input frames from the gLitter dataset were taken with mobile phones that have 1080p cameras. This was too large a size to be used with a deep learning network, as it would require down-sampling to a reasonable size before learning could begin. To reduce the number of layers in the network and speed up processing time, the frames were reshaped prior to training from 1080p to $(1024 \times 1024 \times 3)$ RGB image. The images were first cropped from 1080p to a square image of $(1080 \times 1080 \times 3)$, this could be safely done because the object was guaranteed to always be centred in the frame due to how the dataset was created. Next the images were resized down to the smaller 1024 square, this allowed as much information as possible to be kept for the network to learn from.

ID	Pipeline (1024 × 1024)	Superpixel Type	Average	
			∩/∪	Entropy
A	RGB	N/A	79.4%	3.0478
B	MCG	SLIC	60.2%	1.8837
C	MCG+RGB	SLIC	81.1%	3.6020
D	MCG	DL	75.0%	1.3783
E	MCG+RGB	DL	**81.6%**	**1.2052**

Table 6.1: This table shows the average performance of the five pipelines. Here it can be seen that MCG alone performed the worst out of all the pipelines. MCG plus RGB for both methods performed better than RGB alone. Additionally, it can be seen that the deep-learned MCG versus the handcrafted method performed on average 15% better. In addition, using deep-learned superpixels results in an entropy 3-times lower than using hand-crafted superpixels, which is the second-most accurate system.

Baseline (RGB): For the baseline, the RGB representation alone was used. This proved to be very unstable before the input size was increased for both the input and output from 256 to 1024. As previously mentioned, RGB alone produces a lot of extra noise. In some cases RGB alone can perform admirably, however it often produces fuzzy object boundaries and extra misclassification due to relying on colour alone.

Handcrafted Superpixels (MGC[SLIC]): The second pipeline compared the performance against using only an MCG representation of a handcrafted superpixels algorithm. On its own, this method did not generally yield good results, and performed on average 19.2% worse than the baseline. This drop in performance was most likely because there was a reduced amount of information available to the network. There was only shape information available, all textural and colour information were not present in the input. Additionally, litter has a large variation in shape and number of superpixels per object. However, it did learn that usually the litter would be in the centre of the frame and would have an irregular shape compared to the rest of the image.

Handcrafted Superpixels plus colour (RBG+MCG[SLIC]): The third pipeline combined the use of the MCG obtained from a handcrafted superpixel algorithm together with RGB data. As can be seem in Table 6.1, on average this performed better than RGB alone with an improvement of 1.7%, bringing the average accuracy to 81.1%. However, like the RGB method there still remained a lot of noise in the image, but it did clean up the edge between the object and the background.

Deep-learned Superpixels (MCGDL): The fourth pipeline was designed to test how well the deep-learned superpixels represented via MCG would fare on its own. The mean intersection over union (mIoU) performance for this method was 14.8% better (in absolute terms) than the handcrafted method (pipeline B), even obtaining the highest score for six of the validation images. This showed that the deep-learned superpixels must represent the target image better than the handcrafted superpixel algorithm, with the same limitation of loss of information. In terms of accuracy, this method was only slightly worse than RGB alone, with a drop of 4.4% with regards to mIoU.

Deep-learned Superpixels plus colour (RGB+MCG[DL]): The fifth and final pipeline was the combination of the deep-learned superpixels and RGB. In terms of accuracy, it had the highest performance of all of the pipelines with a 2.2% increase over the baseline, and generally had much cleaner boundaries with minimal noise. Additionally, when examining the performance of the pipelines, the scores on the tests sets per sample were compared and both pipelines using RBG + MCG superpixel had the most images with the highest accuracies.

Tables 6.2, 6.3, and 6.4 show the difference between pipeline E (RGB + Deep-Learned Superpixels) and the baseline. Table 6.2 shows the five instances from the validation set where pipeline E has the greatest improvement over the baseline. In these images, it can be seen that there was much less noise, and generally there was good improvement over the baseline. Additionally, pipeline D (Deep-Learned Superpixels) performed admirably compared to the other pipelines. Take for instance, number 31; this pipeline does not get confused with the lighter patch of flooring like other pipelines using RGB.

Table 6.3 shows the five highest accuracies for pipeline E. These instances were generally white pieces of litter on a grassy background. This makes them easily distinguishable. Even the pipelines that do not use RGB had good performance on these instances. The final table, Table 6.4, shows the five lowest accuracies for pipeline E. These instances were very challenging to segment, as they contained a lot of obstruction, contained noisy backgrounds, or were very small.

6.5 Conclusion

To conclude, the pipelines using superpixel represented by MCGs plus RGB, pipelines C and E, as their input had a higher Jaccard index. However, performing a Student's t-test on the results of pipelines C and E reveals that they were not significantly different. The student's t-test is a statistical test to test a hypothesis under a null hypothesis. A null hypothesis is the opposite of the working hypothesis. The students t-test is calculated using Equation 6.2 and describes if two data collections, *i.e.* two sets of results from different methods, are significantly different. $\bar{X}$ is the sample mean of the data collection. n is the number of data points in the collection. s is the data collections standard deviation.

When the results of the two pipelines are inspected, it can be seen that pipeline E, which uses deep-learned superpixels, has much crisper predictions. Comparing the average entropy values for both of these pipelines shows that pipeline E's entropy is much lower, meaning that the predictions were more confident. Therefore, the use of deep-learned superpixels is preferred over handcrafted superpixels.

$$ttest = \frac{\bar{X}_1 - \bar{X}_2}{s_p \sqrt{\frac{2}{n}}} \tag{6.2}$$

$$s_p = \sqrt{\frac{s_{X_1}^2 + s_{X_2}^2}{2}} \tag{6.3}$$

One problem with how the deep-learned superpixels were used in this chapter was that they were not learned end-to-end, meaning that the loss from the semantic segmentation and the domain-specific intricacies were not passed through backpropagation. This in turn meant that the deep-learned superpixels could not be automatically fine-tuned to the domain-specific data.

Therefore, the next chapter will focus on the creation of an end-to-end network that goes from RGB to semantic segmentation, and extends to multi-class semantic segmentation.

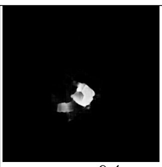
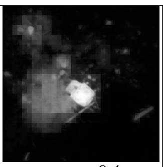
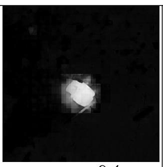
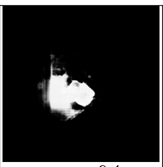
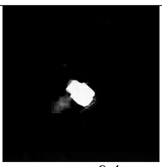
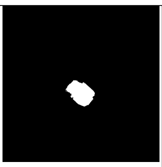
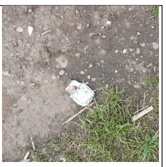
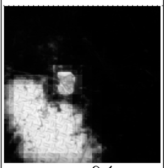
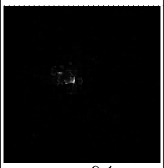
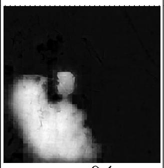
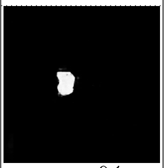
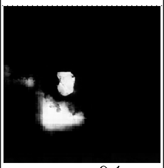
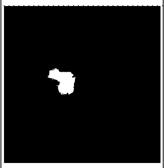
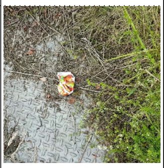
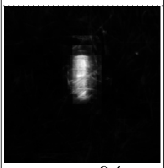
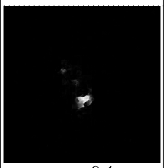
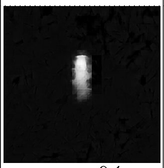
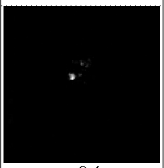
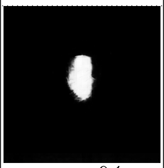
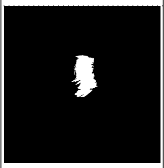
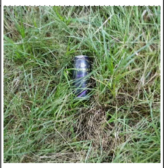
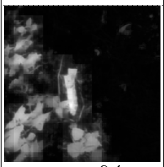
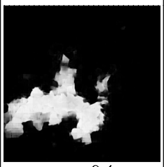
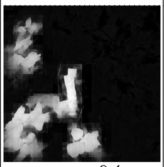
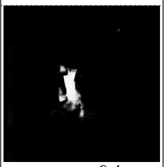
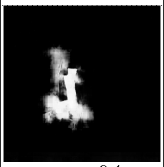
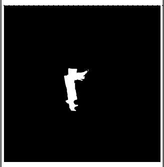
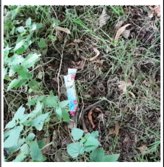
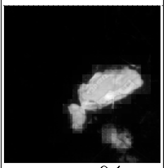
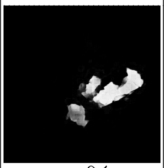
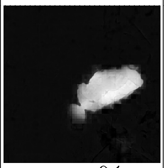
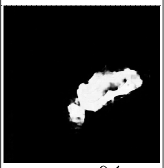
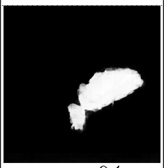
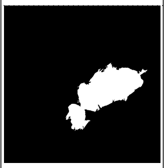
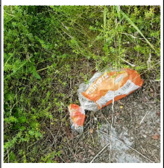
ID	A	B	C	D	E	GT	Input
33							
∩/∪	59.9%	61.2%	85.0%	35.0%	95.2%		
31							
∩/∪	6.2%	1.6%	5.8%	73.6%	23.7%		
3							
∩/∪	57.1%	0.1%	51.8%	0%	66.5%		
18							
∩/∪	14.9%	9.4%	10.4%	51.4%	24.1%		
32							
∩/∪	81.8%	48%	78.3%	78.6%	90.1%		

Table 6.2: This table shows the examples for which pipeline E displayed the largest improvements compared to the baseline. Pipelines using MCGs (all but pipeline E) exhibit much less noise and uncertainty (as measured by Entropy).

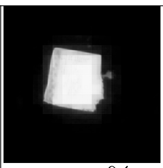
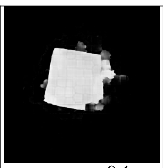
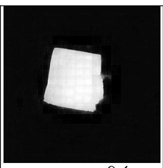
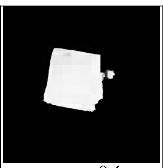
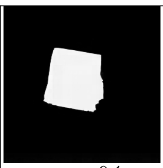
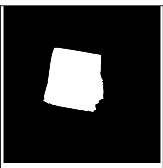
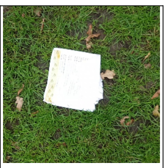
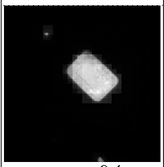
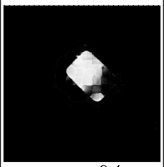
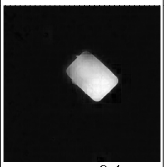
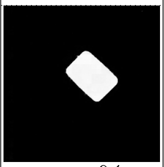
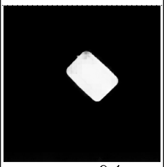
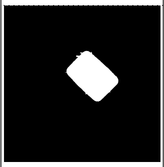
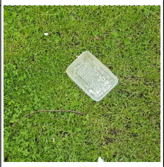
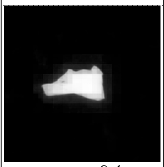
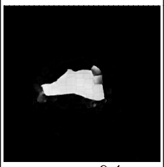
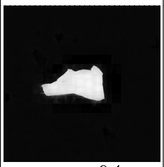
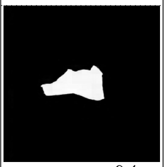
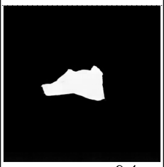
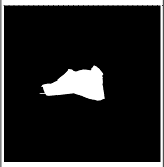
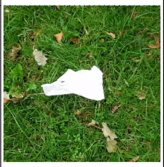
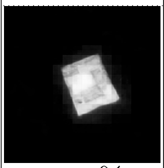
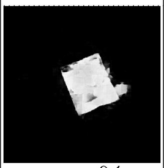
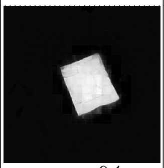
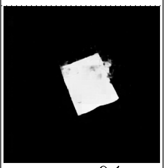
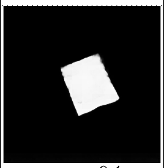
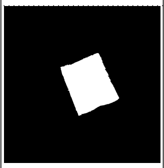
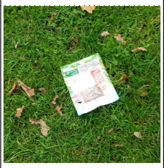
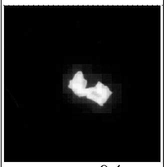
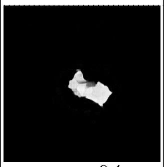
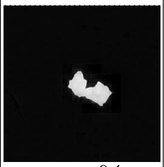
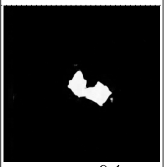
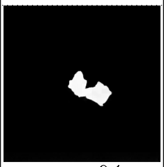
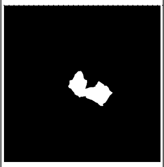
ID	A	B	C	D	E	GT	Input
20							
$\cap/\cup$	98.55%	88.62%	98.62%	96.4%	98.9%		
29							
$\cap/\cup$	98.6%	88.6%	98.6%	96.4%	98.9%		
13							
$\cap/\cup$	96.9%	91.4%	97.7%	97.751%	97.753%		
23							
$\cap/\cup$	95.0%	89.3%	97.4%	95.4%	97.2%		
30							
$\cap/\cup$	96.1%	93.5%	97.0%	96.1%	96.9%		

Table 6.3: This table shows the highest performing instances from pipeline E. All pipelines performed well on these instances because there was a clear distinction between background (green grass), and the object (white litter). Still, pipeline E could be seen to make crisper predictions than pipeline C.

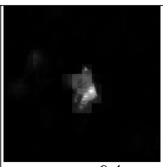
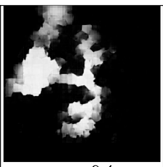
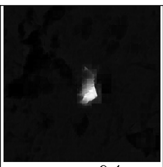
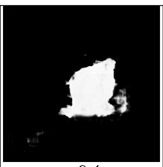
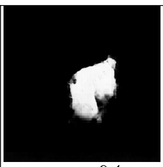
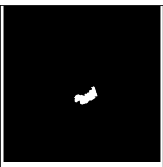
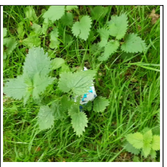
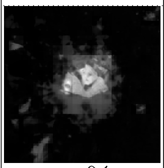
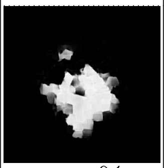
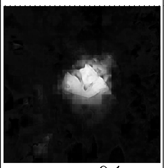
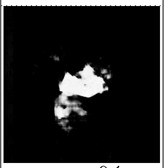
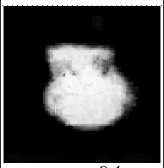
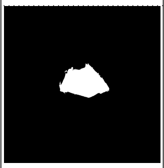
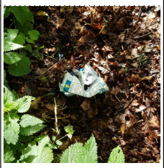
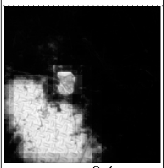
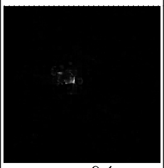
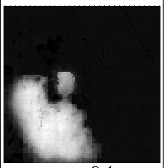
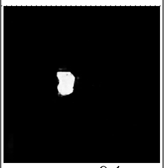
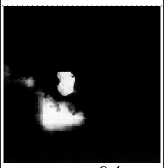
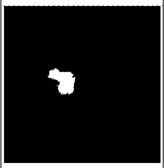
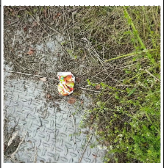
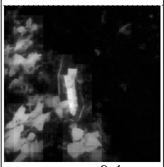
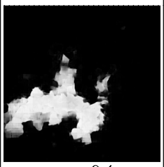
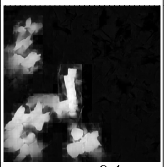
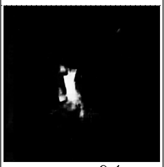
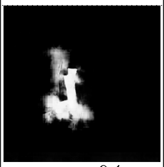
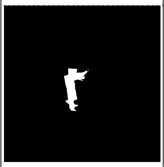
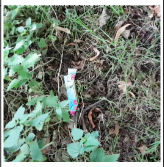
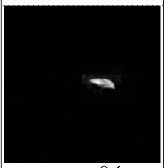
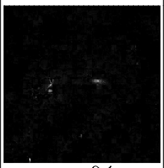
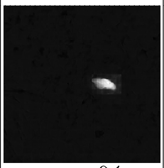
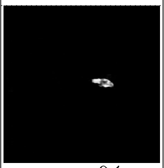
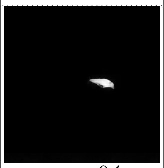
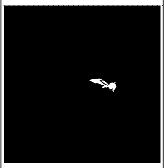
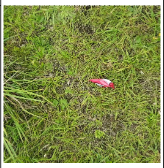
ID	A	B	C	D	E	GT	Input
15							
∩/∪	45.7%	1.2%	46.7%	5%	14.4%		
7							
∩/∪	75%	29.6%	73.6%	62.6%	20.6%		
31							
∩/∪	6.2%	1.6%	5.8%	73.6%	23.7%		
18							
∩/∪	14.9%	9.4%	10.4%	51.4%	24.1%		
34							
∩/∪	49.3%	2.4%	47.8%	39.4%	55.4%		

Table 6.4: This table shows the lowest performing instances from pipeline E. These instances consist of a lot of background noise, obstructions, poor lighting, and smaller objects.

Chapter 7

End-to-End Semantic Segmentation

In Chapter 6, an approach was presented for superpixels to be predicted by a trained deep artificial neural network. These superpixels were used as a feature to improve semantic segmentation, but without changing the weights of the network that predicts the superpixels. This chapter presents the approach for end-to-end semantic segmentation using deep-learned superpixels on a tiny domain-specific datasets, which does allow the superpixel prediction network to be fine-tuned for the domain. The gLitter and the Newborn Resus datasets were used for example use cases for this technique.

7.1 End-to-End Architecture Adaption

This next section goes into detail about how the network structure was adapted for end-to-end learning, and covers the resulting performance of the network upon the gLitter and Newborn Resus datasets.

7.1.1 Methodology

With the available GPUs it was not possible to perform end-to-end deep neural networks that consist of multiple hourglass networks with an input dimension of (1024×1024). Therefore, the dimensionality was reduced from (1024×1024) to (512×512), this allowed the running of multiple hourglass networks one after another. Changing the input resolution allows the hourglass network to compress the representation down further to $(8 \times 8 \times 256)$ with the same number of steps in the hourglass module. However, due to changing the dimensionality of the network's outputs, this meant that the performance metrics could no longer be directly compared to the previous pipelines evaluated in Chapter 6. Additionally, two more changes were made to the gLitter dataset; the addition of negative images that did not contain any litter, and the increase of the size of the dataset to 550 instances. These instances were split into three sets: training, testing, and validation. Each of these sets respectively had 400, 100, and 50 images within them. The

ID	Pipeline (512 × 512)	End-to-End Conditions	Superpixel Type	Average ∩/∪	
				gLitter	Newborn
A	RGB	N/A	N/A	88.8%	12.3%
B	MCG	N/A	SLIC	61.0%	23.4%
C	MCG+RGB	N/A	SLIC	89.0%	45.9%
D	MCG	N/A	DL	79.7%	11.6%
E	MCG+RGB	N/A	DL	88.7%	12.3%
F	End-to-End	Standard	DL	83.2%	43.8%
G	End-to-End	No HG pre-training	SLIC	77.3%	44.2%
H	End-to-End	No Front HG pre-training	SLIC	83.2%	43.0%
I	End-to-End	No Back HG pre-training	DL	78.0%	44.7%
J	End-to-End	Standard + Dual Loss	DL	85.0%	43.0%

Table 7.1: This table shows the average performance of the five pipelines. MCG alone performed the least accurately out of all the pipelines. MCG plus RGB for both methods performed better than RGB alone. Additionally, it can be seen that the deep-learned MCG versus the handcrafted method performed on average 15% more accurately.

addition of the test set allowed comparison between pipelines using genuinely unseen data.

These above-mentioned changes meant that each of the previous pipelines in Chapter 6 had to be rerun so that comparisons could be made between all of the pipelines. The results of the pipelines can be seen in Table 7.1. When comparing the results between the new 512 pipelines (Table 7.1) and the original 1024 pipelines (Table 6.1) for the gLitter dataset, initially it can be seen that for all pipelines the performance increased, with a maximum increase of 9.4%. This increase in general performance could be due to multiple factors such as an increase in dataset size, or the boundary cases around the objects being averaged out into few pixels in the smaller dimensions. Additionally in Table 7.1, it can be seen that the addition of superpixels had a smaller effect on the performance of the pipeline. Moreover, deep-learned superpixels were on par with the baseline, and handcrafted superpixels only performed 0.2% better than the RGB alone baseline.

In addition to the gLitter dataset, the new 512 pipelines were run on the Newborn Resus dataset. In this experiment, all 23 classes from the Newborn Resus dataset were used. Table 7.1 displays that pipeline C performed the most accurately, with a mean Jaccard index of 45.9% over all classes in the dataset and it's predictions on the test set of the can be seen in Figure 7.1.

End-to-End (F): The end-to-end baseline pipeline To attempt to improve the performance of the 512 pipeline E, a new pipeline was created using the end-to-end methodology of having two hourglass networks, with the first feeding directly into the second during training. Each of the hourglass networks have their own role to play. The first hourglass is used to predict the deep-learned superpixels in MCG format, then these, plus the original RGB image, are fed into the second hourglass. The purpose of this second hourglass is to predict the semantic segmentations.

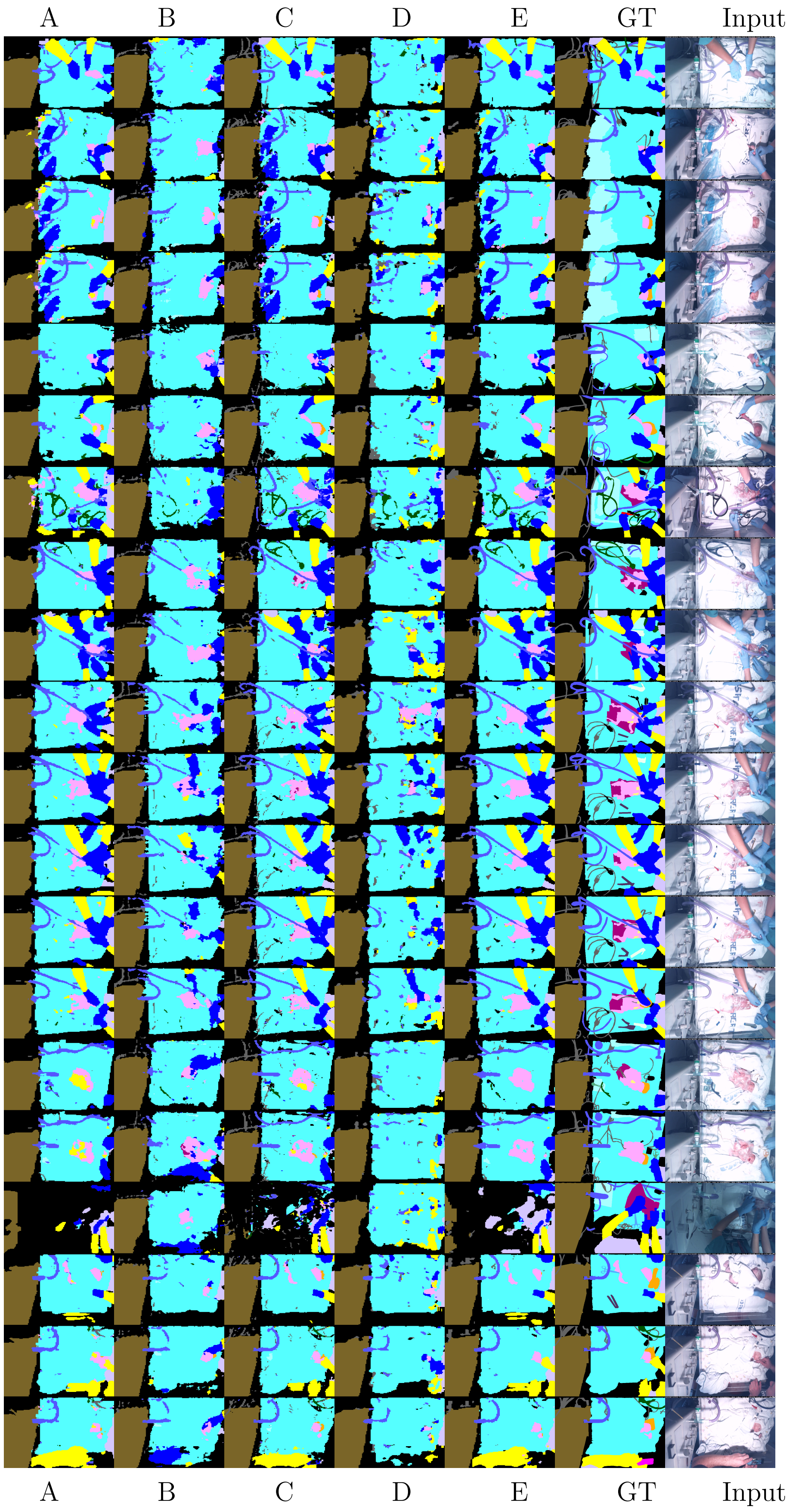


Figure 7.1: Full test set predictions for pipelines A to E and the Newborn Resus dataset.

