Self-Driving Cameras: Automated Camera Capture for Biological Imaging

Joseph McGirr
School of Biosciences/ Computer Science
University of Nottingham

Supervisor
Michael Pound

In partial fulfillment of the requirements for the degree of
Master of Research in Computer Science
October 27, 2020

Acknowledgements

Thank you to my supervisor, Michael for providing guidance and feedback throughout this project. Thanks also to my mum Fran for feeding me throughout.

Abstract

Efficient quantitative analysis of plant traits is critical to keep pace with advances in molecular and genetic plant breeding tools. Machine learning has shown impressive results in automating a lot of these analytical processes however, many of the algorithms rely on a surplus of high-quality biological imagery. This data is currently collected in labs via static camera systems, which provide consistent images but are challenging to tailor to individual plants, species, or tasks. Current research in autonomous camera systems use object detection or tracking methods to control the camera. Unfortunately, this quickly falls apart for static biological imagery as large inter- and intra-species variations, even within the same specimen, make object detection less robust and stationary targets make tracking unusable. Inspired by the success of deep learning in the autonomous driving space, we apply an end-to-end learned approach to directly map saliency-augmented input frames from an RGB monocular camera to a pan-tilt-zoom (PTZ) actuation. Our results show our model correctly classifies which direction to move the camera in 87% of instances and has an average offset error of 250 and 140 pixels for a 1920x1080 image, respectively. Results on a much smaller, plant-only dataset demonstrates the applicability of the model to biological imagery and we demonstrate saliency's effectiveness in improving accuracy by up to 4%.

Contents

List of Figures

List of Tables

Chapter 1

Introduction and Related Work

Plant phenotyping laboratories can benefit greatly from autonomous camera control, leading to better quality images that can be tailored to specific use-cases. The phenotyping labs often consist of heterogeneous plant species which are monitored over time as they grow. This means there is a wide variety in plant structure, size and colour which can make encompassing the relevant parts of the plant difficult for a static camera rig. Furthermore, the scientists that monitor the plants are conducting different experiments for different plants and species. For example, analysing the fruit yield of a strawberry plant versus measuring wheat growth over time. Tailoring the images of each plant to maximise the insight for each specific experiment is difficult for a static camera setup and important parts of the plant may be occluded. Changes in lighting conditions and background clutter exaggerate the problem. This inflexibility can be particularly damaging if the images are going to be analysed via a machine learning technique such as [1]. These algorithms are less robust to poor images than human analysis and could produce inaccurate results if the image does not properly encompass the integral parts of the plant for the experiment. Another use of the images is in the actual training of these machine learning algorithms, particularly deep learning

algorithms. These algorithms rely on large-scale labelled datasets to learn mappings between images and the desired output. Poor quality imagery can therefore be particularly damaging to the performance of these models and will have a cumulative impact on how well the mapping is learnt. The scale of the datasets required for deep learning also means manual postprocessing of the images is time-consuming. Therefore, a dynamic, autonomous camera capture system is preferable as the camera position can be tailored to each plant/experiment giving a higher quality image.

Most approaches to autonomous cameras rely on application-defined object detection, which is then fed to a rule-based camera control system [2]. For the task of biological imaging, object detection quickly breaks down as the plants grow, and their appearance changes, as well as with the introduction of new, unseen plant species. Additional challenges such as occlusion, diverse and cluttered backgrounds further degrade performance. In this work, we propose a camera control pipeline including a multi-headed, deep CNN architecture, that takes the current image as input and outputs the camera control signals required to better encompass the salient object (in our case, plant) in the scene. These signals include the pan and tilt camera offsets required to recentre the object and the field of view adjustments (or zoom) required to limit excessive or insufficient space around the object. Figure 1.1 demonstrates two example outputs of the algorithm, starting from the initial input frame in the left-hand image, to the camera position correction in the middle image and finally, the field-of-view (FOV) correction in the right-hand image. Ideally, the model should complete both position correction and FOV correction in one single step, however in practise it is more likely that the camera will iterate over a few different positions/zooms until a final position is settled upon. The figure demonstrates the model on both a plant (top) and an everyday house item, specifically a lighter (bottom). The solution

generalises to different objects as a direct frame-to-action mapping is learnt as opposed to a task-specific object detector. The model is also generalised to work with different camera rigs by gauging offset predictions in image pixel coordinates. These are then converted to real-world pan-tilt-zoom (PTZ) coordinates, using coefficients configured in an earlier camera calibration step which we explain in more detail in Chapter 5.1. We also utilise transfer learning to improve classification results. With this approach, we hope to simulate the innate capacity that a human camera operator has to pan and zoom to unseen objects, with the intention of generalising to biological imagery.

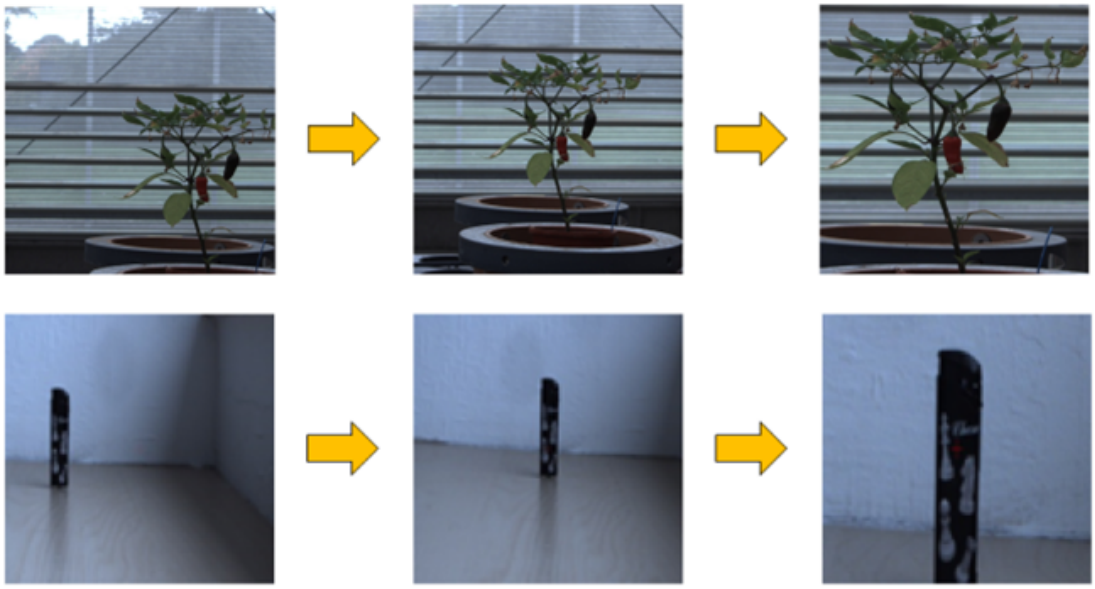

Figure 1.1: Exemplifies how input frames will look during operation, starting with the initial input frame on the left, after the camera position has been corrected in the middle column and after field-of-view (FOV) correction in the right-hand column. Top row: plant. Bottom row: household object (lighter). In the optimal case, camera position and FOV correction will occur simultaneously in one single step. Otherwise, multiple steps could be required before final camera coordinates are settled upon.

A large inspiration for choosing this approach is the recent success of end-to-end learning for self-driving cars such as the NVIDIA project [3] in 2006, which

built on the success of [4] and [5]. The network takes as input the raw pixels of a front-facing camera and learns to directly map this to steering actuation. By utilising a CNN, the algorithm was able to learn useful road features on its own, using only human steering angle as guidance. [6] increased the robustness of NVIDIA's results by including supplementary input, in the form of semantic segmentation to improve performance in lane following tasks. By reposing the autonomous camera problem as a data-driven mapping of input frame to camera pan-tilt-zoom coordinates, we can avoid having to hand-craft features and rules and enable the underlying process of recentring a salient object to be modelled more generally. The applications of this approach have the potential to extend further than just biological imaging, as a general model of autonomous camera relocation can be applied to other imaging modalities, including diffractometric crystal centring [7; 8] and automated microscopy [9]. It could also help provide insights into how humans retarget their attention in a cluttered image . Like [6], we augment our image with supplementary data, in this case, the output of a visual saliency network based on [10]. This enables us to add robustness to cluttered backgrounds by isolating the salient object from the background pixels. This is particularly important in plant phenotyping laboratories, where greenhouse architecture, plants, machinery, and people may be visible in the background of each image.

1.1 Contributions

This thesis offers the following contributions. First, we develop an end-to-end generalised solution to the autonomous camera problem that consolidates the two processes of ROI detection and camera control and can be applied to static scenes. Second, we demonstrate the effectiveness of saliency attention networks in

adding robustness to cluttered backgrounds, showing that this approach matches or exceeds no-saliency so long as effective saliency maps can be produced. Third, we further generalise the network to be camera rig independent by implementing a SIFT-based feature matching algorithm to generate the offset labels for non-optimal camera positions in pixels. This enables us to develop training data from any camera setup, including crowd-sourced data. We then train the network to output offsets predictions in pixels and develop a pipeline starting with camera calibration, that converts the pixel offsets to real-world camera pan-tilt coordinates. Finally, we introduce a suitable accuracy measure that accounts for incremental improvements in camera position and shows accurate results on a dataset of general, everyday objects, as well as promising results on a small plant-based dataset.

This work intends to automate image capture in plant phenotyping laboratories and investigate the effectiveness of an end-to-end learned approach. Inspired by recent advances in self-driving vehicle research, we apply deep learning to avoid designating application-specific features. The results of our experiments show that we can learn a more generalised embodiment of a human-controlled camera system, that is robust to complex backgrounds. To further aid further research in this area, we also make publicly available our novel dataset containing over 12,500 images of objects and their relative offsets from the optimal camera position - available on request (*sbzmpp@exmail.nottingham.ac.uk.*)

1.2 Related Work

This section gives a brief overview of the current research in autonomous cameras, including object detection and tracking approaches as well as approaches that use deep learning. We also explore similar use-cases such as crystal centering and

aesthetic image re-targeting. We go on to explore techniques similar to ours that have been employed in self-driving cars and explore how we can follow the same principles to automate parts of the data collection process.

[2] splits the problem of automatic camera systems into the following 3 problems:

- **Camera Planning:** where the camera should look.
- **Camera Control:** how the camera should move .
- **Camera Selection:** which camera is selected (applies to setups with multiple cameras).

Our use-case is focused on correcting the camera position to fully encompass the salient object in the image plane. This means deciding where to move the cameras' optical centre to recentre the object and how large to set the field-of-view. As such, we are not interested in how the camera moves to this position, so long as it gets there reasonably quickly. This contrasts with use-cases such as autonomous lecture capture [11], sports [12], or TV productions [13], where replicating the visual aesthetics of a human operator is critical to the success of the system. For example, [11] uses mathematical equations to replicate panning speed, acceleration, and deceleration based on the shooting technique of a human camera operator.

Most current approaches to solving the camera planning problem fall into the category of passive tracking, with separate tasks for object tracking and camera control. [2], uses an object detector to outline the region of interest (ROI) in the image and a rule-based camera controller converts this to a camera pan-tilt-zoom coordinate. The autonomous lecture capture algorithm in [11] detects the ROI with temporal differencing, then crops a larger, high-resolution video to give the impression of panning in the x-axis. [2] similarly uses a real-time

multi-person detector to provide a coarse ROI that a robotic PTZ camera can follow and converts the target location in the world system to a PTZ camera movement. [14] demonstrates an assortment of different techniques for the task of ROI estimation.

[15] leverages more complex input features including coordinates for players, ball, referees, and classified game-states in a basketball match to inform camera control. This algorithm is highly tailored to a specific use-case. Our system on the other hand aims to be much more general, as we do not explicitly map the underlying features and processes behind it. [16; 17] demonstrates how additional features such as visual saliency detection can be useful in optimising the path of the camera. We employ a visual saliency model based on [10] to add robustness to cluttered backgrounds by isolating the salient object in the scene. The algorithm integrates both high and low-level features to extract more effective features for saliency detection. High-level features capture rich context features but struggle to approximate area, whereas low-level features contain a lot of noise but have good area predictions. By combining the two and choosing the most effective features from each, a better saliency map can be produced. To do so, both low and high-level features are extracted from a VGG or ResNet's convolutional layers. The high-level features are passed to a context-aware pyramid feature extraction (CPFE) module, to get multi-scale/receptive-field high-level features. Different scales are retrieved by applying multi-scale atrous convolutions on the input volumes. A channel-wise attention mechanism (CA) [18] is then applied to the output of the CPFE module. Attention mechanisms enable more important features (channels in this case) to take precedence by scaling them by a higher value during training and inference. By applying channel-wise attention after CPFE module, the best scale/receptive-field features are selected for generating saliency regions. Low-level features are useful in obtaining detailed boundaries

between salient objects and backgrounds, however low-level background textures can distract the generation of saliency map. Spatial attention (SA) is used to focus more on the foreground regions and help generate effective low-level features for saliency predictions. The algorithm also uses an edge-preservation loss to guide network to learn more detailed boundary localization. [19] were the first to pose camera control as a supervised regression problem. They learn the mapping between player locations detected in a sports game and a camera's pan-tilt-zoom parameter at any given video frame, using SVM and Random Forest learning approaches. More recent works on autonomous camera systems have explored active tracking, which unifies the tasks of object tracking and camera control. [13] uses a cascade-correlation neural network architecture that takes the current image and directly outputs a pan angle to replicate that of a human-controlled camera. They demonstrate that shooting techniques can be embodied in a learning algorithm.

The most recent approaches to the problem have utilised deep learning. [20] were the first paper to propose a direct mapping of frame-to-action tracking using a deep reinforcement-learning (RL) algorithm. Through environmental augmentation, they train the RL algorithm to directly predict camera movement actions (forward, left, right, etc.) from a raw input image by tracking a moving object in the scene. The network can generalise to unseen objects, backgrounds, and moving paths. We similarly want to predict a direct camera movement from an input image however, we are concerned with static scene camera adjustment rather than tracking moving objects. A closer use-case to ours is found in [21], who use a convolutional neural network (CNN) to automate the process of moving a camera's focal point over the centre of a crystal - an intermediary step in macromolecular crystallography. The network uses a Single Shot Multibox Detector to identify a bounding box around the crystal and centres the camera over

the region. [22] alternatively predicts the reflection count at different offsets and chooses the highest. We too want to centre our subject in the scene, but our approach more closely resembles active tracking, where instead of an intermediary step (ROI detection, reflection prediction) determining the actuation, we directly map the input image to the camera signals needed to centre the salient object. It is likely that with enough data, our approach could be applied to crystal centring, with the added benefit of autonomous zoom. Image retargeting also has parallels with our research area. [23] and [24] generate 'attention guided cropping candidates' to find a cropping window that preserves the most important parts of an image. These approaches show that with enough data, deep learning can express moderately subjective decisions, especially when consolidated with supplementary input like visual saliency.

Learned end-to-end actuation from a raw input image has shown encouraging results in the autonomous driving space [25]. [3; 26] trained a CNN to map raw pixels from a single front-facing camera directly to steering commands to achieve lane following. [27] expands on this by applying imitation learning to expand the dataset. To collect their datasets, a time-stamped video is captured by a human operator with the corresponding steering angles at each frame. Similar approaches have been applied to autonomous drone navigation [28; 29]. Our data collection protocol consists of manually centring the camera on the object, applying random offsets to the camera, and recording the new image and its relative offset from the centre image as the label. [6] learns a 'generic vehicle motion model', incorporating the current observation and previous egomotion into the prediction. This decouples the algorithm from the specific driving setup and enables the algorithm to benefit from large-scale, crowd-sourced data. We sought to replicate this in our algorithm by predicting the pixel-offsets, as opposed to direct gimbal units, required to recentre the object. We then apply a camera

calibration algorithm explained in Chapter 5 to convert the pixel offsets to gimbal actuation units.

To predict the offsets in pixels, we had to choose a feature matching algorithm that could identify unique features of an object and match them between the two images. Features are the parts or patterns of an object in an image that help to identify it. Two types of feature detectors exist: traditional detectors such as Harris Corner Detection [30], Scale-Invariant Feature Transform (SIFT) [31] and Speeded-Up Robust Features (SURF) [32] or deep-learning based feature extraction techniques such as those described in [33] and [34]. CNN's have repeatedly proven themselves to be highly capable of extracting meaningful features that describe an image in detail. According to [35], deep learning approaches outperform traditional, hand-crafted feature-based approaches in all tested settings. However, they also note that traditional feature extractors often have much lower time complexity and can still produce similar results to SOTA deep learning methods. Due to the wealth of information and source code samples of the traditional approaches [36], we will use these in our project. They also mention that "handcrafted features can provide complementary information to the deep learning features" which suggests information that is missed from a deep learning approach could also be picked up by traditional approaches. Therefore, it seems fitting to use hand-crafted features to produce the annotations, as we later apply the CNN to learn to predict offset in the images and so we could benefit from the combined effect of the two feature types.

Chapter 2

System Overview

Introduction

In this chapter, we will provide a brief overview of the entire pipeline we have developed and its components. We then go into some detail pertaining to the camera-gimbal setup that was used to both collect data and to test the models we developed. We finally touch upon the dataset required to train the network, however, more detail on this can be found in Chapter 3. This chapter aims to articulate how the system works at a higher level, before drilling down into each of the individual components in later chapters. By the end of the chapter, we hope that the reader has a basic comprehension of how the pipeline is structured and the components we used to achieve robustness to complex backgrounds as well as generalisability to different camera-gimbal rigs.

