University of Nottingham

PhD Thesis

Modularity in Artificial Neural Networks

Student:
Mohammed Amer

Supervisor:
Dr. Tomás Maul

Co-Supervisor:
Dr. Iman Yi Liao

Faculty of Science and Engineering
School of Computer Science

June, 2021

Acknowledgements

I am indebted to Dr. Tomás for his endless support on the professional and the personal levels during the long PhD journey. I was really lucky to have the opportunity to be his student and to work under his supervision. I would also like to thank my co-supervisor, Dr. Iman, for her valuable advice and feedback during my PhD. I wish to show my gratitude to all my colleagues, especially Moataz, Ijaz, Khoa and Tuong. It was really a privilege to be around you.

No words can really express how indebted I am to my father, mother and brother for always being there for me. Their endless caring is the reason I am writing these words right now. I can not express how much I am blessed to have Asmaa, my wife, beside me during the rough and the good times. Without her support, I could not have made it this far. Special mention to my little daughters, Malika and Farida, who were my shining stars during this journey. I dedicate this to Hamid Abdullatif, my grandfather.

Publications and Grants

- Amer, M., Maul, T. A review of modularization techniques in artificial neural networks. Artif Intell Rev 52, 527–561 (2019). https://doi.org/10.1007/s10462-019-09706-7
- Amer, M., Maul, T. Path Capsule Networks. Neural Process Lett 52, 545–559 (2020). https://doi.org/10.1007/s11063-020-10273-0
- Amer, M., Maul, T. Weight Map Layer for Noise and Adversarial Attack Robustness. arXiv:1905.00568 (2019). [Submitted]
- Four quarters, each of 200k CPU hours, at HPC Midlands+ (Athena), which was funded by the EPSRC on grant EP/P020232/1 as part of the HPC Midlands+ consortium.

Abstract

Artificial neural networks are deep machine learning models that excel at complex artificial intelligence tasks by abstracting concepts through multiple layers of feature extraction. Modular neural networks are artificial neural networks that are composed of multiple subnetworks called modules. The study of modularity has a long history in the field of artificial neural networks and many of the actively studied models in the domain of artificial neural networks have modular aspects. In this work, we aim to formalize the study of modularity in artificial neural networks and outline how modularity can be used to enhance some neural network performance measures. We do an extensive review of the current practices of modularity in the literature. Based on that, we build a framework that captures the essential properties characterizing the modularization process. Using this modularization framework as an anchor, we investigate the use of modularity to solve three different problems in artificial neural networks: balancing latency and accuracy, reducing model complexity and increasing robustness to noise and adversarial attacks. Artificial neural networks are high-capacity models with high data and computational demands. This represents a serious problem for using these models in environments with limited computational resources. Using a differential architectural search technique, we guide the modularization of a fully-connected network into a modular multi-path network. By evaluating sampled architectures, we can establish a relation between latency and accuracy that can be used to meet a required soft balance between these conflicting measures. A related problem is reducing the complexity of neural network models while minimizing accuracy loss. CapsNet is a neural network architecture that builds on the ideas of convolutional neural networks. However, the original architecture is shallow and has wide layers that contribute significantly to its complexity. By replacing the early wide layers by parallel deep independent paths, we can significantly reduce the complexity of the model. Combining this modular architecture with max-pooling, DropCircuit regularization and a modified variant of the routing algorithm, we can achieve lower model latency with the same or better accuracy compared to the baseline. The last problem we address is the sensitivity of neural network models to random noise and to adversarial attacks, a highly disruptive form of engineered noise. Convolutional layers are the basis of state-of-the-art computer vision models and, much like other neural network layers, they suffer from sensitivity to noise and adversarial attacks. We introduce the weight map layer, a modular layer based on the convolutional layer, that can increase model robustness to noise and adversarial attacks. We conclude our work by a general discussion about the investigated relation between modularity and the addressed problems and potential future research directions.

Contents

Chapter 1

Introduction

1.1 An Overview of Deep Learning

Machine Learning (ML) is a branch of Artificial Intelligence (AI) that approaches complex tasks, mainly the ones that do not lend themselves to classical types of well-defined algorithms, through tuning mathematical models using a corpus of data in a process known as learning. Learning is a process that aims at increasing the performance of some mathematical model on a given task by exposing it to experience, that is, a suitable form of information or data. While classical ML approaches revolutionized the way many problems are solved, they fall short on many serious applications like complex image recognition, voice synthesis and advanced Natural Language Processing (NLP) tasks like language translation, voice synthesis and semantic analysis, among others.

Many factors contributed to these limitations and two of the main factors were the reliance on shallow models and manual feature engineering. Shallow models, like linear regression, logistic regression and Support Vector Machine (SVM) usually depend on a variant of a linear transformation, that is followed by some non-linearity when needed. This single step of feature processing limits the ability of models to discover and distil defining features from the data and, hence, the performance of the models is very dependant on the process of manually designing good data features. Since these types of complex problems are ill-defined in terms of classic rule-based algorithms, this process of manual engineering was to a great extent a blind process, guided only by rough heuristics.

Deep learning (Goodfellow et al., 2016) is a wide set of related techniques that aims at mitigating the mentioned limitations of classical ML through composition of distributed features, extracted over multiple layers of sequential and parallel processing. Deep learning and Artificial Neural Networks (ANNs) have a long history in computer science and have gone through different waves from cybernetics passing by the connectionism movement till reaching the current resurgence by the term deep learning. While classical ML approaches depended

mostly on shallow learners and manual feature engineering, depth of processing, feature composition and distributed representations are fundamental to deep learning.

While deep learning is a wide umbrella encompassing many techniques, ANNs are the most commonly researched and applied category of models. The inspiration for ANNs came from biological neural circuitry. However, they diverged from their biological origins over the years and took a mainly pragmatic approach, oriented towards engineering problems without clinging faithfully to mimicking their biological counterparts.

Biological neural circuits consist mainly of a densely connected network of cells, called neurons, that are specialized for electrical conduction and chemical transmission. Every neuron consists of a cell body (called soma) having specialized protruding branches, called dendrites, generally responsible for receiving signals from other neurons and another protruding cylindrical structure, called axon, which generally transmits signals to other neurons. These circuits perform information processing through receiving, integrating and transmitting signals between each others (Purves et al., 2004).