Three more pipelines were designed to test the effectiveness of pre-training the hourglass networks.

End-to-End No Pre-Training (G): This pipeline removes all pre-training from the two hourglass networks and starts with randomised weights, thus the learning starts from scratch, and allows determination of whether a two-stacked hourglass could learn to solve the semantic segmentation problem.

End-to-End No Front Hourglass Pre-Training (H): This pipeline is similar to the previous pipeline, but keeps the pre-training of the second network. In theory, this pipeline may not work as interned and take a longer time to converge than other pre-trained pipelines. This is because the second network relies on the input of the first, and with random initialisation of the weight for the first hourglass, the loss could be great enough for the second hourglass to unlearn the weights due to the high losses incurred.

End-to-End No Back Hourglasss Pre-Training (I): This pipeline is a reverse of the previous pipeline, with the pre-training being kept for the first hourglass (deep-learned superpixels), and not for the second (semantic segmentation).

End-to-End Dual Loss (J): In the pipelines discussed so far, the loss has only been calculated on the output layer of the end-to-end network, therefore the backpropagation step does not account for the incorrect superpixel predictions at the end of the first sub-network. Both losses use the MSE loss function and are combined by summation of the two losses and back-propagated from the end of the second network. The first loss is calculated from the superpixel predictions and the SLIC superpixel targets. The second loss is calculated from the predicted semantic segmentation and the ground truth segmentation masks.

7.1.2 Experimental Results

The following results are being compared with regards to their performance on the gLitter dataset.

End-to-End (F): The end-to-end pipeline performed about 5% less accurately than the baseline, which was surprising because the weights from both separate hourglass networks were used as the starting point for this network. A possible explanation for this result is that the first hourglass was not fixed and may have unlearnt the deep learned superpixels.

End-to-End No Pre-Training (G): This network performed less accurately, as hypothesised. This was expected because the first hourglass network was no longer specialised in predicting superpixels, meaning that the network would have to train the two stacked hourglasses to predict semantic segmentation.

End-to-End No Front Hourglass Pre-Training (H): This pipeline performed better than the no pre-training (G), but unexpectedly had the same performance as the original end-to-end pipeline (F). A possibility is that the second hourglass' predictions relied less on the deep-learned superpixels for its predictions at the start of training, and slowly trained the first hourglass to help with predictions.

End-to-End No Back Hourglasss Pre-Training (I): This pipeline's performance was much less accurate than the individual pipeline (C,E) and the original end-to-end pipeline (F). This was very unexpected as the hypothesis of using an end-to-end network should in theory improve the predictions, as all parts of the network can be affected by the backpropagation step.

End-to-End Dual Loss (J): This final pipeline had an improvement of 1.8% compared to the first end-to-end pipeline (F). However, this increase was not enough to make it on par with RGB alone (A), or to beat the third pipeline (C), RGB plus handcrafted superpixels. A comparison of the output for all of these pipelines can be found in Appendix 9.4.

Again these pipelines were trained with all 23 classes from the Newborn Resus dataset. Table 7.1 shows that pipeline *C* still has the highest mean Jaccard index of 45.9%, however the mean Jaccard index values for all of the end-to-end pipelines for the Newborn Resus dataset are much higher than pipelines *A*, *B*, *D*, and *E*. The predictions for the best pipeline's network can be see in Figure 7.2.

When looking at the accuracies between the 1024 and 512 variations, it can be seen that the 512 version pipeline accuracies are higher and, in general, closer together. An assumption that could cause this is that there was additional data added to the 512 dataset, meaning that the network being used becomes saturated at 89% with the amount of data given to it. Additionally, as there was only one class, increasing the number of classes may improve the quality of the predictions because there would be less variation in a class, leading to better performance.

A final experiment was performed on this setup that compared the performance of models trained on varying numbers of training samples, but validated on the same test set of 100 samples. This started with five samples, then the number of training samples was steadily increased up to 225. The order in which the samples were added to the training set was randomised, this in addition to the original randomisation of splitting the data into train, validation, and test sets ensured that there were no patterns in the training set. The structure used was pipeline E, RGB plus deep-learned superpixels, using the MCG format. The accuracy for each of the training samples sizes can be seen in Figure 7.3, and shows that with such a small dataset the performance increased with the number of training samples used. This meant it could be hypothesised that with a larger dataset, the accuracy would increase, but it cannot be said when the network would become saturated, and thus improvements to accuracy would minimise. The hypothesis was that doubling the number of training samples to 500 would give over 90% accuracy on this dataset.

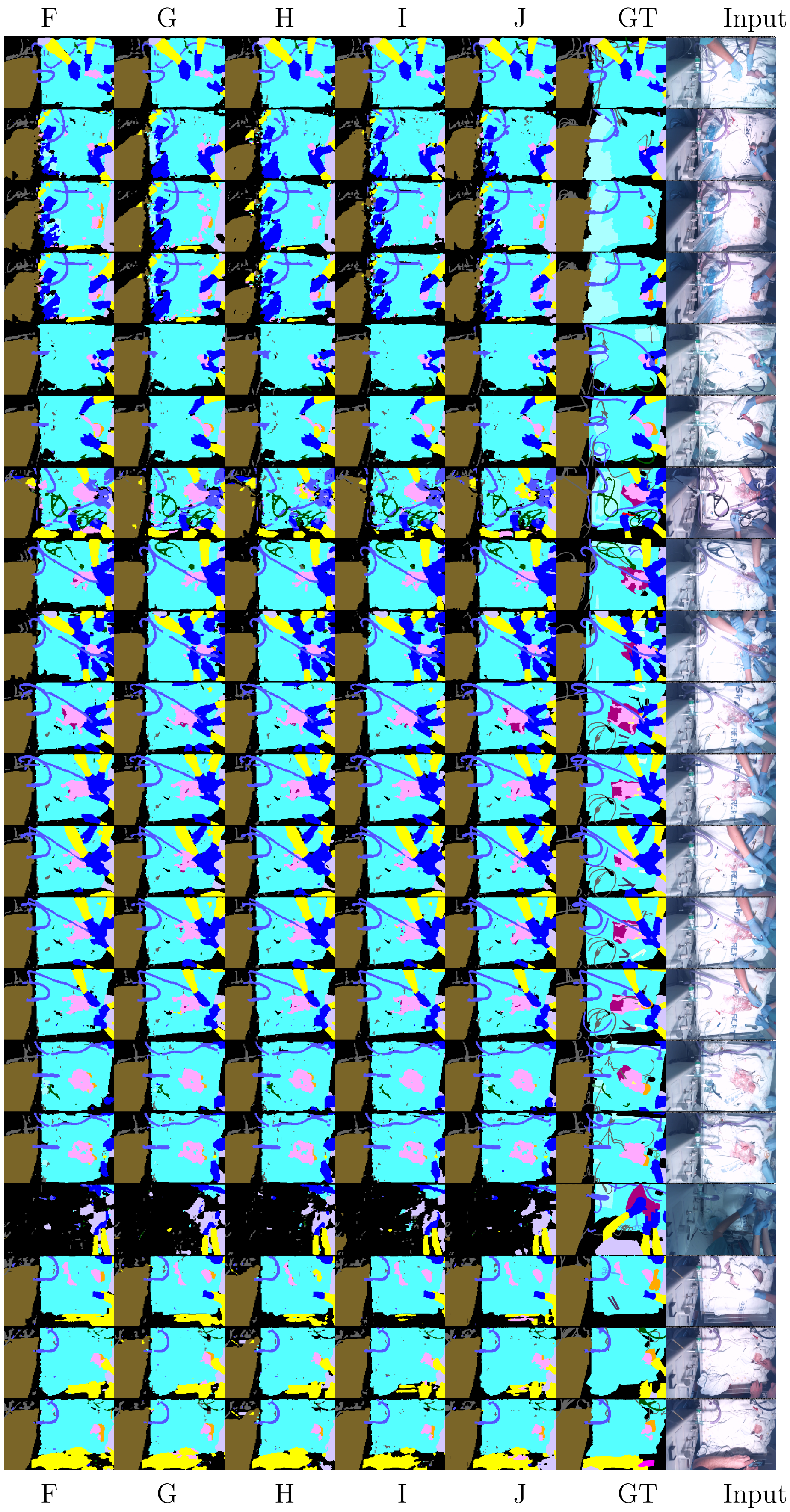


Figure 7.2: Full test set predictions for pipelines F to J and the Newborn Resus dataset.

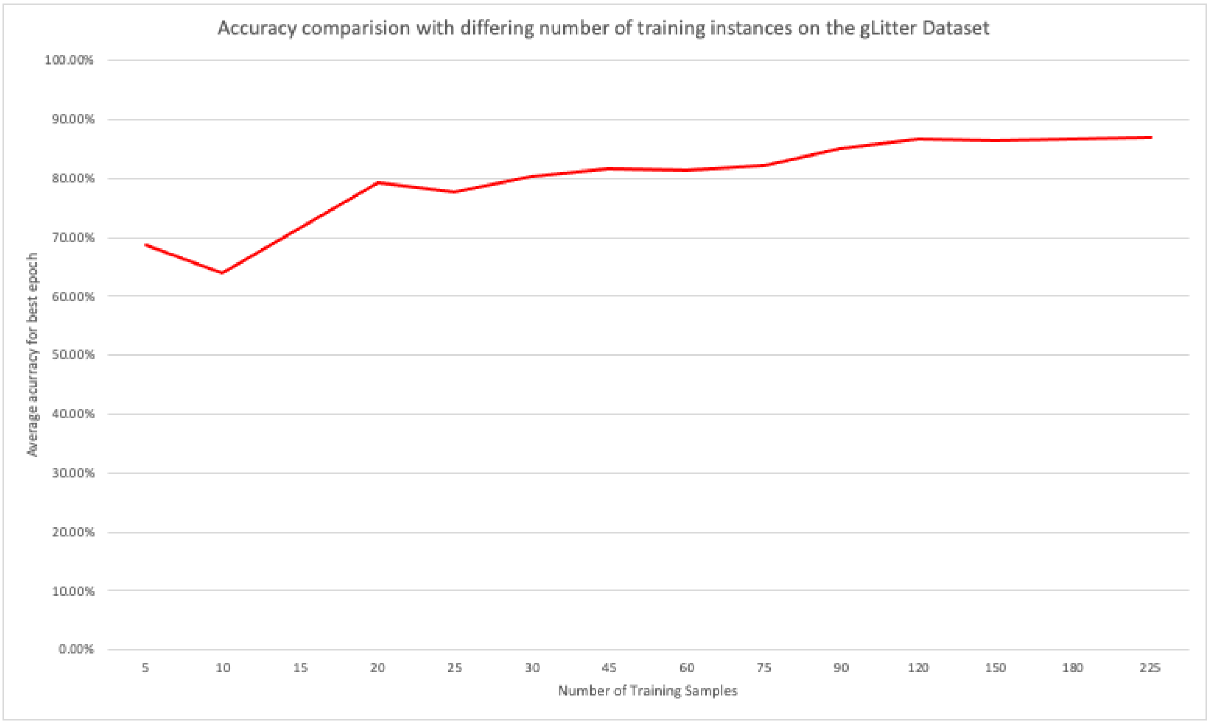


Figure 7.3: This figure shows a graph that represents how the number of training samples effects the accuracy of a network using the gLitter dataset. This network uses the structure from pipeline E. There is a steady increase in performance of the network with each iteration of more training data. The results are calculated on a static test set of 100 samples.

7.2 Multi-Class Refinement

This section explores refining multi-class semantic segmentation with the tiny domain-specific dataset, Newborn Resus, through six experiments. These are: 1) Testing the effectiveness of a predefined unknown class for transfer learning compared to an adaptive unknown class. 2) Checking if learning the least frequent class first improves accuracy for this class. 3) Evaluating if using the same metric for the loss function as calculating the performance metric increases accuracy. 4) Testing if the order of the classes for transfer learning, either ascending or descending with regards to class frequency, matters. 5) Refining the dataset to remove outlier data points to improve accuracy. 6) Increasing the number of semantic segmentation classes from 6 to 32 to see if reducing the complexity of the unknown classes helps with reducing misclassification.

7.2.1 Methodology

To be able to label the Newborn Resus dataset at a later date, both manual and machine annotations were desired in order to achieve a fully labelled dataset in a reasonable time frame for this thesis. Thus when evaluating the performance of the networks in this section, the use of percentage of pixels needed to be updated as an additional metric to F1 score. Percentage of pixels needed to be updated is calculated by $1.0 - accuracy$, where accuracy is the percentage of correct pixel predictions normalised between zero and one. The accuracy for the semantic segmentation is calculated by taking the mean of the union of the predicted and ground truth masks, minus the intersection of these masks.

Despite cooperative learning to fully label datasets becoming common practice, available tools often use pre-trained models to predict on common objects. However, this is not very useful for domain-specific datasets.

Initially, to prove that the concept of semantic segmentation for the Newborn Resus dataset using only 50 frames would work, a subset of five classes (plus the unknown class) from the 23 total classes were used. The five classes selected are seen in Table 7.2. As a baseline, the mean squared error loss function was used for the semantic segmentation sub-network, and the network was trained in a traditional multi-class classification (all classes at once). From the Newborn Resus dataset, 50 random frames were taken to create a sub-dataset that was labelled with the five classes. The labels were created using the Image Annotation Tool (IAT).

To determine the best format for training with such a small dataset, several variables were changed: loss function, transfer learning vs. all at once, and adaptive vs. static unknown class for transfer learning.

The loss functions that were compared were MSE and Lovász-Softmax loss [7]. Lovász-Softmax loss, also known as the Jaccard loss, is an approximation of the intersection over union metric used to determine the accuracy of the network. Berman *et al.* showed that their loss function was able to improve detection of small objects and this property of Jaccard loss function is applicable to the clinical dataset. The dataset they used was the ISBR dataset, consisting of brain scans, to show that very small classes are better learned using the Lovász-Softmax loss compared to cross-entropy loss, see Figure 7.4.

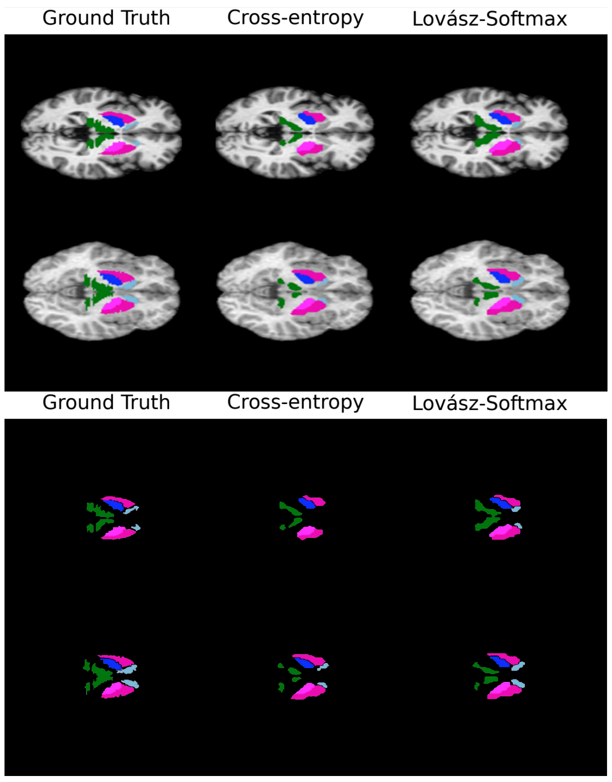


Figure 7.4: Image taken from supplementary material from [7]. Top: original figure from paper, Bottom: masks extracted from figure to increase visibility of segmentations. As can be seen in the bottom segmentation masks, the light blue segmentation is completely missed in the first example when using cross-entropy but is detected when using Lovász-Softmax loss. Additionally, in the second example the network has predicted more of the light blue class compared to the cross-entropy loss.

Transfer learning by using a pre-trained model is common practice in deep learning. However, a pre-trained model (network architecture) was not used in this thesis, instead bootstrapping of a custom architecture network was used. Pre-training the custom architecture was opted against, as it could have taken a long time to train on a large dataset such as MS Coco [64]. Due to the specialised classes in the domain-specific datasets being vastly different to the classes in the public dataset, it may not have been beneficial to use transfer learning in this way. Instead in this work transfer learning was used to train the network one class at a time, and iteratively append new classes to the previously trained models until all classes were learned.

As part of the transfer learning, the use of adaptive unknown class was tested against a static unknown class. The static unknown class is predefined before the start of transfer learning training and does not change between transfer learning runs. Contrastingly, the

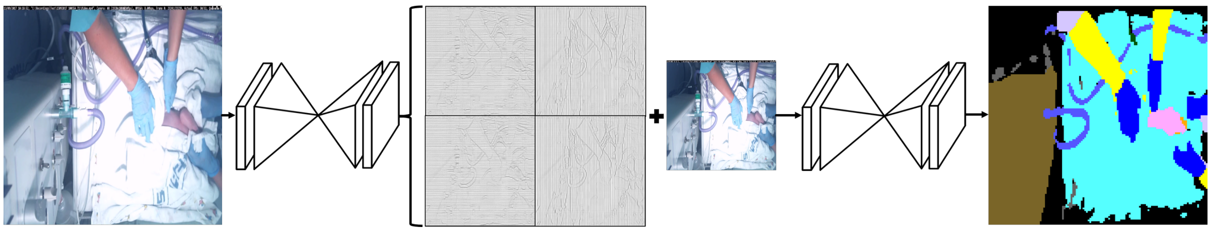


Figure 7.5: This figure shows the network structure that each of the pipelines in this Chapter use. It takes an RGB image as input and first passes through the superpixel generator hourglass sub-network. The first sub-network predicts the four MCGs that represent the deep learned superpixel, one for each direction. Then the deep-learned superpixels output plus the original RGB image are piped directly into the input of the second hourglass sub-network to predict the semantic segmentation. For illustration purposes the output masks have been combined into a single RGB colour image with each class having its own colour.

adaptive unknown is first initialised to everything but the first class, then with each transfer learning run, the unknown class is updated to remove any sections that are now in the next class to learn as part of the transfer learning. Thus on the next iteration of the network, the weights of the new output layer were initialised to be the same as the unknown class, because the new class existed in the previous unknown class. This meant that the network only had to learn to distinguish what was no longer in the unknown class.

To train the network, the 50 frames were partitioned into three sets, with 20 as training, 20 as test, and 10 as validation instances. There were no frames from the same video spanning multiple sets. 14 pipelines were then trained, outlined in Table 7.9, to compare the previously mentioned network parameters. The network structure was similar to the structure in Chapter 6, with the first network being trained on the DAVIS dataset to predict deep-learned superpixels represented using the MCG format, and the second network being trained to predict the semantic segmentation masks, with one channel of the output for a single class. The difference between the two network structures is that instead of the data being manually fed into the second network, it was fed automatically as part of an end-to-end network, allowing for backpropagation to feed all the way from the semantic segmentation predication to the creation of the deep-learned superpixels. This end-to-end structure is illustrated in Figure 7.5
In this section the following six experiments are discussed:

1. The use of predefined unknown class *Vs* the adaptive unknown class for transfer learning against traditional multi-class learning.

2. Comparing the random order for transfer learning performance for small less frequent classes to learning the smallest class first.

3. The use of the Jaccard index as a loss *Vs* MSE.

4. The order of classes for transfer learning: ascending, descending, and random.

5. Refined dataset, to remove inconsistencies in the dataset.

6. Increasing the number of classes to be learned.

Abbreviation	Class
C1	⬛ Unknown
C2	🟦 Bed
C3	🟦 Gloves
C4	🟪 Baby
C5	🟦 Pipes
C6	🟩 Stethoscope

Table 7.2: This table shows a list of the abbreviations used throughout this section.

7.2.2 Experimental Results

Experiment 1: Predefined *Vs* Adaptive Unknown Class

In this experiment the use of predefined unknown class *vs* the adaptive unknown class for transfer learning *vs* traditional multi-class learning was compared.

In Table 7.3, traditional multi-class deep learning is compared to transfer learning using a predefined unknown class and an adaptive unknown class on the Newborn Resus dataset. In the table it can be seen that the adaptive unknown class improved performance for transfer learning methodology in regards to the F1 scores. There was little difference in performance in regards to the percentage of pixel to update for the unknown class type. The mean intersection over union shows transfer learning to perform 11% better than traditional multi-class learning, for both predefined and adaptive unknown classes.

The F1 scores between multi-class and transfer learning using the adaptive unknown were marginal, but there was a much greater decrease in percentage of pixels to update for both transfer learning pipelines. From the F1 scores and percentage of pixels to be updated, the best pipeline was transfer learning with the adaptive unknown class. This was because transfer learning greatly reduced the number of pixels to be updated. Additionally, the adaptive class performed better than using the predefined unknown for the transfer learning, because the network is being informed each pixel's class at all stages of transfer learning, rather than some pixels being in a secondary unknown class. The adaptive unknown class with transfer learning achieved an improvement of 0.091 in regards to the average F1 score.

ID	Network type	Unknown Class type	F1 Score	Pixels Update	Average ∩/∪
A_1	Multi-Class	Predefined	**0.556**	28.80%	71.2%
C_1	Transfer Learning	Predefined	0.460	**17.68%**	82.32%
E_1	Transfer Learning	Adaptive	0.551	17.97%	82.03%

Table 7.3: This table shows the performance of the three pipelines for experiment one. Transfer learning performs much better for both pipelines regarding percentage of pixel to be updated.

Experiment 2: Position of Smallest Least Frequent Class.

In this experiment the position of the smallest least frequent class was changed. Firstly, all of the classes were in a random order, secondly, the smallest class is learned first, and finally, the smallest class is learned last. The smallest class is defined as the class that appears with the smallest number of total pixels in the labelled dataset.

Table 7.4 shows the confusion matrices for the transfer learning using an adaptive unknown class. It can be seen that for class C6, stethoscope, the network was struggling to learn this class and commonly misclassifying it as the unknown class. The stethoscope occurred least frequently and was the smallest class in regards to number of pixels per class.

		Prediction					
		C1	**C2**	**C3**	**C4**	**C5**	**C6**
Actual	**C1**	**2,147,823**	249,309	56,760	125,228	5,351	848
	C2	283,922	**1,742,168**	20,338	18,381	7,132	0
	C3	57,089	69,759	**163,274**	1,346	872	0
	C4	31,773	12,962	860	**61,169**	1,670	0
	C5	41,259	41,285	2,829	2,163	**58,196**	0
	C6	16,566	18,741	403	2,129	1,243	**32**

Table 7.4: This table shows the confusion matrices for the transfer learning methodology using the adaptive unknown class.

		Prediction					
		C1	**C6**	**C3**	**C4**	**C5**	**C2**
Actual	**C1**	**2,224,150**	2,790	215,750	51,297	74,288	17,044
	C6	17,601	**2,416**	15,161	748	2,352	836
	C3	353,288	1,070	**167,5013**	18,297	19,024	5,249
	C4	61,856	0	61,051	**165,949**	1,759	1,725
	C5	38,397	0	16,568	1,230	**51,143**	1,096
	C2	42,772	12	35,625	1,807	1,802	**63,714**

Table 7.5: Confusion matrix for the pipeline where the stethoscope class (C6) is learned in the first network.