2.1 Pipeline Overview

A simplified block diagram of our pipeline is shown in Figure 2.1. The pipeline consists of a single, deep neural network constructed of a saliency network and

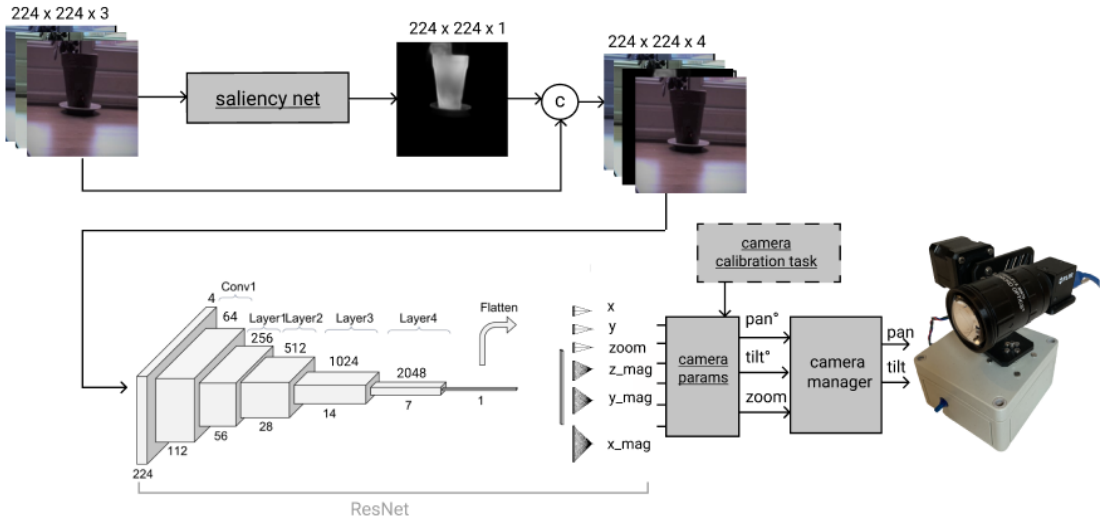

Figure 2.1: The overall pipeline of our system. From the top-left, the 224x224x3 image is passed to the saliency net which generates a saliency map for the image. The saliency map is then augmented with the input image via channel-wise concatenation, symbolised by 'C' in the diagram. The augmentation is then fed to the ResNet classifier to produce a direction and magnitude classification to move the camera in. The different layers of the ResNet architecture are depicted and labelled with their width/height and depth dimensions below and above each layer, respectively. A pan(x), tilt(y) and zoom direction classification are produced from the ResNet, consisting of positive, negative or no movement classes. Also, a discretized bin index is produced for each direction's magnitude which can later be converted to the delegated value for each bin, measured in pixels. The pixel values are then converted back to the gimbal-specific units based on the camera parameters configured in an earlier camera calibration task. The movement can then be actuated by the camera-gimbal rig.

a ResNet classifier. Images are fed into the saliency network which generates a saliency map for the image and concatenates it (channel-wise) with the input image. The augmented image is then passed to a multi-headed ResNet [37] classifier. It is passed through the convolutional layers of the network until a 1x2048 feature vector is attained. At this point, six different fully connected layers representing pan, tilt and zoom direction and pan, tilt and zoom magnitudes take the feature vector as input and generate a probability distribution. An argmax function is

then applied to each of the fully connected layer outputs to find the class with the largest predicted probability for each. The direction classes consist of positive (1), negative (-1) or no movement (0). The magnitude classification refers to a discretized bin value, measured in pixels. The resulting magnitude value can then be multiplied by the direction and converted to real-world, pan-tilt-zoom coordinates using parameters generated from the camera calibration task. These positions are then serialized into gimbal instructions to actuate the movement.

The saliency network is a pyramid feature attention (PFA) saliency network [10] and the classifier is a ResNet architecture, with 101 layers. The saliency network uses a combination of high-level and low-level features, to generate an edge-preserving saliency map (see Section 4.2 for more detail). The ResNet has been adapted with six fully connected layers for each of the separate classification outputs (pan direction, tilt direction, zoom direction, pan magnitude, tilt magnitude, zoom magnitude). Each fully connected layer takes the 1x2048 feature vector produced from the convolution layers and produces a probability distribution for each of their possible classes: The three direction classifications have 3 classes: 1) tilt-up/pan-left/zoom-in, 2) tilt-down/pan-right/zoom-out, 3) no axis movement. The three magnitude predictions predict the magnitude with which to move in each direction. These are divided into fixed-size bins and are expressed in pixels. More detail on how the magnitudes are binned can be found in Section 3.1.2.2. Argmax is applied to each of the fully connected layers to find the class with the largest predicted probability.

Finally, the outputs are converted from pixels to pan-tilt-zoom coordinates. This is done by multiplying the pixel offsets by the proportional rotation of the camera in degrees per pixel (degrees-per-pixel). The degrees-per-pixel value is calculated for the x and y axes during the camera calibration task and only needs to be calibrated once per camera-gimbal rig. Once we have the pan and tilt

coordinates, we can serialize them and send them to the gimbal to be actuated. Because our camera setup does not currently have a programable zoom function, the zoom is simulated by performing an N*5% crop for each zoom unit, see Section 3.1.1 for more detail. A single pass through the pipeline (having calibrated the camera-gimbal), enables the input from the camera to be converted into a pan-tilt-zoom coordinate that should move towards recentring and resizing the object in the scene. Once convergence is achieved, a picture of the object is taken and saved.

2.2 Camera Gimbal Setup

2.2.1 Camera

The camera we use is a Flea3 FL3-U3-32S2C-CS, a 3.2MP camera. It operates at 60 FPS and uses a Sony IMX036 Colour sensor with sensor size 1/2.8" (6.49 mm). This gives it a maximum resolution of 2080 x 1552 (3.27MP) effective pixels, with a pixel size of 2. 5µm. However, we shoot the camera in 1920x1080 mode for our experiments. We use Point Grey's Spinnaker SDK python bindings to interface with the camera over USB3.0.

To the camera, we attach a 12mm, fixed focal length HP Series lens. The fixed focal length means that we cannot program zoom directly, so we simulate it instead using image cropping (see Section 3.1.1). The lens has a variable aperture f/1.8-f/16. In our experiments, we manually increase the aperture as high as is possible in the environment without saturating exposure. This enables us to attain a shallow depth of field, blurring backgrounds, and keeping the foreground objects sharp [38]. In doing so, better separation between the subject and the background can be achieved, reducing the complexity required to find the salient object.

2.2.2 Gimbal

To actuate predictions into real camera movements, the team at the University of Nottingham's Make Studio have designed and built a motorised pan-tilt gimbal mount. The motor control system is based on a microcontroller development board (Arduino Uno R3)—this incorporates a 16 MHz ATmega328P controller on an inexpensive breakout board with multiple input/output connections including a USB serial connection to a host PC or laptop [39]. An expansion shield (CNC Shield V3) is connected to the board to allow the deployment of up to three stepper motor drivers in the widely-used "StepStick" format [40]. The motor drivers selected for this system (DRV8825, TI) can be configured to single-stepping, 1/2, 1/4, 1/8, 1/16, or 1/32 microsteps and operate at a maximum drive current of 2.5 A at 24 V. All electronic components are housed in a case with a connector for stepper motor power. The motors are powered by a 24 V, 2.71 A power adaptor.

The microcontroller board is powered by a USB connection to the host computer, which provides serial communication. The microcontroller runs a sketch written in the Arduino Integrated Development Environment [41] that uses the AccelStepper library [42] to control the stepper motors. This sketch allows the setting of acceleration parameters for each motor and monitors the serial connection.

On receiving a serial string with positional information for the pan and tilt motors via the USB port, the motors are moved to their respective positions using the pre-defined acceleration parameters to ensure smooth acceleration and deceleration. The L-bracket and 'tilt shaft', pictured in Figure 2.2, were printed using a fused filament fabrication 3D printer (Model S5, Ultimaker) using tough polylactic acid (PLA) filament. A picture of the camera and gimbal setup can also be found in Figure 2.2.

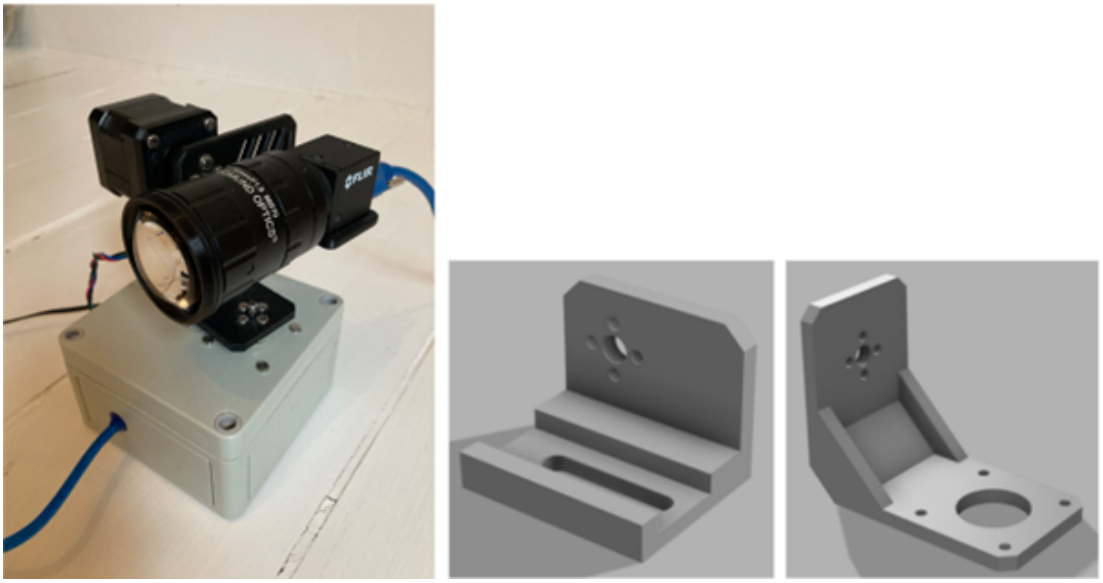

Figure 2.2: Left: A photograph of the camera-gimbal rig we used for the project. It consists of a programmable FL3-U3-32S2C-CS 3.2MP camera and a pan-tilt gimbal which serializes actuation commands over a USB3.0 connection. Right: The 3D models for the 3D printed L-bracket and tilt shaft that is found on the gimbal and enable the pan and tilt motion when connected to the x and y-axis stepper motors.

2.3 Dataset

Training the network requires a dataset of input images containing a salient object and a set of annotated labels describing where to move the camera to recentre and resize the object in the image plane. The unit for zoom is recorded in 5% crops. To keep the solution general to all camera rigs, x and y offsets are recorded in pixel values.

To generate the labels, a human operator centres the camera over the object in the scene. This is recorded as the 'optimal', centred camera position. Next, the camera autonomously iterates over random offsets from the optimal position, taking a picture at each position and recording the offset. To measure the offset from the centred image, a SIFT-based feature detector [31] is applied to the offset

image and the centred image to detect common, scale-invariant features between them. The mean difference of the matched feature coordinates (measured in pixels) is used as the label for the pan (x-axis) and tilt (y-axis) offsets of the image. To generate the zoom label, we manually label each centred image and then extrapolate this zoom label to all the offset images. This process is repeated twice for an object, rotating it halfway through.

The dataset we use to train the deep network has been collected both with both our camera rig (1920x1080 resolution) and a generic mobile phone (2048x1536/1000x750 res.) and contains a diverse set of everyday household objects, including plants, and backgrounds.

2.4 Training the network

During training, a batch of images is input into the network, augmented with a saliency map and a classification is produced. A combined loss function is calculated over all the outputs and backpropagated through the ResNets' weights. The saliency network weights are frozen, so backpropagation does not affect them.

This chapter provides a high-level introduction to the key software and hardware components in our autonomous camera system. The next three chapters will explore these topics in more detail and explain the reasoning behind the different design decisions that were made.

Chapter 3

Data Collection

Introduction

In this chapter, we describe how we generate the data to train and test the network. We explain the properties required by the datasets, the methodologies of capturing and labelling images, and the characteristics of the different datasets that make up the final dataset. We also provide some basic statistics.

A key contribution of this work is the introduction of a new dataset for autonomous camera control. Some autonomous camera control datasets do exist for task-specific cases such as autonomous sports capture [19]. This dataset consists of a video taken from a human-controlled camera operator, labelled with the estimated pan angles at each frame. The labels are generated by modelling the rotation matrix at each frame, using the standard pinhole model [43]. Currently, there is no such large-scale archive of human-operators recentring objects in an image with which we could apply a similar methodology to. We attribute this mainly to the nuance of the task, especially if we are only considering biological imaging. Another reason we chose to create the dataset was that our solution was originally camera-rig-specific, so we would require not just the input frames,

but also a manual annotation of the movement required by our specific camera rig.

From our dataset, our key aim was to annotate a mapping between an input image, and a real-world gimbal actuation for a camera-rig. Many datasets currently exist for the analogous task of autonomous driving [44; 45]. These consist of a set of inputs, including images from front and rear-facing cameras, alongside their respective labels of speed and steering wheel angle which is automatically recorded as a human-operator drives the car. The network can then learn to predict the optimal steering angle for each road image, based on these labels and generalise to unseen images. The cumbersome nature of manually annotating datasets large enough to train deep neural networks meant it was not a viable option. Therefore, we wanted to imitate the automated data collection methodology of autonomous driving datasets, by computing the offset of the current camera position from the optimal camera position for each frame. Initially, our solution was camera-rig-specific. However, after deciding to generalise the algorithm, we developed a rig-agnostic method by detecting offsets between the optimal and sub-optimal camera positions in pixels, rather than the specific units of our gimbal. This is discussed in detail in Section 3.1.

Currently, most plant datasets are related to the standard tasks of plant classification, detection, localization, and segmentation. These are usually hand-annotated by experts which is prone to human error as discussed in [46]. The reduction of human error is an added benefit of automating our data collection process. Also, A lot of the images in these standard plant datasets are taken against plain, or explicitly clinical backgrounds. This reduces their generalisability to other settings which is a critical motivation in our study. Hence, we aspired to capture as diverse a set of backgrounds, and spatial variations as we could to prepare for the glasshouse conditions.

Local lockdown rules meant we struggled to gain access to the university greenhouses routinely over the course of the year, so we decided to first prove our algorithm on a variety of everyday, household objects rather than plants. This included items such as bottles, packaging, and anything we could find around the house. By using a wide variety of objects, we anticipated that the algorithm would generalise to plants and would strengthen our argument that end-to-end deep learning approaches can achieve better generalisability.

3.1 Methodology

Figure 3.1: An example of the annotated ground truths generated for the network. Vertical arrows represent tilt direction and horizontal arrows represent pan; Z represents zoom. Left: The offset image, with the tilt and pan pixel offsets given in pixel values normalised between -1 and 1. The negative tilt offset means tilt the camera down, a positive pan offset means pan the camera to the right. The sign of the offset dictates the direction classification predicted by the network. The magnitude prediction is the absolute value of the pixel offset. No zoom occurs in the left-hand image as the object is too far from the centre of the image. Right: The zoom value of 7 will result in a 7x5% centre crop of the image from the edge.

A collection of general, household objects were collected. This population was selected based on the objects' variety in size, shape, colour, and opacity to make the problem space as complex as possible to increase generalisability. To collect and annotate the data, an automated capture program was written. The capture program enables the user to manually centre the object in the image plane and select a suitable zoom to encompass the whole object without excessive or deficient empty space surrounding it. The image is then captured and stored as the 'optimal' image for the object. After this, the camera randomly iterates through various offsets from the centre (within predetermined bounds), capturing images at every position and recording its offset from the optimal position. Resulting in an input frame and labels for pan, tilt and zoom coordinates with which to relocate the camera over the salient object in the scene.

To record the camera offset of each offset image from the centred image, one approach we tried was to annotate the difference between the gimbal stepper motor positions at the two positions. We trained a rudimentary network using this approach and the results were promising, however, it meant that the network would be tightly coupled with the camera-rig . This means that if we swap the camera rig, the model will no longer work as the offset predictions are measured in the units specific to our first rig and will not necessarily generalise to the new rig. It also means that we can only collect training data from our original camera-rig, whereas if we can measure the offsets between two images in a universal metric system, we can process images taken from any camera medium. This enables quicker large-scale collection of data as we can use a mobile phone, rather than the less-mobile camera rig for data collection. It also means some data can be crowd-sourced by anyone with a camera.

So, we decided to calculate the offsets between images in normalised pixel values instead, to make the network camera-rig agnostic. To achieve this, A

SIFT-based feature detector is applied to the offset image and the centred image. This identifies matching features between the two images that are invariant to scale, rotation, and illumination [31]. The mean difference between the matched features' coordinates can then be calculated to find the x and y pixel offsets from the centred image. This can be done between two images taken on any device. We found the SIFT algorithm to be very reliable in detecting features effectively between scenes which means we can reliably calculate the offsets between two images most of the time. If no features are found, we move the camera to a different position and try again. This was a rare occurrence in practice. It is important for the feature matching to be reliable as it will be the limiting factor on how effective our deep learning model is. This is because neural networks learn their mapping function from the data alone and consequently are only as good as the data used to train them. If the feature matching algorithm can only detect features between 50% of images, then the 50% that are missed will also be missed from the training corpus and hence the network may not be able to produce a valid mapping for them either. Each saved image path is stored in a CSV format with its x and y pixel offsets (including 0,0 for the centred images). These offsets are also normalized over half of the resolution of the image (see equation 3.1) and stored in the file, alongside the zoom value, which offset images inherit from their centred image. The effect of normalisation gives the offsets as a percentage of the total image, rather than absolute pixel values, enabling the data to generalise to different camera resolutions.

$$x' = \frac{x}{(0.5 * x\_res)} \in [-1, 1] \tag{3.1}$$

The object is rotated and then moved to another position where the process is repeated. This helps to ensure that the algorithm does not just learn a background subtraction function. The whole process is then repeated for all objects

in the dataset. An example of an annotated instance from the dataset is given in Figure 3.1

3.1.1 Zoom

Since our cameras' fixed focal length meant we could not programmatically control zoom, we decided to simulate zoom using image cropping. It is quite feasible to crop these images as the resolution of the CNN (224x224) is much lower than the capture resolution. This means any information lost when the image is cropped would likely be lost when the image is resized for the network anyway. Cropping can easily be converted into a zoom signal when available in the camera rig.