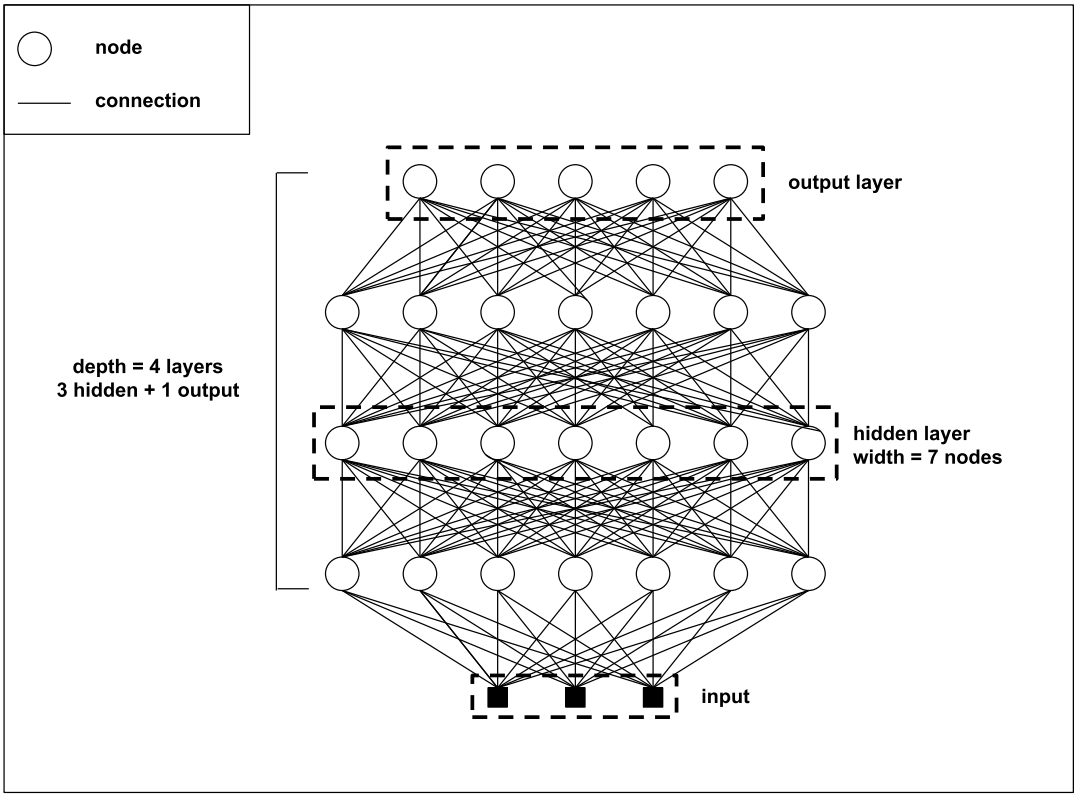


Figure 1.1. A conceptual diagram of a fully-connected feedforward neural network with 3 hidden layers and output size of 5.

Inspired by this distributed processing model, ANNs are parametric models that conceptually consist of a set of interconnected neurons. Each neuron (or node) is a parameterized function that performs a linear transformation on the inputs followed by a non-linearity (called activation). These neurons are usually arranged in sequential layers, where neurons in each layer receive input from the previous layer and pass their output to the subsequent layer. Fig. 1.1 shows a conceptual diagram of a feedforward fully-connected network which is often called an Multilayer Perceptron (MLP); it is one of the earliest types of ANNs. Formally, the output $h_j^{(l)}$ of a given node j in layer l in an MLP is calculated as follows,

$$\begin{aligned} \widehat{h}_j^{(l)} &= b_j + \sum_i w_{ij}^{(l)} * x_i^{(l-1)} \\ h_j^{(l)} &= \sigma(\widehat{h}_j^{(l)}) \end{aligned} \tag{1.1}$$

where $\widehat{h}_j^{(l)}$ is the intermediate state of the node after applying the linear transformation, $w_{ij}^{(l)}$ is the weight (the learned parameter) connecting node i in the previous layer $l-1$ to node j in layer l, $x_i^{(l-1)}$ is the output of node i in the previous layer $l-1$ (or the input if layer l is the first layer), b_j is the bias term for node j, σ is the non-linearity function and $*$ is the multiplication operator. Eq. (1.1) can be considered the fundamental equation of modern ANNs and almost any other variant is more or less building on it.

Through training, the weights get adjusted to fit the data and, subsequently, each layer in the network extracts a set of features by composing features from the previous layer. The final layer in the network produces the output and is called the output layer, while all the previous layers produce intermediate states and are called the hidden layers. Like the training of many other classical ML models, an ANN is trained by defining an objective or a loss, which is a measure of the network performance on the task dataset, and then optimizing the network parameters based on this loss or error signal.

Due to the extensive non-linear properties of an ANN, there is no closed-form solution to the optimization process. Hence, ANNs are optimized using an iterative algorithm that alternates between evaluating the loss using the current weights (parameters) and optimizing the weights based on the loss to get a new set of weights. The two most commonly used techniques for optimizing ANNs are backpropagation and Evolution Algorithms (EAs) (Paul and Singh, 2015).

EAs are a set of optimization techniques inspired by the theory of evolution by natural selection (Bäck, Fogel, and Michalewicz, 2018). The main steps in any EA are: fitness evaluation, selection and reproduction (or mating). In an EA, a set of solutions (individuals) are initialized and encoded. The performance of each solution is evaluated according to a problem-specific fitness function. Then, a subset of the solutions are chosen to proceed to the next

evolutionary generation by a selection process that depends on the fitness of each solution. The selected solutions are then reproduced, a process which introduces diversity and random mutations in the offspring. Then the process is repeated for the new evolutionary generation until satisfactory criteria are met.