When manually analysing the subdataset that had been created, it was noted that the stethoscope class only occurred 14 times in the training set, and six times in the validation set. A hypothesis was formed that the order of the classes may affect the performance of the network on smaller classes, thus classes C2 and C6 were switched.

When comparing the data Table 7.4 and Table 7.5, in particular the results of C6, stethoscope, it can be seen that if the sparse stethoscope object is learned earlier in the transfer learning, it can be learned even with small amounts of training samples to learn from. The confusion matrices also show that the network was still having trouble with misclassification. This was suspected to be caused by the lack of data and vast differences in appearance of the object (going from black ear inserts to silver hardware and tube). When comparing the average performance of the pipelines in Table 7.6, it can be seen that the average performance for pipelines where C6 was learned first perform the best in regards to average F1 scores, but the pipelines where C6 was last have lower average percentage of pixels to update.

ID	Stethoscope Position	F1 Score	Pixels Update	Average $\cap/\cup$
E_1	Last	0.551	**17.97%**	82.03%
I_1	First	**0.578**	19.67%	80.33%

Table 7.6: This table shows the performance of the two pipelines for experiment two. Both pipelines use transfer learning methodology, whilst varying the position of the stethoscope position from being the first class to be learned, to being the final class to be learned. This table also shows the performances of the adaptive and predefined pipelines. When the stethoscope was learned last, the pipeline performs best, in regards to the average F1 score.

Experiment 3: MSE *Vs* Jaccard Loss

In this experiment the use of the Jaccard index as a loss function was compared against the MSE loss function.

As previously mentioned, the stethoscope had been a tricky class to predict thus far. A second possible cause of this problem was the size of the object in the scene and its effect on the MSE loss function. A smaller object in a scene will have less influence upon MSE loss function. This is because if the majority of pixels are correct in the prediction and there is more to gain from optimising for larger classes, more pixels will be updated, and thus the function will favour the larger classes.

To combat the problem of small objects having less importance, a better loss function was needed. The Jaccard index was already being used to calculate the accuracy of the network, so could it be used to train the network? The Jaccard index is an accuracy measure which is used in segmentation and works by calculating the union and intersection of a prediction with the ground truth, then dividing the intersection by the union, $JaccardIndex = \frac{|A \cap B|}{|A \cup B|}$. However as the Jaccard index is not differentiable, it cannot be used in backpropagation, and therefore cannot be used as a loss function. Berman et al. [7] solved the issue of the Jaccard index not being differentiable by approximating it as a sub-modular set function.

Table 7.7 shows the confusion matrix from the pipeline using the Jaccard index loss functions. Class C6 predictions had less confusion when using the Jaccard loss function than when using MSE loss function. However, when comparing the average F1 scores and the percentage of pixels to update for both MSE and Jaccard loss pipelines, the MSE pipeline performed marginally better, as seen in Table 7.8. In regards to F1 score with a small improvement of 0.09, and 1.67% better.

		Prediction					
		C1	**C6**	**C3**	**C4**	**C5**	**C2**
Actual	**C1**	**2,264,529**	152	142,794	34,634	122,683	20,527
	C2	26,243	**1,118**	8,871	427	922	1,533
	C3	535,559	90	**1,476,906**	21,104	17,023	21,259
	C4	81,080	0	53,629	**152,476**	1354	3,801
	C5	36,281	0	13,535	709	**54,144**	3,765
	C6	46,452	0	36,409	3,082	5,362	**54,427**

Table 7.7: Confusion matrix for the pipeline using Lov*á*sz-Softmax loss where the stethoscope class (C6) is learnt in the first network.

ID	Loss Function	F1 Score	Pixels Update	Average $\cap/\cup$
I_1	MSE	0.578	19.67%	80.33%
J_1	Jaccard	0.569	21.34%	78.66%

Table 7.8: This table shows the average performance of the MSE loss function and Jaccard Loss function, in regards to F1 score and percentage of pixels to update.

Experiment 4: Order of All Classes

Does the order of classes for transfer learning matter? In this experiment this refers to the order of the classes, from random to ascending and descending.

To further expand upon experiment two, the ordering of the classes was changed to test if the order of classes, in regards to number of occurrences, impacted performance on the less frequent classes. The results of this experiment can be seen in Table 7.9, with IDs E_1,F_1,K_1,L_1,M_1, and N_1. As can be seen in this table, pipeline M_1 performed the best in regards to F1 score, which was 0.622, much higher than previous pipelines. However, the number of pixels to be updated was much higher at 28.20%.

Pipelines F_1, $_1L$ and N_1, all use the Jaccard loss and have lower percentages of pixels to be updated than pipeline M_1. Thus, it was decided to expand the number of pipelines to further investigate the performance benefit of the Jaccard index. The full results of this are shown in Table 7.9. Pipeline M still remained the highest performing network with regards to F1 score, and pipeline C_1 had the lowest percentage of pixels to update at 17.68%.

ID	Network Type	Order	Unknown Class Type	Loss Function	F1 Score	Pixels Update	Average ∩/∪
A_1	Multi-Class	Random	Predefined	MSE	0.556	28.80%	71.2%
B_1	Multi-Class	Random	Predefined	Jaccard	0.578	24.86%	75.14%
C_1	Transfer	Random	Predefined	MSE	0.460	**17.68%**	82.32%
D_1	Transfer	Random	Predefined	Jaccard	0.537	18.86%	81.14%
E_1	Transfer	Random	Adaptive	MSE	0.551	17.97%	82.03%
F_1	Transfer	Random	Adaptive	Jaccard	**0.556**	19.65%	80.35%
G_1	Transfer	Stethoscope First	Predefined	MSE	0.558	**18.11%**	81.89%
H_1	Transfer	Stethoscope First	Predefined	Jaccard	**0.581**	21.34%	78.66%
I_1	Transfer	Stethoscope First	Adaptive	MSE	0.578	19.67%	80.33%
J_1	Transfer	Stethoscope First	Adaptive	Jaccard	0.569	21.34%	78.66%
K_1	Transfer	Ascending	Adaptive	MSE	0.587	27.61%	72.39%
L_1	Transfer	Ascending	Adaptive	Jaccard	0.565	23.27%	76.73%
M_1	Transfer	Descending	Adaptive	MSE	**0.622**	28.20%	71.8%
N_1	Transfer	Descending	Adaptive	Jaccard	0.571	**22.69%**	77.31%

Table 7.9: This table shows the results for the pipelines used in the four experiments, plus additional variations on the existing pipelines to give a wider view on the effects of each parameter. For example, pipeline B_1 uses the Jaccard loss with the predefined unknown for traditional multi-class predictions. From this table it can be inferred that pipeline M_1 is the best performing pipeline with an F1 score of 0.622.

Experiment 5: Refined Dataset

For this experiment, the previous pipelines were run again to compare how much of an influence a refined dataset had on such a small dataset. The dataset was refined after manually inspecting each specific instance of the test set, because it was noted that some of the instances in the dataset had large enough variation to the rest of the dataset to be seen as completely different classes. This meant that the networks were unable to generalise to this wider class of objects. These instances were of frames that came from the babies born by ‘any route’, as these videos consisted of completely different lighting situations and work space setups. To replace these images, additional frames from the ‘good’ videos were added to create a more consistent dataset.

Table 7.10 shows the performance of the same pipelines but on the new refined dataset. When comparing the performance of the two datasets, it can be clearly seen that the second refined dataset had a higher average performance across all pipelines. However, the difference between the best and worst performing pipelines had decreased. Using two metrics made it difficult to determine the order of performance of the pipelines. To help evaluate this, the F1 score and percentage of incorrect pixels were combined with equal weighting, one-to-one. When looking at the top four performing pipelines, M_2, K_2, F_2, and N_2, it was clear that the ordering of the classes did not have a large impact on the performance of the network.

Pipeline M_2 upon the refined dataset had the highest combined score of 0.779, closely followed by pipelines K_2, F_2, and N_2, with respective scores of 0.776, 0.771, and 0.771. This is within the a margin of error of less than 0.008%, thus showing there is no benefit for changing the order of the inputs to the network. However, in regards to the type of unknown class, this showed that the adaptive class is superior, with all four pipelines using it. The Jaccard loss was not consistently better than MSE, and often the performance between the two loss functions was similar. In the ordering pipelines the Jaccard loss actually performed worse than MSE in both F1 and percentage of incorrect pixels.

ID	Network Type	Order	Unknown Class Type	Loss Function	F1 Score	Pixels Update	Score
A_2	Multi-Class	Random	Predefined	MSE	0.591	17.602%	0.707 (14)
B_2	Multi-Class	Random	Predefined	Jaccard	**0.633**	**15.136%**	0.741 (12)
C_2	Transfer	Random	Predefined	MSE	0.695	**17.811%**	0.759 (7)
D_2	Transfer	Random	Predefined	Jaccard	0.701	18.662%	0.757 (9)
E_2	Transfer	Random	Adaptive	MSE	0.625	17.834%	0.723 (13)
F_2	Transfer	Random	Adaptive	Jaccard	**0.727**	18.493%	0.771 (3)
G_2	Transfer	Stethoscope First	Predefined	MSE	0.706	**17.466%**	0.766 (5)
H_2	Transfer	Stethoscope First	Predefined	Jaccard	0.699	18.237%	0.759 (8)
I_2	Transfer	Stethoscope First	Adaptive	MSE	**0.710**	17.878%	0.766 (6)
J_2	Transfer	Stethoscope First	Adaptive	Jaccard	0.694	18.049%	0.757 (10)
K_2	Transfer	Ascending	Adaptive	MSE	0.726	17.378%	0.776 (2)
L_2	Transfer	Ascending	Adaptive	Jaccard	0.701	18.796%	0.757 (11)
M_2	Transfer	Descending	Adaptive	MSE	**0.727**	**17.031%**	**0.779** (1)
N_2	Transfer	Descending	Adaptive	Jaccard	0.716	17.484%	0.771 (4)

Table 7.10: This table shows the results for the pipelines used in the rerun for all 14 pipelines on the refined dataset. The results clearly illustrate that the order of classes for transfer learning does not matter for a dataset of this size.

Macro- vs Micro-Averages

The training results on the Newborn Resus dataset, although positive for such small amounts of data, struggled to improve the performance above an F1 score of 0.727. Thus the distribution of the classes being learnt was investigated to see if there was a reason for this. The distribution for both the classes per set (training, validation, test) and number of pixels per class can be seen in the graphs in Figure 7.6.

Both graphs show there is a large class imbalance, in regards to both number of occurrences and number of pixels for each class. The class imbalance is emphasised in the graph which shows the number of pixels. For standard machine learning it is common to use the traditional averaging, however this method does not take into account this

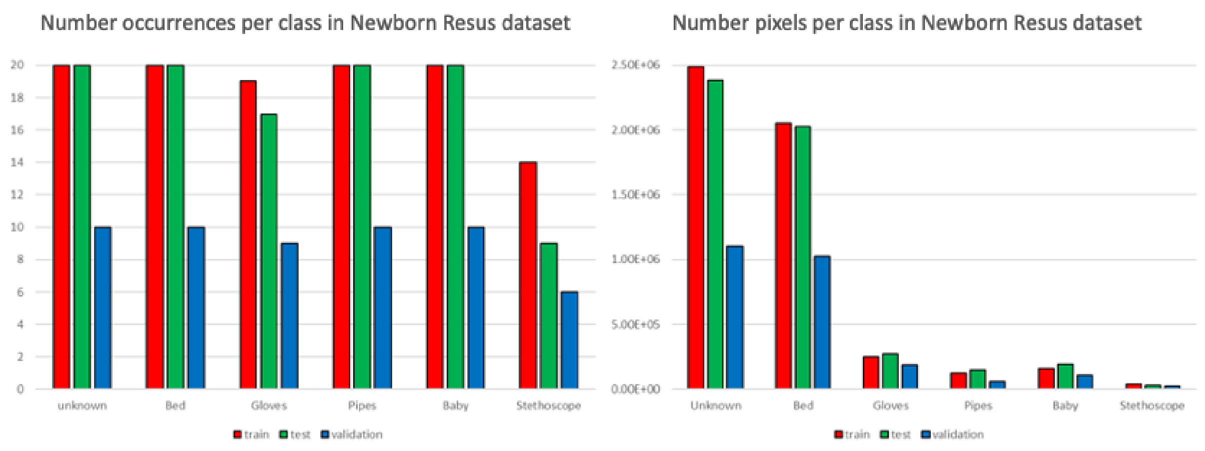


Figure 7.6: This figure shows the distribution of the semantic classes in the Newborn Resus dataset for the subset used in experiments one to five. The left graph shows the number of frames in which the classes occur for each set in the dataset. The right graph shows the number of pixels per class per set in the dataset.

imbalance. This type of averaging is referred to as macro-averaging. Another type of averaging is micro-averaging, this method does take into account the weighting of the classes.

Micro-averaging is a method in which the individual true positives, false positives, and false negatives for each class are summed together. These combined values are then used to calculate the statistics for the network. Micro-averaging is used when the classes in the dataset are unbalanced, because it gives a higher weighting to the classes with more examples. Conversely, macro-averaging is taking the statics of each class, adding them together and dividing by the number of classes.

The following equation shows how the micro- and macro-averages differ for the example data in Table 7.11.

Class x		Actual	
		1	0
Pred	1	TPx	FPx
	0	FNx	TNx

Class 1		Actual	
		1	0
Pred	1	15	10
	0	5	5

Class 2		Actual	
		1	0
Pred	1	50	20
	0	10	30

Table 7.11: Toy data for explaining micro- and macro-averaging.

Recall

$$R_1 = 15/(15+5) = 0.75 \tag{7.1}$$

$$R_2 = 50/(50+10) = 0.83 \tag{7.2}$$

Precision

$$P_1 = 15/(15+10) = 0.60 \tag{7.3}$$

$$P_2 = 50/(50+20) = 0.71 \tag{7.4}$$

F1

$$F1_1 = 2\frac{R_1 \times P_1}{R_1 + P_1} = 2\frac{0.75 \times 0.60}{0.75 + 0.60} = 0.67 \tag{7.5}$$

$$F1_2 = 2\frac{R_2 \times P_2}{R_2 + P_2} = 2\frac{0.83 \times 0.63}{0.83 + 0.63} = 0.77 \tag{7.6}$$

Micro-average of recall

$$MiR = \frac{TP1 + TP2}{TP1 + TP2 + FN1 + FN2} \Rightarrow \frac{15+50}{15+50+5+10} = 0.81 \tag{7.7}$$

Macro-average recall

$$MaR = \frac{R1 + R2}{2} \Rightarrow \frac{0.75 + 0.83}{2} = 0.79 \tag{7.8}$$

Micro-average of precision

$$MiP = \frac{TP1 + TP2}{TP1 + TP2 + FP1 + FP2} \Rightarrow \frac{15+50}{15+50+10+20} = 0.68 \tag{7.9}$$

Macro-average precision

$$MaP = \frac{P_1 + P_2}{2} \Rightarrow \frac{0.60 + 0.71}{2} = 0.66 \tag{7.10}$$

Micro F1

$$MiF1 = 2\frac{MiR * MiP}{MiR + MiP} \Rightarrow 2\frac{0.81 \times 0.68}{0.81 + 0.68} = 0.74 \tag{7.11}$$

Macro F1

$$MaF1 = 2\frac{MaR * MaP}{MaR + MaP} \Rightarrow 2\frac{0.79 \times 0.66}{0.79 + 0.66} = 0.72 \tag{7.12}$$

Average F1

$$F1_{avg} = \frac{F1_1 + F1_2}{2} = \frac{0.67 + 0.77}{2} = 0.72 \tag{7.13}$$

The micro- and macro-averages were calculated and the results can be seen in Table 7.12. Using the micro-averaging, the weighted average for the F1 score was as high as 0.805. However, the networks still followed the same trend in regards to performance, but this representation gives a better understanding of how the network would perform on a hypothetical dataset with an even distribution of classes.

Experiment 6: Adding More Classes

In this experiment, the addition of the rest of the semantic classes in the Newborn Resus dataset were added to test the effect upon the performance of the best performing previous pipeline. It was only performed on the best performing pipeline, because the transfer

ID	Network Type	Order	Unknown Class Type	Loss Function	Macro F1 Score	Micro F1 Score
A_2	Multi-Class	Random	Predefined	MSE	0.591	0.761
B_2	Multi-Class	Random	Predefined	Jaccard	0.633	**0.790**
C_2	Transfer	Random	Predefined	MSE	0.695	0.785
D_2	Transfer	Random	Predefined	Jaccard	0.701	0.791
E_2	Transfer	Random	Adaptive	MSE	0.625	0.796
F_2	Transfer	Random	Adaptive	Jaccard	0.727	**0.801**
G_2	Transfer	Stethoscope First	Predefined	MSE	0.706	0.790
H_2	Transfer	Stethoscope First	Predefined	Jaccard	0.699	0.787
I_2	Transfer	Stethoscope First	Adaptive	MSE	0.710	**0.798**
J_2	Transfer	Stethoscope First	Adaptive	Jaccard	0.694	0.764
K_2	Transfer	Ascending	Adaptive	MSE	0.726	0.765
L_2	Transfer	Ascending	Adaptive	Jaccard	0.701	0.759
M_2	Transfer	Descending	Adaptive	MSE	0.727	**0.805**
N_2	Transfer	Descending	Adaptive	Jaccard	0.716	0.804

Table 7.12: This table shows the results for the pipelines used in the rerun for all 14 pipelines on the refined dataset in regards to their F1 score using both macro- and micro-averaging.

learning technique would take too long to process all pipelines. The additional 18 classes for 12 pipelines would not have been feasible in regards to time spent training the networks with the setup at the university. Hypothetically, if each stage in transfer learning takes a minimum of 3 hours, the total training time would be at least 800 hours. The full list of annotated classes can be found in Table 7.13, the micro average F1 score for all of classes was 0.752 and the average micro intersection over union was 76.54%.

Using the network predictions trained on the refined dataset, the full dataset was labelled with almost full scene segmentation using the model created. Only small sections of the frames were left as unknown. This was where there was not enough information in the parts of the images to label them, and they were not related to any part of the equipment used during the resuscitation procedures. With each frame taking between 30 and 60 minutes to fully annotate, depending on the complexity of the scene, this gave a worst-case estimate for annotation time of 50 hours. The use of the IAT did make certain classes in the dataset much faster, but some of the more complex smaller objects, or object with hard-to-distinguish boundaries, had to be manually annotated with the slower polygon tool.

Using pipeline M_2, the transfer learning was ran for all 23 classes. The average performance decreased compared to the smaller set of classes, but still performed admirably for such a small dataset, Table 7.13. When inspecting the distribution of classes, Figure 7.7, and the confusion matrix, Figure 7.8, it can be seen that not all classes exist in the training set. This means that the network would never be able to predict these

Class	F1	Precision	Recall	Pixels Update	Average $\cap/\cup$
⬛ Unknown	0.672	0.564	0.833	14.03%	85.97%
🟦 Gloves	0.795	0.812	0.780	15.62%	84.38%
🟦 Bed	0.857	0.831	0.884	22.19%	77.81%
🟪 Baby	0.762	0.779	0.745	22.97%	77.03%
🟦 Pipes	0.676	0.791	0.590	23.36%	76.64%
🟩 Stethoscope	0.504	0.580	0.445	23.45%	76.55%
🟨 Arms	0.739	0.623	0.909	23.97%	76.03%
🟧 Hat	0.320	0.515	0.232	23.98%	76.02%
🟫 Machines	0.865	0.987	0.770	24.21%	75.79%
🟥 Syringe	1.000	1.000	1.000	24.21%	75.79%
🟩 Blue Towel	0.000	0.000	1.000	24.70%	75.3%
🟩 Scissors	0.000	1.000	0.000	24.71%	75.29%
🟨 Electric Patches	0.000	1.000	0.000	24.71%	75.29%
🟪 Mobile	0.000	1.000	0.000	24.72%	75.28%
🟪 Plastic Bag	0.000	1.000	0.000	24.72%	75.28%
🟦 Packaging	0.134	0.747	0.074	24.75%	75.25%
🟦 Umbilical Cord Clamp	0.000	1.000	0.000	24.75%	75.25%
🟩 Pink Jacket	1.000	1.000	1.000	24.75%	75.25%
⬛ Wires	0.163	0.363	0.105	24.75%	75.25%
🟪 Name tag	0.000	1.000	0.000	24.75%	75.25%
🟥 Umbilical Cord	1.000	1.000	1.000	24.75%	75.25%
🟪 Clothing	0.570	0.508	0.648	24.75%	75.25%
⬜ Airway Opener	0.000	1.000	0.000	24.75%	75.25%
Macro Average	0.437	0.787	0.479	23.46%	76.54%
Micro Average	0.752	0.752	0.752	23.46%	76.54%

Table 7.13: The 23 classes shown with their respective F1, precision, recall and incorrect pixel percentages.

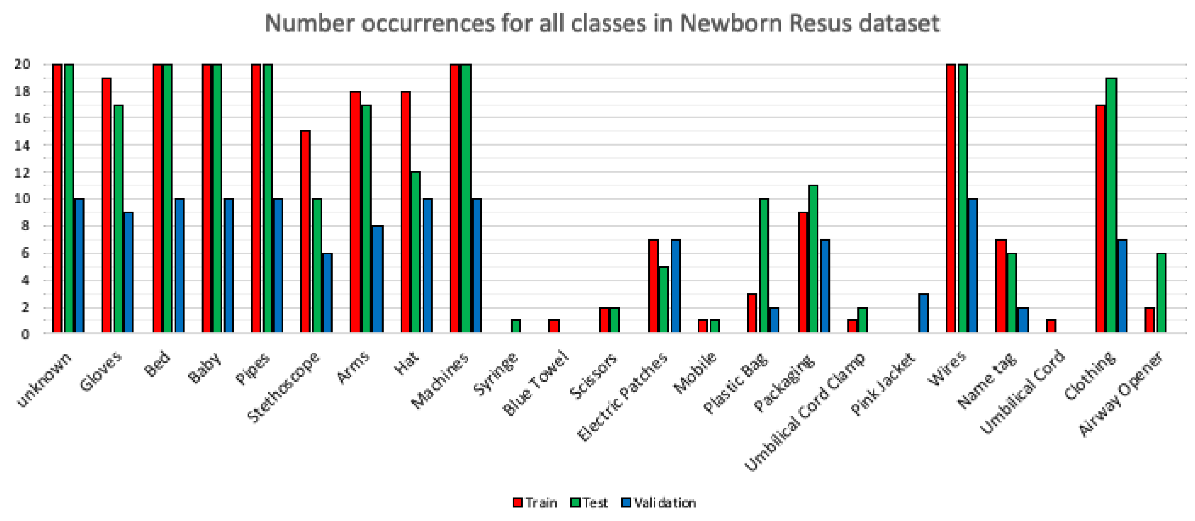


Figure 7.7: Class occurrences for the refined dataset between training, test, and validation sets.

unseen classes. This issue arose due to the nature of the dataset, since each video does not follow an exact procedure so some of the objects do not appear in some videos.

Additionally, taking only a few frames for each video drastically reduces the probability of the majority of classes being present in every frame. The performance of the classes that existed in the training sets are a great deal better than the average. The network misclassified classes that are very similar to the 'unknown' or 'bed' classes. It was hypothesised that this was due to the difficulties that arose by over-exposure of the dataset. Over-exposure resulted in most of the classes being white-washed. Despite this, the network still managed to detect almost all classes in the training set.

The prediction of the network on the test set can be seen in Figure 7.9. When taking into consideration that this network had only been trained on 20 training examples, the predictions are very close to the ground truth segmentation masks. If a given dataset is well-created with good ground truth data, inputs that are very consistent, with no harsh variability such as exposure, and the classes are well-balanced, then semantic segmentation using tiny domain-specific clinical datasets is possible. This is especially true when using methods such as transfer learning and deep-learned superpixels.