During data collection, the user specifies the optimal zoom for the centred image of each object, by previewing an n * 5% centre crop of the image and incrementing or reducing n until a suitable field-of-view is attained. This zoom value n is then inherited by all offset images of the object so long as the object is close enough to the image centre i.e. the x and y ground truth magnitudes are within the bottom 25% and 50% of x and y magnitude bins, respectively. Figure 3.2 demonstrates the ground truth cropping process. Each coloured line represents an incrementing 5% crop of the image. For the star object we would select the light blue line (4*5% crop) as the final crop value 4.

We could train the network on this value alone, however, this would mean it only learns to predict zooming in. We cannot manually set a zoom out value for images because we cannot crop outwards. Therefore, we need to augment the data, which we do during training. This means performing a random n*5% centre crop of each input image and updating the zoom label accordingly by calculating the difference between n and the image's ground truth (g.t) zoom value. If n $<$ g.t, the new zoom will be positive (zoom in). If n $>$ g.t, the new zoom will be

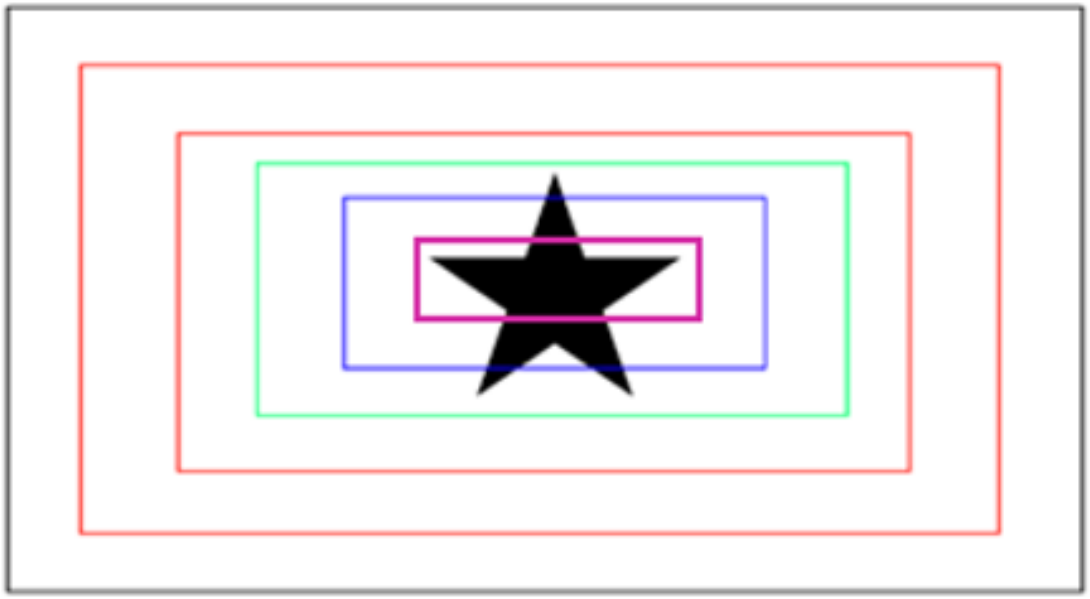

Figure 3.2: Demonstrates how the ground truth zoom values are generated as a portion of 5% image centre crops. The light-green crop, 4 from the edge represents the optimal crop for this object as it contains the object without excessive or insufficient space around it.

negative (zoom out), otherwise no zoom.

For example, the image in Figure 3.2 may be augmented during training and cropped to the purple line, 6 crops from the edge of the image. The difference between the purple crop (6) and the light-green ground-truth crop (4) is -2. Therefore, the augmented image label is updated with a zoom out direction classification and zoom magnitude bin of 2. A positive difference would result in a zoom-in classification.

If an object is heavily offset from the centre, performing a centre crop will crop the object out of the image and zoom in on background clutter instead as demonstrated in Figure 3.3. This will lead to the camera trying to centre background clutter, rather than the foreground object. To prevent this, we ensure that we only apply the random crop augmentation during training to images where the object is suitably close to the centre of the image. For the x-axis, we

use the bottom 25% of bins and for the y-axis, we use the bottom 50% of the bins as the resolution is roughly half of x. This will prevent the model from applying a centre crop before the object is suitably close to the image plane.

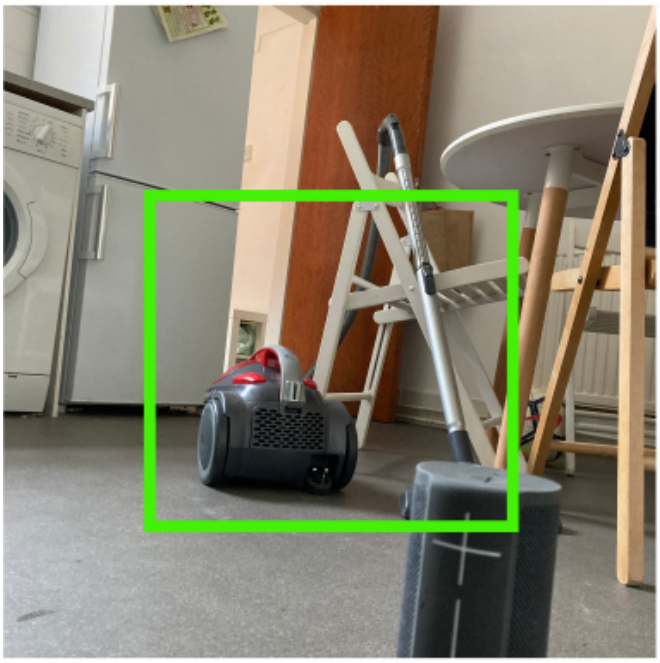

Figure 3.3: Exemplifies how cropping the image too early can lead to background clutter dominating the image rather than the salient object. To avoid this occurring, we only apply cropping to the ground-truth images with objects suitably close to the centre of the image plane. This will force the model to only zoom when the object is near the centre of the image plane.

This methodology enables the network to map input images to a zoom in or zoom out action. This can then be actuated in real-world zoom or simulated by cropping the current image.

3.1.2 Post Processing

3.1.2.1 Data sanitizing

After collecting the dataset using the automated capture process , the data was sanitized to remove misleading or erroneous images where i) the salient object was missing or occluded from the shot, ii) the calculated offset was misleading. For both problems, a program was written that enabled us to examine each image in the dataset with an overlay of the annotated offsets. If the salient object was occluded or the offset looked inaccurate, the image was removed from the dataset.

3.1.2.2 Binning

Both classification and regression approaches were evaluated for predicting offset and zoom magnitudes. For classification, we reposed the problem by discretizing the magnitude values into many small bins and use cross-entropy to measure performance against the ground truth. For regression, the direction predictions are still classified but we directly regress the magnitude values. Results of both approaches are given in Section 6.1. We started with the classification approach based on the discussion in [6] that found that discretizing the continuous action of steering actuation into many small bins to be more effective than directly regressing the parameters. The X and Y offset magnitude values have been normalized between 0 and 1, where 1 represents 50% of the total image width or height, respectively. To discretise these values into bins, there are two possible approaches: **Quantile-based discretization**: allocates bins such that there is an equal number of observations in each bin. This has the benefit that due to the equal probability distribution of the bins there is no class imbalance introduced. Adversely, the bins have unequal sizes. which means a large loss of accuracy when the bin predictions are converted back to offsets, particularly for magnitudes at

the extremes of either scale which have a much smaller probability distribution. **Fixed-size bin discretization**: This approach sets fixed, equispaced bin edges, where there is no guarantee about the distribution of observations in each bin. The drawback of this method is that the bins which will introduce some class imbalance as they do not have an equal probability distribution. However, the equal spacing of the bins means the maximum loss of accuracy when converting the bin back to the offset is limited to the width of the bin.

We chose to discretize our values into fixed-size bins as the loss in accuracy was a much greater risk to the overall efficacy of the system, whereas class imbalance could be countered by introducing a weighted loss function (discussed in Section 4.1.5).

3.1.3 SIFT

To label each offset image with a pixel offset, we applied a SIFT-based feature detector[31] between the offset and centred image as described in Section 3.1. We decided to use keypoint detectors between the images rather than more simplistic traditional features such as edge detection [47] or corner detection [30]. Our reasons for chosing traditional approaches over deep-learning methods are described in section 1.2. We chose keypoints as our traditional method because keypoint matching approaches are invariant to rotation, transformations, illumination, and scale [31]. Conversely, edge and corner detection fall apart as rotation and scale changes are introduced. The added robustness is at the expense of efficiency, however, because all the label processing happens offline, real-time efficiency was not important so long as it was less than 20 seconds. An evaluation of different traditional feature detectors in [48] found that dense SIFT detectors were the superior keypoint detectors in detecting correspondences between images. The other detectors compared include Harris-Laplace, SURF and MSER detectors,

among others. Once the keypoints have been identified between our centred and offset images, the mean difference between the matched features' is taken as the pixel offset ground truth. If no features were identified between the images, due to excessive occlusion or error, the camera is moved to a new rotation and the process tries again. This does mean that our algorithm is limited by the effectiveness of SIFT as we cannot introduce these unmatched images into the training corpus.

3.1.4 Datasets

Over the course of our experiments, we collected three datasets which vary slightly concerning the objects,T backgrounds, and capture mediums used in each. They are arranged in the same order as they were created, and as a new dataset was introduced, it was appended to the existing dataset and the model retrained. The final model was trained on the combination of all the datasets (minus the test sets.) Three examples of each dataset are given in Figure 3.4.

3.1.4.1 Uncluttered

This dataset was generated for our initial proof-of-concept. It consists of 6736, 1920x1080 images of a variety of household objects that range in shape, size, and colour. It consists of 26 objects, with just one scene. However, note that each scene in all of these datasets is taken at 2 different angles per object to ensure the network doesn't just learn a background subtraction function. The images were captured and labelled using the automated capture software running on the camera rig. All images are taken in a highly controlled, clinical environment. Ideal lighting conditions are provided with the use of a strong, overhead lamp. Objects are situated in monochromatic, white backgrounds and surfaces to reduce the complexity of the problem.

3.1.4.2 Cluttered

After proving the networks could learn valid camera offsets, we introduced images with increasingly cluttered backgrounds and surfaces. 5283 1920x1080 images were captured and annotated with the camera rig capture software. 1544 images with varying resolutions were captured on mobile phones and then auto labelled. The auto labelling is applied with a variant of the original automated capture software that instead takes the centred and offset images of an object stored in two separate directories. This means the person who captures the images should note whether the image is centred or not or manually sort them. The offset images are then matched with their centred image and the offset is calculated and recorded. 83 objects were captured, with a total of 27 different background scenes.

3.1.4.3 Plants

Our final experiment was to test our model on a dataset of plant images. 624 1920x1080 images were taken of plants in a cluttered glasshouse environment at the university, using the gimbal. A range of 14 different plants were captured, and background clutter was varied by taking pictures of each plant in-situ, resulting in 12 different background scenes. Due to time constraints, we could only collect a small number of plants which meant the network would learn a bias towards the general household objects. To resolve the imbalance, we use oversampling in our data loader to virtually increase the size of the plants' dataset equal to the number of general objects. This is explained in more detail in Section 4.5.1.

3.1.4.4 Test Sets

To provide an unbiased evaluation of each model, three test sets were captured alongside the training sets defined above (except for the cluttered test set, which

was captured separately). The test datasets present a held-out set of images that have not been used in the training or validation loops. All datasets are split at the object level, so no same object is in both train and test datasets. Each test set provides a testbed for the two objectives of adding robustness via saliency and applicability to biological datasets. Finally, the three test sets are combined into a single test set that is used in Chapter 6 to evaluate each of the models. This provides an unbiased evaluation tool that embodies the range of conditions that the camera could be used in.

3.1.4.4.1 Cluttered Test Set This dataset is very similar to the cluttered dataset described in Section 3.1.4.2, however, it contains a different variety of objects and was taken in a variety of different scenes to the cluttered training set. It consists of 508 images and is intended to demonstrate the efficacy of saliency in adding robustness to clutter. 3 different objects were captured with a single background scene (taken from 2 different angles, per object.) The fact that the dataset is collected separately from the cluttered training set, offers some assurance that we are not just overfitting to the varied, but still limited number of scenes in the datasets. See Figure 3.4 to see how it differs from the cluttered training set.

3.1.4.4.2 Uncluttered Test Set This dataset, consisting of 350 images, is the antithesis to the cluttered test set and is intended to demonstrate that saliency and non-saliency models perform similarly in controlled, uncluttered scenes – confirming the overall hypothesis that saliency is a tool to add robustness to clutter. This dataset was collected at the same time as the Uncluttered training set and consists of 10 different objects in a total of 4 different background scenes.

3.1.4.4.3 Plant Test Set This data set is made up of 235 images, taken in the same conditions as the plant dataset of Section 3.1.4.3. It consists of 5 different plants, taken in-situ. Hence, 5 different background scenes are included. It intends to demonstrate the applicability of the system to plants. This dataset was collected at the same time as the Plants training set.

This section has presented an overview as to how data is collected, annotated, and recorded, as well as describing the 3 types of dataset produced for both training and test sets. At this point, the reader should have a good idea of the data required to train the network and to run inference, as well as how to collect their own dataset if required.

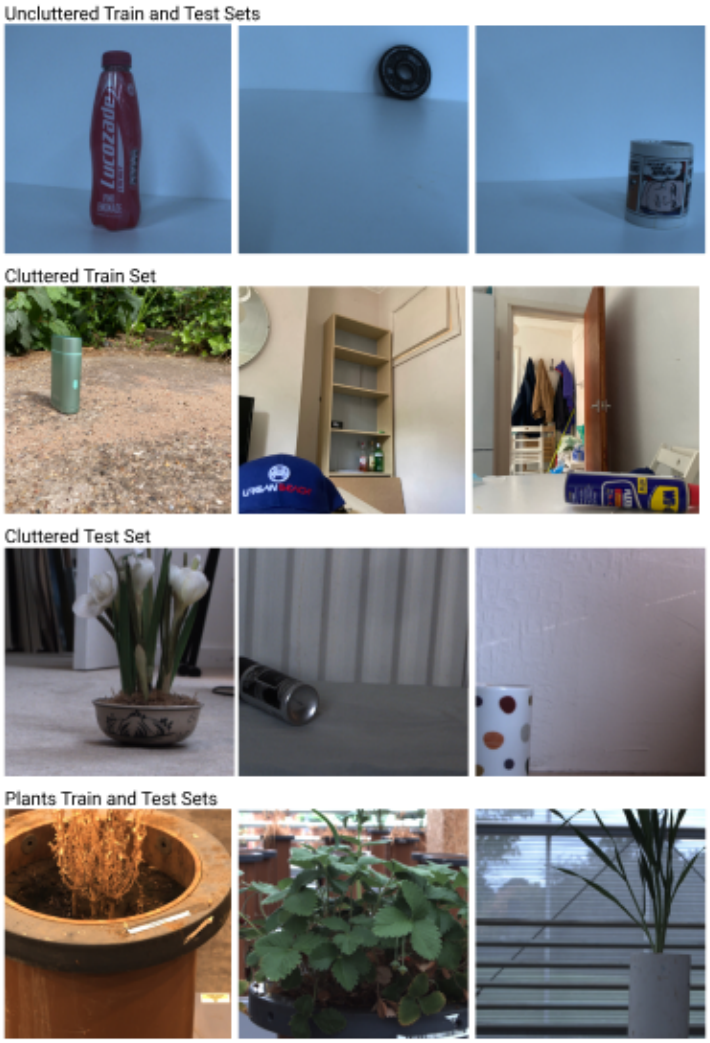

Figure 3.4: Examples from each of the three dataset categories. First row: examples from the uncluttered train and test sets. These images consist of everyday objects situated in very simple, controlled backgrounds. Second row: examples from the cluttered training dataset. These images consist of everyday objects situated in a variety of cluttered backgrounds and scenes. Third row: examples from the cluttered test dataset. These images have similar properties to the cluttered training set but are taken in a different set of environments. Fourth row: examples from the plant's train and test sets. These images consist of plants, taken in greenhouse conditions, including a variety of cluttered backgrounds as the images were taken in-situ. All datasets are split into train and test at the object level. This means no two same objects/plants will appear in both train and test sets. This is true also for the train-validation split, so no object, offset or centred, exists in both.

Chapter 4

Deep Learning Approaches

Introduction

In this chapter, we present some of the deep learning architectures and parameters that were trialled over the course of our experiments. Our network is composed of a saliency segmentation side task and a frame-to-action function approximator, which leverages the saliency mask to improve its performance. We explore different architectures and adaptions for both components. We then exhibit the multi-task loss function used to train our network and the accuracy measures used for validation and testing. Finally, we describe training details and hyper-parameters, including augmentation, transfer learning and learning rate decay.

The overall network is an end-to-end function approximator which takes the current camera frame as input and maps it to the camera offsets (including zoom) from the centre of the image, measured in pixels. The network is a synthesis of two networks, one which augments the input image with a saliency map and the other which maps the augmented image to the camera offsets. A loss function is required to backpropagate the errors for each of the predictions to update the network weights. A suitable accuracy measure is required to provide an unbiased

performance evaluation of each model.

This end-to-end approach differs from most autonomous camera algorithms which although they may use deep learning to perform object detection, usually rely on rule engines to convert the detection into camera actuation. Using this approach, we hope to achieve an increased generalisability, enabling the network to predict accurate classifications for any object that is put in front of the camera, in front of a host of different backgrounds.