There are three main variations of EAs, namely Genetic Algorithm (GA), Evolution Strategies (ES) and Genetic Programming (GP). GA is the most faithful to biological evolution. The main characteristic of a GA is that the solutions are encoded into a genotype or a genome, i.e. an encoding scheme that can be translated into the actual solution. In contrast, ES operate on the actual solution or the phenotype directly. In ES, two sets of parameters are used to represent the solution: decision parameters which represent the solution itself and strategy parameters which control the mutation of the decision parameters. GP is an EA intended for evolving programs, i.e. a set of instructions to be executed as an algorithm. A program in a GP is often represented as a tree.

EAs have the advantage that the neuron activations and the loss can be any arbitrary functions. However, EAs are very computationally expensive and demand a serious scale of distributed processing to be sufficiently efficient and, hence, practical. On the other hand, backpropagation requires the activation functions and the loss to be differentiable since it depends on gradient information. Backpropagation, however, is by far more computationally efficient than EAs since it can be accelerated using commercial Graphical Processing Units (GPUs) and is much faster to converge. Many optimization algorithms based on backpropagation exist like Stochastic Gradient Descent (SGD), Adagrad (Duchi, Hazan, and Singer, 2011), Adadelta (Zeiler, 2012), RMSprop (Ruder, 2017) and Adam (Kingma and Ba, 2017), among others, where each of them has its advantages and disadvantages.

1.2 Basic Topologies of ANNs

Like any kind of network, there has to be a general plan (i.e. topology) that defines how individual neurons are connected together to form the network structure. There are virtually an infinite number of possible topologies; many variants are in use and architectural innovations to find better performing networks is an open question and one of the biggest active research areas in ANNs research. However, there is a fundamental dichotomy in the topologies of ANNs which is based on graph theory. Since Neural Networks (NNs) are essentially a graph where the nodes are the vertices and interconnections are the edges, we can classify NNs into either feedforward NNs, which are NNs with Directed Acyclic Graph (DAG), i.e. a graph containing no loops, and Recurrent Neural Networks (RNNs), which are NNs with a Cyclic Graph (CG), i.e. containing loops.

Feedforward NNs process their input by a unidirectional flow of information, where the first hidden layer receives the input and passes its output to the

subsequent hidden layer. Every subsequent layer similarly processes its input and passes the output to the next layer, until the last layer, called the output layer, which produces the final output. Feedforward networks are mainly used to process independent inputs with no inter-dependencies, however, some recent variants have modified architectures that can process sequential data. Some notable examples are WaveNet (Oord et al., 2016), an audio generative model, and Transformer (Vaswani et al., 2017), a state-of-the-art (SOTA) NLP model.

Two feedforward variants are of special importance since they lay the foundation for many other feedforward architectures, which are MLPs, also called Fully Connected (FC) networks, and Convolutional Neural Networks (CNNs). The MLP was one of the first architectures to be used in the deep learning field. An MLP is a multilayer NN where each node in a given layer is connected to every node in the previous layer. Historically, the MLP was used for any data type, including visual data, but later it was superseded by CNNs for visual datasets. However, MLPs are still used for many other types of data and as a component subnetwork/module of almost every ANN to serve different purposes like producing the classification output, routing input in routed NNs (Rosenbaum, Klinger, and Riemer, 2017; Rosenbaum et al., 2019) and attentional mechanisms (Bahdanau, Cho, and Bengio, 2014; Xu et al., 2015; Luong, Pham, and Manning, 2015; Yang et al., 2016), among others.

The other famous variant of feedforward networks is CNNs (Fukushima and Miyake, 1980; LeCun and Bengio, 1995). A CNN is a type of NN that is architecturally modified to be efficient in processing visual data and images. Unlike MLPs, a CNN layer node is connected only to a subset of the outputs of the previous layer. Hence, CNNs depend on two main concepts, Receptive Field (RF) and weight sharing. An RF is the size of the subset of the previous layer output to which each node in the subsequent layer is connected to. Weight sharing means that weights of each node across all the possible RFs are tied together so that they have the same values. The reason for this arrangement is the spatial arrangement inherent to visual data. Images, and similar visual data, are hierarchically composed of building units at different levels of abstraction. Hence, it makes sense to try to identify these different units at all the possible locations of the image.

The other main type of NN is RNNs. An RNN is an NN containing feedback connections, i.e. a connection from a node to itself or to a previous node in the hierarchy. This means that some nodes depend on past inputs and, hence, have temporal dependency. This makes an RNN suitable for processing temporal and other types of sequential data, e.g weather forecasting. RNNs, however, suffer from difficulties in learning due to the long term dependencies between their states, which manifests mainly in vanishing and exploding gradients. These learning problems are usually solved by either a modified learning algorithm, a modified architecture, or both. Learning algorithms like Truncated Backpropagation Through Time (TBPTT) solve this by truncating the steps used to

calculate the gradient. Architectural innovations like Long Short-Term Memory (LSTM) (Hochreiter and Urgen Schmidhuber, 1997) and Gated Recurrent Unit (GRU) (Cho et al., 2014) address the problem by introducing different gating mechanisms that control the information flow and time dependency.

1.3 Learning Types

Different ML learning problems require different types of learning. While all of the ML problems dictate estimating some probability distribution over the data, the output required to solve the problem differs in a significant way. One common type of ML problem is associating an input with an output. The most commonly encountered problem in this category is data classification, where an input has to be placed in one of multiple classes. Supervised learning is usually used to solve this kind of problem. In supervised learning, the model is presented with an input and its corresponding output and the model has to implicitly estimate the conditional probability distribution, so that at inference time, the NN can classify similar patterns correctly.

In contrast, for some tasks, we need only to discover interesting patterns in the data. Clustering problems fall under this category, where we want to discover similarity between patterns and then place similar data into the same cluster/group. Unsupervised learning is the type of learning used for this kind of problem, where the model needs to implicitly estimate the probability distribution over the input data in order to be able to map similar patterns to similar representations. Another very useful common application of unsupervised learning is pretraining NNs in order to extract useful representations that can be used in another downstream task, like object classification using supervised learning.