7.3 Conclusion

To conclude, the best network structure for using deep-learned superpixels for semantic segmentation is an end-to-end methodology with two hourglass networks joined together. The first of the two sub-networks pre-trained to predict SLIC superpixels represented by a MCG, with it being trained on a large varied dataset and targeting multiple sized superpixels. For input to this network, an RGB image of $(512 \times 512 \times 3)$ was used. The

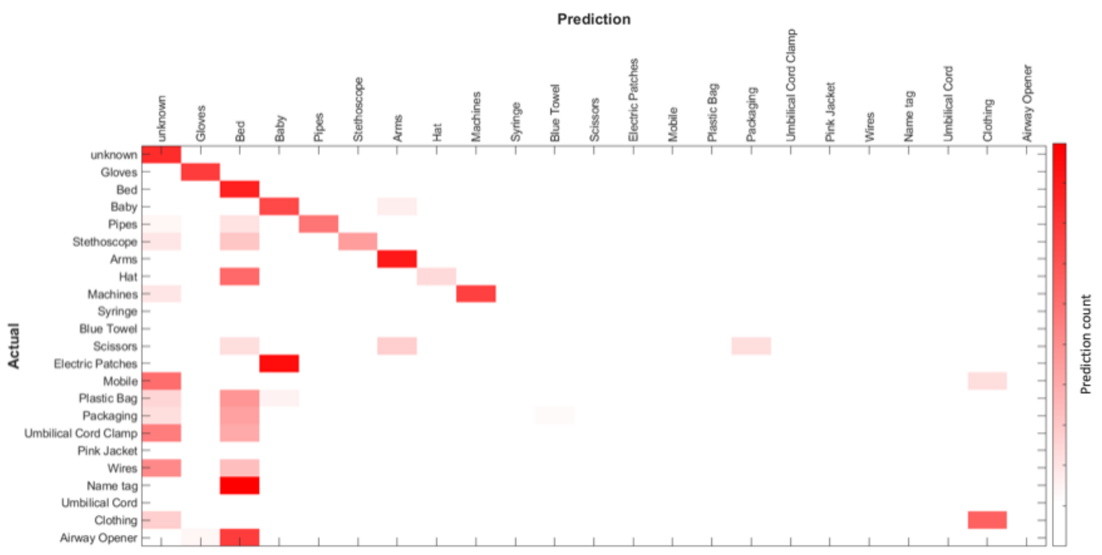


Figure 7.8: Confusion matrix for the refined dataset. The horizontal axis represents the prediction, and the vertical axis represents the ground truths. The darker the colour, the higher the count of predictions in that cell. True positive predictions are in the diagonal cells from the upper left to lower right.

second hourglass sub-network takes in as input the predicted deep-learned superpixels plus a copy of the RGB image. This sub-network can be pre-trained on the given dataset to improve performance, but it is also able to train from scratch. For the loss functions, MSE was used for both sub-networks, and their losses are added together in a dual loss configuration. When training the network, a transfer learning approach is recommended, by training the network multiple times and increasing the number of classes each time. The ordering of classes did not result in a statistical benefit for such a small dataset. It is highly recommended that if the dataset consists of a large variety of classes, then the number of instances per class are balanced, because less frequent classes did not learn well. Additionally, refining the dataset to remove any large inconsistency between instances is recommended, because there was a 0.105 increase in F1 score when doing this.

In the next chapter, the best achieving semantic segmentation network from this chapter is used to label all of the frames in the Newborn Resus dataset, and these predictions are used as the input to an RNN network using LSTM to perform action detection.

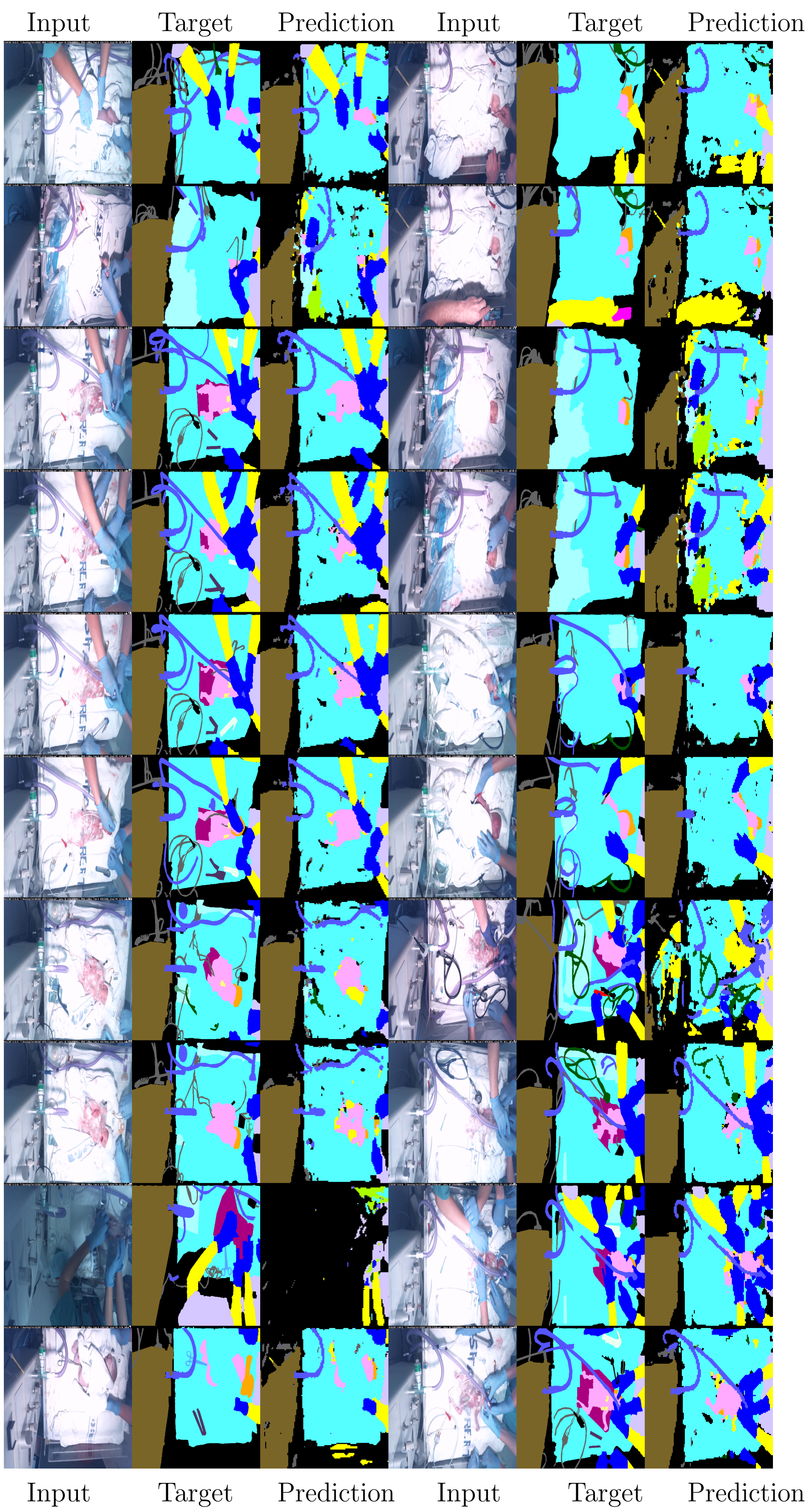


Figure 7.9: Full test set of images, from left to right: input image, ground truth and prediction. Performance for images with similar exposure performs well, however the underexposed image on row nine performs poorly.

Chapter 8

Action Detection

The one of the goals of this thesis was to find techniques to allow small domain-specific medical datasets to be used in deep learning, and what are the key factors in achieving useful predictions. The dataset that best fits this description was the Newborn Resus dataset. The aim for this dataset was to perform action detection on videos of postnatal resuscitation to aid in evaluation of how well the correct procedure has been followed, and possibly help in training of new paediatric staff by detecting when certain actions are performed.

In this chapter, the exploration of deep learning action recognition using the Newborn Resus dataset is discussed. When using temporal data in deep learning, recurrent neural networks (RNN) are often used. For the dataset, LSTMs and Bi-directional LSTM nodes are used in the architecture of the network. The best performance achieved for action recognition with the Newborn Resus dataset was an F1 score of 0.5181, averaged over seven actions, and an average accuracy of 90.72%. This network used an input dimensionality of $(32 \times 32 \times 2)$.

The input for the action recognition network is the output of the end-to-end semantic segmentation network. As the output of the semantic segmentation network is of resolution $(512 \times 512 \times N)$, where N is the number of semantic classes, this gives a large feature space for the RNN to learn from. Having a smaller feature space reduces the amount of information the network has to find patterns in. Additionally, using small dataset with few examples of each action should help focus the network.

To reduce the feature space, the output is first converted from multiple output masks, $(512 \times 512 \times 23)$, to a single output mask, (512×512), via argmax, which takes the class with the highest prediction confidence to be the predicted class. Next a custom pooling process is used to iteratively down-sample the output from (512×512) to $(N \times N \times C)$, where N is the new output dimension and C is the number of frequency layers. Each frequency layer consists of the C^{th} most frequent class for each cell. The custom pooling orders the unique predicted classes per pooling area by the total number of predicted cells. In Section 8.2 the input dimensionality, $(N \times N \times C)$, is experimented with to find a good solution, first starting with $N = 16$ and $C = 1$.

Therefore using this reduction technique the input matrix was converted into a (16×16) matrix. This matrix represents the most common semantic class for the area is stored, Figure 8.1. The input resolution went from 5,767,168 features, $(512 \times 512 \times 22)$, to 256 features.

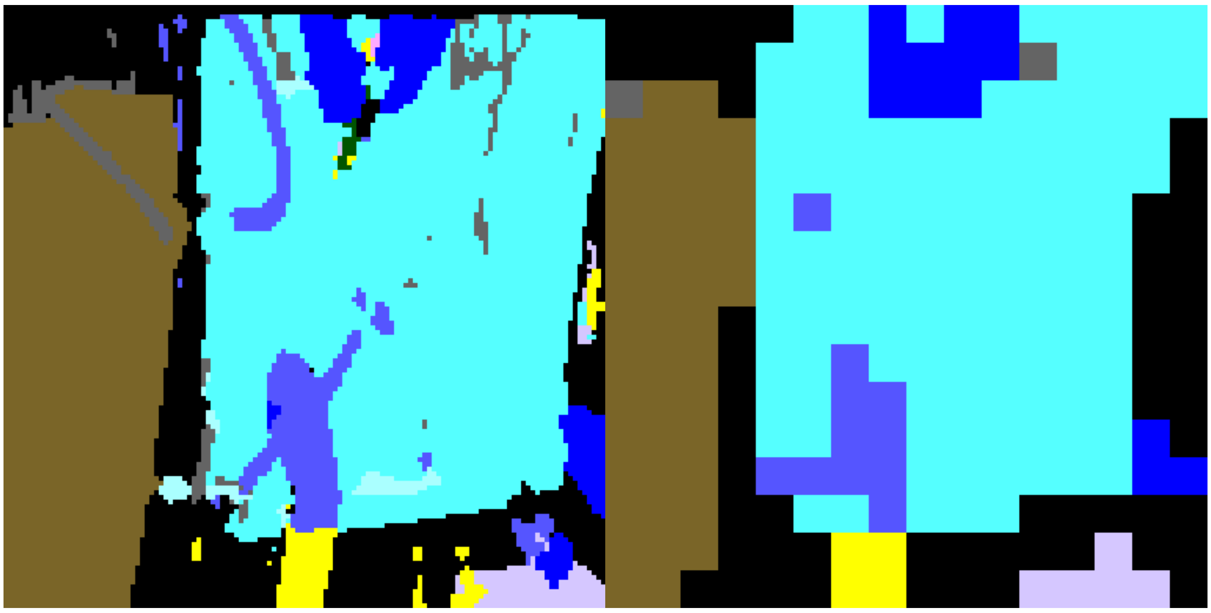


Figure 8.1: This figure shows the down-scaling from (512×512) to (16×16) using a single colour image to represent all classes. This reduces the dimensionality by a factor of 22528. If using N channels to represent masks per class, the input image would have a depth of 22 and the output depth would be one as each class is represented by a single number.

Additionally, the dataset was streamlined by removing as much invariance from the videos in the dataset as possible. This is done by excluding any videos that are different to the majority of the videos. Half of the dataset was already excluded as there was too much variation in lighting, position and objects, resulting in the total used videos to 35 out of 70. The videos were then streamlined further to 22 videos to reduce the variation in the inputs to the networks, thus improving the predictive capability of the action recognition network.

The Newborn Resus dataset consists of 19 possible actions that can be seen in Table 8.1. From the provided labels for the dataset, action D occurred most frequently. Action D is where the paediatric staff use a stethoscope to assess the baby's heart rate. The stethoscopes in the dataset are mostly black with silver hardware, however there are a few cases where a different coloured stethoscope is used. This may have affected the stethoscope prediction in the semantic segmentation network, therefore impacting the prediction in the action detection network.

The provided labels for the Newborn Resus dataset were not in a compatible format for creating a script to create labels for a network to train from. This was because

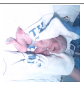
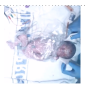
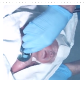
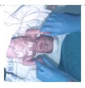
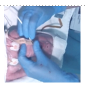
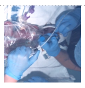
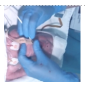
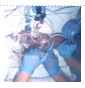
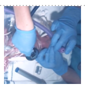
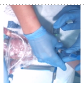
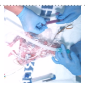
Class ID	Description	Example	Class ID	Description	Example
A	Dried with towel		K	Provide five inflation breaths lasting three seconds	
B	Wrapped in polythene bag		L	Ventilation breaths given incorrect ratio	
C	Cap placed on head		M	Ventilation breaths given for 30s and 1s each	
D	Heart rate assessed stethoscope		N	Stimulation the infant	
E	Attachment of pulse oximeter		O	Cleared away suctioning	
G	Place ECG on chest		P	Incubation attempt	
F	Pule oximeter adjusted		Q	Attach mask on after incubation	
H	ECG adjusted		R	Administer suffoctant	
I	Airway manoeuvre		S	Insert NGT (feeding tube)	
J	Inflation breaths given incorrect ratio				

Table 8.1: List of all actions provided in the Newborn Resus dataset assigned to an ID for use in later tables.In Appendix A.2 larger version of the example images can be found in Figures B.1-B.19.

the provided labels were in Excel files with varying structure. These Excel files were interpretable by a human, and the most efficient way to relabel the dataset was to use NoVA. Initially, only action D was labelled with NoVA because the concept of using the reduced size of input had not yet been proven to work, and additionally the relabelling of the dataset was time consuming.

8.1 Labelling Dataset

To fully label the Newborn Resus dataset, the model from the end-to-end semantic segmentation using the deep-learned superpixel network was used to predict each of the 23 semantic classes for every frame in the dataset. However, as this network was only trained on 20 samples this was not enough to encapsulate the full dataset variations as shown in the previous chapter, where a differently exposed sample image performed much worse.

To combat this, more data was needed to fine-tune the model. This was accomplished by testing two pipelines after running the full dataset though the model. Pipeline one was to choose N worst performing frames, pipeline two was to choose N random frames.

To choose the worst frames for pipeline one, the prediction confidence from the model output over the full dataset was used to determine which images the network performed worst on. The lowest N frames were then used.

To determine N, the number of frames to add to the dataset, the time it would take to annotate different number of frames was calculated. It was decided that the number of frames to be added would always be a percentage of the training set. The original training set $h0$ consisted of 20 instances. The percentage of numbers was varied between 10% and 50% in steps of 5%, and shown for five iterations. The results of this can be seen in Table 8.2. 25% and three iterations were the parameters that were decided upon.

Percentage	h0		h1		h2		h3		h4		h5		Total	
	Frames	Time	Frames	Time	Frames	Time	Frames	Time	Frames	Time	Frames	Time	Frames	Time
10%	22	30_m	24	30_m	26	30_m	29	45_m	32	45_m	35	45_m	15	225_m
15%	23	45_m	26	45_m	30	60_m	35	75_m	40	75_m	46	90_m	26	390_m
20%	24	60_m	29	75_m	35	90_m	42	105_m	50	120_m	60	150_m	40	600_m
25%	25	75_m	31	90_m	39	120_m	49	150_m	61	180_m	76	225_m	56	840_m
30%	26	90_m	34	120_m	44	150_m	57	195_m	74	255_m	96	330_m	76	1140_m
35%	27	105_m	36	135_m	49	195_m	66	255_m	89	354_m	120	465_m	100	1500_m
40%	28	120_m	39	165_m	55	240_m	77	330_m	108	465_m	151	645_m	131	1965_m
45%	29	135_m	42	195_m	61	285_m	88	405_m	128	600_m	186	870_m	166	2490_m
50%	30	150_m	45	225_m	68	345_m	102	510_m	153	765_m	230	1155_m	210	3150_m

Table 8.2: This table shows the total number of frames and the time it would take to annotate each, using 15 minutes as a minimum time for annotation, for each iteration of the training set increase. This tables shows up to five iterations for percentages 10% to 50% in steps of 5%.

Using the two pipelines, 30 more instances were added to each of the training sets. To evaluate which of the two pipelines performed best, the accuracies were compared. Additionally, the model training times, and inference times were recorded, this can be seen in Table 8.3. When comparing the inference times there was not much difference between the two pipelines, and the time taken for the pipelines to converge did not correlate to either better scores or better methods. Pipeline one, worst predictions, was decided to be the better pipeline due to the higher accuracy values. Each pipeline took approximately 80-plus hours to complete, with each step of the process waiting on the previous step. Moreover, with this setup, each step had to be manually activated. This was also factored into why only three iterations were used for increasing the dataset.

Pipeline one was run over the full dataset of frames one last time to create a fully semantically segmented version of the Newborn Resus dataset.

Iteration	Accuracy		Training Time		Inference Time	
	Random	Worst	Random	Worst	Random	Worst
h0	0.655		49_H 44_M 26_S		4_H 47_M 4_S	
h1	0.653	0.663	3_H 41_M 8_S	4_H 27_M 50_S	4_H 11_M 26_S	3_H 45_M 17_S
h2	0.648	0.667	5_H 4_M 51_S	3_H 19_M 23_S	5_H 15_M 47_S	5_H 19_M 53_S
h3	0.657	0.658	5_H 13_M 3_S	9_H 4_M 15_S	-	-

Table 8.3: Average accuracy of iteration and the time taken for training the model.

8.2 Single Class

The section discusses the proof of concept that action recognition can be achieved with tiny datasets by first learning to predict a single action from the Newborn Resus dataset. The input to proof of concept RNN pipeline makes use of the down-sampled semantic segmentation predictions. The single class chosen for this experiment was class D due to it being the most frequent class in the dataset. Using only class D from the Newborn Resus dataset meant that there would be a huge class imbalance between positive and negative labels. To solve the imbalance problem, each occurrence of action D in the dataset was iterated over, and the frames were marked to be included in the new sub-dataset. The length of the occurrence ($D_{duration}$) of the current action D was calculated, and padding of $D_{duration}/2$ was added to either side of the action. In the current state, the new dataset would only work on data similar to it in regards to the pattern of negative-positive-negative, but in a real situation, the there would be more cases of negative-negative-negative. To include this into the dataset, random samples of the original dataset were added that were not already included in the new dataset, that had roughly the average length of the occurrence of action D.

The network structures used for this dataset followed the pattern of having at least one convolution, followed by at least one LSTM. For both the convolutional layers and the LSTM layers, the numbers ranged between one and three, giving nine possible networks. To reduce overfitting to such a small dataset, dropout layers were added after the LSTM layers and after the first dense layer, with the dropout value set to 0.2. The final layer in the network was another dense layer with soft-max activation. These networks were then compared against each other to find which performed the best. Detail on all of the networks' performances can be found in Appendix B.

To compare performance between networks, the F1 score was used. Because a confusion matrix is needed to calculate F1, this also allowed for comparisons of recall and precision of the network. Table 8.4 shows the layout of a confusion matrix. For the input of (16×16), the best network structure found was two convolutional layers followed by a single LSTM layer, and can be seen in Figure 8.2. This network architecture achieved an F1 score of 0.2717, and an accuracy of 65.56%. Since the F1 score was so low, a bidirectional LSTM was tried to find out if having more temporal data would help the network. This increased the F1 score to 0.3435, however this was still on the low side. This led to the assumption that some information in the feature reduction step had been lost and this was preventing the network from learning.

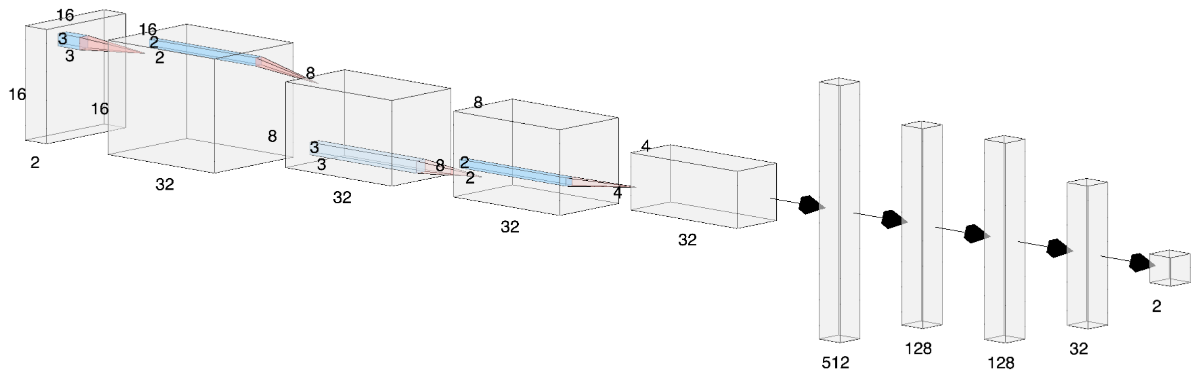


Figure 8.2: This figure shows the network structure for the best performing network and consists of 10 layers (L). From left to right the layers in the network are: (L1) Input of $(16 \times 16 \times 1)$, (L2) 2D Convolution with filter (3×3), (L3) Max pool (2×2), (L4) 2D Convolution with filter (3×3), (L5) Max pool (2×2), (L6) Flatten, (L7) Bi-directional LSTM + Dropout 0.2 + Batch Normalization, (L8) Flatten, (L9) Dense layer of size 32 with ReLU activation + Dropout 0.2, (L10) Dense layer of size 2 with sigmoid activation. This figure was created using the NN-SVG: Publication-Ready Neural Network Architecture Schematics by LeNail [61].

		Actual	
		1	0
Pred	1	TP	FP
	0	FN	TN

Table 8.4: This confusion matrix layout is used in each of the tables in this chapter which have the column heading Confusion Matrix.

Another metric that can be used is the average accuracy of the network. This is calculated by computing the sum of the true positives and true negatives, divided by the total number of predictions. Average accuracy shows how frequently the network's predictions are correct. However, average accuracy does not account for false positive and false negatives predictions.

The first way to increase the amount of information passed to the network was to add a second channel to the input data, thus going from an input of (16×16) to $(16 \times 16 \times 2)$. The first channel was used for the class with the highest prediction value, and the second channel for the second highest prediction. This was done because if there are two objects in an area such as a hand and a stethoscope, the hand would take up much more of the area and the stethoscope would be lost. Adding an extra channel to the input only doubles the input features to the network, compared to doubling the size of the input which results in a quadrupling of the input feature size. This allows the input features to be kept as small as possible, whilst increasing the information given to the network.

Contradictory to the hypothesis, the addition of this extra channel had the opposite effect and reduced the F1 score. Next, the input size was doubled from (16×16) to (32×32) check if this would improve the prediction. However, this again resulted in a worse performance than the second network structure. The results for the best performing structure given a particular input can be seen in Table 8.5.