The overall aim of this chapter is to provide a more complete description of the deep neural network discussed briefly in Chapter 2. The descriptions are specific enough that the reader can replicate our results, while the discussions on alternate approaches can guide future works on what does not work well.

4.1 Frame-to-Action Network

The frame-to-action network takes the input frame of the camera, resizes it to 224x224 and sends it through a forward pass of the network, outputting the offsets and zoom parameters required by the camera to better encompass the salient object in the scene. The pan, tilt and zoom directions are posed as classifications while the magnitude of the movements can be posed as a direct regression or by discretizing the offset magnitudes into bins and posing it as a classification problem. For both approaches, a slightly different loss function is required for training. Another consideration for the frame-to-action network is the choice of architecture. We experiment with ResNet and MobileNet architectures, along with their variants, see Section 4.1.3.1. We also explore the loss functions we employed which includes weighted class balancing techniques to avoid the model overfitting to overrepresented classes in the dataset.

4.1.1 Continuous Approach

In this approach, we attempt to regress the offset (pan, tilt, zoom) magnitude labels based on the input image. As described in Section 3.1, the X and Y offsets are recorded as normalized values $x, y \in [-1, 1]$, where 1 represents 50% of the total image width or image height, respectively. The zoom label is denoted in units of 5% crops of the image. To implement this approach, we still classify the direction for x, y and zoom but the fully connected layers of each of the magnitude predictions are adapted to output scalar values, as opposed to a multinomial distribution. This approach will output a continuous value for each of the magnitude predictions. We then calculate the loss between the predicted magnitude value and the ground truth at each batch using the root-mean-square error (RMSE). We then combine the RMSE loss from the magnitude predictions and the Cross-Entropy loss from the direction classifications into a single loss that is backpropagated through the weights. To ensure neither direction nor magnitude predictions dominate the combined loss, a weighting factor is assigned to each of the losses, described in Section 4.1.5.2.

4.1.2 Discrete Approach

In this approach, we discretize the magnitude values into fixed-width bins as described in Section 3.1.2.2. The fully connected layers are then adapted to output a multinomial distribution, based on the number of bins in each classification. To train the network, we use a cross-entropy [49] loss function. This method outputs a multinomial distribution. Argmax is then applied to return the bin class with the largest predicted probability for each task.

4.1.3 ResNet

Since 2015, the ResNet architecture has become the de-facto base model for computer vision tasks including image classification, semantic segmentation, and object detection. In these tasks, the architecture achieves excellent accuracy and generalisation performance [50]. The key to ResNet's success is its ability to train with hundreds or even thousands of layers without suffering from the 'vanishing gradient problem' [51]. This is the problem where, as the number of layers in a network is increased, the repeated multiplication of the error gradient with each neurons' local gradient during backpropagation causes the gradient to become increasingly small. ResNet overcomes this by introducing the use of so-called identity shortcut connections. The identity shortcuts prevent prevents gradient atrophy when multiplied with the local gradients as the identify function's gradient is always 1.

The large capacity for layers enables increased model complexity and representational ability. Ergo, we chose ResNet as the base model for our task, which essentially is an image classification exercise.

4.1.3.1 ResNet Variants

The ResNet has several variants that relate to the number of layers in each model. For example, resnet34, resnet50, resnet101 and resnet154. As the number of layers in a CNN increase, so does the model's representational capability, with each layer learning progressively more complex, high-level features. However, more layers also mean increased model complexity, leaving us with a trade-off between accuracy and inference time (storage space is negligible for our use case). With these constraints in mind, we chose the resnet101 model for our task as it offers good performance and suitable inference times. See Section 6.1 for a detailed comparison of the models.

4.1.4 MobileNet

We also implemented an alternative, lightweight MobileNet_v2 [52] architecture. The architecture uses depthwise separable convolutions to reduce the number of parameters at each convolution step. Similarly, to the ResNet architecture, the network uses a shortcut connection in its block structures to increase the depth of the network. These 'inverted residual blocks', act like regular residual blocks except that the skip connections are between the thin bottle-neck layers, rather than the wide layers.

The low number of parameters required for the architecture enables us to reduce the size of the network and inference time for real-time operation. However, the complexity of the saliency network, which requires a GPU to run in real-time, creates a bottleneck and makes the time-saved on our classifier redundant for saliency-enabled models.

4.1.5 Multi-Task loss function

4.1.5.1 Classification

To train the network, we use a multi-task loss function to measure the error between the output predictions and their respective ground truths. For the discrete classification task discussed in Section 4.1.2, we use weighted cross-entropy loss to measure the divergence of each task from its respective label. The loss function is defined in equation 4.1

$$WCE(y') = -\frac{1}{K}\sum_{i}^{I}\sum_{k=1}^{K} w_k y_{ik}\log(y'_{ik}) \qquad w_k = \frac{1}{|y_k|} \tag{4.1}$$

where WCE denotes the weighted cross-entropy, I is the number of training examples (batch items), K is the total number of classes, y' is the output of the network, y'_{ik} is the output of the network for training example i, class k. y are the ground truths, y_{ik} denotes the target label for training example i, class k. log denotes the natural logarithm and w_k denotes the weight assigned to class k.

The cross-entropy loss function penalises probabilistic false positives. If for example, a given training example at class k has a probability of 0.6 and the ground truth is 1, from a Bayesian perspective, the prediction has a probabilistic false negative of 40%. To penalise this false negative, the natural log is used which incentivises values closer to 1 and penalises values closer to 0. i.e. in the perfect case: $\log(1) = 0$ [53].

Our dataset contains class imbalances. For example, instances predicting no movement are underrepresented as there is only one perfectly centred image per object. Figure 4.1 demonstrates the underrepresentation of the no-movement class (0) of the pan(x) and tilt(y) direction classification. Magnitude distributions are more uniform, but there is still a decreasing distribution towards extremely high magnitudes. To prevent overfitting to the dominant classes, we add a weighting factor w_k which assigns more weight to minority classes and over-penalises instances of the underrepresented classes. The weights are defined by the inverse of class support for each class. The intuition behind this is that under-represented classes will be penalised more aggressively and hence have a larger impact on the network weight adjustments. This is equivalent to increasing the ratio of the under-represented classes in the dataset.

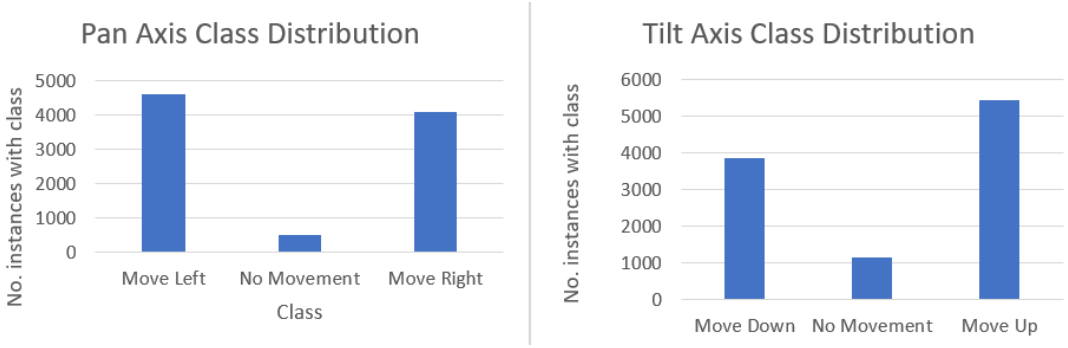

Figure 4.1: Demonstrates the class imbalances within the pan and tilt classifications. The underrepresented examples of centred images versus offset images is the cause of both.

4.1.5.2 Regression

For the continuous regression approach described in Section 4.1.1, the loss function is given in equation 4.2

$$\begin{aligned} Loss &= L1 + (0.125 * L2) \\ RMSE\,(y') &= \sqrt{\frac{1}{n}\sum_{i}^{I}(y - y\prime)^2} \end{aligned} \tag{4.2}$$

Where L1 is the sum of Cross-Entropy losses for x_{dir}, y_{dir} and $zoom_{dir}$. L2 is the sum of RMSE losses for x_{mag}, y_{mag} and $zoom_{mag}$. The L2 loss is balanced with the value 0.125 as it has a similar ratio to the L1 and L2 losses produced during experimentation.

In the RMSE equation, instead of summing the log loss for each class, we calculate the root mean square error for the scalar predictions in the batch. Because the value is directly regressed, there are no classes, so we do not use class balancing. On the other hand, since the x and y offsets are normalised, and the zooming

magnitude is not, we need to add a weight to each of the individual losses of each fully connected layer to balance their overall influences on the combined loss function (0.125).

The gradient of the error rate produced by the loss functions is then back-propagated through each layers' weights, which are updated accordingly to yield an incremental improvement to the error in the next forward pass.

4.2 Saliency Network

This section introduces the saliency attention network we use to add robustness to background clutter. The saliency network takes the cameras' input frame, performs a forward pass through the network and outputs a saliency mask. Each scalar value in the mask represents the conspicuity or visual allure of that location. The features that represent visual allure were learnt by the network as it trains, comparing each mask that it outputs against a human-annotated ground truth of the image saliency such as the dataset in [54]. In our preliminary experiments, we found our frame-to-action network was apt at recentring objects that reside in very simple backgrounds. However, we found that the accuracy decreased as new datasets were introduced with increasingly complex backgrounds. To tackle this, the frame-to-action network is synthesised with the output of a pretrained saliency network. By concatenating the input image with a saliency mask, we can isolate the salient object from their backgrounds. This enables us to guide our frame-to-action network to learn features more relevant to salient objects during training. Two example outputs of the saliency network are given in Figure 4.2. Because we use channel-wise concatenation to augment the input image and saliency map, no adjustments need to be made to the ResNet layer's dimensions, other than ensuring the kernels in the first convolutional layer have the same

depth as the input volume, which in this case is 4. This will ensure both the filter and the local region of the input volume (the receptive field) will have the same dimensions when flattened. Since a convolution is essentially a dot product between the filter and the local region of the input, it can be formulated as a dot product of the two flattened vectors. For example, a 7x7x3 filter and 7x7x3 local region of the input can both be flattened into two 1x147 column vectors and a dot product applied to the two vectors gives us the convolution result for a single feature map pixel.

The saliency network we use is a pretrained model based on Pyramid Feature Attention Networks that utilises multi-scale features. The network integrates high-level contextual features and low-level spatial features to optimise the edge boundaries of the saliency map and achieves SOTA performance [10]. The low and high-level features are extracted from a block of convolutions based on either a VGG or ResNet backbone architecture. We chose the ResNet backbone for improved accuracy.

4.2.1 Thresholding

Inferring from the residual learning principle specified in [37], we consider $S(x, W_i)$ as a forward pass through the saliency network with W_i denoting the networks' weights and x the RGB image input. The output of the saliency network is then concatenated with the original image to create input $y = S(x, W_i) + x$, which is input to our CNN. The concatenation of the saliency output with the RGB input means if no saliency is produced, y will present an identity mapping $S(x, Wi) = x$. From this logic, we infer that our classifier augmented with saliency will achieve performance that is at least as good or better than the non-augmented input (unless the saliency map is misleading).

However, having amalgamated the saliency network into ours, we reran our

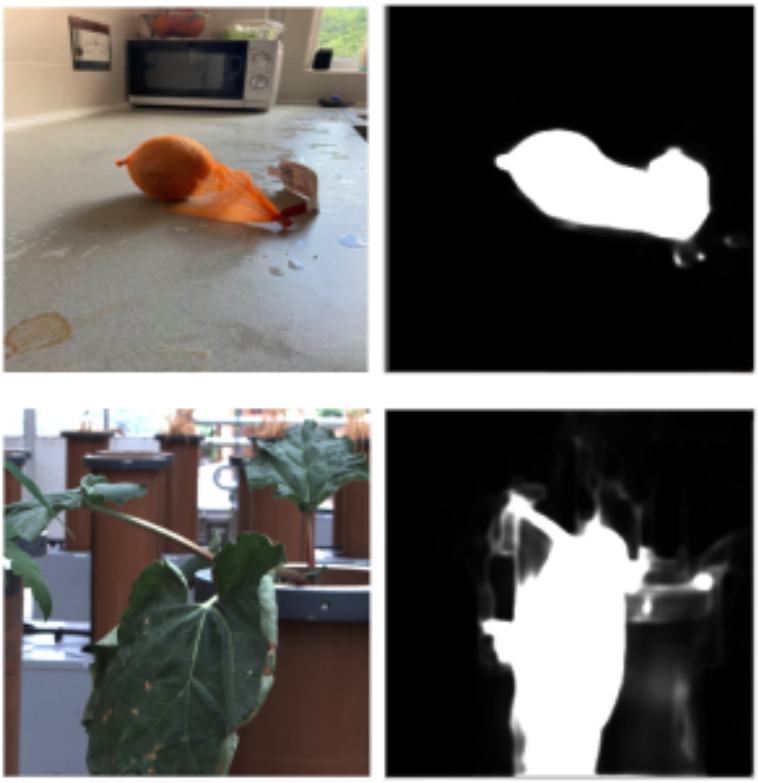

Figure 4.2: Two examples of the saliency masks produced by our saliency network. Top: saliency mask produced for an instance from the cluttered training dataset. The centre onion is approximately 7.62cm in diameter. Bottom: the saliency mask produced for an instance from the plant's dataset. The mask seems to highlight the most prominent leaf in the frame. For reference, the plant pot is 30cm in diameter.

experiments to find that the results were significantly worse with saliency than the original network. After further investigation, we found that most of the misclassified instances had produced a seemingly blank saliency map. We found this puzzling as we inferred that introducing no new information in the saliency map should revert the network to learning the original, non-saliency mapping. On closer inspection, we found that the blank maps were not actually null matrices but had very small activations that were indistinguishable by manual observation. We hypothesized that these uninformative values were introducing noise into the

loss and damaging learning. To minimise the noise, we introduced an intensity threshold that would zero the mask if the total mask intensity were less than 1% of its maximum capacity. This enables to model to achieve a performance equal to or greater than non-saliency. This holds so long as the saliency map is valid – see the evaluation in Chapter 6 for further detail.

4.3 Accuracy Measure

To evaluate the model's performance, an effective accuracy criterion had to be developed that was representative of the systems' real-world performance. Initially, the network output two classifications: one dictated the direction of the camera movement, with nine classes: up, down, left, right, up-left, up-right, down-left, down-right and no movement. The other classification decided the magnitude of the movement. The direction classification was split in this way to provide granular enough movements that all positions can be reached eventually, but be coarse enough so as to constrain the problem's complexity To measure accuracy, we focused solely on the accuracy of these two metrics but quickly found that the results were not representative of how well the algorithm worked in practice.

On closer inspection, we realised that evaluating whether the movement was suitable at a given frame was not always a binary decision, as the camera could still make small improvements to its position through a series of sub-optimal movements. For example, if the object is in the top-left corner of the image, the optimal direction for the camera to move would be up-left, to recentre it. However, if the camera were to move just left or just up, the new position of the camera will still be closer to the target position than before. By observing the operation of the camera, we realised that these kinds of sub-optimal movements occurred quite regularly in practice and would still culminate in a valid final

position.

We performed an analysis of the misclassified instances over our validation set, presented in Figure 4.3. The diagram shows each misclassified instance from a sample of 450 images. Each point demonstrates the position of the object in the image frame for that instance. The arrow on the point shows the direction that was predicted by the model. The diagram supported our theory since most misclassified instances correctly classified one of the axes but not the other, e.g. predicted left-down, rather than left-up. This was particularly prevalent close to the axes of the image plane, where decisions could be more ambiguous.

To overcome this shortfall, we decided we would get a more expressive metric of accuracy if we separated the direction prediction into two classifications. This would enable us to provide a separate loss for each axis, which were combined into the final loss function. With this loss function, incremental improvements are still rewarded by the loss function though not as highly as optimal solutions. Completely inaccurate classifications, where both axes are incorrect, are penalized more severely. It also enabled us to develop a more realistic accuracy measure for use in testing. Instead of measuring a single axis prediction, we split it into two separate predictions, we three classes each and measure the accuracy of both. We also record the number of instances that got both x and y direction predictions wrong and record the inverse of this as the new model accuracy measure. This is a more accurate way of monitoring how well each model performs in practice, noting every incremental improvement as an accurate prediction.

In addition to the direction predictions (x, y and zoom), a magnitude for each movement is output from the network. In the classification approach 4.1.2, a simple accuracy measure is to count the number of magnitude classifications that were correct as a percentage of all predictions. Although this provides an insight into how accurate the magnitudes are, a more expressive metric was required to

provide insight into 'how far' each prediction was from the correct class - i.e. how far the algorithm over or undershot the camera movement. Therefore, we measure the accuracy of the magnitude predictions using RMSE. This measures the error between the predicted class and the ground truth, having converted the magnitude integer to positive or negative based on the direction prediction. This gives us a more informative indication of how wrong the magnitude classification is, as each class presents just a small amount of movement (0.0333 and 0.066 for x and y, respectively).

4.4 Augmentation

To increase the robustness of the model, we adopt some augmentation techniques to increase the variability in the dataset. Random horizontal flip is implemented to increase the invariance of the network to different object rotations. This improves generalisation performance when the algorithm is applied in different conditions, including new objects and backgrounds. Similarly, random colour jitter including brightness, contrast, saturation, and hue changing is applied to increase robustness to these variables. This is particularly important if the algorithm is to generalise to new settings and different cameras, where image capture settings may vary. As described in Section 3.1.1, a random crop augmentation is applied to the images during training. This enables the network to simulate a zoomed-in instance of an image and produce a zoom-out prediction for the network to train on. The final augmentation that occurs before/if the image is passed to the saliency network is pixel rescaling. This performs local pixel centring by subtracting the mean RGB values of the dataset provided in [10] from the image's respective channels, giving pixels a zero mean. The MobileNet architecture requires us to normalize the images' pixel values based on the per-channel mean

and std. dev of the ImageNet dataset.