Self-supervised learning is another type of learning which has similarity to unsupervised learning in that it needs no labeled data, however, it is also similar to supervised learning in that it utilizes a supervisory signal for learning. The idea is to use some inherent structure in the data or an auxiliary source to automatically extract a supervisory signal. For example, word embedding models like BERT (Devlin et al., 2019) and skip-gram (Mikolov et al., 2013) are trained to vectorize words into useful representation by predicting missing words in a text corpse using their surrounding context. These types of self-supervised tasks are usually called pretext tasks to differentiate from final or downstream tasks. Self-supervised learning also has applications that are themselves final tasks, like image colorization (Vitoria, Raad, and Ballester, 2020) and inpainting (Zeng et al., 2020).

Another type of task, which has similarity to supervised learning, is where we want to make decisions based on input states. Hence, it is effectively a task of associating states with actions. However, unlike the case for supervised learning, we do not have pairs of states and their corresponding actions. Instead,

the model is presented with a reward or a score that acts as a proxy for the model's performance. Reinforcement Learning (RL) is the type of learning that is used for these kinds of tasks and it has many useful application when it comes to domains like robotic control (Kormushev, Calinon, and Caldwell, 2013), AI in video games (Mnih et al., 2015) and mastering board games (Silver et al., 2016; Silver et al., 2017b; Silver et al., 2017a).

1.4 Application Domains

Deep learning has a vast number of applications that are infeasible to count. Many of these applications were practically outside of ML reach before the recent era of deep learning, mainly due to the curse of dimensionality, the scale of the data and the high level of expertise needed to extract useful features. Research on applying ML models to visual data is an old quest that stretches back across many decades of the previous century and, for quite a long time, practical Computer Vision (CV) applications were mainly dominated by classical CV algorithms. CNNs have revolutionised the field and their variants are currently considered the SOTA in this field. Visual-modality applications of ANNs are numerous, including image classification, segmentation (Ronneberger, Fischer, and Brox, 2015), colorization (Vitoria, Raad, and Ballester, 2020), inpainting (Zeng et al., 2020), style transfer (Jing et al., 2018), sample generation (Barua, Erfani, and Bailey, 2019), face detection (Zhang et al., 2016b) and recognition (Guo and Zhang, 2019) and guidance of self-driving cars (Nugraha, Su, and Fahmizal, 2017) and Unmanned Aerial Vehicles (UAVs) (Padhy et al., 2018), among others. One of the triumphs of this subfield is Generative Adversarial Networks (GANs) (Goodfellow et al., 2014), which are generative models that can generate images in a given context with unprecedented realism.

On acoustic data, such as voice and music, ANNs have achieved SOTA on different complex tasks such as voice recognition (Newatia and Aggarwal, 2018), text-to-speech and music synthesis (Oord et al., 2016), musical instrument separation (Chandna et al., 2017) and voice cloning (Arik et al., 2018), among others. NLP is the subfield of processing human languages. The NLP subfield had been dominated for a long time by combined Hierarchical Markov Models (HMMs) and Gaussian Mixture Models (GMMs), termed GMM-HMM. With the introduction of RNNs and, later, Transformers (Vaswani et al., 2017), SOTA has been pushed extensively on many NLP tasks, like word representation (Devlin et al., 2019; Mikolov et al., 2013), language translation (Wu et al., 2016), text generation (Lu et al., 2018), document summarization (Zhang, Wei, and Zhou, 2019) and similarity (Ranasinghe, Orasan, and Mitkov, 2019), sentiment analysis (Wadawadagi and Pagi, 2020) and chatbots (Csaky, 2019), among others.

The applications discussed above are only a subset of the most profound territories that ANNs are revolutionizing. Many other applications are heavily

researched and new applications are being continuously explored as the field expands its understanding and toolkit. One of the newly emerging subfields that is gaining traction is Graph Neural Networks (GNNs) (Zhou et al., 2019), a specialized architecture that is adapted to processing graphs. GNNs are being applied to many interdisciplinary fields, like rational drug discovery (Li, Cai, and He, 2017), molecular simulation (Husic et al., 2020) and properties prediction (Hao et al., 2020), combinatorial optimization (Gasse et al., 2019) and social network analysis (Fan et al., 2019).

1.5 Challenges to Deep Learning

1.5.1 Data Greediness

Despite the fact that the concept of stacking multiple non-linear layers dates back to the 1960s, two main challenges worked against the popularization and practical application of deep learning: scarcity of data and computational needs. At the dawn of the 21st century and as big data became more available and computers became more powerful, specially after commercial General-Purpose Graphical Processing Units (GPGPUs) became widely available and could be utilized to accelerate ANNs, the real potential of ANNs could be harnessed.

Later, a trend appeared in ANNs where greater performance could be achieved by increasing model capacity and, hence, more data can be fitted by the model and the generalization can be improved in turn. The data greediness of ANNs turned to be problematic, specially for supervised learning, where obtaining large amounts of labeled data is expensive and requires extensive manual labor. Several research directions aim at mitigating this problem. Data augmentation aims at generating more data variations from the existing samples through different transformations or through generative models like GANs (Goodfellow et al., 2014). Another emerging approach is few-shot learning (Wang et al., 2020), which is a type of meta-learning where the model has to generalize from few examples of each class or, in the extreme case, from a single example of each class (one-shot learning).

1.5.2 Overfitting and Underfitting

Another problem that is related to capacity is overfitting. Underfitting and overfitting are two problems that plague not only ANNs but ML models in general. Underfitting occurs when ML models with small capacity struggle to fit large data and, hence, tend to have poor test performance. On the other hand, overfitting happens when models with very high capacity can fit noise and statistical artifacts in the data and, hence, their performance will also suffer at test time. The bias-variance trade off is the analytical framework that explains these phenomena, where the performance of the underfitting models will tend

to have lower variance across many runs, however, the mean performance will be biased compared to the ground truth. On the other hand, overfitting models will on average have less bias, but their performance will be of higher variance across multiple runs.