Input Dimensions	LSTM Type	Best Structure	Best Accuracy	Best F1	Best Recall	Best Precision	Confusion Matrix	
$(16 \times 16 \times 1)$	LSTM	2 Conv 1 LSTM	0.7819	0.2717	0.2528	0.2937	158	380
							467	1766
$(16 \times 16 \times 1)$	BLSTM	2 Conv 1 LSTM	0.6556	**0.3435**	0.5872	0.2427	367	1145
							285	1001
$(16 \times 16 \times 2)$	BLSTM	3 Conv 1 LSTM	0.7856	0.1832	0.1600	0.2141	100	367
							588	2059
$(32 \times 32 \times 2)$	BLSTM	2 Conv 1 LSTM	0.7362	0.2539	0.2986	0.2208	189	667
							444	2138

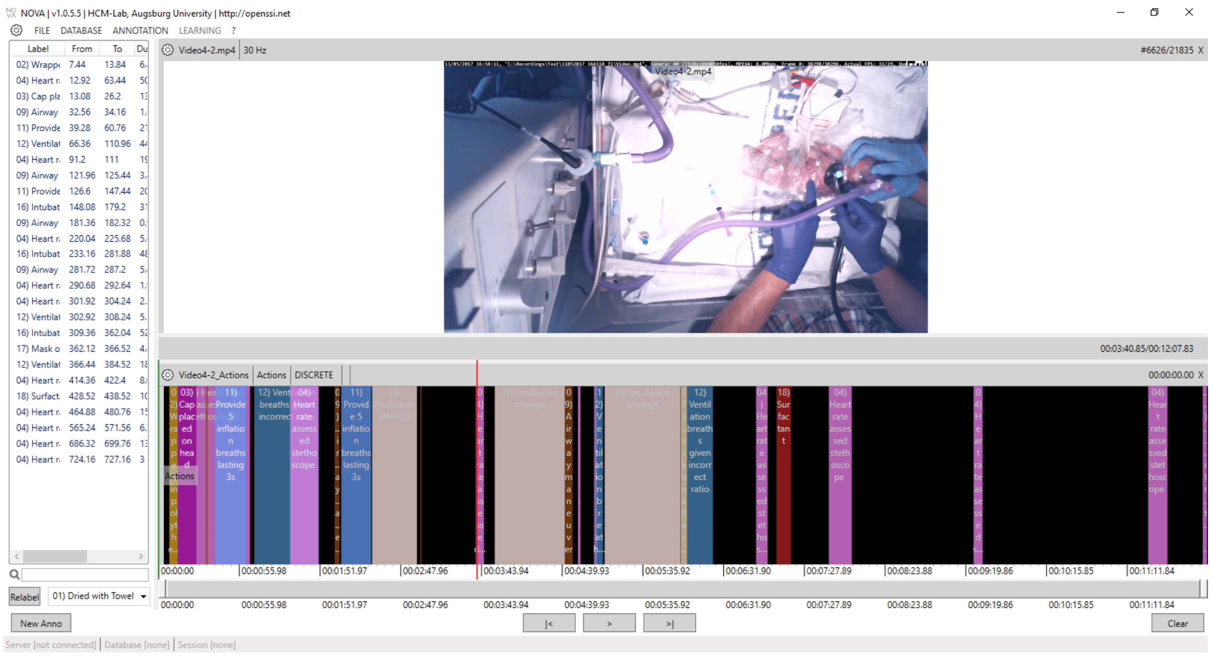
Table 8.5: These tables show the results for prediction class D for the best performing network structures for each input and LSTM type. The network with input $(16 \times 16 \times 1)$ with two convolutions and one bidirectional LSTM performs the best with an F1 score of 0.3435.

An extensive search was performed to find the best architecture parameters. This included varying the input dimensionality, the number of channels for class frequency, the number of convolutions, the number of LSTMS, and the types of LSTM (standard or bidirectional). The best accuracy and F1 scores for single action detection were 85.56% and 0.2469 respectively. These results used the configuration of $(32 \times 32 \times 2)$ input to three convolutional layers and one bidirectional LSTM layer. Other network configurations and their performance for detection of a single action can be found in Appendix B, Tables B.1, B.2, B.3, B.4.

8.3 Multi-Class

In an attempt to further increase the performance of the action recognition network, the hypothesis of additional classes was formed. This hypothesis came from the assumption that the network would be able to classify some of the negative examples into their correct classes and thus reduce the misclassification of the original action D.

The final dense layer in the network had to be changed from soft-max to sigmoid activation, because this allows multiple actions to be activated at the same time. Soft-max only allows for a single action to be active at one time.



Label	From	To	Du
02) Wrappe	7.44	13.84	6.
04) Heart r.	12.92	63.44	50
03) Cap pla	13.08	26.2	13
09) Airway	32.56	34.16	1.
11) Provide	39.28	60.76	21
12) Ventilat	66.36	110.96	44
04) Heart r.	91.2	111	19
09) Airway	121.96	125.44	3.
11) Provide	126.6	147.44	20
16) Intubat	148.08	179.2	31
09) Airway	181.36	182.32	0.
04) Heart r.	220.04	225.68	5.
16) Intubat	233.16	281.88	48
09) Airway	281.72	287.2	5.
04) Heart r.	290.68	292.64	1.
04) Heart r.	301.92	304.24	2.
12) Ventilat	302.92	308.24	5.
16) Intubat	309.36	362.04	52
17) Mask o	362.12	366.52	4.
12) Ventilat	366.44	384.52	18
04) Heart r.	414.36	422.4	8.
18) Surfact	428.52	438.52	10
04) Heart r.	464.88	480.76	15
04) Heart r.	565.24	571.56	6.
04) Heart r.	686.32	699.76	13
04) Heart r.	724.16	727.16	3

Figure 8.3: This figure shows a video from the Newborn Resus dataset that has been fully labelled with action annotations using the NoVA tool.

To start the multi-class action recognition, the Newborn Resus dataset needed to be fully re-annotated in NoVA with all action classes labelled. The relabelling process took anywhere from 30 minutes to two hours to annotate each video fully. An example of a video that has been fully annotated using the NoVA tool can be seen in Figure 8.3. This varied drastically depending on the length of the video, the number of actions present, and the quality of the original dataset labels. Whilst relabelling the dataset, it was found that there had been several action labels that were missed by the original annotator, and the original annotations only specified when an action was to start in a five second window.

Whilst re-annotating the dataset it was noted that there seemed to be a class imbalance of the actions. This can be seen in Table 8.6. Instead of trying to learn actions with very few occurrences or not at all, these actions were either ignored or combined into similar actions, as can be seen in Table 8.7. For example, any assisted breathing had been combined together into a single class because the input frames are indistinguishable between them, and thus in theory the machine should have learnt them better as single class.

The full re-annotation of the Newborn Resus dataset took approximately 27 hours using NoVA. Because this was such a slow process, it was decided to train the network on half of the fully annotated data to see if adding the extra classes would align with the hypothesis.

Action	22 Videos			11 Videos		
	Occurrences	Total Duration (seconds)	Average Duration (seconds)	Occurrences	Total Duration (seconds)	Average Duration (seconds)
A	30	4913.56	163.79	11	478.52	43.50
B	7	216.84	30.98	2	30.08	15.04
C	40	5629.04	140.71	20	3007.32	150.38
D	104	31572.76	303.58	62	19807.28	319.47
E	22	1806.16	82.10	10	948.6	94.86
F	13	3103.84	238.76	6	1278	213
G	19	887.12	46.69	9	541.76	60.20
H	2	196.56	98.28	0	0	0
I	67	24889.2	371.48	35	12571.96	359.20
J	3	435.4	145.13	2	207.64	103.82
K	6	727.24	121.21	5	650.68	130.14
L	20	8307.64	415.38	12	3009.04	250.75
M	5	1114.08	222.82	2	434.2	217.10
N	28	9297.92	332.07	11	3412.68	310.24
O	9	5911.88	656.88	1	224.44	224.44
P	7	2773.76	396.25	3	778.12	259.37
Q	4	1731.84	432.96	1	350.52	350.52
R	2	1250.72	625.36	1	421.52	421.52
S	0	0	0	0	0	0

Table 8.6: This table shows the total occurrences of each action in the Newborn Resus dataset once fully completed, and at the halfway point. Additionally it shows the total and average duration of each action. It can be seen that for some actions there are no or very few occurrences.

8.3.1 Half Dataset

To find the best network architecture, the same methodology was used as with the single class, with nine networks for each input configuration being tested. The results were much better than for the full dataset, with the best F1 score being 0.5013. The network had an input of $(32 \times 32 \times 2)$ and its structure was two convolutions followed by three bidirectional LSTMs. This network structure can be seen in Figure 8.4. This network also had an average accuracy of 89.51%. Furthermore, the confusion matrix, recall, and precision for this network can be found in Table 8.8. This best network supports both of the earlier hypotheses, that the addition of an extra dimension containing the second most frequent objects, and the increasing of the horizontal and vertical dimensionality of the input to the network, both improve the performance of the network.

Action	22 Videos			11 Videos		
	Occurrences	Total Duration (seconds)	Average Duration (seconds)	Occurrences	Total Duration (seconds)	Average Duration (seconds)
A	30	4913.56	163.79	11	478.52	43.50
C	40	5629.04	140.73	20	3007.32	150.37
D	104	31572.76	303.58	62	19807.28	319.47
E+F	35	4910.00	140.29	16	2226.60	139.16
I	67	24889.20	371.48	35	12571.96	359.20
J+K+L+M	34	10584.36	311.30	21	4301.56	204.84
P	7	2773.76	396.25	3	778.12	259.37

Table 8.7: This table shows the number of occurrences, total and average durations for the selected actions/action combinations for both the full dataset and half dataset.

Input Dimensions	LSTM Type	Best Combined Structure	Best Combined Accuracy	Best Combined F1	Best Combined Recall	Best Combined Precision	Confusion Matrix	
$(16 \times 16 \times 1)$	LSTM	1 Conv 1 LSTM	0.8914	0.4213	0.3627	0.5024	1661 2919	1645 35782
$(16 \times 16 \times 1)$	BLSTM	1 Conv 2 LSTM	0.8778	0.4159	0.3991	0.4342	1828 2752	2382 35045
$(16 \times 16 \times 2)$	BLSTM	1 Conv 1 LSTM	0.8912	0.4695	0.4417	0.5011	2023 2557	2014 35413
$(32 \times 32 \times 2)$	BLSTM	2 Conv 3 LSTM	0.8951	**0.5013**	0.4836	0.5203	2215 2365	2042 35385

Table 8.8: This table shows the results of the best performing network structures for each input and LSTM type for the multi-class action recognition, using only half of the dataset. The network with input $(32 \times 32 \times 2)$ with two convolutions and three bidirectional LSTMs performed the best, with a 0.5013 F1 score.

8.3.2 Full Data

Again the same methodology was used to find the best network for the full dataset as was used for half dataset and the single class. Contradictory to predictions, the F1 scores dropped for all networks when using the full dataset, with the best network only getting an F1 score of 0.2960. The network that achieved this F1 score also did not use a bidirectional LSTM but a standard LSTM. The best network in this case had an input of $(16 \times 16 \times 1)$ and used a single convolutional layer and two LSTM layers. The other network performances can be seen in Table 8.9.

8.4 Discussion

The best-performing networks for the three experiments (single class, multi-class half dataset, and multi-class full dataset) are discussed. The best two networks both make use of the bidirectional LSTMs and have two convolutional layers. However, the best network

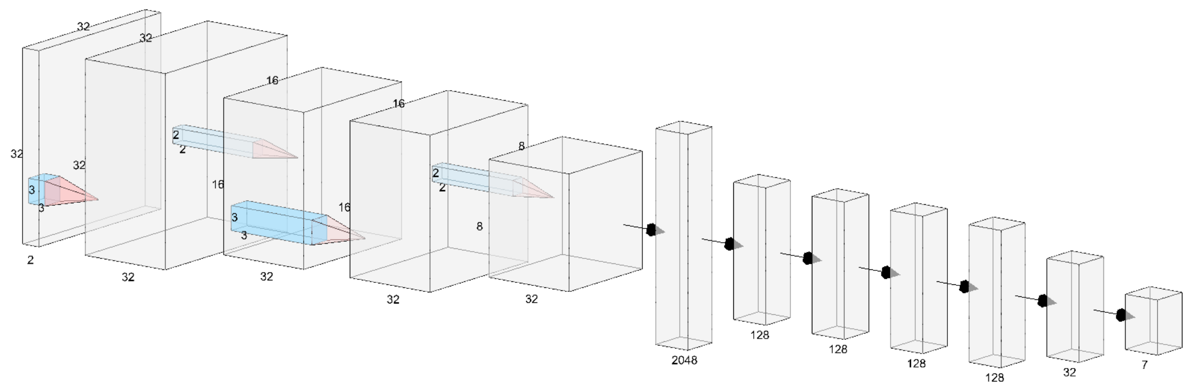


Figure 8.4: This figure shows the network structure for the best performing network and consists of 12 layers (L). From left to right the layers in the network are: (L1) Input of $(32 \times 32 \times 2)$, (L2) 2D Convolution with filter (3×3), (L3) Max pool (2×2), (L4) 2D Convolution with filter (3×3), (L5) Max pool (2×2), (L6) Flatten, (L7) Bi-directional LSTM + Dropout 0.2 + Batch Normalization, (L8) Bi-directional LSTM + Dropout 0.2 + Batch Normalization, (L9) Bi-directional LSTM + Dropout 0.2 + Batch Normalization, (L10) Flatten, (L11) Dense layer of size 32 with ReLU activation + Dropout 0.2, (L12) Dense layer of size 7 with sigmoid activation.
This figure was created using the NN-SVG: Publication-Ready Neural Network Architecture Schematics by LeNail [61].

uses eight times more input features than the other networks, with 2048 input features.

Contradictory to expectations, adding the other half of the data to the dataset used to train and evaluate the networks, reduced the performance of the network. A possible cause of this could be due to artefacts from the semantic segmentation end-to-end networks, since this network is used to predict on all frames of the dataset. If the predictions within it are incorrect, this would mean the input to the action recognition network does not correlate to the ground truth actions. For example, if items such as the pulse oximeter or ECG are not detected and labelled with the segmentation network, then the action recognition inputs would never see these objects being placed and would have to try to learn the location of the hands in regards to which action is being performed.

An evaluation of whether the addition of extra classes improved the predictions, the breakdown of each of the confusion matrices for the best performing network can be found in Table 8.10. It can be seen that the F1 score for action D slightly decreased compared to learning the action on its own. Additionally, there are several actions in the test set that failed to be learnt at all, these were actions $A, E+F$, $J+K+L+M$, and P. Only actions C, D, and I had any true positive predictions. Figure 8.5 displays the relationships between the average and the normalised F1 scores, the number of occurrences, the total duration, and the average duration. This graph shows that F1 score follows the same trend as each of the metrics, except for action P in regards to average duration. This exception is likely due to the low number of occurrences and the low total duration of

Input Dimensions	LSTM Type	Best Combined Structure	Best Combined Accuracy	Best Combined F1	Best Combined Recall	Best Combined Precision	Confusion Matrix	
$(16 \times 16 \times 1)$	LSTM	1 Conv 2 LSTM	0.8559	**0.2960**	0.2388	0.3892	2554	4008
							8142	69639
$(16 \times 16 \times 1)$	BLSTM	1 Conv 3 LSTM	0.8643	0.2773	0.2052	0.4272	2195	2943
							8501	70704
$(16 \times 16 \times 2)$	BLSTM	2 Conv 2 LSTM	0.8587	0.0812	0.0493	0.2314	527	1750
							10169	71879
$(32 \times 32 \times 2)$	BLSTM	1 Conv 1 LSTM	0.8424	0.0526	0.0345	0.1105	369	2969
							10327	70678

Table 8.9: This table shows the results of the best performing network structures for each input and LSTM type for the multi-class action recognition, using the full dataset. The network with an input of $(16 \times 16 \times 1)$ with one convolution and two LSTM performs the best with an F1 score of 0.2960.

the action, thus resulting in the machine not having enough examples to learn the trend. Moreover, the average length of the action is longer than other actions. When a Pearson correlation was performed on the F1 scores, each of the individual metrics were calculated and a high correlation between total duration and average duration was shown. This had a confidence value of 95%, and a number of occurrences fell just shy of the 0.10 significance value (0.669). The values for each result can be seen in Table 8.11.

It can be seen that there was a high correlation between having a high number of occurrences and a large total duration, because as the number of occurrences increases, the total duration will increase. The F1 score is highly correlated to the total duration and number of occurrences, as when there is more training data, the actions will have a higher F1 score. However, the F1 score is not highly correlated to the average duration, because actions $J + K + L + M$ and P have high average durations, but the F1 score remains zero. This is due to the low number of occurrences and low total duration. For a larger dataset, it could be hypothesised that the average duration would play more of a role upon the performance of the action detection. If an action is too short, the network may not be able to encapsulate enough data to recognise it as an activation. At the other extreme, if the action is too long, the network may need to be adjusted to cope with very long term memory necessary to detect the activation.

Due to the predicted ground truths being generated using the previous end-to-end semantic segmentation network, the dataset only consisted of very few occurrences. Because of this, and that the inputs to this action recognition network had a relatively low dimensionality, the network was unable to find any correlations between the inputs and all the actions. There are a few key factors which may have contributed to the uncertainty in the inputs. For instance, the position of the paediatrician's hand is often around the baby's face, which could obscure any equipment they might be holding. Additionally, the baby is not always visible due to being wrapped in blankets. In this case, the input to the

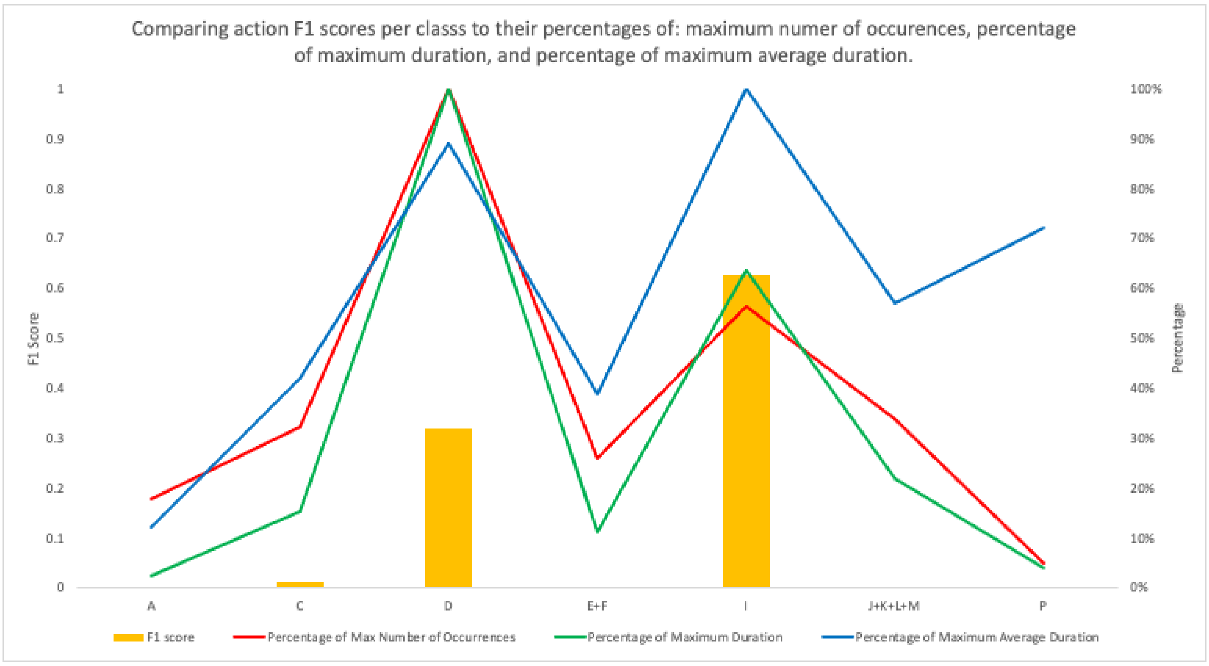


Figure 8.5: This figure shows the relationship between the average F1 score for each action and; the percentage of the maximum action occurrence, the percentage of total duration for the longest action, and the percentage of the maximum average duration of the actions. Firstly, it can be seen that the three metrics follow the same trend per action, except for in the case of action P for average duration. From visual inspection of this graph, the F1 score for each is highly correlated to the metrics, except in regards to action P and the average duration.

Actions	F1	Recall	Precision	Confusion Matrix	
A	0.0000	0.0000	0.0000	0	0
				165	5836
C	0.0123	0.0062	0.5000	1	1
				160	5839
D	0.3180	0.2124	0.6329	350	203
				1298	4150
E+F	0.0000	0.0000	0.0000	0	0
				284	5717
I	0.6277	0.8284	0.5053	1864	1825
				386	1926
J+K+L+M	0.0000	0.0000	0.0000	0	8
				72	5921
P	0.0000	0.0000	0.0000	0	5
				0	5996
Macro Average	0.1369	0.1496	0.2340	*N/A*	
				N/A	
Micro Average D + I	0.5440	0.5680	0.5219	*N/A*	
				N/A	
Micro Average C + D + I	0.5335	0.5457	0.5219	*N/A*	
				N/A	
Micro Average	0.5013	0.4836	0.5203	2215	2042
				2365	35385

Table 8.10: This table shows the breakdown of the performance for the network that was trained on half of the dataset, with the network structure: ($32 \times 32 \times 2$), followed 2 convolutions and 3 bidirectional LSTMs. Only actions D and I have been learnt well by the network.

network would mainly consist of hands/gloves and bedding/towels/blankets. A technique that may improve the results is to increase the input resolution, allowing less information to be lost in the compression.

8.4.1 Comparison of Amount of Training Data

Taking the best performing network from the experiments conducted, the amount of training data was varied to test the effect on the performance of the network. The previously defined process of performing end-to-end semantic segmentation using deep-learned superpixels, followed by feeding the output into a simple RNN network (consisting of a small number of convolutional layers and bidirectional LSTM layers) was used to detect actions. The three best networks from this, in their respective configurations, are shown in Table 8.12. The network that was chosen was the network with ($32 \times 32 \times 2$) input, two convolutional layers, and three bidirectional LSTMs layers, because of its much higher F1 score of 0.5013. Figure 8.6 shows an example of the network's trained model

	Number of Occurrences	Total Duration	Average Duration Duration	Significance Value (0.05)
Pearson Correlation to F1 score	0.660	0.774	0.778	0.754

Table 8.11: This table shows the Pearson correlation between the average F1 scores for the actions and the number of occurrences, total duration, and average duration. Total duration and average duration showed a high correlation with 95% confidence. Number of occurrences showed no correlations, with its Pearson score of just less than the 0.10 significance (0.669) by 0.09.

being used to perform inference on a whole video from the Newborn Resus dataset.

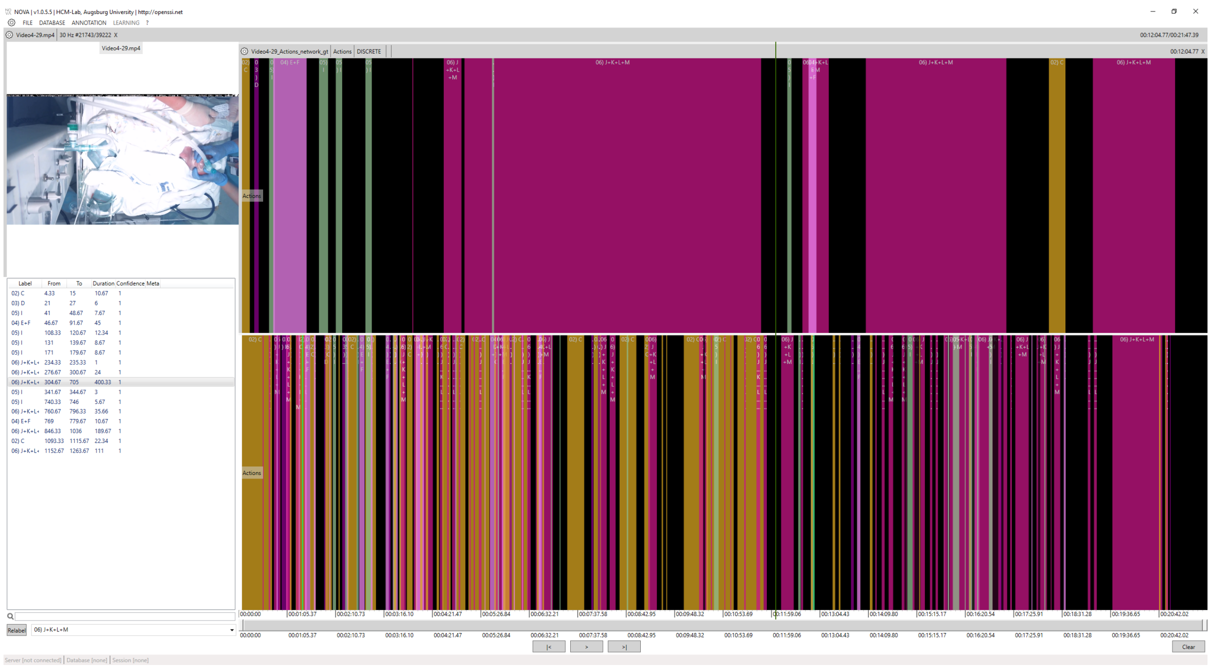
Dataset	Network Structure	F1 Score
Full Dataset Single Action	($16 \times 16 \times 1$) 2 Conv 1 BLSTM	0.3435
Full Dataset Multiple Actions	($16 \times 16 \times 1$) 1 Conv 1 LSTM	0.2960
Half Dataset Multiple Actions	($32 \times 32 \times 2$) 2 Conv 3 BLSTM	**0.5013**

Table 8.12: Comparison of the best performing networks for each dataset and action combination.