4.5 Training Details

Our dataset of 12,537 images is split with a 75:25 train-validation ratio, however, images of the same object must not appear in both datasets, so the train set is rounded down to the closest object, leaving 9528 images in the training set and 3009 in the validation set. Independent, task-specific test datasets are collected for evaluating each model as discussed in Section 3.1.4.4 of 508, 1561 and 235 images. All images are resized to 224x224, before being processed by the model. We chose this resolution based on constraints on training time and GPU memory and is important if the system is to run in real-time in practice. Additionally, the use of transfer learning, described below, requires our resolution to be the same as the pertained ImageNet model. Transfer learning is applied by initializing the ResNet's weights based on a pre-trained ImageNet [55] model. This enables us to leverage the rich image feature representation learnt from the large benchmark dataset and then fine-tune them to our specific problem. This accelerates training and significantly improves accuracy by 6% as we demonstrate in the evaluation Chapter 6. Random horizontal flip, colour jitter and centre crops are applied to the training set. Random centre crops are the only augmentation applied to the validation set. The models were trained on an NVIDIA TITAN X Pascal GPU with 3 CPU cores for each job.

The models are trained end-to-end using Stochastic Gradient Descent with momentum of 0.5. We use a multi-task cross-entropy loss function with an initial learning rate of 1.0x10-3, which is decayed by a factor of 10 halfway through training to refine the local optima and converge faster. Training is run for 20 epochs and the best performing model out of all the epochs is selected. The du-

ration of training is approximately 3 hours for a ResNet-101 model. We trialled three different batch sizes: 4, 16 and 32 and found that the larger batch sizes had lower accuracy but much quicker training times, taking almost half the time as it allows for increased parallelisation. Figure 4.4 demonstrates the accuracy of the different batch sizes. This is supported by [56], who attribute the superior generalisation performance of small batches to their ability to find flat minima, which generalise to new data better than sharp minima. They explain that small batches introduce more noise in error calculations than large batches, which enables them to exit the loss valley of sharp minima and find flat minimizers further away.

4.5.1 Oversampling Plants

Plant images are significantly underrepresented in the dataset, so to better demonstrate our applicability to biological imaging, a resampling technique is used that adds bias to the plant images. We artificially double the size of the dataset and then resample any indexes from the mock second half of the dataset to the index of an image from the plant dataset to increase the yield of plant images. This approximates increasing the number of plant images without having to physically collect any new data. This helps to reduce bias and improve generalisation.

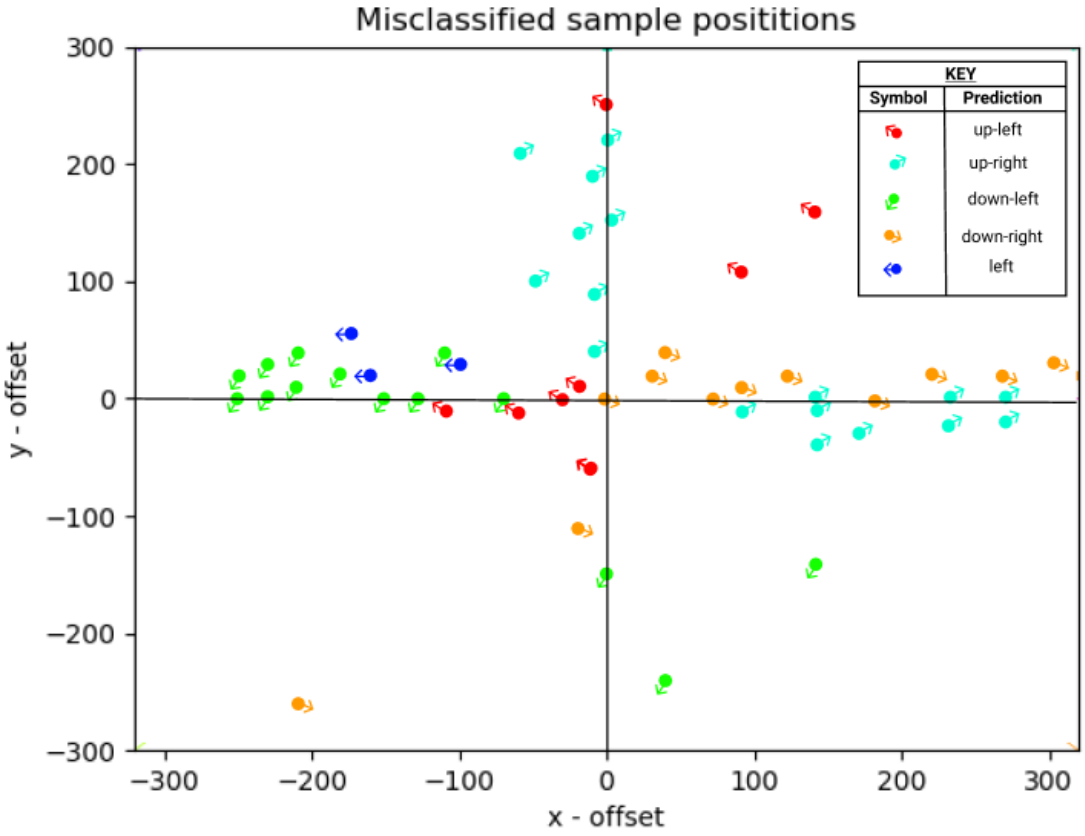

Figure 4.3: A subsample of misclassified instances were taken from the original 9-class direction prediction, before x and y-axis directions were divided into separate classifications. Sample size = 55. The grid depicts the camera's image plane, and the point where the cross-hairs meet is the camera's principle point. Each point on the grid represents a misclassified image instance and its position is relative to the image plane. The key in the top-right corner defines the direction classification of each instance, related to the colour and arrow direction of each point. e.g. For red instances, the camera moves up and left, moving the instances at the top left of the image plane towards the centre crosshairs. In all instances. at least one of the X or Y direction is correct but the other is not. Also, most instances are close to the axes where decisions are more ambiguous. Therefore, we split our X and Y directions into separate classifications to get a more realistic measure of accuracy.

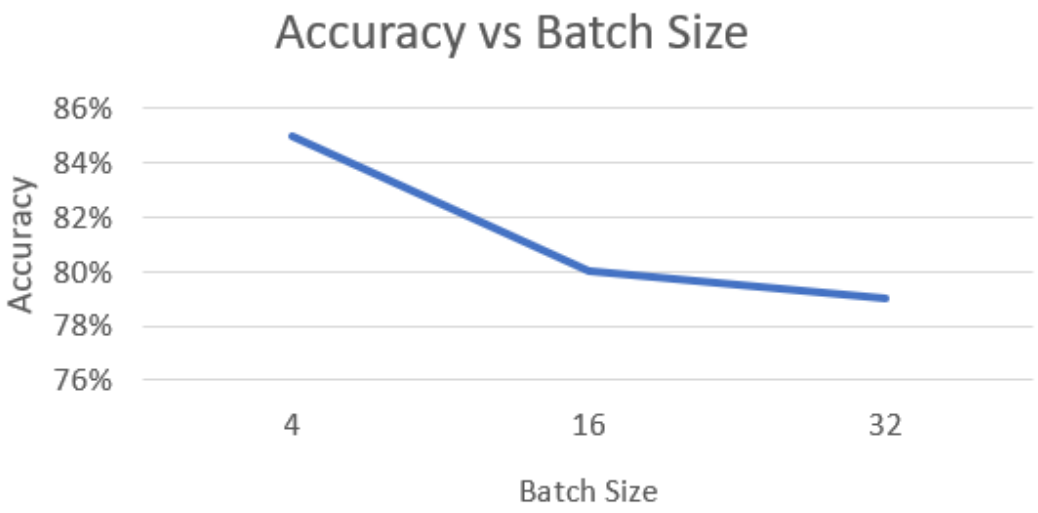

Figure 4.4: Illustrates the accuracy of the ResNet-152 model as the batch size is increased from 4 to 16, to 32.

Chapter 5

Generalisation

Introduction

One of our aims for the model was to make it as general as possible. Initially, this meant the ability to generalise to different objects and recentre whatever is put in front of the camera. This was achieved by introducing a greater variety of objects into the training corpus including everyday household objects and plants, which is described in more detail in Section 3.1.4. The next objective was to generalise the model to different hardware, specifically the camera-gimbal rig. Initially, the model outputted camera offsets in the specific units of our rig e.g. 1 unit in the X-axis = 0.107°. This will not generalise well to new rigs as they may have their own, hard-coded unit system to represent axis rotation. To fix this, we wanted to output offset predictions from the network in pixels which are common to every camera, we could then write a program to convert these pixel offsets back into the rig-specific units. This was achieved by using a SIFT-based feature matching algorithm to generate offset ground truth labels in pixel values rather than rig-specific gimbal units. The model is then trained to output these pixel values. To make sure the pixel values were not tightly coupled with the

image resolution of our camera, we normalise all offsets between 0 and 1 so they will extrapolate to different resolution cameras. More details on this approach can be found in Section 3.1.3. The final step that is required then is to convert these pixel values back to the rig's specific units, which we explain in the next section, Camera Calibration.

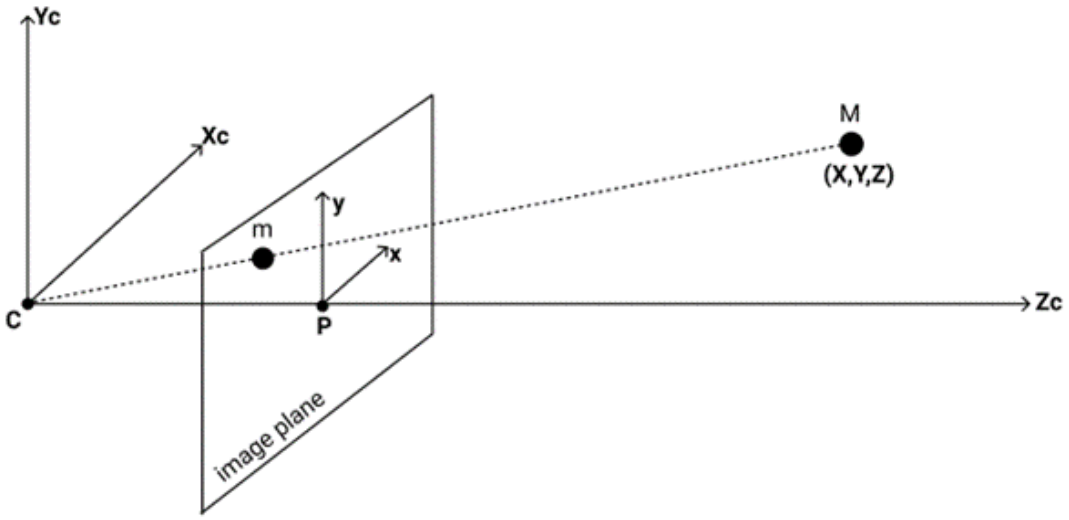

Figure 5.1: Diagram demonstrating how a point in the 3D world space is projected through the camera coordinates and onto the 2D image plane. Camera coordinates are denoted by Xc, Yc and Zc and has the camera's optical centre C at its centre. The world coordinates are represented by X, Y, Z and represents the 3D object coordinates e.g. the calibration corner coordinates in mm. The image coordinates are denoted by x,y on the image plane and are measured in pixels. P represents the principal point of the image plane.

5.1 Camera Calibration

Now that we have the network producing the cameras' X and Y-axis offset in pixels, a method is required to convert the pixel offsets back into gimbal actuation units for our specific camera-gimbal rig. Most gimbal setups are programmed to receive actuation commands in terms of an angle, measured in degrees or radians.

Our setup has a unit of movement for the X-axis stepper motor that equates to 0.107° and 0.057° for the Y-axis motor. To convert the pixel offsets into a portion of these units, we need to estimate how many degrees each pixel equates to. This is achieved through a process of camera calibration.

Camera calibration aims to determine the relationship between the camera's natural units, in pixels and the real-world units in degrees. To do so, we need to estimate the cameras' intrinsic and extrinsic parameters. With these coefficients, we can project the pixel offsets in 2D image coordinates to the 3D rotation of the camera in the world coordinate system. This enables us to calculate the correlation between a single-pixel offset and its corresponding rotation in degrees.

To describe the projective aspects of the camera, we use the pinhole model. This model enables us to project from the world to camera, to image coordinates. The difference between the three coordinates systems is described in Figure 5.1. Based on the principles of similar triangles, the world point M, depicted in the figure is proportional to the image point m, and as such, the 2D image plane coordinate $\boldsymbol{m} = \begin{bmatrix} x & y \end{bmatrix}$ is defined in Equations 5.1 and 5.1

$$x = \frac{Xf}{Z} \tag{5.1}$$

$$y = \frac{Yf}{Z} \tag{5.2}$$

$\frac{f}{Z}$ acts as a scaling factor here, which when multiplied by the object's 3D X or Y coordinate, projects the coordinate to the 2D image plane (x or y). However, since the projection of X means dividing by Z, the expression is no longer linear and cannot be represented as a matrix multiplication. To remedy this, we can write the equation linearly using homogeneous coordinates. The linear form is

given in Equation 5.3.

$$s \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{5.3}$$

s is an arbritrary scaling factor, which maps to Z. f is the camera's focal length, measured in pixels. Thus a a camera can be considered as a system that performs a linear projective transformation from the projective space $\mathbb{R}^{\mathbf{3}}$ into the projective plane $\mathbb{R}^{\mathbf{2}}$. The above expression assumes that the origin in the image plane is at the point where the Z axis hits the plane's principal point. In most cases, this is not true, so we introduce principal point offsets c_x and c_y. Hence, Formulas 5.1 and 5.2 change to:

$$x = \frac{Xf}{Z} + c_x \tag{5.4}$$

$$y = \frac{Yf}{Z} + c_y \tag{5.5}$$

Again, to write this in a linear form we homogenise the coordinates and use the intrinsic matrix K, defined in Equation 5.6 to project the points. The intrinsic matrix represents the focal lengths f_x, f_y measured in pixels and optical centre offsets c_x, c_y of the camera and enables the projection of a point in camera coordinates X_c, Y_c, Z_c into pixel coordinates x, y. It can also contain a skew coefficient to account for the axis-skew of non-rectangular pixels. However, most modern camera's axes are at 90° to each other, so we have ignored this coefficient. Also note that we simulate zoom with image cropping, so focal length will remain

unchanged if the camera zooms in or out.

$$\boldsymbol{K} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{5.6}$$

We can now insert K in place of the projection matrix given in equation 5.3. So far, we have assumed that the origin of the 3D coordinate system is located at the pinhole. To project the world coordinates to the camera coordinates, we need to apply rotation and translation to the points, to match that of the camera. To do so, we apply the rotation and translation matrices to the 3D world coordinates. The Rotation matrix is part of the extrinsic parameters of the camera and represents the rotation of the camera in a 3D scene, with the origin of the coordinate system at the cameras optical centre. This matrix enables us to transform a point in world coordinates X, Y, Z into camera coordinates X_c, Y_c, Z_c. The translation vector defines the translations required the world coordinates into camera coordinates. This gives us 6 degrees-of-freedom: 3 in the X,Y,Z rotation axes and 3 in the X,Y and Z translation axes.

Hence, to project a 3D point $\boldsymbol{M}$ to its 2D image projection $\boldsymbol{m}$, we use the formula in Equation 5.7.

$$\boldsymbol{m} = \boldsymbol{KRM} \tag{5.7}$$

Where $\boldsymbol{K}$ is the intrinsic matrix from 3D camera to 2D image coordinates and $\boldsymbol{R}$ is the rotation matrix from 3D world to 3D camera coordinates.

In order to calibrate our camera, we need to find the intrinsic and extrinsic parameters of our camera to be able to project a 3D world point to its corresponding 2D image position and vice versa. To estimate the intrinsic and extrinsic param-

eters of our camera-gimbal setup, we need a set of 3D real-world points (object points) and their corresponding 2D projections (image points). To achieve this, we use a well-defined checkerboard calibration pattern with a known grid size and dimensions, illustrated in Figure 5.2. We start by taking a picture of the pattern centred in the image plane. We move the camera in a series of different rotations taking a picture at each position while the pattern remains static. We assume the pattern is a planar object by keeping the camera stationary along the Z-axis (Z=1). At each position, we record the object points and image points of the pattern in its new location. The object points are locations where two black squares touch each other on the board. The points are measured in the patterns' coordinate space, which in our case, where the 6x9 grid has 24.5mm grid squares, will increment in multiples of 24.5mm e.g. (0, 0), (24.5, 0), (49, 0)..(196, 122.5). The image points are the corresponding pixel locations of these points in the image plane. We tried to take pictures of the pattern from as many angles as possible by the camera to ensure a good spread of points and to make sure that the calibrated parameters generalise well to the positional extremes. As the camera's tilt increases towards, the pan will have a decreasing impact, so it is important to model these characteristics during calibration. This is how we collect our calibration points. We also record the pixel offsets of each pattern from the initial, centred pattern to cross-reference with the rotation matrix later. To do so, we use the same SIFT-based matcher as described in Chapter 3.1.3.

To project a 3D point in the camera coordinate system to a 2D homogenous image coordinate we first need to calibrate the intrinsic matrix $\boldsymbol{K}$. The formula for the projective transformation using the intrinsic matrix is given in Equation

5.8

$$s\begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} \tag{5.8}$$

The left-hand side represents a point in the 2D image plane using homogenous coordinates x, y, w, where $w = 1$ or $w = Z$. The matrix operation represents a system of linear equations where f_x, f_y, c_x and c_y are the unknowns and with every correspondence in the calibration points resulting in a new equation. To solve the equation, we need at least 2 image/object point pairs [57], however, using 10 or more correspondences (we used 50) will form a more well-posed equation and improve accuracy. Based on the point correspondences and using the homographies, there are closed form solutions that determine the intrinsic camera parameters. We used Singular value decomposition (SVD) to find the least-squares solution to the homogenous system [58]. Implementations of the algorithm can easily be found in common computer vision libraries such as Open-CV. Once we solve the equation, we have an approximation of the intrinsic matrix, common to all views of the calibration image.