Overfitting is more manifested in ANNs since they usually have large capacity. Hence, research in regularization techniques have special importance in the deep learning subfield. Regularization is a set of techniques that introduces inductive bias, that is a prior inherent bias, in the model selection process such that a subset of models is favoured over others. In the context of ANNs, different regularization techniques (Kukačka, Golkov, and Cremers, 2017) are used to bias the model selection towards models with lower effective capacity. Weight decay and different forms of L-norm are used to bias the learning process towards models with sparser or smaller weight magnitude. Dropout (Srivastava et al., 2014) is another regularization technique that randomly drops different nodes during training to decrease effective capacity and introduce more independence between the learned features. Similar drop techniques followed such as DropConnect (Wan et al., 2013). We will later use DropCircuit (Phan et al., 2018), a related modular regularization technique in Chapter 4. Data augmentation is itself considered a regularization technique since it aims at reducing overfitting by increasing the dataset size.

1.5.3 Complexity

One of the obvious problems that comes with models with large capacities is the explosion in time and memory complexity. Large ANNs require large matrix-matrix multiplications which translate to a huge number of Floating Point Operations Per Second (FLOPS) and large memory is needed to store the millions, and sometimes, billions of parameters, intermediate states and the associated gradients calculated during training. Despite the fact that the industry of GPUs before the recent popularization of ANNs was mainly fueled by other applications like video games, graphics rendering and simulation of biological and physical systems (Kirk and Wen-Mei, 2016), GPUs proved to be very handy when it comes to performing the huge number of FLOPS required for ANNs. Tensor libraries adopted support for GPU acceleration and many of them are extended to support multiple GPUs and distributed computing. Emerging recent research trends aim at devising new training algorithms that can accelerate ANNs on Central Processing Unit (CPU) clusters and multi-core machines (Chen et al., 2020).

This computational revolution aided the conventional paradigm of compute-centric learning. In compute-centric applications, a central repository of data and powerful machines are utilized for learning models and usually also remote inference. With the advancement of handheld and Internet of Things (IoT) devices, a new potential opened for exploiting the huge amount of distributed

data gathered by these devices. However, it is not feasible to pursue applications in this domain using the compute-centric approach due to many limitations and concerns like safety, privacy and bandwidth. The data-centric approach was developed as a mitigation to these concerns. Instead of a central computational repository of machines and data, the data-centric approach depends on small distributed models that can learn locally without the need for central data collection.

The data-centric approach, along with the need for inference on limited-resource devices, stimulated a line of research directed at miniaturizing models. Since model performance is on average correlated with generalization, a delicate balance needs to be achieved such that shrinking the model does not lead to severe degradation in performance. Several techniques are actively researched in this research line such as Neural Architecture Search (NAS), separable convolutions, quantization, hashing, factorization, pruning and distillation. We review some of these techniques when we address the complexity issue using modularity in Chapter 3.

1.5.4 Adversarial Attacks

The commonly researched and most performing ANNs are differentiable models. This differentiability facilitated their training by exploiting gradient information. However, this very same differentiability also made these models sensitive to adversarial attacks, an intriguing vulnerability that was discovered by Szegedy et al. (2013). ML models, and deep learning models being no exception, suffer from sensitivity to noise. However, random noise existing in the data can usually be accounted for by using large datasets, regularization techniques and data augmentation. However, adversarial attacks are a special kind of engineered noise that can seriously harm the performance of ANNs. By using the same kind of gradient information used for training the models, an attacker can differentiate with respect to the ANN input to find the directions in the input space that seriously affect the network performance. Then, by adding very small perturbations in these directions to inputs like images, the network performance, e.g classification accuracy, can be severely harmed without any noticeable disruption as perceived by a human observer.

Adversarial attacks can be either white-box or black-box (Chakraborty et al., 2018). White-box attacks assume the target model to be directly accessible and, hence, gradient information can be readily obtained. Black-box attacks assume no access to the target model and they generally rely on training a surrogate model that can be used to design the attacks that will be finally targeted at the victim model. This depends on the transferability assumption between ML models, that is, the assumption that models trained on similar datasets can be affected by the same attacks. Research in adversarial attacks is a feedback arms race between devising more powerful attacks and enhancing adversarial

defences, which are techniques that aim at reducing model susceptibility to the attacks. An adversarial defence can be through detection, modifications to the architecture or the learning process or a combination of these techniques. We elaborate on adversarial defences when we use modularity to increase robustness of convolutional-based models to noise and adversarial attacks in Chapter 5.

1.5.5 Catastrophic Forgetting

ANNs are learned through weight updates guided by an error signal that depends on the dataset. Learning, or fitting an ANN to a dataset, means finding a set of weights with a good generalization on the target dataset. If a trained ANN is later learned on a different dataset, it will gain performance on the new dataset, however, at the cost of performance degradation on the old dataset. This phenomenon is called catastrophic forgetting (Kirkpatrick et al., 2016). Catastrophic forgetting happens due to the shift of the network weights from the discovered local minimum, as measured by the loss, of the old dataset into a local minimum of the new dataset. This problem is related to the stability-plasticity problem (Abraham and Robins, 2005) in Neuroscience, which remains an open problem to date.

Life-long learning is the term used to describe the ability of an AI agent to continuously acquire new knowledge and skills without forgetting what was learned in the past. Catastrophic forgetting is a serious obstacle to achieving life-long learning and to knowledge integration across different tasks; mitigating its effect is an open question and an active research area in the deep learning field.

Five main approaches that try to address this problem currently exist in the literature. One of the very early approaches is regularization. Regularization in the context of catastrophic forgetting aims at regularizing the model weights to stay close to the previous local minimum, while leaving enough flexibility to allow the model to learn new tasks. Notable examples are Elastic Weight Consolidation (EWC) (Kirkpatrick et al., 2016) and Synaptic Intelligence (SI) (Zenke, Poole, and Ganguli, 2017). Ensemble methods (Polikar et al., 2001; Dai et al., 2007; Fernando et al., 2017) depend mainly on assigning a new network or a new module in a larger network for each new task. Replay methods (Shin et al., 2017; Ven and Tolias, 2018; Isele and Cosgun, 2018) reduce forgetting by frequently injecting samples from the old datasets into the learning process to regularize training. The dual-memory approach (McClelland, McNaughton, and O'Reilly, 1995; Kumaran, Hassabis, and McClelland, 2016) is a biologically inspired technique, where two types of memories, namely Short-Term Memory (STM) and Long-Term Memory (LTM), are used. STM processes the new task information and integrates it later into the LTM. Finally, sparse-coding (Kruschke, 1991; Coop, Mishtal, and Arel, 2013; Murdock, 1983; Eich, 1982)

aims at reducing the interference in the internal network representations across tasks by introducing sparsity.