The test set for this experiment was kept the same to allow for a fair comparison between networks. The only data changed in this experiment was the percentage of training data used. However, if a straight percentage of the data was taken, this would split some videos across multiple training sets. To combat this, the percentages of the total videos were taken, and the training sets were split according to this. To further validate the experiment, the networks were trained three separate times per training set percentage, and an average was taken. This reduced any poor performances due to the random initialisation of the networks.

When the training set consisted of 20% of the total number of instances, the network performed best with an F1 score of 0.5181, and an accuracy of 90.72%. This was very close to the turning point of the training sets at 60% training data. At 70% training data, the performance started to decrease. This is shown in Figure 8.7 and the performance for the other percentages of training data can be found in Table 8.13.

There are several hypotheses for why this may have happened. Firstly, there is an inherent class unbalance due to certain videos consisting of different procedures. Not all videos consist of the baby receiving the full resuscitation procedure, as some babies are


NOVA | v1.0.5.5 | HCM-Lab, Augsburg University | http://openssi.net
FILE DATABASE ANNOTATION LEARNING ?
Video4-29.mp4
Video4-29_Actions_network_gt
Actions
DISCRETE
06) J+K+L+M
Actions
Relabel
06) J+K+L+M
Clear

Label	From	To	Duration	Confidence	Meta
02) C	4.33	15	10.67	1	
03) D	21	27	6	1	
05) I	41	48.67	7.67	1	
04) E+F	46.67	91.67	45	1	
05) I	108.33	120.67	12.34	1	
05) I	131	139.67	8.67	1	
05) I	171	179.67	8.67	1	
06) J+K+L	234.33	235.33	1	1	
06) J+K+L	276.67	300.67	24	1	
06) J+K+L	304.67	705	400.33	1	
05) I	341.67	344.67	3	1	
05) I	740.33	746	5.67	1	
06) J+K+L	760.67	796.33	35.66	1	
04) E+F	769	779.67	10.67	1	
06) J+K+L	846.33	1036	189.67	1	
02) C	1093.33	1115.67	22.34	1	
06) J+K+L	1152.67	1263.67	111	1	

Figure 8.6: This figure shows the multi-class action predictions by the best performing network in Table 8.12 on a example video from the Newborn Resus dataset.

only checked over and wrapped up to keep warm. With such a small amount of data, this would not give enough examples for a network to be able to generalise for each action.

Another hypothesis is that the data in the final 40% of the training data may not be reliable, because the predictions from the semantic segmentation may not have picked up all of the objects in the videos. If any important objects, such as the hands, are not detected, then the action recognition system would be missing vital information to be able to learn from. This could be the case if the gloves which the paediatric staff are wearing are a different colour than the ones in the training set for the semantic segmentation network.

When looking at the individual action breakdown predictions, Table 8.14, the performance slightly increased from the previous best network, Table 8.10. The F1 score of action I, airway manoeuvre, increased from 0.6277 to 0.6849.

8.5 Conclusion

In conclusion, the methodology of using a $(32 \times 32 \times 2)$ input with multiple convolutions layers followed by bi-directional LSTMS, Figure 8.4, works adequately for a small number of actions trained on a very small amount of data. This achieved an accuracy of 90.72% and a F1 score of 0.5181. This high accuracy and lowers F1 score shows that the network is overfitting to some classes. This overfitting has likely occurred because of the lower occurrences for classes A and P, and low total duration of C, E+F, and J+K+L+M. Both the total duration and number of occurrences are low compared to classes D and I.

Percentage Training Data	Micro Accuracy	Micro F1	Micro Recall	Micro Precision
10%	0.9048	0.5122	**0.4583**	0.5806
20%	**0.9072**	**0.5181**	0.4574	0.5974
30%	0.8988	0.4965	0.4574	0.5429
40%	0.8950	0.4679	0.4234	0.5228
50%	0.8955	0.4808	0.4435	0.5266
60%	0.9072	0.5137	0.4496	**0.5992**
70%	0.9016	0.4592	0.3831	0.5731
80%	0.8954	0.3667	0.2777	0.5397
90%	0.8889	0.3090	0.2282	0.4792
100%	0.8822	0.2361	0.1670	0.4026

Table 8.13: This table shows the performance of the network trained on varying amounts of training data against a fixed test set. The best performing network used only 20% of the training data, but was very closely followed by the network which used 60%. After 60%, the performance started to decrease.

Actions	F1	Recall	Precision
A	0.0000	0.0000	0.0000
C	0.0158	0.0104	0.0342
D	0.3197	0.2186	0.5951
E+F	0.0046	0.0023	0.1032
I	0.6849	0.7699	0.6169
J+K+L+M	0.0000	0.0000	0.0000
P	0.0000	0.0000	0.0000

Table 8.14: This table shows the breakdown performance for the individual actions for the best performance network when trained on 20% of the training data.

To improve the performance upon the actions that the network failed to learn would require addressing the issues mentioned throughout this chapter. Firstly, the data fed to the network must be of high precision. This would require re-annotating the 22 videos, achieved by manually finding the worst predictions for all of the videos, then re-annotating them and including them in the training set for the end-to-end semantic segmentation network. This would need to be repeated until the entire dataset was labelled without any objects being missed. It could take many weeks for this to be done to a high standard. With this higher quality dataset, the action recognition network could be retrained to see if the addition of more data improves the network further.

To combat the issue of lost information during compression of the input to the network, several experiments must be performed. This would include increasing the resolution of the input until all of the most important semantic classes were visible. This would require varying the width and height of the input as well as the number of channels, where each additional channel represents the next most frequent class for an area. For this to be

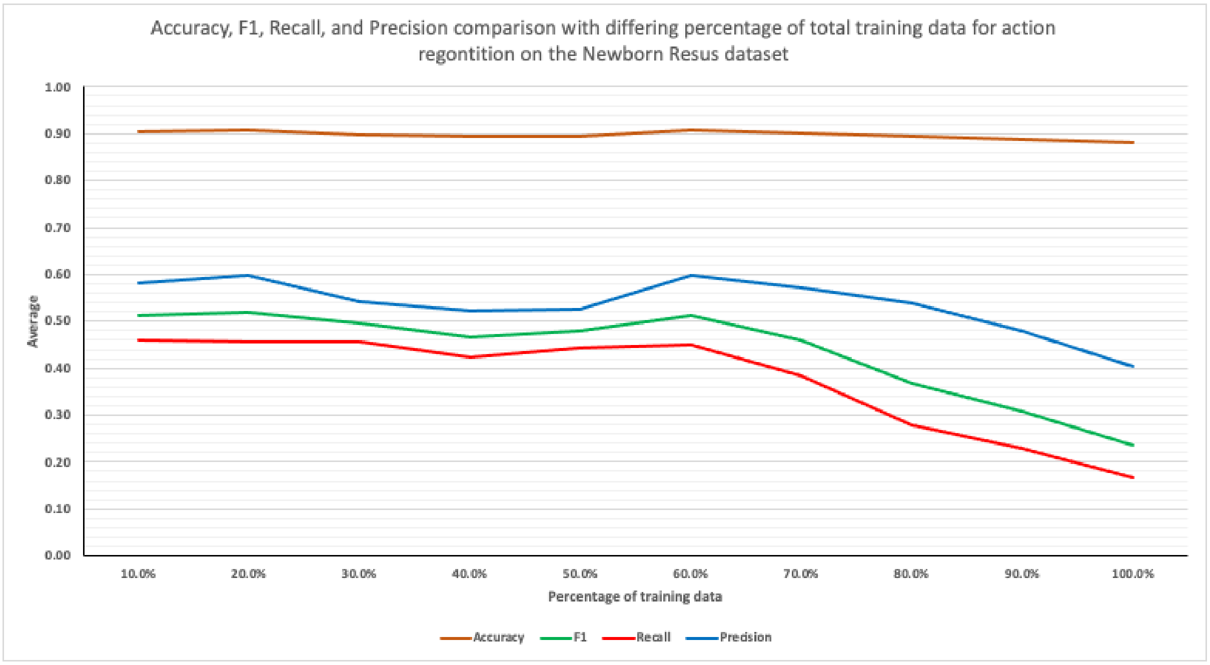


Figure 8.7: Graph comparing the average F1, accuracy, recall, and precision values when varying the percentage of training data given to the network.

feasible it would need to be automated and at each step the action recognition network would be retrained until the most optimal network input was found. A too-large input could drastically increase both training and prediction times, as each instance that the input increases, the number of parameters in the network proportionally increase. If the input dimensionality required was too large for enough information to be present during training, then the training times could be reduced by altering the network structure to accommodate for this. This would be a lengthy process, which was not possible to complete within this thesis due to time constraints.

Given that such small amounts of data have been used to train these networks, it can be confidently assumed that given a well-labelled dataset following strict constrains, such as maintaining the same view and a small number of object in the dataset, that it is indeed possible to get adequate action recognition results on very small domain-specific datasets. The main take away from this is that the quality of the inputs to the network have a larger impact on performance than when using a traditional sized dataset with many more variations. Thus meaning that the input semantic segmentations for the frames of the videos should be of good quality.

Chapter 9

Conclusion

9.1 Research question

This thesis set out to discover if useful results for the task of semantic segmentation and action recognition using deep learning architectures with tiny domain specific dataset is possible. From the evidence given throughout the chapters in this thesis I have concluded that, given a modest amount of instances, it is indeed possible to use complex deep learning architectures with tiny datasets.

Semantic segmentation with tiny domain-specific datasets has been accomplished by making use of an end-to-end network consisting of two hourglass structures, with the first hourglass being pre-trained to predict deep-learned superpixels, by being trained on a large varied dataset. The second hourglass takes the RGB image, and the deep-learned superpixel represented in the novel MCG representation as input. This second hourglass is trained to perform semantic segmentation. Good semantic segmentation results have been achieved with as little as 20 training instances on the Newborn Resus dataset.

A key part of training a dataset is first labelling the dataset. In this thesis, the Image Annotation Tool has been updated to make use of superpixels to aid in annotating images to near pixel-perfect with large improvements to the duration it take to annotate such images.

Action recognition with tiny datasets was achieved by making use of semantic segmentation as a pre-processing set to reduce the number of features fed to the RNN network. In addition to semantic segmentation, a custom pooling was applied to the input to reduce the dimensionality from $(512 \times 512 \times 23)$ for the Newborn Resus dataset, down to $(32 \times 32 \times 2)$, where each channel in the new output represents the N^{th} most frequent class for a section. Using multiple convolutional layers followed by bi-directional LSTM layers the architecture was able to detect two of the seven classes from the Newborn Resus dataset. The main contributing factor for struggling to learn the other five classes was the lack of training data for complex tasks.

Being able to perform train deep learning architectures with a tiny domain-specific dataset and achieve usable predictions is an important step in allowing deep learning to be able to be used by a wider community without the need for such large datasets. The generation of such large datasets for some communities is not feasible due to the expertise and time required to annotate those datasets.

9.2 Recommendations

For semantic segmentation, I would highly recommend a well-balanced dataset with an absolute minimum of 30 instances of each class for both action recognition and semantic segmentation. Ideally, the dataset would contain 50+ instances. Using 50 instances allows for the dataset to be split into 20:10:20 sets for training, validation and testing, respectively. With regard to semantic segmentation and the Newborn Resus dataset, classes with less than 30 total instances failed to learn. There was a similar story told in regards to action recognition, where a large number of classes that had fewer than 30 instances failed to learn.

Including superpixels as an input feature to a network, represented via Multi-channel Connected Graphs (MCGs), has proven useful for small amounts of data, but when increasing the number of samples there is less of an improvement. Using deep-learned superpixels over handcrafted superpixels again showed larger improvements when using small amounts of data, but when larger amounts of data were used, handcrafted superpixels had a minor improvement over deep-learned superpixels.

9.3 Un-answered questions

An unanswered question regarding MCGs is their ability to be represent other pixel relationships for use as input features for CNN architectures, as well as the MCG's ability to be used with different pixel layouts, such as a triangular layout.

9.4 Future Work

In regards to Newborn Resus dataset and its use in development of a system to aid in training of paediatric staff, more data would be required to perform adequate action recognition for medical training. A larger amount of training data that is also well-labelled could be used to train a modern fast architecture, such as the adapted U-nets [95]. These adapted U-nets could replace the stacked hourglass modules in Chapters 2.4 for a faster inference time. Then, in an end-to-end manner, multiple RNN layers could be added to the end of the adapted stacked U-nets for a single model. The use of the down-sampling technique discussed in Chapter 8 would also be included in the network to reduce the amount of variables for the RNN layers.

The gLitter project was put on hold due to financial and logistical reasons. To enable this project to become active again, much more data collection would be required. With larger amounts of data, the super litter class could be split into subclasses which would allow for better sorting of the litter during collection with the rover. The video footage gathered by humans is a good place to start, but if the detection system is needed to be embedded into the drone and rover, additional data from these sources would be needed to fine-tune the final models to ensure that the network can handle the variations in distance from the subject, and motion caused by the autonomous vehicles. The major thing currently missing from the gLitter project is what would be classed as ‘not litter’, and different terrains that the vehicles would encounter, such as treetops for the drone. Not being able to identify the difference between a bottle that belongs to someone and is not litter, and a bottle that has been left on the ground as litter, is one of the challenges that will need to be solved during prototyping.

Taking deep-learned superpixels to the next level by training a network with many different handcrafted superpixel algorithms, whilst also varying the superpixel algorithm’s parameters to create differently shaped and sized superpixels, and making use of a very large image dataset such as ImageNet, could possibly result in a better over-segmenting method. Due to the large number of superpixels that would be created by the varying algorithms, a new problem would arise on how to represent all of the resulting superpixel masks. A possibility that may work well, would be to use the MCG representations, and instead of having multiple graphs (one per superpixel algorithm creation), a single combined MCG could be created by summing all individual masks together, and then normalising between zero and one. This new MCG mask would, in turn, highlight where multiple different algorithms create boundaries between their respective superpixels.

Being able to reconstruct superpixels from the predicted MCG masks would be both beneficial to the above-stated improvement to deep-learned superpixels, and to the superpixels created in this thesis. By being able to reconstruct the individual superpixels from predicted MCGs would allow for them to be used outside of the proposed end-to-end network. However, this remains an unsolved problem; how to enforce the ability to reconstruct the superpixels during training. I believe this would require a new loss function that could determine how many nodes in the MCGs need to be updated to be able to reconstruct to whole superpixels or super-regions. This new loss function could have a parameter to enforce the size of the predicted superpixels, which would make them tunable per domain.

References

[1] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk. Slic superpixels. page 15, 2010. URL `http://infoscience.epfl.ch/record/149300`.

[2] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk. Slic superpixels compared to state-of-the-art superpixel methods. *IEEE transactions on pattern analysis and machine intelligence*, 34(11):2274–2282, 2012.

[3] V. Badrinarayanan, A. Kendall, and R. Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017.

[4] M. Bai and R. Urtasun. Deep watershed transform for instance segmentation. *arXiv preprint arXiv:1611.08303*, 2016.

[5] T. Baur, G. Mehlmann, I. Damian, F. Lingenfelser, J. Wagner, B. Lugrin, E. André, and P. Gebhard. Context-aware automated analysis and annotation of social human-agent interactions. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 5 (2):11, 2015.

[6] M. Berman and M. B. Blaschko. Optimization of the Jaccard index for image segmentation with the Lovász hinge. *ArXiv e-prints*, May 2017.

[7] M. Berman, A. Rannen Triki, and M. B. Blaschko. The lovász-softmax loss: A tractable surrogate for the optimization of the intersection-over-union measure in neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4413–4421, 2018.

[8] A. H. Bhatti, A. Rahman, and A. A. Butt. Video segmentation using spectral clustering on superpixels. In *Image Processing (ICIP), 2016 IEEE International Conference on*, pages 869–873. IEEE, 2016.

[9] C. M. Bishop. *Pattern recognition and machine learning.* springer, 2006.

[10] A. Blake, P. Kohli, and C. Rother. *Markov random fields for vision and image processing.* Mit Press, 2011.

[11] A. Bréhéret. Pixel Annotation Tool. `https://github.com/abreheret/PixelAnnotationTool`, 2017.

[12] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah. Signature verification using a" siamese" time delay neural network. In *Advances in neural information processing systems*, pages 737–744, 1994.

[13] G. J. Brostow, J. Fauqueur, and R. Cipolla. Semantic object classes in video: A high-definition ground truth database. *Pattern Recognition Letters*, 30(2):88–97, 2009.

[14] F. Caba Heilbron, V. Escorcia, B. Ghanem, and J. Carlos Niebles. Activitynet: A large-scale video benchmark for human activity understanding. In *Proceedings of the ieee conference on computer vision and pattern recognition*, pages 961–970, 2015.

[15] J. Carreira and A. Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6299–6308, 2017.

[16] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. *arXiv preprint arXiv:1412.7062*, 2014.

[17] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2017.

[18] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017.

[19] M. Cordts, M. Omran, S. Ramos, T. Scharwächter, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. The cityscapes dataset. In *CVPR Workshop on The Future of Datasets in Vision*, 2015.

[20] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[21] L. Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[22] M. Dong and Z. Sun. On human machine cooperative learning control. In *Intelligent Control. 2003 IEEE International Symposium on*, pages 81–86. IEEE, 2003.

[23] Dräeger. Resuscitaire, 2021. URL `https://www.draeger.com/en_uk/Products/Resuscitaire`.

[24] C. Du and S. Gao. Image segmentation-based multi-focus image fusion through multi-scale convolutional neural network. *IEEE Access*, 5:15750–15761, 2017.

[25] Y. Du, C. Yuan, B. Li, L. Zhao, Y. Li, and W. Hu. Interaction-aware spatio-temporal pyramid attention networks for action classification. In *Proceedings of the European conference on computer vision (ECCV)*, pages 373–389, 2018.

[26] L. Duan, C. Huang, G. Chen, L. Xiong, Q. Liu, and W. Yang. Determination of rice panicle numbers during heading by multi-angle imaging. *The Crop Journal*, 3 (3):211–219, 2015.

[27] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

[28] A. Dutta and A. Zisserman. The VIA annotation software for images, audio and video. In *Proceedings of the 27th ACM International Conference on Multimedia*, MM '19, New York, NY, USA, 2019. ACM. ISBN 978-1-4503-6889-6/19/10. doi:10.1145/3343031.3350535. URL `https://doi.org/10.1145/3343031.3350535`.

[29] A. Dutta, A. Gupta, and A. Zissermann. VGG image annotator (VIA). http://www.robots.ox.ac.uk/ vgg/software/via/, 2016. Version: X.Y.Z, Accessed: INSERT$_DATE_HERE$.

[30] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html, 2012.

[31] L. Fei-Fei, R. Fergus, and P. Perona. One-shot learning of object categories. *IEEE transactions on pattern analysis and machine intelligence*, 28(4):594–611, 2006.

[32] C. Feichtenhofer, A. Pinz, and R. P. Wildes. Spatiotemporal multiplier networks for video action recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4768–4777, 2017.

[33] C. Feichtenhofer, H. Fan, J. Malik, and K. He. Slowfast networks for video recognition. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6202–6211, 2019.

[34] D. A. Forsyth and J. Ponce. *Computer vision: a modern approach.* Prentice Hall Professional Technical Reference, 2002.

[35] J. Fu, J. Liu, Y. Wang, J. Zhou, C. Wang, and H. Lu. Stacked deconvolutional network for semantic segmentation. *IEEE Transactions on Image Processing*, 2019.

[36] F. Galasso, N. Nagaraja, T. Cardenas, T. Brox, and B.Schiele. A unified video segmentation benchmark: Annotation, metrics and analysis. In *IEEE International Conference on Computer Vision (ICCV)*, Dec 2013. URL `http://lmb.informatik.uni-freiburg.de//Publications/2013/NB13`.

[37] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[38] S. N. Gowda, M. Rohrbach, and L. Sevilla-Lara. Smart frame selection for action recognition. *arXiv preprint arXiv:2012.10671*, 2020.

[39] A. Graves and J. Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural networks*, 18(5-6):602–610, 2005.

[40] A. Graves, N. Jaitly, and A.-r. Mohamed. Hybrid speech recognition with deep bidirectional lstm. In *2013 IEEE workshop on automatic speech recognition and understanding*, pages 273–278. IEEE, 2013.

[41] M. Grundmann, V. Kwatra, M. Han, and I. Essa. Efficient hierarchical graph based video segmentation. *IEEE CVPR*, 2010.

[42] S. Gu, J. Wang, L. Pan, S. Cheng, Z. Ma, and M. Xie. Figure/ground video segmentation via low-rank sparse learning. In *Image Processing (ICIP), 2016 IEEE International Conference on*, pages 864–868. IEEE, 2016.

[43] B. Hariharan, P. Arbeláez, R. Girshick, and J. Malik. Simultaneous detection and segmentation. In *European Conference on Computer Vision*, pages 297–312. Springer, 2014.

[44] D.-C. He and L. Wang. Texture unit, texture spectrum, and texture analysis. *IEEE transactions on Geoscience and Remote Sensing*, 28(4):509–512, 1990.

[45] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[46] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[47] C. Henry, L. Shipley, C. Ward, S. Mirahmadi, C. Liu, S. Morgan, J. Crowe, J. Carpenter, B. Hayes-Gill, and D. Sharkey. Accurate neonatal heart rate monitoring using a new wireless, cap mounted device. *Acta Paediatrica*, 110(1):72–78, 2021.

[48] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

[49] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[50] S. Hussain, S. M. Anwar, and M. Majid. Brain tumor segmentation using cascaded deep convolutional neural network. In *Engineering in Medicine and Biology Society (EMBC), 2017 39th Annual International Conference of the IEEE*, pages 1998–2001. IEEE, 2017.

[51] S. Hussain Raza, M. Grundmann, and I. Essa. Geometric context from videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3081–3088, 2013.

[52] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[53] J.-H. Jeong, K.-H. Shim, D.-J. Kim, and S.-W. Lee. Brain-controlled robotic arm system based on multi-directional cnn-bilstm network using eeg signals. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 28(5):1226–1238, 2020.

[54] W. Kay, J. Carreira, K. Simonyan, B. Zhang, C. Hillier, S. Vijayanarasimhan, F. Viola, T. Green, T. Back, P. Natsev, et al. The kinetics human action video dataset. *arXiv preprint arXiv:1705.06950*, 2017.

[55] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[56] M. A. Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE journal*, 37(2):233–243, 1991.

[57] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre. Hmdb: a large video database for human motion recognition. In *2011 International conference on computer vision*, pages 2556–2563. IEEE, 2011.

[58] S. Kwak, S. Hong, B. Han, et al. Weakly supervised semantic segmentation using superpixel pooling network. In *AAAI*, volume 1, page 2, 2017.

[59] H. Larochelle, D. Erhan, and Y. Bengio. Zero-data learning of new tasks. In *AAAI*, volume 1, page 3, 2008.

[60] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[61] A. LeNail. Nn-svg: Publication-ready neural network architecture schematics. *Journal of Open Source Software*, 4(33):747, 2019. doi:10.21105/joss.00747. URL `https://doi.org/10.21105/joss.00747`.

[62] F. Li, T. Kim, A. Humayun, D. Tsai, and J. M. Rehg. Video segmentation by tracking many figure-ground segments. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 2192–2199. IEEE, 2013.

[63] G. Lin, A. Milan, C. Shen, and I. Reid. Refinenet: Multi-path refinement networks for high-resolution semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1925–1934, 2017.