Once we have an initial estimate for the intrinsic matrix, we can try and solve the complete equation for a typical pinhole camera model, given in 5.9. This enables us to project a 3D world coordinate to 2D image coordinate.

$$s\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} \tag{5.9}$$

s on the left-hand side is an unknown scaling factor which maps to the Z_c coordinate of the point in the camera coordinate system. This is required as objects in the image plane seem bigger if the 3D object is closer to the camera. The right side, $\boldsymbol{KR}$ is the projection matrix which maps a 3D point in the world coordinate system (i.e. a grid corner) into 2D image coordinates. It consists of the extrinsic matrix $\boldsymbol{R}$ (rotation and translation coefficients) and the estimated intrinsic matrix $\boldsymbol{K}$ we solved in the previous step. The extrinsic matrix represents a rigid transformation from 3D world coordinates to the 3D camera coordinate system. The intrinsic matrix represents a projection from the 3D camera coordinates to the 2D image coordinates [57].

Because the transformation between the camera coordinate system and image coordinate system loses the Z dimension, we add the scale factor s. This makes the equation non-linear and much harder to compute. This problem is defined as the Perspective-n-Point (PnP) problem. To solve it, we employ the Levenberg-Marquardt optimization function [59] to minimize the reprojection errors of our object points. See Equation 5.10

$$\boldsymbol{K}, \boldsymbol{R} = \text{argmin} \sum_{i}^{I} \|\boldsymbol{m_i} - \hat{\boldsymbol{m}}_{\boldsymbol{i}}\|^2 \tag{5.10}$$

Here, I is the total number of point correspondences. $\boldsymbol{m_i}$ is the known projection of object point $\boldsymbol{M_i}$ and $\hat{\boldsymbol{m}}_i$ is the projection of object point $\boldsymbol{M_i}$ by projection $\hat{\boldsymbol{m}}_{\boldsymbol{i}} = \boldsymbol{KRM_i}$. We then minimise the reprojection error (sum square distance) between the image point ground truths and the projected object points which we project using the current estimates for intrinsic and extrinsic parameters [57].

With the optimization complete, we now we have the rotation vectors at each pattern offset measured in radians. We convert them to Euler angles (ZYX Tait–Bryan angles to be precise), measured in degrees. With the known rota-

tion angles and pixel offsets between each pattern and the initial centred pattern, we can calculate the degrees per pixel by dividing each axes' pixel offset from the centred pattern by its corresponding rotation in degrees. Each axes' degrees per pixel value is then averaged over every position we collected to get an accurate measure.

Because the camera is a perspective projection, a camera movement in the pan-axis will result in objects that are closer to the camera moving more than objects that are further away from the camera. Therefore, when the camera's tilt is increased, the pan will have a decreasing impact. This occurs when we capture images of the calibration camera from different angles and certain camera angles (e.g. top-right) will produce a much smaller scaling factor than the camera angle where the calibration pattern is close to the object (e.g. centred image). However, because we use a very large (>50 images) range of positions while calibrating the image and take an average of the degrees-per-pixel for each axis, we essentially take the average of all the scale factors which is sufficient to get an accurate estimate at all camera angles. However, it should be noted that this is an assumption that we are making.

Another assumption we are making during calibration is that the camera is positioned parallel to the calibration pattern. Specifically we mean that there is negligible rotation in the roll axis of the camera. This assumption is enforced by the fact that we calculate the degrees-per-pixel by dividing the predicted axes rotation by the pixel offset generated. This means that we are comparing x degrees of gimbal pan against x degrees of pixel movement. If the camera is rolled, then gimbal pan and pixel movement will no longer align as x gimbal pan will result in a combined x and y pixel offset when the camera is rolled (also the case for the y-axis). Therefore, to provision for this possibility in the future, we should divide the gimbal angle by a combined magnitude of the x and y pixel

offsets. However, for this use-case the camera is close enough to parallel that we need not address it.

Now we have the degrees per pixel values for both X and Y axes for our given camera-gimbal setup. These coefficients are saved to a file to be loaded during inference. When the network outputs a camera offset prediction in pixels, the value is converted into the rig-specific pan and tilt rotations, measured in degrees (or whichever angle quantification the rig uses). This means the network can work with any PTZ camera-gimbal rig. This extends the generalisability of our algorithm not just to different objects and scenes but also different hardware setups which will be useful when the system is deployed in production. Note that the calibration step only needs to be run once per camera-gimbal rig.

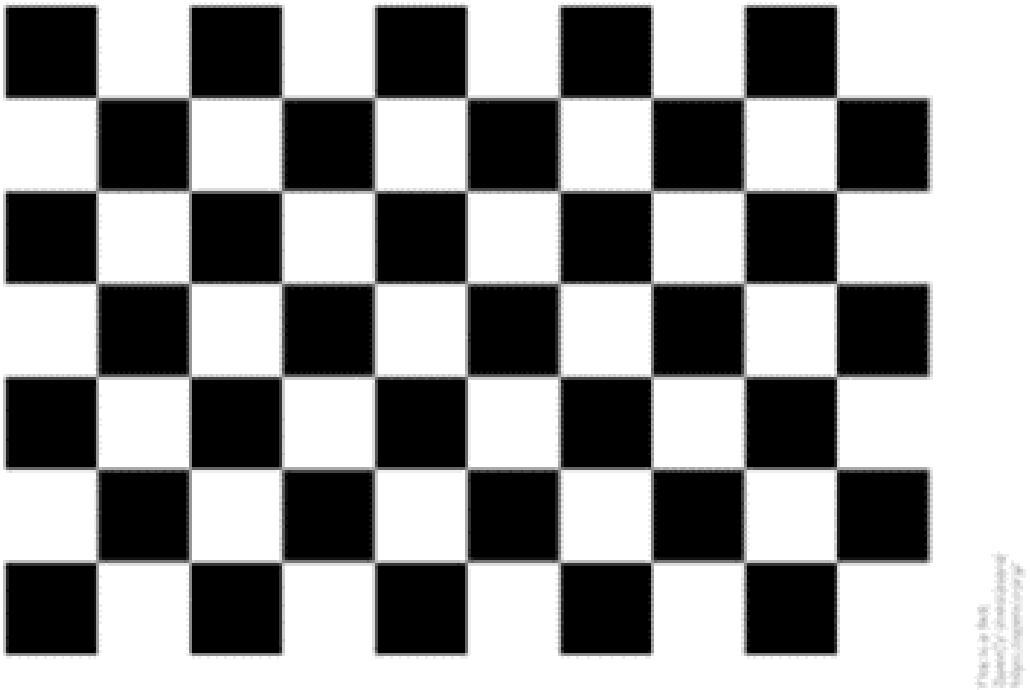

Figure 5.2: The 9x6 calibration pattern used in the camera calibration step. Each square is 24.5mm.

Chapter 6

Evaluation

Introduction

We built an autonomous camera system that directly maps the input frame of the camera to a pan-tilt-zoom coordinate to recentre the object in a static scene. In this section, we present the results of the experiments we conducted to analyse and evaluate our approach. We aim to assess our systems' success in solving the *active* autonomous camera control problem for a static, biological scene. We do so by describing the overall performance and efficiency of each model architecture that was tested to scrutinise each architectural decision and compare alternate approaches. We then evaluate the effectiveness of saliency in adding robustness to background clutter and finish by demonstrating the systems' applicability to plants. The criteria by which we judge our results are the accuracy measure described in Section 4.3, the X and Y-axis direction accuracy, magnitude RMSE's, and the inference time of each of the models on the GPU. All experiments are carried out on the combined test set of 1065 images, consisting of both cluttered, uncluttered and plant images. We also present qualitative evaluations of the camera paths and saliency maps produced, to help elucidate the figures and

Model	Saliency	Transfer Learning	Model Accuracy	X accuracy	Y accuracy	X_mag RMSE	Y_mag RMSE	Zoom RMSE	Inference Time(s)
CLASS_RESNET_101	**yes**	**yes**	**87%**	**80%**	**67%**	**5.0**	**3.0**	**2.9**	**0.27**
REGR_RESNET_101	yes	yes	82%	73%	67%	5.4	3.5	2.4	0.20
CLASS_RESNET_101	yes	no	84%	73%	67%	5.6	3.7	3.4	0.27
CLASS_RESNET_101	no	yes	83%	72%	69%	6.0	3.7	2.9	0.06
CLASS_RESNET_50	yes	yes	83%	74%	69%	5.2	3.2	2.3	0.26
CLASS_RESNET_152	yes	yes	85%	77%	71%	5.3	3.5	3.1	0.26
CLASS_MOBILE_V2	yes	yes	81%	72%	67%	4.0	3.0	2.2	0.18
CLASS_MOBILE_V2	no	yes	82%	77%	70%	4.3	2.9	2.3	0.04

Table 6.1: Table demonstrating the performance and efficiency of each model. The CLASS_ substring determines whether the model's magnitude predictions are classification based, while REGR_ is regression-based. Model accuracy here refers to the total instances that had at least a correct pan or tilt direction prediction. X and Y accuracy is the percentage of correct pan and tilt predictions. RMSE demonstrates how far, on average, each magnitude prediction (classified or regressed) is to the ground truth.

demonstrate the systems' efficacy in generalising to different camera-rigs. Due to the lack of research in this area, there are no alternative models to compare with, so we offer these results as a preliminary baseline. We aim to demonstrate that a general embodiment of human-controlled camera systems in a static scene can be learnt by a deep neural network. This includes generalisability to different backgrounds, objects (including plants), and camera rigs.

6.1 Model Evaluations

In this section, we present and compare the test results of eight different variations of the model. Each of the models was trained on the combined training dataset with the hyperparameters described in Section 4.5, including plant oversampling.

The models are tested on the combined test set. We evaluate each model in terms of performance and inference times for each. The first of the performance metrics is model accuracy. This measures how often the model makes a correct prediction in at least one of the x or y-axis classifications. Or put simply, how often the camera improves its position in relation to the salient object. We also include the RMSE for each axes' magnitude prediction, which measures 'how far' the predicted value is from the ground truth. We treat model accuracy as the primary metric of performance as it directly correlates with the objective of inching the camera towards the target at each step. This takes precedence over RMSE, which is treated as a secondary metric as, without an accurate direction classification, the magnitude that the camera moves with is irrelevant. Magnitude errors, on the other hand, can be easily corrected in subsequent steps. Also, lots of small magnitude errors are better than one or two large magnitude errors because large errors could move the camera to a position that obscures the object. The inability to differentiate between these two types of magnitude errors makes the RMSE metrics less reliable and hence subsidiary to model accuracy. We also present the average inference time of each model. Inference time here helps to guide model choice based on the requirement for the model to run in real-time and is measured by running inference over the combined test set and calculating the average time for each forward pass through the network. All experiments are run on an NVIDIA TITAN X Pascal GPU with 3 CPU cores.

The results of the experiments are presented in Table 6.1. The ResNet_101 classifier with saliency and transfer learning achieved the highest accuracy of all the models and is highlighted in bold. It accomplishes model accuracy of 87%, meaning just 13% of instances resulted in an incorrect movement prediction. As such, all models discussed henceforth will be compared with this optimal model. In the second row, an identical ResNet architecture is applied to the regression

formulation of the problem. Here, a continuous value is regressed for each of the axis' magnitude predictions, as opposed to a discretized bin classification. The formulation has a lower overall model accuracy than CLASS_RESNET_101. The magnitude RMSEs worsen as well, except for zoom magnitude which experiences a decrease in RMSE. This suggests that directly regressing the magnitudes does not improve its accuracy and damages the model's ability to learn direction classifications. Due to time constraints, we did not consider how a differently weighted loss function might impact the result. Neither did we attempt to reoptimize the hyperparameters for the regression model, instead we kept those that were optimized for the classification task.

To demonstrate how saliency augmentation and transfer learning impacted model performance, we performed an ablation study, removing each of the features from the ResNet_101 model to gain an insight into their effects on performance. Row 3 presents the results of removing transfer learning from the model, resulting in a 3% drop in model accuracy. The x-axis accuracy drops by 7% and all magnitude RMSE values deteriorate as well. This illustrates the improved generalisability introduced with transfer learning, as the network is initialised with rich image features learnt from the ImageNet dataset [55] before training begins. These features can then be fine-tuned to our specific task to reduce training time. Without these weight initialisations, the features must be learnt purely from our own, limited dataset which results in less-effective features being learnt and may require longer training times to find the optima. Row 4 presents the results of removing saliency augmentation from the model. Removing saliency incurs a 4% model accuracy reduction alongside increased RMSE scores, except for zoom which is unchanged. Conversely, a 2% improvement in y-axis accuracy is demonstrated by the model and inference time is reduced by a factor of 4.5. This makes the

model suitable for real-time inference on a CPU, rather than a GPU and demonstrates a trade-off between reduced accuracy but improved efficiency compared with the saliency-enabled model. Choosing between saliency and non-saliency is therefore a decision that depends on the complexity of the background scenery and the hardware requirements for the intended use-case. For biological imaging, we favour improved accuracy over efficiency as we are more likely to deal with complex backgrounds and have access to a GPU for real-time inference.

As described in Section 4.1.3.1, the ResNet architecture has multiple variants relating to the number of residual layers in the models. We ran our model with two alternative variants to the 101-layer model in row 1. These include a shallower model with 50 layers and a deeper, 152-layer model. The results of the variants are presented in rows 5 and 6. The smaller ResNet_50 model in row 5 incurs a 4% drop in model accuracy and RMSE scores increase except for zoom which decreased by 0.6 for x and zoom magnitudes. The larger ResNet-152 model sustains 2% lower model accuracy than its 101-layer counterpart but benefits from a 4% improvement in y-axis accuracy. Inference time is comparable between all three ResNet models. We suspected the shallower ResNet_50 model accuracy would decrease compared with the 101-layer model but were surprised to find that ResNet_152 also suffered an accuracy decrease. We attribute this to the hyperparameters used in the experiments, which were initially chosen to optimise the 101-layer ResNet variant. Reoptimizing the hyperparameters for the deeper model could result in an improvement in the accuracy. Also, more data might be needed before the accuracy benefits of the deeper network are realised. However, we did not have time to test either approach. Therefore, we chose the ResNet_101 model over the two variants.

Also, please note that in all of the models, x accuracy is higher than y accuracy. The reason for this was a miscalculation in the weights for the classes in the tilt (y) axis. Unfortunately we we did not have time to correct the error and so the high class imbalance for the tilt-axis depicted in Chapter 4, Figure 4.1 added a bias towards the 'Move Up' class and damaged its performance.

As described in Section 4.1.4, we also provided an alternative, lightweight model in the form of a MobileNet_V2 architecture. The results of the MobileNet model with saliency augmentation included are presented in row 7 and without saliency in row 8. Both models have a lower model accuracy that the ResNet_101 models. The non-saliency model has a higher X and Y direction accuracy than the saliency-enabled model. The saliency-enabled model is 0.09 seconds quicker than saliency-enabled ResNet_101 model and the non-saliency MobileNet model is 0.02 seconds faster than the non-saliency ResNet_101 model. This suggests that the MobileNet improves the efficiency of the model compared to the ResNet backbones. The lower magnitude RMSE scores of the MobileNet architecture suggest that magnitude predictions in the MobileNet architecture are more accurate, however, the decrease in accuracy makes these improvements redundant and this is reflected in the real-world operation of the model. Hence, as Figure 6.1 demonstrates, even though magnitude predictions may be more accurate, the decay in accuracy makes the MobileNet architecture less effective than the ResNet_101 model. This is because model accuracy directly correlates with the objective of inching the camera towards the target at each step. Whereas RMSE is an indicator of how far (measured in bins) the magnitude predictions are from their ground truths. An RMSE difference of 1 bin correlates to less than 70 pixels in a 1080p image which will not detriment performance in practice if the camera

is moving in the correct dimension. Conversely, if the camera is moving in the wrong direction more often, the camera is more likely to lose sight of the salient image and focus on the background instead.