1.6 Biological Origins of Modularity

The early inspiration of ANNs as ML models came from biology (Goodfellow et al., 2016). Biological neural circuitry is composed of densely connected networks of cellular units called neurons. A neuron is a cell specialized in communicating signals through electrical conduction. Each neuron consists of a cell body (soma) which has several arborizations called dendrites and a cylindrical protruding structure called axon. Information processing in nervous systems takes different shapes. In the most common mechanism, a neuron receives input signals from other neurons, processes and integrates them, and then decides to fire, i.e. send a signal to its downstream neurons, or not. Connection sites between neurons are called synapses and firing occurs by sending an electrical signal, called action potential, that travels down the axon till it reaches its terminals. When the action potential reaches the axonal terminals, it stimulates the release of a chemical neurotransmitter that will cross the connection gap (synaptic gap) between the axon and the receiving neurons to stimulate the post-synaptic neurons. It is notable to mention that different synapses have different strengths that reflect how much a pre-synaptic neuron can affect a post-synaptic neuron. The change in these synaptic strengths, known as plasticity, is the main mechanism behind learning processes in nervous systems.

ANNs started as significantly simplified models of their biological counterparts. Neurons are represented by simple non-linear mathematical functions that are conceptually arranged in layers and interconnected by weights, which represent the synaptic strengths between them. Over the development of the field, ANNs became more oriented towards solving engineering probelms and departed more and more from being a faithful replication of their biological origins. Nonetheless, cross-fertilization between the neuroscientific and ANN fields continues to emerge and inspire researchers on both sides from time to time (Hassabis et al., 2017).

A very important organizing principle in almost every biological nervous system, specially in animals having enough brain complexity to show sophisticated social and intelligent behaviour, is modularity. In the neural context, modularity is used to describe the structural or functional arrangement of functional units into aggregations, where each aggregation of units shows dense internal interaction and sparse interaction with other aggregations. Modularity shows at many levels in the brain, in a spectrum ranging from the micro-level of synapses (Kastellakis et al., 2015) to the macro-level of brain regions exhibiting functional specialization (Schwarz, Gozzi, and Bifone, 2008). Lesion studies and fMRI imaging provided ample evidence that brain regions can be very specialized in specific functions. Other theoretical and analytical evidence

comes from graph theoretical studies of brain networks. In an evolutionary context, many reasons have been hypothesized to explain the natural pressure towards modular brain networks. In one hypothesis, it was proposed that modularity facilitates evolving brain functions without introducing disruption to old functionality (Clune, Mouret, and Lipson, 2013). Another proposed reason is facilitating exaptation, where ancient biological structures can be readapted to different functions over evolutionary time (Kashtan and Alon, 2005). Modular structures are proposed to be more exaptation-friendly due to their contained functionality. We elaborate more on the biological origins of modularity in Section 2.3.

1.7 Modularity in ANNs

A Modular Neural Network (MNN), in the artificial context, is an ANN that can be decomposed into a set of subnetworks based on its connectivity pattern. As ANNs can be regarded as graphs (in a graph theoretical sense), different techniques exist for analysing modularity in ANNs and they originate from two different domains, computer science and sociological studies (Newman, 2004; Newman, 2006), where modularity is usually established using graph partitioning techniques. These techniques were mostly developed for other applications before the interest in the modularity of ANNs. Computer science, for example, has used graph partitioning for optimizing parallel processing and sociology applied these techniques to community detection. In more recent approaches, more advanced modularity measures were studied (Newman, 2004; Newman, 2006; Newman, 2016; Tyler, Wilkinson, and Huberman, 2005; Radicchi et al., 2004).

Based on our work in Chapter 2, modularizing an ANN has to be done at multiple levels of abstraction. These are: domain, topology, formation and integration. Domain modularity is an optional step in modularization. Domain modularity is concerned with partitioning the input space to allow specialized processing. For example, input decomposition can be done to separate different modalities, data with different characteristics or different pre-processing. Domain modularization can be manual or learned. While manual decomposition is an obvious solution in some cases, learned decomposition is a more flexible technique in general since the problems addressed by ML are generally complex and their input space is by definition hard to separate. We discuss domain modularity more extensively in Section 2.4.1.

Choosing a modular topology is the next step in the modularization chain. Topology refers to the general blueprint or skeleton of the MNN (Miikkulainen, 2010). Modular topologies have multiple modules, which are densely connected internally and sparsely connected to other parts of the network. We identified and systemically categorized many modular topologies that exist in the literature and we discuss them in more details in Section 2.4.2. Many variants of

modular architectures exist; some of them are highly regular while others are complex irregular structures.

While the modular topology is concerned with the general plan of the MNN architecture, formation on the other side is concerned with how these modules can be constructed, learned and interconnected between each others to form the network. This can be done manually, using EAs or other forms of learning techniques. We elaborate on formation in Section 2.4.3. After the topology is selected and the formation technique is established, a final step is how the different modules can be integrated to give the final output. Integration can be either cooperative, where all of the modules contribute to the final output, or competitive, where a module or a subset of modules is selected over the others. We further discuss integration in Section 2.4.4.

1.8 Why Modularity?

As we discussed, from an evolutionary and biological perspective, modularity has evolutionary advantages like facilitating the evolution of new functions and exaptation. In the artificial context, modularity can serve many functions. As we discuss and show several examples in Section 2.4, decomposable problems can be easily modularized, which makes debugging easier and adding enhancements less disruptive to the existing functions.