[64] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[65] M.-Y. Liu, O. Tuzel, S. Ramalingam, and R. Chellappa. Entropy rate superpixel segmentation. In *CVPR 2011*, pages 2097–2104. IEEE, 2011.

[66] Z. Liu, X. Li, P. Luo, C.-C. Loy, and X. Tang. Semantic image segmentation via deep parsing network. In *Proceedings of the IEEE international conference on computer vision*, pages 1377–1385, 2015.

[67] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015.

[68] H. Lu, Z. Cao, Y. Xiao, Y. Li, and Y. Zhu. Region-based colour modelling for joint crop and maize tassel segmentation. *Biosystems Engineering*, 147:139–150, 2016.

[69] Y. Lu, X. Bai, L. Shapiro, and J. Wang. Coherent parametric contours for interactive video object segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 642–650, 2016.

[70] I. Luengo, M. Basham, and A. P. French. Smurfs: superpixels from multi-scale refinement of super-regions. 2016.

[71] B. Marcotegui and F. Meyer. Bottom-up segmentation of image sequences for coding. In *Annales des télécommunications*, volume 52, pages 397–407. Springer, 1997.

[72] Ø. Meinich-Bache, S. L. Austnes, K. Engan, I. Austvoll, T. Eftestøl, H. Myklebust, S. Kusulla, H. Kidanto, and H. Ersdal. Activity recognition from newborn resuscitation videos. *IEEE Journal of Biomedical and Health Informatics*, 2020.

[73] R. Mester and U. Franke. Statistical model based image segmentation using region growing, contour relaxation and classification. In *SPIE Symposium on Visual Communications and Image Processing*, pages 616–624, 1988.

[74] F. Meyer. Color image segmentation. In *1992 international conference on image processing and its applications*, pages 303–306. IET, 1992.

[75] A. Myronenko and A. Hatamizadeh. Robust semantic segmentation of brain tumor regions from 3d mris. In *International MICCAI Brainlesion Workshop*, pages 82–89. Springer, 2019.

[76] U. National Health Service (NHS). Hands-only cardiopulmonary resuscitation (cpr) and cpr with rescue breaths, 2018. URL `https://www.nhs.uk/conditions/first-aid/cpr/`.

[77] A. Newell, K. Yang, and J. Deng. Stacked hourglass networks for human pose estimation. In *European Conference on Computer Vision*, pages 483–499. Springer, 2016.

[78] H. Noh, S. Hong, and B. Han. Learning deconvolution network for semantic segmentation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1520–1528, 2015.

[79] N. Otsu. A threshold selection method from gray-level histograms. *IEEE transactions on systems, man, and cybernetics*, 9(1):62–66, 1979.

[80] M. Palatucci, D. Pomerleau, G. E. Hinton, and T. M. Mitchell. Zero-shot learning with semantic output codes. In *Advances in neural information processing systems*, pages 1410–1418, 2009.

[81] K. Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11): 559–572, 1901.

[82] J. M. Perlman, J. Wyllie, J. Kattwinkel, M. H. Wyckoff, K. Aziz, R. Guinsburg, H.-S. Kim, H. G. Liley, L. Mildenhall, W. M. Simon, et al. Part 7: neonatal resuscitation: 2015

international consensus on cardiopulmonary resuscitation and emergency cardiovascular care science with treatment recommendations. *Circulation*, 132(16_suppl_1):S204–S241, 2015.

[83] J. Pont-Tuset, F. Perazzi, S. Caelles, P. Arbeláez, A. Sorkine-Hornung, and L. Van Gool. The 2017 davis challenge on video object segmentation. *arXiv preprint arXiv:1704.00675*, 2017.

[84] A. Ray, S. Rajeswar, and S. Chaudhury. Text recognition using deep blstm networks. In *2015 eighth international conference on advances in pattern recognition (ICAPR)*, pages 1–6. IEEE, 2015.

[85] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[86] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi:10.1007/s11263-015-0816-y.

[87] K. Saarinen. Color image segmentation by a watershed algorithm and region adjacency graph processing. In *Proceedings of 1st International Conference on Image Processing*, volume 3, pages 1021–1025. IEEE, 1994.

[88] M. Schuster and K. K. Paliwal. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681, 1997.

[89] E. Shelhamer, J. Long, and T. Darrell. Fully convolutional networks for semantic segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(4):640–651, 2017.

[90] T. J. Smith. Image annotation tool, 2018. URL `https://github.com/ThomasJamesSmith/ImageAnnotation-Tool`.

[91] K. Soomro, A. R. Zamir, and M. Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*, 2012.

[92] P. Sundberg, T. Brox, M. Maire, P. Arbeláez, and J. Malik. Occlusion boundary detection and figure/ground assignment from optical flow. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 2233–2240. IEEE, 2011.

[93] W. Tang, Y. Zhang, D. Zhang, W. Yang, and M. Li. Corn tassel detection based on image processing. In *2012 International Workshop on Image Processing and Optical Engineering*, volume 8335, page 83350J. International Society for Optics and Photonics, 2011.

[94] T. Tieleman and G. Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4 (2):26–31, 2012.

[95] A. Tkachenka, G. Karpiak, A. Vakunov, Y. Kartynnik, A. Ablavatski, V. Bazarevsky, and S. Pisarchyk. Real-time hair segmentation and recoloring on mobile gpus. *arXiv preprint arXiv:1907.06740*, 2019.

[96] J. J. Tompson, A. Jain, Y. LeCun, and C. Bregler. Joint training of a convolutional network and a graphical model for human pose estimation. In *Advances in neural information processing systems*, pages 1799–1807, 2014.

[97] M. T. Torres, M. F. Valstar, C. Henry, C. Ward, and D. Sharkey. Small sample deep learning for newborn gestational age estimation. In *2017 12th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2017)*, pages 79–86. IEEE, 2017.

[98] M. T. Torres, M. Valstar, C. Henry, C. Ward, and D. Sharkey. Postnatal gestational age estimation of newborns using small sample deep learning. *Image and Vision Computing*, 83:87–99, 2019.

[99] M. Treml, J. Arjona-Medina, T. Unterthiner, R. Durgesh, F. Friedmann, P. Schuberth, A. Mayr, M. Heusel, M. Hofmarcher, M. Widrich, et al. Speeding up semantic segmentation for autonomous driving. In *MLITS, NIPS Workshop*, volume 2, 2016.

[100] M. Van den Bergh, X. Boix, G. Roig, B. de Capitani, and L. Van Gool. Seeds: Superpixels extracted via energy-driven sampling. In *European conference on computer vision*, pages 13–26. Springer, 2012.

[101] T. Verelst, M. Blaschko, and M. Berman. Generating superpixels using deep image representations. *arXiv preprint arXiv:1903.04586*, 2019.

[102] H. Vu, K. Engan, T. Eftestøl, A. Katsaggelos, S. Jatosh, S. Kusulla, E. Mduma, H. Kidanto, and H. Ersdal. Automatic classification of resuscitation activities on birth-asphyxiated newborns using acceleration and ecg signals. *Biomedical Signal Processing and Control*, 36:20–26, 2017.

[103] L. Wang and D.-C. He. Texture classification using texture spectrum. *Pattern Recognition*, 23(8):905–910, 1990.

[104] S. Wolf, L. Schott, U. Köthe, and F. Hamprecht. Learned watershed: End-to-end learning of seeded segmentation. *arXiv preprint arXiv:1704.02249*, 2017.

[105] X. Xiong, L. Duan, L. Liu, H. Tu, P. Yang, D. Wu, G. Chen, L. Xiong, W. Yang, and Q. Liu. Panicle-seg: a robust image segmentation method for rice panicles in the field based on deep learning and superpixel optimization. *Plant Methods*, 13(1):104, 2017.

[106] C. Yang, Y. Xu, J. Shi, B. Dai, and B. Zhou. Temporal pyramid network for action recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 591–600, 2020.

[107] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.

[108] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia. Pyramid scene parsing network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2881–2890, 2017.

[109] W. Zhao, Y. Fu, X. Wei, and H. Wang. An improved image semantic segmentation method based on superpixels and conditional random fields. *Applied Sciences*, 8(5):837, 2018.

[110] F. Zhong, X. Qin, Q. Peng, and X. Meng. Discontinuity-aware video object cutout. *ACM Trans. Graph. (SIGGRAPH Asia 2012*, 31(6):175:1–175:10, 2012. ISSN 0730-0301.

[111] X. Zhu. Semi-supervised learning literature survey. *Computer Science, University of Wisconsin-Madison*, 2(3):4, 2006.

Appendix A

Deep Learned Superpixels

A.1 Multi-Channel Connected Graphs

A.2 End-to-End Litter

The following figures: A.1, A.2, and A.3 display the outputs of the best epochs for each of the pipelines tested.

```
import numpy

def combineMCG(up, down, left, right):
    # MCG inputs of (image height, image width) containing 1s & 0s
    n_rows, n_cols = up.shape()
    output = numpy.zeros((n_rows, n_cols))-1
    output = output.astype(int)
    output[0][0]=0
    next_id = 1

    for row_i in range(n_rows):
        for col_i in range(n_cols):
            # (MCG direction map, row index, col index,
            #  within output array, name)
            for direction in [(right, 0, 1, col_i<n_cols-2, 'right'),
                              (down,  1, 0, row_i<n_rows-2, 'down'),
                              (up,   -1, 0, row_i>0,'up'),
                              (left,  0,-1, col_i>0,'left')]:

                # Check if MCG direction map is 1
                if direction[0][row_i][col_i] == 1:
                    # Check if not already assigned
                    o_r = row_i+direction[1]
                    o_c = col_i+direction[2]
                    if output[o_r][o_c]] == -1:
                        output[o_r][o_c] = output[row_i][col_i]
                    # Check that previous ID's are correct
                    elif not output[o_r][o_c] == output[row_i][col_i]:
                        output[row_i][col_i] = output[o_r][o_c]
                # Check if indices within output dimensions
                elif direction[3]:
                    # Check if not already assigned
                    if output[o_r][o_c] == -1:
                        output[o_r][o_c] = next_id
                        next_id += 1

    # Ensure that IDs are indexed once after another
    uniques = list(numpy.unique(output))
    uniques.sort()
    min_num = 0
    for u in uniques:
        if not min_num == u:
            output[output==u]=min_num
        min_num += 1

    return output
```

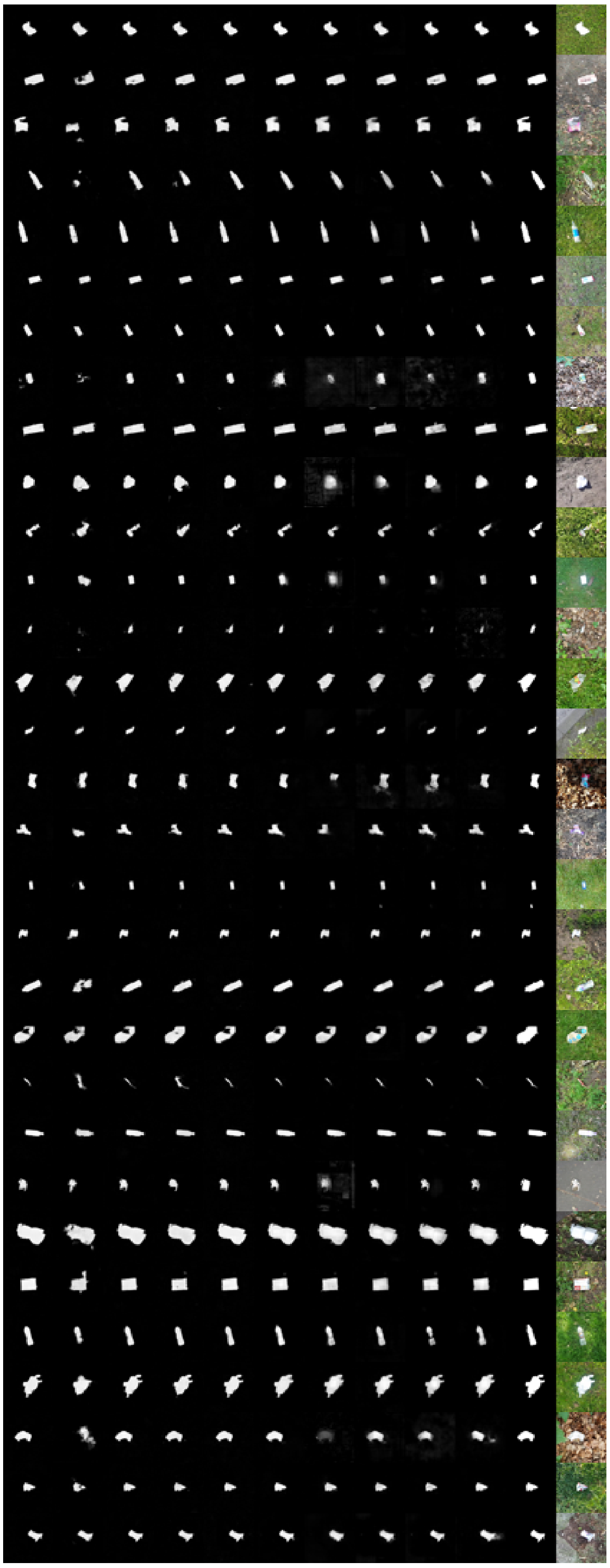


Figure A.1: 512 test data predictions, images 1-31. From left to right: RGB, MCG(SLIC), MCG(SLIC)+RGB, MCG(DLSP), MCG(DLSP)+RGB, End-to-end, No Pre-train, No Front, No Back, Dual Losses, ground truth, and RGB input.

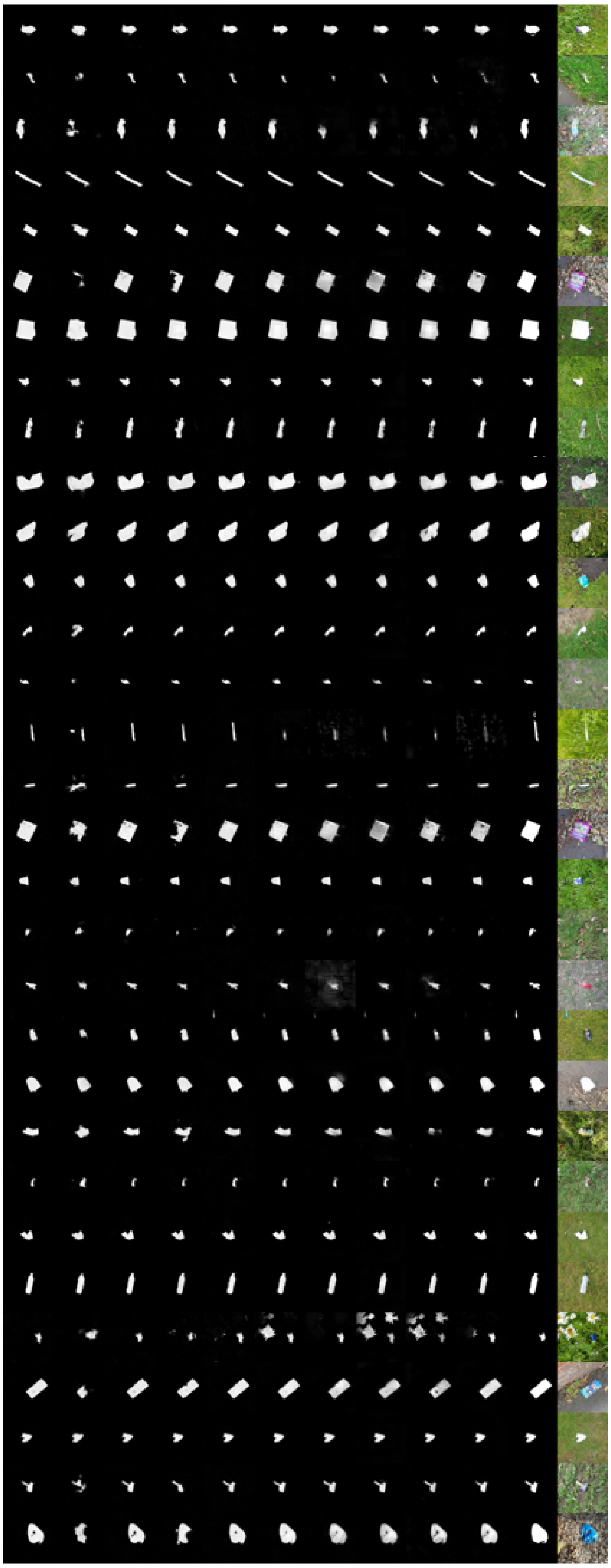


Figure A.2: 512 test data predictions, images 32-62. From left to right: RGB, MCG(SLIC), MCG(SLIC)+RGB, MCG(DLSP), MCG(DLSP)+RGB, End-to-end, No Pre-train, No Front, No Back, Dual Losses, ground truth, and RGB input.

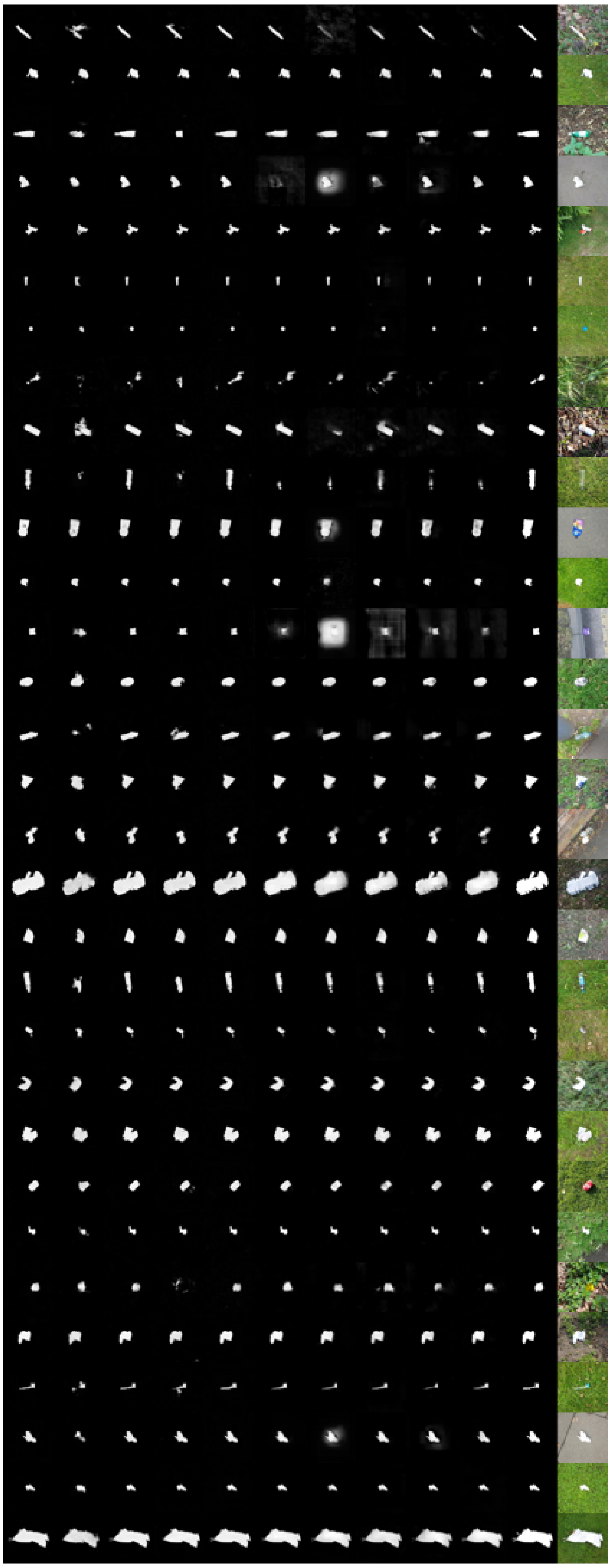


Figure A.3: 512 test data predictions, images 63-93. From left to right: RGB, MCG(SLIC), MCG(SLIC)+RGB, MCG(DLSP), MCG(DLSP)+RGB, End-to-end, No Pre-train, No Front, No Back, Dual Losses, ground truth, and RGB input.

Appendix B

Action Detection

B.1 Actions Examples

The following 19 figures: B.1 - B.19 display example images of the corresponding actions in 8.

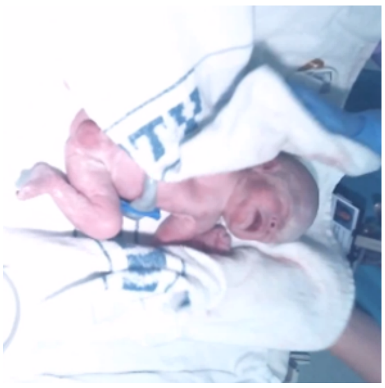


Figure B.1: Example image for Action: A) Dried with towel.

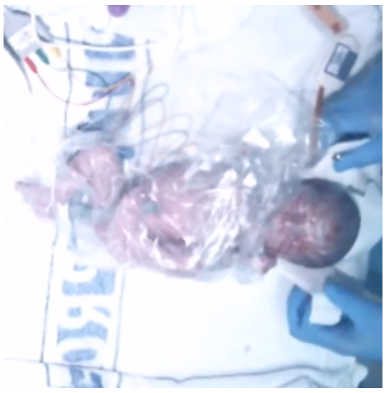


Figure B.2: Example image for Action: B) Dried with towel.

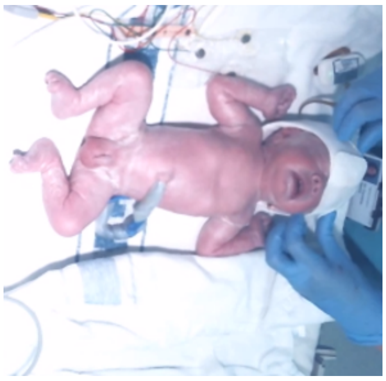


Figure B.3: Example image for Action: C) Cap placed on head.

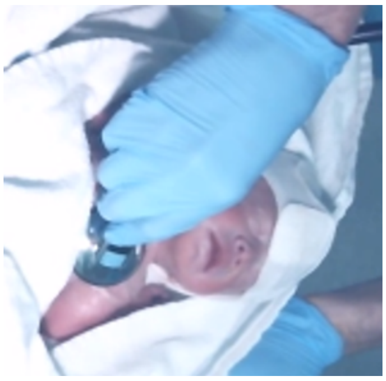


Figure B.4: Example image for Action: D) Heart rate assessed stethoscope.

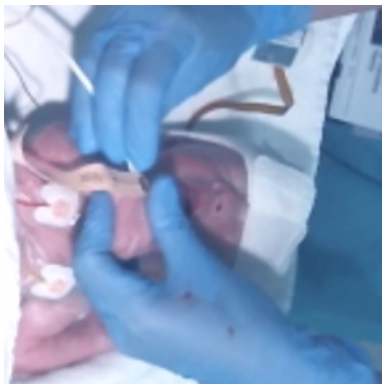


Figure B.5: Example image for Action: E) Attachment of pulse oximeter.

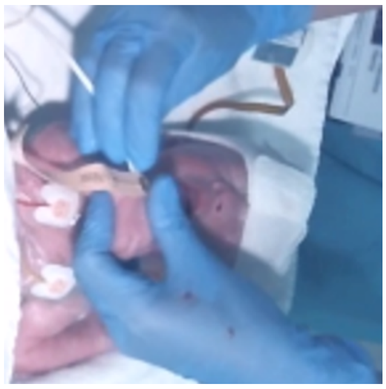


Figure B.6: Example image for Action: F) Pule oximeter adjusted.

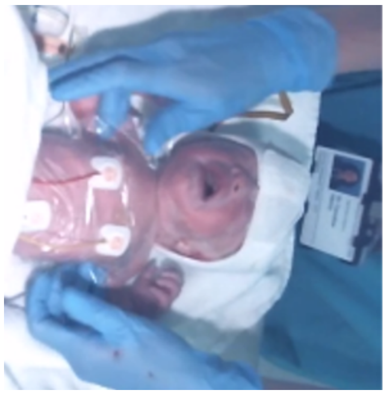


Figure B.7: Example image for Action: G) Place ECG on chest.

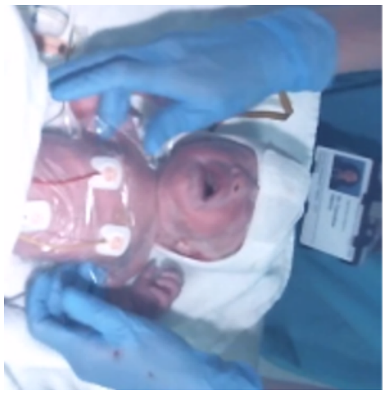


Figure B.8: Example image for Action: H) ECG adjusted.