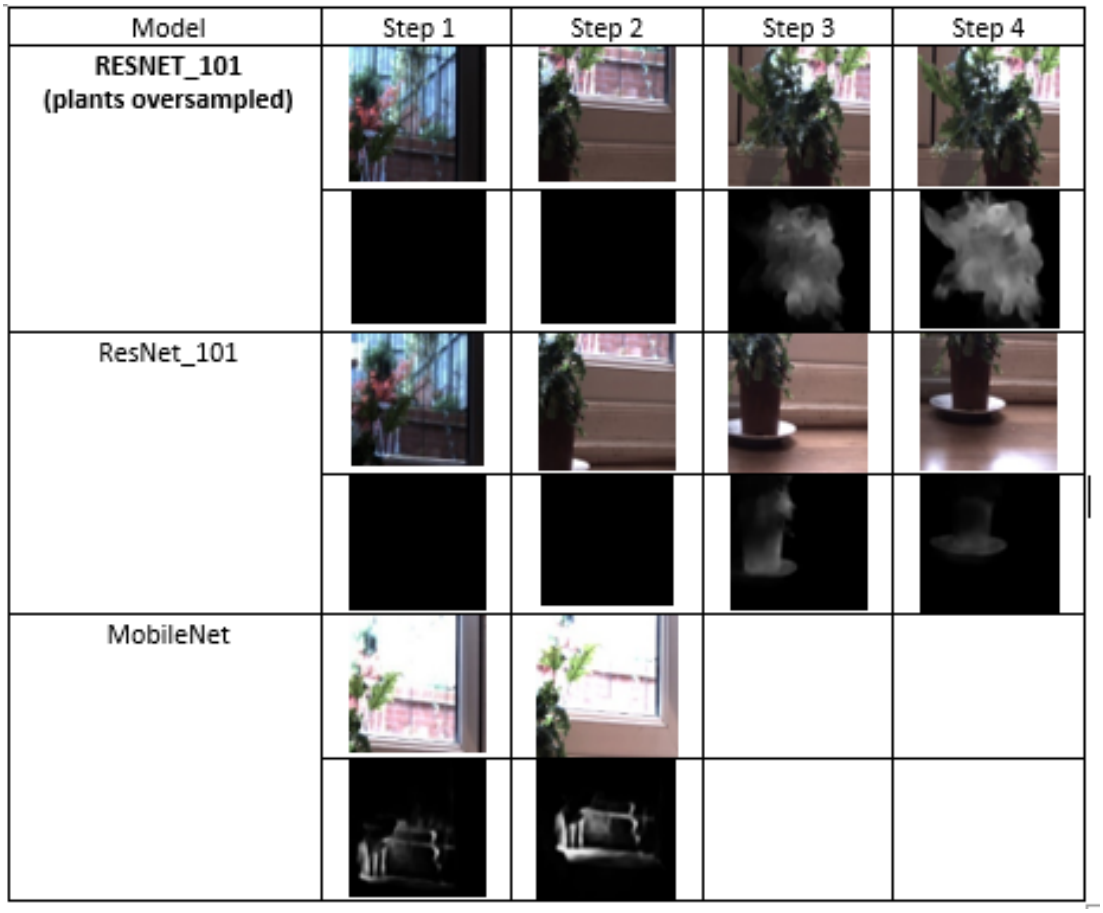

Figure 6.1: Illustrates the camera paths taken by different models and the input frame/saliency map produced after each movement classification. First row: ResNet_101 model with plants oversampled. Second row: ResNet_101 model without plants oversampled. Last row: MobileNet model with plants oversampled. All model clearly has saliency enabled and uses transfer learning.

Figure 6.1 demonstrates the camera paths taken by different models on a plant within an arbitrary scene. The models compared are the ResNet_101 model with

plant oversampling, ResNet_101 without plant oversampling and a MobileNet architecture (with saliency). The MobileNet architecture is subjectively the least effective model as it takes small steps towards the centre but generates a 'no-movement' prediction too early. This can be attributed to the slightly lower accuracy of the model. The ResNet without oversampling model is more accurate than MobileNet but seems to recentre the plant pot rather than the plant. We attribute this to a bias to the type of objects used in the training set which does not contain any plant images. This observation is explored further in Section 6.3. The ResNet_oversampled model successfully learns to centre the plant itself rather than the pot. Looking at the saliency maps, it seems clear that the masks of the oversampled model seem to define the plant boundaries much more clearly than other models. The model seems to manoeuvre itself towards the centre of the plant during the first two steps where no saliency mask is produced. Then, when the plant is better in shot, a more effective saliency mask is produced, and the camera position can be finetuned. This shows how it is still important for classifications that are not supplemented with a saliency map to produce an effective movement.

6.2 Saliency Evaluations

We know that saliency improves performance based on the results in Table 6.1, where removing saliency from the ResNet model results in a 4% drop in accuracy. However, there are still some unresolved questions regarding the efficacy of saliency augmentation, namely:

1. How strong is the correlation between saliency and accuracy?
2. Whether the performance boost is caused by added robustness to backgrounds or some other factor?

3. Whether saliency boosts performance on biological imagery, specifically plants?

At the time of writing, the saliency attention model that we use has state of the art performance [60], however, it still contains a level of uncertainty. When the saliency network is unsure about the salient object in the scene, it will return a blank saliency map. This means that a saliency map is not necessarily produced for every image, which can make evaluating the correlation between saliency and accuracy more difficult. To answer the questions above and gain a more conclusive understanding of how saliency improved the results, we, therefore, needed to isolate instances with and without saliency maps produced (i.e. maps surpass 1% intensity threshold) and compare their accuracies. This was achieved during inference by marking whether each instance produced a saliency map and recording its respective accuracy. The results for the combined test set are presented in Figure 6.2. The bar chart shows the quantity, model accuracy, x-axis accuracy and y-axis accuracy of instances which produce a saliency map. It demonstrates that of the 1065 images in the dataset, only 40% of instances produced a saliency map. Of this 40%, a 92% accuracy is achieved, compared to 74% accuracy when no saliency map is produced 60% of the time. This means the production of a saliency map can increase accuracy by 18% and demonstrates a strong correlation between saliency and accuracy. There are also improvements in the magnitude predictions such as the Y magnitude, which can reduce error by almost a full bin (equivalent to 57 pixels in a 1920x1080 image). This demonstrates the ability to improve accuracy when a good saliency map is produced by the saliency network. However, the probability of a saliency map being generated for an instance is still quite low at 40%, and so this suggests that our approach will improve linearly with the development of more reliable saliency detection networks, or those trained datasets more appropriate to our use-case.

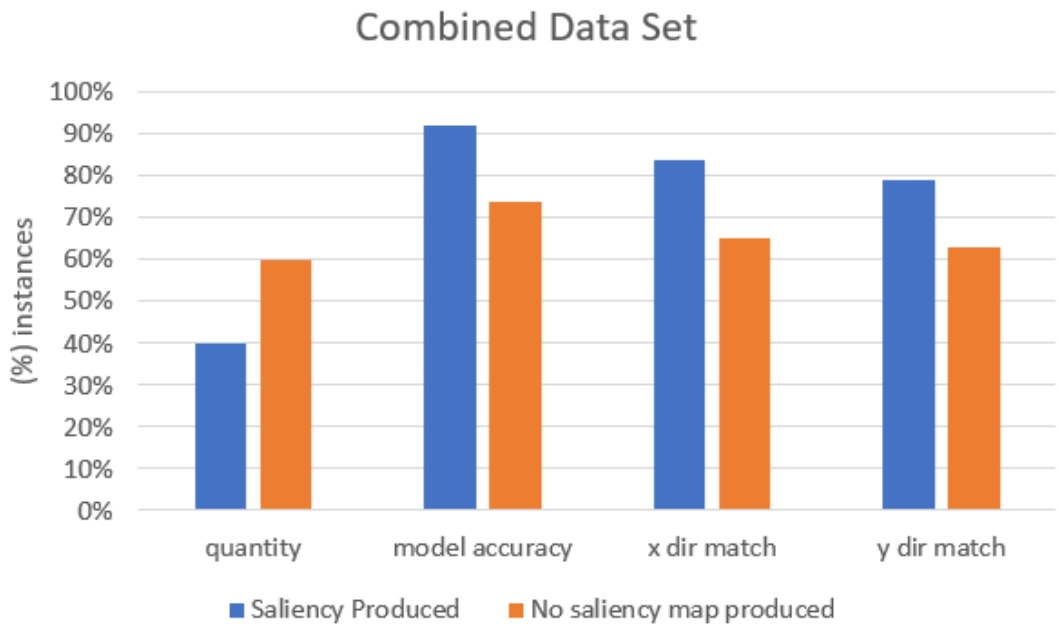

Figure 6.2: Demonstrates the accuracy of instances with and without a saliency map produced for the combined test set. Model accuracy is the % of instances with at least a correct pan or tilt prediction. X and Y dir match are % instances with correct pan and tilt predictions. Even though fewer instances have saliency maps, the accuracy of those instances is much higher than instances without a saliency map produced.

So, by analysing individual instances, we can validate that augmenting images with a saliency map significantly improves performance. To answer whether this can be attributed to better robustness to backgrounds, we use the same approach but applied to both uncluttered and cluttered test sets. As explained in Section 3.1.4, the uncluttered dataset contains objects amidst very simple, monochrome backgrounds. The cluttered datasets subjects appear within a much wider variety of texturally complex and cluttered backgrounds. Figure 6.3 compares the accuracy of instances with and without saliency maps produced, for the two datasets. A similar ratio of saliency maps is produced for both datasets - 54% saliency in cluttered and 46% in non-cluttered. The right-hand figure shows how in the uncluttered dataset, instances that produce a saliency map provide little to no

improvement in accuracy compared with instances that do not produce a saliency map. This suggests that the frame-to-action network can recentre objects with simple backgrounds easily as accuracy is close to 100% both with or without saliency augmentation.

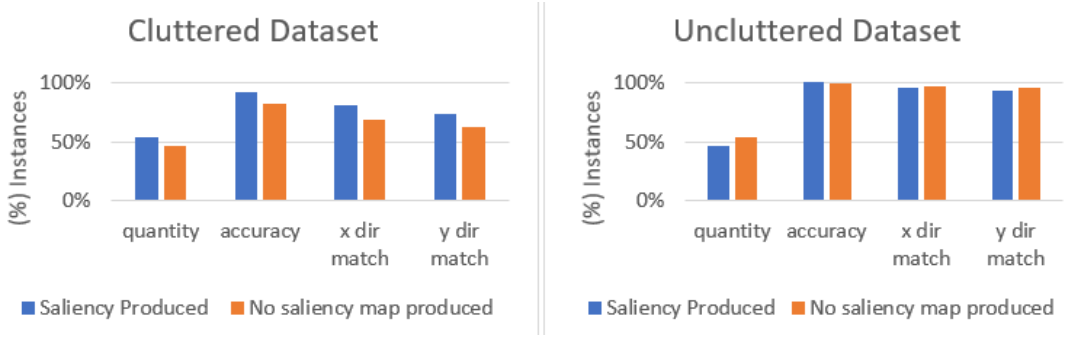

Figure 6.3: Compares the accuracy of instances with and without a saliency map produced for both the cluttered and uncluttered test sets. Model accuracy is the % of instances with at least a correct pan or tilt prediction. X and Y dir match are % instances with correct pan and tilt predictions. Top: Results for the cluttered test set. Bottom: Results for the uncluttered test set. Instances with saliency maps produced are more accurate than those without in the cluttered dataset, whereas the difference in accuracy is undetectable in the uncluttered dataset.

In the cluttered dataset (left-hand chart), however, an accuracy gap begins to appear between examples with and without saliency maps produced. This includes a 10% drop in model accuracy and 13% and 12% in X and Y axis accuracy where no saliency map is produced for an instance. Magnitude predictions also benefit from saliency maps, resulting in RMSE value improvements of 3.48, 2.98 and 0.16 for x, y and zoom predictions compared with instances where no map is produced. Because the introduction of clutter is the main characteristic between the two datasets, we infer that clutter is what causes performance to drop for instances where no saliency maps are produced. Instances that are augmented with saliency maps must be producing some resilience to the clutter as their performance drop is much smaller between the two datasets. Hence, we argue

that saliency is adding robustness to clutter. The small drop in performance that is experienced by saliency augmented instances can be explained by misleading saliency maps – i.e. segmenting the surface or part of the background as well as the object. This again suggests that our algorithms' accuracy will continue to improve alongside new advances in saliency detection networks. However, this does not completely discount a non-saliency approach, as the non-saliency models' performance is still promising. Additionally, the removal of the saliency network bottleneck can enable the algorithm to run in real-time on a CPU.

To answer the final question concerning saliency's applicability to plant imagery, we experimented once more with the plants test set. The result of the experiment is presented in Figure 6.4. Only 10% of the images in the plant dataset produced a saliency map, compared with 90% that did not. There is a 5% improvement in accuracy and some improvement in the X direction prediction for instances that produced saliency maps. This suggests that when saliency is effective, it can improve robustness to backgrounds and improve accuracy for plants. However, the rate at which saliency maps are produced for plant imagery (10%) with our current implementation is not suitable. The reason for poor saliency mask output on plants could be accredited to a lack of plant-based imagery in the dataset used to train the saliency network. Therefore, a suggestion for future works is to add some plant data to the saliency dataset and retrain the network to improve saliency map output for plant imagery.

6.3 Applicability to Plants

Before collecting our plants' dataset, our rudimentary network was trained on a combination of the controlled and cluttered training datasets discussed in Section 3.1.4, with no plant images in them. They consisted instead of household objects

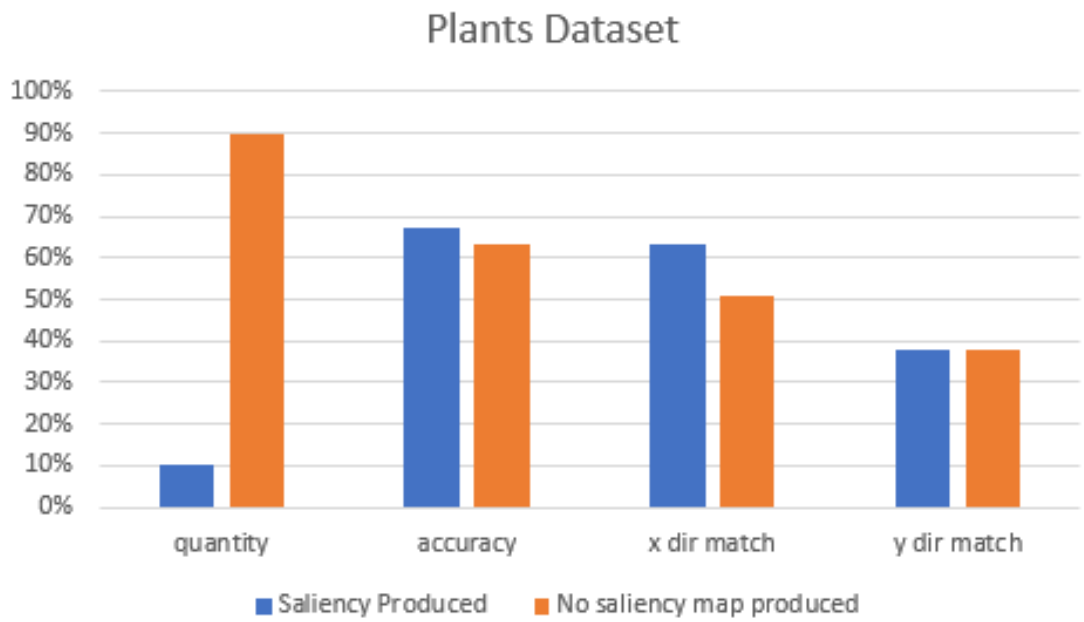

Figure 6.4: Compares the accuracy of instances with and without a saliency map produced for the plant's dataset. Model accuracy is the % of instances with at least a correct pan or tilt prediction. X and Y dir match are % instances with correct pan and tilt predictions. The quantity of saliency maps produced is much lower for the plant dataset compared with the other test sets. The saliency maps that are produced do still improve accuracy compared to instances without saliency.

that varied in shape, size, colour, and background with the hope that the function learnt would generalise to unseen plant data. Figure 6.5 shows two example camera paths taken by this model on two different plants, in chronological order from left to right. Both paths tend to centre the plant pots rather than the plants themselves. The bias to general, household objects that constitute the training data could explain this behaviour. These objects do not contain a lot of noise and are usually simple, self-contained shapes. Plants, on the contrary, are much more complex structures with lots of intricate components such as branches, leaves and roots. They also consist of a wider variety of shapes as each plant is unique. Therefore, with no prior knowledge of plants, it seems logical that the network

will pan to the simplest, most self-contained form in the image, which in most cases is the plants' pot.

To alleviate this bias and improve performance on plants, we collected a plant dataset, appended it the existing training set and retrained the network. The inference results on the plant test set before and after training on plants are presented in the first two rows of Table 6.2. We can see that introducing plants into the training corpus has a positive impact on the accuracy of around 3%, however, the results still do not look promising as it has a 53% chance of predicting a 'completely wrong' direction.

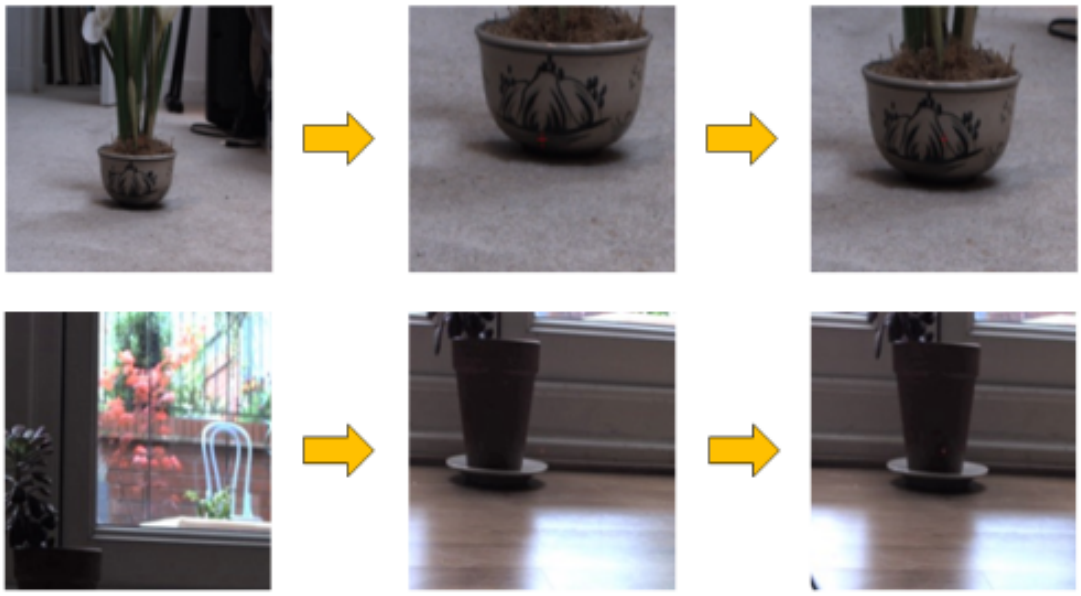

Figure 6.5: Demonstrates two example network outputs for the ResNet-101 model without plant oversampling applied. The model tends to centre the plant pot, rather than the plant itself when plant oversampling is not applied.