Transfer learning is an ML technique where a model developed for a task is reused to solve a different task. Pretrained modular components can be used in different ways to make it easier to transfer and integrate knowledge from different sources. We list multiple examples in Section 2.4 of similar usage in the literature. For example, by modularizing a robotic control network, Zhang et al. (2016a) decoupled perception from control and, hence, made it easier to transfer knowledge between simulation and real environments. Another example is Calabretta et al. (2000), where learned modules can be reused by duplication in an EA by mutating a set of duplicating genes.

Model capacity has proved to be of great importance to generalization, specially for complex tasks that need large amounts of data. The main downside of large models is time and memory complexity. Latency is the increased processing runtime of a neural network, usually as measured at inference time. Hence, latency is a proxy measure of the time complexity of a neural network. However, due to the difficulty of measuring the actual runtime latency, the model parameter count is often used as a proxy measure of the latency. Since the model parameter count is also a proxy measure of the memory complexity, it is an overall good measure of the suitability of a model to limited-resource applications. Other things being equal, an MNN is usually sparser than its monolithic counterpart (i.e. a non-modular model with all of the other architectural hyperparameters being the same). In Chapter 3, we show that we can

balance between complexity and accuracy in a systematic way such that a required trade off between latency and accuracy can be realized. Also, we show in Chapter 4 that we can modularize CapsNet to decrease its complexity and, at the same time, achieve better or comparable results. Another perspective regarding capacity is that modularity can act as a regularizer. Models that are well-engineered to have lower effective capacity can be less susceptible to overfitting.

Another important aspect of an MNN that results from the sparse connectivity between its modules is its readiness for parallelization and distribution. Since, by definition, modules connect sparsely between each other, the overhead of distribution across multiple computing nodes is decreased. Moreover, independent modules can be executed in parallel. This allows MNNs to benefit from parallel and distributed computing, either in multi-GPU or distributed computing settings.

Modularity can also benefit scenarios where a modular structure can exploit the properties of the problem being tackled. CNNs use filters to process their inputs. A convolutional filter can be considered a small linear module, and indeed it was generalized to a more complex subnetwork in Lin, Chen, and Yan (2013). The usage of filters exploited the intrinsic structure of images, mainly, the translation invariance and the decomposability of a scene into discrete objects. This kind of prior knowledge that is integrated into a model intrinsically is called inductive bias (Hüllermeier, Fober, and Mernberger, 2013). We use a similar concept in Chapter 5 to increase the resistance of convolutional-based networks to noise and adversarial attacks.

We can summarize the main benefits that can be achieved through modularity in the following:

- Accumulating functionality with less disruption to existing functions and easing debugging.
- Transfer learning.
- Less complexity and more regularization.
- Computational distribution and parallelization.
- Problem-specific inductive bias.

1.9 Challenges to Modularity

MNNs have a long history in the deep learning field and they have been studied either implicitly or explicitly (Happel and Murre, 1994; Caelli, Guan, and Wen, 1999; Sharkey, 1996; Xu, Krzyzak, and Suen, 1992). One of the main obstacles to defining a modular architecture is that there are usually a lot of

hyperparameters that control it. A lot of decisions regarding architectural design have to be made such as the number of modules, the connectivity pattern and the architecture of separate modules. In Section 2.4.2, Section 2.4.3 and Section 2.4.4, we describe many techniques that were used in the literature to rationally make such choices either manually or through different optimization techniques. However, this is still generally an open question and a very active research area that has branched out into different directions like using EAs or NAS.

Another open question regarding architectural design is finding suitable training objectives that can optimize naturally for modularity. Two notable strategies pertaining to EAs are penalizing connection cost and cooperative coevolution. Another technique involves restricting the resulting architectures to be modular in nature through some inductive bias to the learning process. We further discuss these techniques in Section 2.4.3.

Another complication in learning modular structures is ensuring diversity and functional specialization of different modules. Obviously, a set of modules that compute the same function is redundant and generally not desired. Some techniques can enforce such diversity. Similar to what Dropout (Srivastava et al., 2014) aims to do generally in ANNs, DropCircuit (Phan et al., 2016; Phan et al., 2017) is a regularization technique for Multipath NNs (a topology of MNNs) where modular paths are dropped randomly to introduce independence of features between paths. In the cooperative coevolution used in Garcia-Pedrajas, Hervas-Martinez, and Munoz-Perez (2003), two fitness measures were used to enforce competition and cooperation, where the former ensures that different modules do not converge to the same function and the latter ensures that the module functions are complementary. For some datasets and applications, diversity can be introduced by feeding different types of inputs, either from different modalities or from different preprocessing steps. For example, Wang (2015) used a multipath NN, where the input to the first path is the raw image, while the input to the other is a filtered version of the same image.

We believe that a major problem that contributes to the difficulty of modularity research is the efficiency of implementation. One of the main factors that contributed to the acceleration of ANN research in recent years was the significant efficiency of learning and inference made possible by GPUs. GPU acceleration depends on the fast parallel multiplication of dense matrices. MNNs, however, are usually sparse and sometimes have irregular structure which make them unable to benefit from their sparsity and reduced complexity when it comes to implementation efficiency.

Since the research we did in Chapter 5 has a similarity to conventional convolutional layers, we could make use of the acceleration already established in the existing tensor libraries. Convolutional layers are regular structures that can be reduced to dense matrix-matrix multiplication and hence can benefit from GPU acceleration. We could apply a similar trick to the work we did

in Chapter 4. By making the paths regular, i.e. having the same width and depth, we could reduce the parallel computations across the independent paths into dense matrix-matrix multiplications. We achieved that by reducing the convolutional layers across the different paths into a computational form similar to depthwise-separable convolutions and we could, again, utilize the optimized convolutional operators found in existing tensor libraries. We discuss this in more detail in Chapter 4. Unfortunately, the same was not possible in the case of Chapter 3 since we had to deal with irregular paths having different widths. So, eventually, we relied on approximating the sparse matrices with dense matrices having zero entries. Still, however, this was possible since we had to rely on this approximation only at inference time.