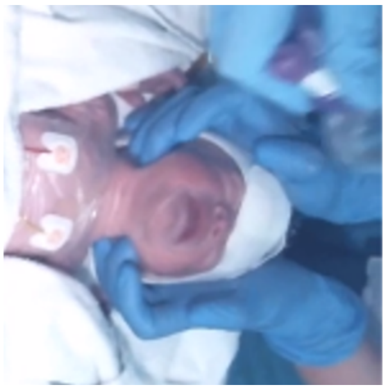


Figure B.9: Example image for Action: I) Airway manoeuvre.

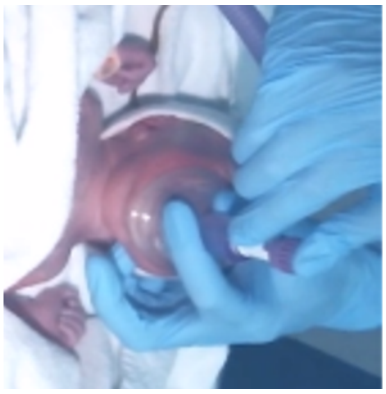


Figure B.10: Example image for Action: J) Inflation breaths given incorrect ratio.

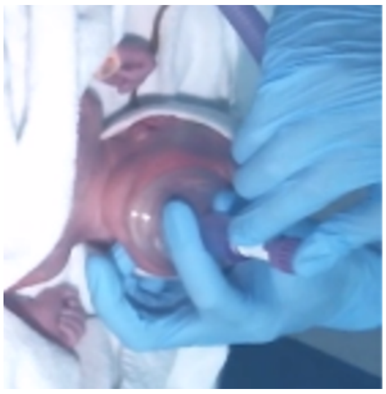


Figure B.11: Example image for Action: K) Provide five inflation breaths lasting three seconds.

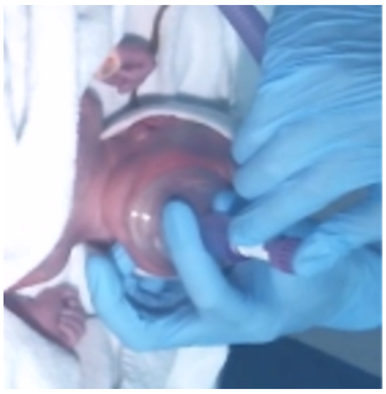


Figure B.12: Example image for Action: L) Ventilation breaths given incorrect ratio.

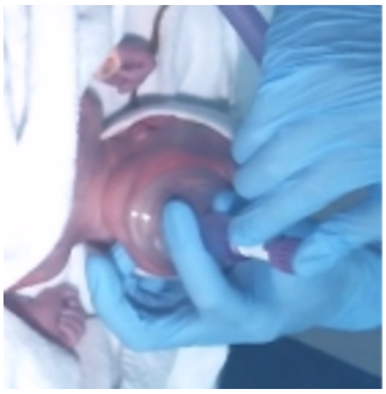


Figure B.13: Example image for Action: M) Ventilation breaths given for 30s and 1s each.

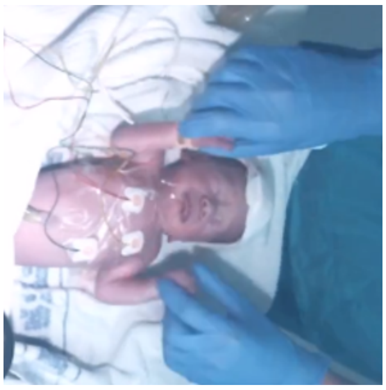


Figure B.14: Example image for Action: N) Stimulation the infant

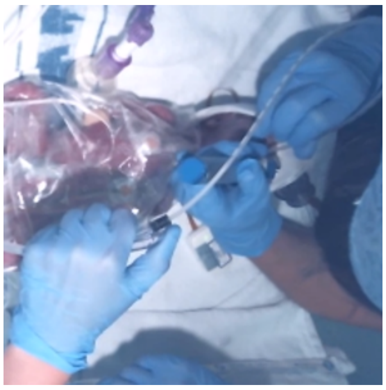


Figure B.15: Example image for Action: O) Cleared away suctioning.

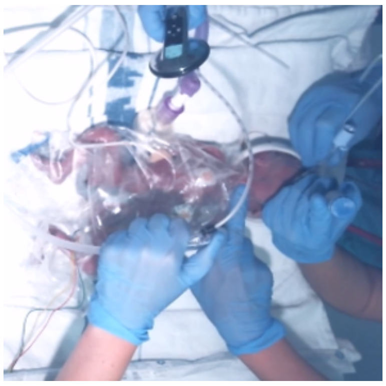


Figure B.16: Example image for Action: P) Incubation attempt.

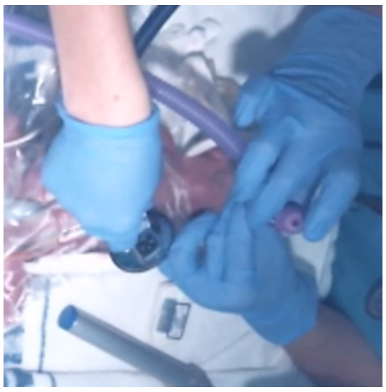


Figure B.17: Example image for Action: Q) Attach mask on after incubation.

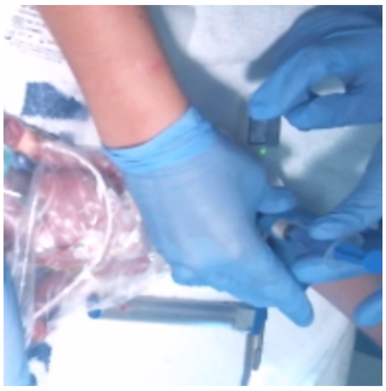


Figure B.18: Example image for Action: R) Administer suffoctant.

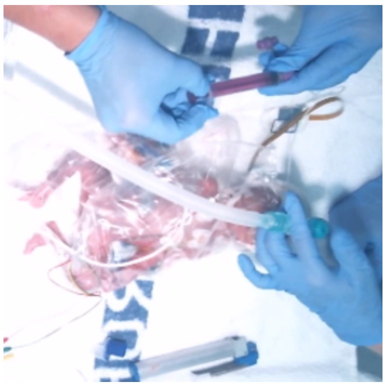


Figure B.19: Example image for Action: S) Insert NGT (feeding tube).

B.2 Single Action

Configuration	# Epochs to Converge	Loss	Accuracy	F1	Recall	Precision	Confusion Matrix	
1×Conv 1×LSTM	17	0.5717	0.7853	0.2118	0.1808	0.2557	113	329
							512	1817
1×Conv 2×LSTM	20	0.6957	0.8073	0.2170	0.1776	0.2789	111	287
							514	1859
1×Conv 3×LSTM	20	0.5045	0.8032	0.0791	0.0560	0.1346	35	225
							590	1921
2×Conv 1×LSTM	20	0.4546	0.7819	**0.2717**	0.2528	0.2937	158	380
							467	1766
2×Conv 2×LSTM	22	0.5364	0.8257	0.1161	0.0768	0.2376	48	154
							577	1992
2×Conv 3×LSTM	23	0.4988	0.8445	0.0062	0.0032	0.0833	2	22
							623	2124
3×Conv 1×LSTM	22	0.4541	0.7823	0.1342	0.1120	0.1675	70	348
							555	1798
3×Conv 2×LSTM	27	0.4748	0.7864	0.1440	0.1200	0.1799	75	342
							550	1804
3×Conv 3×LSTM	17	**0.4515**	**0.8494**	0.0000	0.0000	0.0000	0	1
							625	2145

Table B.1: Single action detection with an input 16x16x1 with a patience of 15 epoch for validation. Using a range of convolutions and LSTMs.
training - 1=41% 0=59% 1=4592 0=6493 total=11085
testing - 1=%23 0=77% 1=625 0=2146 total=2771

Configuration	# Epochs to Converge	Loss	Accuracy	F1	Recall	Precision	Confusion Matrix	
1×Conv 1×BLSTM	20	0.7315	0.8068	0.2063	0.1664	0.2715	104	279
							521	1867
1×Conv 2×BLSTM	19	0.5080	0.8380	0.0506	0.0288	0.2093	18	68
							607	2078
1×Conv 3×BLSTM	17	**0.4150**	**0.8507**	0.0251	0.0128	0.6667	8	4
							617	2142
2×Conv 1×BLSTM	19	0.5930	0.6556	**0.3435**	0.5872	0.2427	367	1145
							285	1001
2×Conv 2×BLSTM	21	0.5178	0.8243	0.2004	0.1440	0.3297	90	183
							535	1963
2×Conv 3×BLSTM	21	0.4625	0.8477	0.0335	0.0176	0.3548	11	20
							614	2126
3×Conv 1×BLSTM	23	0.4173	0.4173	0.0536	0.0368	0.0987	23	210
							602	1936
3×Conv 2×BLSTM	21	0.4509	0.8181	0.0237	0.0144	0.0677	9	124
							616	2022
3×Conv 3×BLSTM	17	0.4651	0.8481	0.0000	0.0000	0.0000	0	6
							625	2140

Table B.2: Single action detection with an input 16x16x1 with a patience of 15 epoch for validation. Using a range of convolutions and bidirectional LSTMs.
training - 1=41% 0=59% 1=4592 0=6493 total=11085
testing - 1=%23 0=77% 1=625 0=2146 total=2771

Configuration	# Epochs to Converge	Loss	Accuracy	F1	Recall	Precision	Confusion Matrix	
1×Conv 1×BLSTM	26	0.6537	0.8389	0.0000	0.0000	0.0000	0 625	5 2141
1×Conv 2×BLSTM	26	0.4945	0.8453	0.0123	0.0064	0.1538	4 621	22 2124
1×Conv 3×BLSTM	19	**0.4720**	**0.8458**	0.0000	0.0000	0.0000	0 625	15 2131
2×Conv 1×BLSTM	19	0.5383	0.8099	0.0643	0.0432	0.1256	27 598	188 1958
2×Conv 2×BLSTM	20	0.5697	0.8315	0.0641	0.0384	0.1935	24 601	100 2046
2×Conv 3×BLSTM	19	0.5639	0.8457	0.0031	0.0016	0.0556	1 624	17 2129
3×Conv 1×BLSTM	24	0.4793	0.7856	**0.1832**	0.1600	0.2141	100 525	367 1779
3×Conv 2×BLSTM	24	0.4844	0.8422	0.0787	0.0448	0.3218	28 597	59 2087
3×Conv 3×BLSTM	22	0.5371	0.8376	0.0988	0.0592	0.2988	37 588	87 2059

Table B.3: Single action detection with an input 16x16x2 with a patience of 15 epoch for validation. Using a range of convolutions and bidirectional LSTMs.
training - 1=41% 0=59% 1=4592 0=6493 total=11085
testing - 1=%23 0=77% 1=625 0=2146 total=2771

Configuration	# Epochs to Converge	Loss	Accuracy	F1	Recall	Precision	Confusion Matrix	
1×Conv 1×BLSTM	19	0.7111	0.8264	0.0237	0.0126	0.1905	8 625	34 2104
1×Conv 2×BLSTM	25	0.6081	0.8482	0.0156	0.0079	0.6250	5 628	3 2135
1×Conv 3×BLSTM	17	0.5162	0.8477	0.000	0.000	0.000	0 633	0 2138
2×Conv 1×BLSTM	24	0.5346	0.7362	**0.2539**	0.2986	0.2208	189 444	667 1471
2×Conv 2×BLSTM	18	0.4956	0.8221	0.1443	0.0995	0.2625	63 570	177 1961
2×Conv 3×BLSTM	23	0.6755	0.8433	0.0686	0.0379	0.3582	24 609	43 2095
3×Conv 1×BLSTM	20	**0.4081**	**0.8556**	0.2469	0.1564	0.5858	99 534	70 2068
3×Conv 2×BLSTM	19	0.4361	0.8512	0.1144	0.0632	0.6061	40 593	26 2112
3×Conv 3×BLSTM	22	0.4476	0.8486	0.0538	0.0284	0.5000	18 615	18 2120

Table B.4: Single action detection with an input 32x32x2 with a patience of 15 epoch for validation. Using a range of convolutions and bidirectional LSTMs.
training - 1=41% 0=59% 1=4584 0=6501
testing - 1=%23 0=77% 1=633 0=2138

B.3 Multiple Actions

B.3.1 Full Data

Configuration	# Epochs to Converge	Combined Loss	Combined Accuracy	Combined F1	Combined Recall	Combined Precision	Confusion Matrix	
1×Conv 1×LSTM	14	0.4060	0.8278	0.2878	0.2308	0.3823	2469 8227	3990 69657
1×Conv 2×LSTM	39	0.3685	0.8559	**0.2960**	0.2388	0.3892	2554 8142	4008 69639
1×Conv 3×LSTM	31	0.3438	0.8643	0.2772	0.2052	0.4269	2195 8501	2947 70700
2×Conv 1×LSTM	33	0.3739	0.8446	0.1482	0.1066	0.2432	1140 3547	3547 70100
2×Conv 2×LSTM	28	0.3711	0.8600	0.1023	0.0629	0.2737	673 10023	1786 71861
2×Conv 3×LSTM	28	0.3526	**0.8713**	0.2385	0.1588	0.4782	1699 8997	1854 71793
3×Conv 1×LSTM	41	**0.3128**	0.8660	0.2212	0.1501	0.4204	1605 9091	2213 71434
3×Conv 2×LSTM	28	0.3606	0.8645	0.1533	0.0968	0.3691	1035 9661	1769 71878
3×Conv 3×LSTM	22	0.3528	**0.8713**	0.2666	0.1845	0.4807	1973 8723	2130 71517

Table B.5: Multiple action detection with an input 16x16x1 with a patience of 15 epoch for validation. Using a range of convolutions and LSTMs. Using the full dataset.

Configuration	# Epochs to Converge	Combined Loss	Combined Accuracy	Combined F1	Combined Recall	Combined Precision	Confusion Matrix	
1×Conv 1×BLSTM	26	0.3664	0.8358	0.1553	0.1190	0.2235	1273 9423	4422 69225
1×Conv 2×BLSTM	25	0.3756	0.8370	0.2002	0.1609	0.2650	1721 8975	4774 68873
1×Conv 3×BLSTM	31	0.3437	0.8643	**0.2773**	0.2052	0.4272	2195 8501	2943 70704
2×Conv 1×BLSTM	33	0.3736	0.8451	0.1509	0.1085	0.2475	1161 9535	3529 70118
2×Conv 2×BLSTM	28	0.3792	0.8580	0.0916	0.0565	0.2428	604 10092	1884 71763
2×Conv 3×BLSTM	28	0.3547	0.8709	0.2247	0.1475	0.4708	1578 9118	1774 71873
3×Conv 1×BLSTM	41	**0.3094**	0.8668	0.2273	0.1545	0.4303	1652 9044	2187 71460
3×Conv 2×BLSTM	21	0.4253	**0.8712**	0.0022	0.0011	0.0615	12 10684	183 73464
3×Conv 3×BLSTM	19	0.4169	0.8461	0.0455	0.0289	0.1066	309 10387	2591 71056

Table B.6: Multiple action detection with an input 16x16x1 with a patience of 15 epoch for validation. Using a range of convolutions and bidirectional LSTMs. Using the full dataset.

Configuration	# Epochs to Converge	Combined Loss	Combined Accuracy	Combined F1	Combined Recall	Combined Precision	Confusion Matrix	
1×Conv 1×BLSTM	31	0.4448	0.8524	0.0540	0.0332	0.1443	355 10341	2105 71542
1×Conv 2×BLSTM	38	0.4350	0.8573	0.0414	0.0243	0.1400	260 10436	1597 72050
1×Conv 3×BLSTM	23	0.4273	0.8566	0.0360	0.0211	0.1222	226 10470	1623 72024
2×Conv 1×BLSTM	33	0.4221	0.8639	0.0317	0.0176	0.1622	188 10508	971 72676
2×Conv 2×BLSTM	31	0.3930	0.8587	**0.0812**	0.0493	0.2314	527 10169	1750 71897
2×Conv 3×BLSTM	21	0.4299	0.8645	0.0232	0.0127	0.1351	136 10560	871 72776
3×Conv 1×BLSTM	22	**0.3847**	**0.8721**	0.0271	0.0140	0.3846	150 10546	240 73407
3×Conv 2×BLSTM	21	0.4253	0.8712	0.0022	0.0011	0.0615	12 10684	183 73464
3×Conv 3×BLSTM	19	0.4169	0.8461	0.0455	0.0289	0.1066	309 10387	2591 71056

Table B.7: Multiple action detection with an input 16x16x2 with a patience of 15 epoch for validation. Using a range of convolutions and bidirectional LSTMs.Using the full dataset.

Configuration	# Epochs to Converge	Combined Loss	Combined Accuracy	Combined F1	Combined Recall	Combined Precision	Confusion Matrix	
1×Conv	35	0.5533	0.8424	**0.0526**	0.0345	0.1105	369	2969
1×BLSTM							10327	70678
1×Conv	19	0.4597	0.8625	0.0366	0.0206	0.1639	220	1122
2×BLSTM							10476	72525
1×Conv	18	0.5280	0.8618	0.0212	0.0118	0.1036	126	1090
3×BLSTM							10570	72557
2×Conv	19	0.5027	0.8660	0.0152	0.0081	0.1113	87	695
1×BLSTM							10609	72952
2×Conv	26	0.4844	0.8689	0.0097	0.0050	0.1146	54	417
2×BLSTM							10642	73230
2×Conv	24	0.4817	0.8678	0.0180	0.0095	0.1555	102	554
3×BLSTM							10594	73093
3×Conv	19	0.5226	0.8674	0.0075	0.0039	0.0736	42	529
1×BLSTM							10654	73118
3×Conv	21	**0.4249**	0.8724	0.0009	0.0005	0.0685	5	68
2×BLSTM							10691	73579
3×Conv	22	0.4698	**0.8729**	0.0017	0.0008	0.2045	9	35
3×BLSTM							10687	73612

Table B.8: Multiple action detection with an input 32x32x2 with a patience of 15 epoch for validation. Using a range of convolutions and bidirectional LSTMs. Using the full dataset.

B.3.2 Half Data

Configuration	# Epochs to Converge	Combined Loss	Combined Accuracy	Combined F1	Combined Recall	Combined Precision	Confusion Matrix	
1×Conv 1×LSTM	41	0.4059	0.8914	**0.4213**	0.3627	0.5024	1661	1645
							2919	35782
1×Conv 2×LSTM	35	0.3519	0.8703	0.2877	0.2402	0.3585	1100	1968
							3480	35459
1×Conv 3×LSTM	34	0.4185	0.8659	0.3441	0.3227	0.3686	1478	2532
							3102	34895
2×Conv 1×LSTM	41	0.3433	0.8875	0.3647	0.2961	0.4748	1356	1500
							3224	35927
2×Conv 2×LSTM	34	**0.3229**	0.8865	0.3872	0.3288	0.4709	1506	1692
							3074	35735
2×Conv 3×LSTM	40	0.3470	**0.8916**	0.3834	0.3092	0.5046	1416	1390
							3164	36037
3×Conv 1×LSTM	49	0.4079	0.8836	0.3938	0.3467	0.4557	1588	1897
							2992	35530
3×Conv 2×LSTM	33	0.3304	0.8780	0.2713	0.2083	0.3891	954	1498
							3626	35929
3×Conv 3×LSTM	43	0.3251	0.8790	0.2915	0.2282	0.4033	1045	1546
							3535	35881

Table B.9: Multiple action detection with an input 16x16x1 with a patience of 15 epoch for validation. Using a range of convolutions and LSTMs. Using the half dataset.

Configuration	# Epochs to Converge	Combined Loss	Combined Accuracy	Combined F1	Combined Recall	Combined Precision	Confusion Matrix	
1×Conv 1×BLSTM	23	0.4437	0.8546	0.3688	0.3897	0.3501	1785 2795	3314 34113
1×Conv 2×BLSTM	26	0.3812	0.8778	**0.4159**	0.3991	0.4342	1828 2752	2382 35045
1×Conv 3×BLSTM	23	0.3836	0.8671	0.3270	0.2961	0.3651	1356 3224	2358 35069
2×Conv 1×BLSTM	22	0.3560	0.8773	0.3904	0.3605	0.4258	1651 2929	2226 35201
2×Conv 2×BLSTM	47	0.3823	**0.8827**	0.3873	0.3400	0.45	1557 3023	1903 35524
2×Conv 3×BLSTM	30	0.4425	0.8760	0.3736	0.3391	0.4159	1553 3027	2181 35246
3×Conv 1×BLSTM	46	0.3880	0.8822	0.3315	0.2679	0.4348	1227 3353	1595 35832
3×Conv 2×BLSTM	35	**0.3206**	0.8706	0.3544	0.3258	0.3886	1492 3088	2347 35080
3×Conv 3×BLSTM	31	0.3654	0.8770	0.2208	0.1598	0.3571	732 3848	1318 36109

Table B.10: Multiple action detection with an input 16x16x1 with a patience of 15 epoch for validation. Using a range of convolutions and bidirectional LSTMs. Using the half dataset.

Configuration	# Epochs to Converge	Combined Loss	Combined Accuracy	Combined F1	Combined Recall	Combined Precision	Confusion Matrix	
1×Conv 1×BLSTM	23	**0.3388**	**0.8912**	**0.4695**	0.4417	0.5011	2023 2557	2014 35413
1×Conv 2×BLSTM	34	0.3749	0.8750	0.3442	0.3009	0.4020	1378 3202	2050 35377
1×Conv 3×BLSTM	29	0.3909	0.8825	0.4219	0.3932	0.4550	1801 2779	2157 35270
2×Conv 1×BLSTM	25	0.3594	0.8826	0.3513	0.2917	0.4417	1336 3244	1689 35738
2×Conv 2×BLSTM	45	0.3987	0.8702	0.2882	0.2410	0.3582	1104 3476	1978 35449
2×Conv 3×BLSTM	27	0.4452	0.8615	0.3362	0.3216	0.3522	1473 3107	2709 34718
3×Conv 1×BLSTM	31	0.4481	0.8840	0.3583	0.2969	0.4517	1360 3220	1651 35776
3×Conv 2×BLSTM	28	0.4338	0.8390	0.2552	0.2531	0.2574	1159 3421	3343 34084
3×Conv 3×BLSTM	33	0.3638	0.8756	0.2009	0.1434	0.3352	657 3923	1303 36124

Table B.11: Multiple action detection with an input 16x16x2 with a patience of 15 epoch for validation. Using a range of convolutions and bidirectional LSTMs. Using the half dataset.

Configuration	# Epochs to Converge	Combined Loss	Combined Accuracy	Combined F1	Combined Recall	Combined Precision	Confusion Matrix	
1×Conv 1×BLSTM	34	0.3322	0.9092	0.4725	0.3731	0.6439	1709 2871	945 36482
1×Conv 2×BLSTM	32	**0.2844**	0.9021	0.3632	0.2561	0.6243	1173 3407	706 36721
1×Conv 3×BLSTM	32	0.3750	**0.9119**	0.4978	0.4002	0.6582	1833 2747	952 36475
2×Conv 1×BLSTM	44	0.4335	0.8782	0.3873	0.3531	0.4289	1617 2963	2153 35274
2×Conv 2×BLSTM	56	0.4121	0.8891	0.4223	0.3718	0.4887	1703 2877	1782 35645
2×Conv 3×BLSTM	27	0.3762	0.8951	**0.5013**	0.4836	0.5203	2215 2365	2042 35385
3×Conv 1×BLSTM	31	0.4115	0.8641	0.2379	0.1945	0.3061	891 3689	2020 35407
3×Conv 2×BLSTM	38	0.3847	0.8768	0.3878	0.3579	0.4233	1639 2941	2233 35194
3×Conv 3×BLSTM	32	0.3645	0.8775	0.2474	0.1847	0.3747	846 3734	1412 36015

Table B.12: Multiple action detection with an input 32x32x2 with a patience of 15 epoch for validation. Using a range of convolutions and bidirectional LSTMs. Using the half dataset.