The 624 plant images that were introduced to the dataset make up only 6% of the total images in the training set, compared with 8280 images of general objects. This imbalance causes a large bias towards general objects and explains the incremental results before and after training on plants. To resolve this, we

	Model Accuracy*	**X accuracy**	**Y accuracy**	**X_mag RMSE**	**Y_mag RMSE**	**Zoom RMSE**
Before Training on plants	44%	28%	29%	2.3	3.0	5.9
After Training on plants	47%	31%	30%	2.6	3.4	5.8
Oversampling plants	**64%**	**53%**	**34%**	**4.0**	**4.3**	**6.1**

Table 6.2: Demonstrates the impact on accuracy of the ResNet-101 model with saliency before training on plants (first row), after training on plants (second row) and after training on plants with the plant instances oversampled (third row) on the plants test set. Model accuracy is the % of instances with at least a correct pan or tilt prediction. X and Y dir match are % instances with correct pan and tilt predictions. RMSE demonstrates how far, on average, each magnitude prediction (classified or regressed) is to the ground truth. Introducing plants to the training corpus increases accuracy. Oversampling plants to compensate for the underrepresentation in the dataset improves accuracy by 20%, while reducing magnitude accuracy.

implemented a form of oversampling described in Section 4.5.1. This enables us to artificially increase the number of plant images in the dataset without losing any information by removing non-plant objects. The results of the oversampled method are presented in the last row of Figure 6.2. It demonstrates a significant boost in accuracy of 20% compared with the model trained on no plants and 17% compared with the model trained on plants without oversampling applied. Overfitting does not explain the improvement as no images in the plant test set is used during training.

These results suggest that with a larger plant dataset, the system could achieve accuracy rates closer to the combined test set (87%). This exemplifies the applicability of our algorithm to biological imaging. As discussed in Section 6.2, by retraining our saliency network with application-specific images, an even higher accuracy could be attained. We believe the network shows promise to be implemented in production in plant phenotyping laboratories and has the potential to be generalised to other use-cases such as diffractometric crystal centring.

Chapter 7

Conclusions

We introduced an end-to-end deep neural network pipeline which directly maps input frames from a camera to gimbal actuation commands. The model can learn to recentre a variety of different objects in a static scene and adjust the field of view to better encompass the object in the image plane. This enables the camera to simulate a human's innate capacity to pan and zoom to the important object in a scene. We have empirically demonstrated that the model can learn the task of autonomous camera control without decomposition into separate ROI detection and camera control tasks. The model generalises well to new objects and backgrounds and can be further improved by adding more task-specific data where necessary. The model can take advantage of a saliency detection side task, which augments the input image with information to help inform the network's decision. Saliency augmentation improves the accuracy of the network when a saliency map is found, however current saliency detection network's result in relatively few saliency maps being produced. It is anticipated then that our approach will improve with the introduction of more accurate saliency detection networks, or those trained on appropriate data for our use-case. The algorithm is also generalised to different camera-gimbal rigs by means of a calibration algorithm which converts pixel offsets back to gimbal-specific units. This will enable future

works to fluently transition the algorithm to new hardware as well as enable the model to be trained on data collected from any camera device. This opens up the potential for using crowd-sourcing to increase the quantity of data available for training. The model is applicable to biological imagery, specifically images of plants taken in a phenotyping laboratory. Further data collection is required to add robustness to the algorithm, especially for plants which are disproportionally represented in the dataset, however, initial results are promising. The accuracy achieved on plants suggests that an end-to-end learned approach could be suitable for similar biological applications including diffractometric crystal centring and automated microscopy. The use of deep learning to directly map input frames to actuation enables a more general model to be learnt than that of separate, task-specific ROI detector followed by a camera control algorithm.

7.1 Future Work

More work is needed to improve the robustness of the network, specifically the collection of more data from a variety of contexts and more data relating to plants needs to be collected to retrain our network. Plant data also needs to be collected and labelled for retraining the saliency network to improve saliency outputs for plant imagery. Future works could also start to predict additional camera parameters including aperture/focus, actual zoom and white balance based on human-controlled ground truths. To improve the efficiency of the model, we can remove the bottleneck of the saliency network during inference, instead utilising the saliency side-task purely at training time only by implementing the Learning under privileged information (LUPI) paradigm [61]. This would enable the network to leverage saliency map augmentation at training time and learn features that discriminate between the salient object in the scene and the background. Without the saliency network bottleneck, the model can perform inference in

real-time on a CPU, which would reduce the hardware costs required to implement the algorithm.

One important analysis that was missed from this thesis is an evaluation of our algorithm against alternative approaches to the autonomous camera problem. This would require a more traditional approach, such as object detection (using a traditional or deep learning approach) followed by a rule-based system to be implemented and compared with our end-to-end approach. A set of tests can then be executed which can test the different criteria of our algorithm. Specifically, the following criteria should be evaluated: accuracy, generalisability, inference time, applicability to plants and ability to handle (and ignore) background noise. To achieve this, datasets need to be collected with image attributes specific to each test, much in the same way as we collected our test sets for uncluttered, cluttered and plant datasets in our evaluation. For example, the generalisability dataset should consist of as wide a range of objects as is possible to create, the background robustness dataset should consist of objects in a variety of complex backgrounds where it is difficult to distinguish between the salient object and background noise. Having tested on all of the datasets, the two approaches can be compared in order to evaluate whether our end-to-end approach offers any significant benefit over more traditional approaches to image relocation. Another suggestion for future work, is to increase the resolution of the images that are fed into the network. Currently, the input images are resized to 224x224 pixels, however it could be possible that this is causing some critical information to get lost, specifically regarding smaller features like textures and patterns. The model will still work with any square height and width dimensions if they are a multiple of 8. Hence, the size could be increased to, for example, 520x520 or greater. This will increase the amount of information fed into the network and could refine the learning of smaller features. The caveat, however, is that more GPU memory will

be required to train the model in parallel, so any practitioners should ensure that they have suitable hardware that can support the memory requirements before attempting this adaption. A final suggestion for future work is to implement mechanical zoom, rather than simulating it with cropping. The disadvantage of using cropping to simulate zoom is that you lose resolution when zooming in. By implementing mechanical zoom, the resolution will remain unchanged after zooming into objects which means images taken of the centred object while zoomed in will be of higher quality. It also could enable the camera to zoom out much further, as our current setup is limited to the original image as it's minimum zoom. It may also enhance performance as better resolution images means smaller features can be identified. Once mechanical zoom is implemented, it will mean that the calibration algorithm needs to be adapted to provision for adaptive focal lengths. This could mean calibrating the camera at each zoom level and switching between intrinsic parameters as the camera zooms in. Alternatively, [62] present a novel camera calibration method with which focal length of camera can be variable.

References

[1] T. Burrell, S. Fozard, G. Holroyd, A. French, M. Pound, C. Bigley, C. J. Taylor, and B. Forde, "The microphenotron: A robotic miniaturized plant phenotyping platform with diverse applications in chemical biology," *Plant Methods*, vol. 13, 12 2017. 1

[2] P. Carr, M. Mistry, and I. Matthews, "Hybrid robotic/virtual pan-tilt-zom cameras for autonomous event recording," MM '13, (New York, NY, USA), p. 193–202, Association for Computing Machinery, 2013. 2, 6

[3] M. Bojarski, D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to end learning for self-driving cars," 04 2016. 3, 9

[4] Y. LeCun, E. Cosatto, J. Ben, U. Muller, and B. Flepp, "Dave: Autonomous off-road vehicle control using end-to-end learning," *Courant Institute/CBLL, http://www. cs. nyu. edu/yann/research/dave/index. html, Tech. Rep. DARPA-IPTO Final Report*, 2004. 4

[5] D. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," in *Proceedings of Advances in Neural Information Processing Systems 1* (D. Touretzky, ed.), pp. 305 –313, Morgan Kaufmann, December 1989. 4

[6] H. Xu, Y. Gao, F. Yu, and T. Darrell, "End-to-end learning of driving models

from large-scale video datasets," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017. 4, 9, 26

[7] P. Dera and A. Katrusiak, "Diffractometric crystal centering," *Journal of Applied Crystallography*, vol. 32, pp. 510–515, 06 1999. 4

[8] A. Jain and V. Stojanoff, "Are you centered? An automatic crystal-centering method for high-throughput macromolecular crystallography," *Journal of Synchrotron Radiation*, vol. 14, pp. 355–360, Jul 2007. 4

[9] Y. Rivenson, Z. Göröcs, H. Günaydin, Y. Zhang, H. Wang, and A. Ozcan, "Deep learning microscopy," *Optica*, vol. 4, pp. 1437–1443, Nov 2017. 4

[10] T. Zhao and X. Wu, "Pyramid feature selective network for saliency detection," *CoRR*, vol. abs/1903.00179, 2019. 4, 7, 13, 41, 45

[11] T. Yokoi and H. Fujiyoshi, "Virtual camerawork for generating lecture video from high resolution images," in *2005 IEEE International Conference on Multimedia and Expo*, pp. 4 pp.–, 2005. 6

[12] C. Chen, O. Wang, S. Heinzle, P. Carr, A. Smolic, and M. Gross, "Computational sports broadcasting: Automated director assistance for live sports," in *2013 IEEE International Conference on Multimedia and Expo (ICME)*, pp. 1–6, 2013. 6

[13] M. Okuda, S. Inoue, and M. Fujii, "Machine learning of shooting technique for controlling a robot camera," in *2009 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pp. 1111–1116, 2009. 6, 8

[14] R. Xu and J. Jin, "Camera control and multimedia interaction using individual object recognition," *J. Multim.*, vol. 2, pp. 77–85, 2007. 7

[15] J. Quiroga, H. Carrillo, E. Maldonado, J. Ruiz, and L. M. Zapata, "As seen on tv: Automatic basketball video production using gaussian-based actionness and game states recognition," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 3911–3920, 2020. 7

[16] D. Pang, S. Madan, S. Kosaraju, and T. V. Singh, "Automatic virtual camera view generation for lecture videos," in *Tech report*, Stanford University, 2010. 7

[17] K. K. Rachavarapu, M. Kumar, V. Gandhi, and S. Ramanathan, "Watch to edit: Video retargeting using gaze," *Computer Graphics Forum*, vol. 37, 2018. 7

[18] L. Chen, H. Zhang, J. Xiao, L. Nie, J. Shao, W. Liu, and T.-S. Chua, "Sca-cnn: Spatial and channel-wise attention in convolutional networks for image captioning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017. 7

[19] J. Chen and P. Carr, "Mimicking human camera operators," in *2015 IEEE Winter Conference on Applications of Computer Vision*, pp. 215–222, 2015. 8, 18

[20] W. Luo, P. Sun, F. Zhong, W. Liu, T. Zhang, and Y. Wang, "End-to-end active object tracking and its real-world deployment via reinforcement learning," *CoRR*, vol. abs/1808.03405, 2018. 8

[21] S. Ito, G. Ueno, and M. Yamamoto, "DeepCentering: fully automated crystal centering using deep learning for macromolecular crystallography," *Journal of Synchrotron Radiation*, vol. 26, pp. 1361–1366, Jul 2019. 8

[22] J. Schurmann, I. Lindhè, J. W. Janneck, G. Lima, and Z. Matej, "Crystal centering using deep learning in x-ray crystallography," in *2019 53rd Asilomar Conference on Signals, Systems, and Computers*, pp. 978–983, 2019. 9

[23] D. Cho, J. Park, T. Oh, Y. Tai, and I. S. Kweon, "Weakly- and self-supervised learning for content-aware deep image retargeting," *CoRR*, vol. abs/1708.02731, 2017. 9

[24] W. Wang, J. Shen, and H. Ling, "A deep network solution for attention and aesthetics aware photo cropping," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, no. 7, pp. 1531–1544, 2019. 9

[25] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu, "A survey of deep learning techniques for autonomous driving," *Journal of Field Robotics*, vol. 37, p. 362–386, Apr 2020. 9

[26] U. Muller, J. Ben, E. Cosatto, B. Flepp, and Y. Cun, "Off-road obstacle avoidance through end-to-end learning," in *Advances in Neural Information Processing Systems* (Y. Weiss, B. Schölkopf, and J. Platt, eds.), vol. 18, pp. 739–746, MIT Press, 2006. 9

[27] J. Zhang and K. Cho, "Query-efficient imitation learning for end-to-end autonomous driving," *CoRR*, vol. abs/1605.06450, 2016. 9

[28] K. Amer, M. Samy, M. Shaker, and M. ElHelw, "Deep convolutional neural network-based autonomous drone navigation," *arXiv preprint arXiv:1905.01657*, 2019. 9

[29] A. Giusti, J. Guzzi, D. C. Cireşan, F. He, J. P. Rodríguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. D. Caro, D. Scaramuzza, and

L. M. Gambardella, "A machine learning approach to visual perception of forest trails for mobile robots," *IEEE Robotics and Automation Letters*, vol. 1, no. 2, pp. 661–667, 2016. 9

[30] C. Harris and M. Stephens, "A combined corner and edge detector," in *Procedings of the Alvey Vision Conference 1988*, Alvey Vision Club, 1988. 10, 27

[31] D. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, pp. 91–, 11 2004. 10, 16, 22, 27

[32] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," in *Computer Vision – ECCV 2006* (A. Leonardis, H. Bischof, and A. Pinz, eds.), (Berlin, Heidelberg), pp. 404–417, Springer Berlin Heidelberg, 2006. 10

[33] D. DeTone, T. Malisiewicz, and A. Rabinovich, "Superpoint: Self-supervised interest point detection and description," *CoRR*, vol. abs/1712.07629, 2017. 10

[34] Y. Liu, X. Xu, and F. Li, "Image feature matching based on deep learning," in *2018 IEEE 4th International Conference on Computer and Communications (ICCC)*, pp. 1752–1756, 2018. 10

[35] T. Georgiou, Y. Liu, W. Chen, and M. Lew, "A survey of traditional and deep learning-based feature descriptors for high dimensional data in computer vision," *International Journal of Multimedia Information Retrieval*, vol. 9, pp. 135 – 170, 2019. 10

[36] https://docs.opencv.org/master/dc/dc3/tutorial_py_matcher.html, "Opencv: Feature matching," accessed January 2021. 10

[37] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. 12, 41

[38] www.nikonusa.com, "What is aperture?: Understanding camera aperture from nikon," october 2020. 14

[39] Y. Badamasi, "The working principle of an arduino," pp. 1–4, 09 2014. 15

[40] https://reprap.org/wiki/StepStick, "Stepstick—repra," accessed october 2020. 15

[41] https://www.arduino.cc/en/main/software, "Arduino—software," accessed october 2020. 15

[42] https://www.airspayce.com/mikem/arduino/AccelStepper/index.html, "Accelstepper: Accelstepper library for arduino," accessed october 2020. 15

[43] P. Sturm, *Pinhole Camera Model*, pp. 610–613. Boston, MA: Springer US, 2014. 18

[44] https://waymo.com/open/data/, "Waymo open driving data," accessed october 2020. 19

[45] https://www.nuscenes.org/nuscenes, "nuscenes open drivning dataset," accessed october 2020. 19

[46] M. Pound, A. Burgess, M. Wilson, J. Atkinson, M. Griffiths, A. Bulat, Y. Tzimiropoulos, D. Wells, E. Murchie, T. Pridmore, and A. French, "Deep machine learning provides state- of-the-art performance in image-based plant phenotyping," *GigaScience*, vol. 6, 05 2016. 19

[47] J. Canny, "A computational approach to edge detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679–698, 1986. 27

[48] A. Hietanen, J. Lankinen, J.-K. Kämäräinen, A. Buch, and N. Krüger, "A comparison of feature detectors and descriptors for object class matching," *Neurocomputing*, vol. 184, 12 2015. 27

[49] G. E. Nasr, E. Badr, and C. Joun, "Cross entropy error function in neural networks: Forecasting gasoline demand.," in *FLAIRS conference*, pp. 381–384, 2002. 35

[50] https://paperswithcode.com/sota/image-classification-on imagenet, "Imagenet benchmark (image classification)," accessed october 2020. 36

[51] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, pp. 107–116, 04 1998. 36

[52] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation," *CoRR*, vol. abs/1801.04381, 2018. 37

[53] Y. Ho and S. Wookey, "The real-world-weight cross-entropy loss function: Modeling the costs of mislabeling," *IEEE Access*, vol. 8, p. 4806–4813, 2020. 38

[54] http://saliencydetection.net/duts/, "The duts image dataset," accessed october 2020. 40

[55] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009. 46, 63

[56] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang,

"On large-batch training for deep learning: Generalization gap and sharp minima," *arXiv preprint arXiv:1609.04836*, 2016. 47

[57] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 11, pp. 1330–1334, 2000. 56, 57

[58] W. Burger, "Zhang's camera calibration algorithm: in-depth tutorial and implementation," *Hagenberg, Austria*, 2016. 56

[59] T. Shao-xiong, L. Shan, and L. Zong-ming, "Levenberg-marquardt algorithm based nonlinear optimization of camera calibration for relative measurement," in *2015 34th Chinese Control Conference (CCC)*, pp. 4868–4872, 2015. 57

[60] https://paperswithcode.com/sota/saliency-detection-on-dut omron, "Papers with code - dut-omron benchmark (saliency detection," accessed october 2020. 68

[61] J. Lambert, O. Sener, and S. Savarese, "Deep learning under privileged information using heteroscedastic dropout," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8886–8895, 2018. 76

[62] G. Liu, W. Wang, J. Yuan, X. Liu, and Q. Feng, "A novel camera calibration method of variable focal length based on single-view," in *2009 Second International Symposium on Electronic Commerce and Security*, vol. 2, pp. 125–128, IEEE, 2009. 78

[63] Y. Liu, X. Xu, and F. Li, "Image feature matching based on deep learning," in *2018 IEEE 4th International Conference on Computer and Communications (ICCC)*, pp. 1752–1756, 2018.