1.10 Our Research

In this work, the main problem we are trying to address is:

> **"How can we use modularity to improve some of the essential performance measures of neural networks?"**

In order to address the research problem, we needed to know the existing modularization techniques and how modularity is used in the literature to address different problems. Hence, in Chapter 2 we did an extensive review of the existing literature that addresses modularity in ANNs. We substantially expanded previous reviews and established a systematic taxonomy of MNNs and identified a characteristic process of modularizing NNs. This review lays the foundation and context for the methods and architectures used later in the subsequent chapters.

Since NNs have many different performance measures that can be studied, we had to select a subset of these measures to address using modularity. We focused on three essential problems in the field of deep learning. The problems were chosen based on their impact and practical importance. The first and second problems focused on how the accuracy of an NN is affected by its capacity. The relation between capacity and accuracy is a very important problem that affects the application of NN in limited resource environments. The high-capacity nature of ANNs makes overfitting noise in the data a serious complication in their practical applications. Hence, the third problem focused on how the accuracy of an NN is affected by noise. Noise is an integral part of any realistic dataset and the ability of ANNs to generalize is largely affected by their robustness to noise. Moreover, engineered noise like adversarial attacks represents a serious complication to the usage of NNs in mission critical applications.

In the first problem (Chapter 3), we addressed the trade-off between the accuracy of an NN and its latency. By modularizing a monolithic NN and hence

making it more sparse, we could draw a relation between the NN accuracy and latency which can be used to satisfy a mutual accuracy-latency requirement. In the second problem (Chapter 4), we addressed the latency measure but with the stronger constraint of maintaining the target NN accuracy. We could redesign a target NN architecture to be more modular. The modular design had lower capacity but at the same time had a similar or better performance compared to the original architecture. In the third problem (Chapter 5), we focused on how the accuracy is affected by noise. We introduced a modular layer that can be integrated into CNNs and increase their robustness to random and engineered noise.

The main Research Objectives (ROs) we addressed are:

1. RO: Do an extensive review of modularity practices in the literature in order to formulate a general modularization framework which will provide the context and the toolbox needed to improve a subset of NN performance measures.

2. RO: Use differential NAS to optimize the modularity of a multipath NN and establish a relation between its accuracy and latency.

3. RO: Apply modular techniques to CapsNet in order to reduce its latency while maintaining its accuracy.

4. RO: Design a modular layer that can be integrated into CNNs and increase their robustness to noise.

Our main contributions:

1. We established a general framework that captures how modularity is applied in a variety of settings in the literature.

2. We used differential NAS to modularize a monolithic network into a modular multipath NN and used sampled architectures to establish a relation between the accuracy and latency of the target family of architectures.

3. By applying multipath modularization, DropCircuit regularization and a modified routing technique, we could reduce the latency of CapsNet while obtaining similar or better performance compared to the original architecture.

4. We designed a modular layer that resembles convolutional layers and can be readily integrated into CNNs to increase their robustness to random noise and adversarial attacks.

In Section 1.10.1, Section 1.10.2, Section 1.10.3 and Section 1.10.4, we describe in more details the research we have done in subsequent chapters, which research objectives we target, what are the research questions we want to answer and the contributions we achieved.

1.10.1 Review of Modularity in ANNs

Since we aim at doing a horizontal study of using modularity in solving different problems that face the deep learning field, we need a solid foundation of the existing contributions in this domain and how modularity was previously studied and used to solve problems in deep learning. Moreover, during reviewing the literature, we spotted that previous reviews in this domain were outdated and focused only on applications and lacked a systematic framework that captured the modularization process in ANNs. Hence, in Chapter 2 we extensively analyse the implicit and explicit study of modularity in the literature and, based on that, we establish a general modularization framework and a systematic taxonomy of modularity techniques that capture the essential properties of the modularization process. Finally, we present a set of case studies that apply our framework to prominent research in the field to test its consistency and practicality. Through this review, we aim to address the following ROs:

1. RO: Cover a sufficiently wide range of literature addressing modularity to spot explicit and implicit relevant studies.
2. RO: Analyse the existing contributions to capture the defining patterns that characterizes the modularization process.
3. RO: Establish a general framework and taxonomy that captures the essence of the modularization process.

We answer the following Research Questions (RQs):

1. RQ: Which of the previous research in the deep learning field contains elements of modularity?
2. RQ: What are the main elements that are common to existing modular techniques?
3. RQ: Are there any common steps that are being followed by modular techniques?
4. RQ: What are the logical steps that need to be followed to turn a monolithic network into a modular network?
5. RQ: How can modularity be quantified?

We achieve the following contributions:

1. We significantly expand previous reviews to include the various modularity variants that exist in the literature.

2. We provide general systematic principles organized in a framework that describes the modularization process across different MNNs.

3. We fit our framework to several prominent use cases in the literature to show how it captures the big picture and provide insights into MNN modeling.

1.10.2 Balancing Latency and Accuracy

As discussed, ANNs are growing in capacity in order to accommodate larger datasets and approach higher generalization standard. While this paradigm may be suitable for the compute-centric approach, it is not suitable for the data-centric approach which utilizes distributed learning on devices with limited resources. Modularizing an FC network by slicing it into multiple paths can introduce sparsity and, hence, reduce model complexity. However, this will need a set of architectural hyperparameters that are critical to the accuracy of the model like the number of paths and the width of each path. Optimizing these parameters by brute force is infeasible for practical applications, so the modeler has to have a way of predicting the interplay between these hyperparameters and how they affect latency and accuracy in order to make a guided decision based on the required latency-accuracy trade off. In Chapter 3, we aim to address this by seeking the following ROs:

1. RO: In the context of a multipath network, use differential architectural search to optimize for the number and widths of paths starting from an FC network.

2. RO: Use data derived from this optimization process to evaluate sampled architectures.

3. RO: Establish a relationship between latency and accuracy from the evaluated architectures and use it to predict the performance of any potential architecture.

We try to answer the following RQs:

1. RQ: Does the optimization process yield multipath networks with consistent relative balance between latency and accuracy?

2. RQ: Does the information derived from the optimization process have good predictability of the latency-accuracy relationship for unseen models?

3. RQ: Do the predictions have good correlation with the actual test performance?

4. RQ: Is evaluating a small sample from the architectural space sufficient for good predictability?

We achieve the following contributions:

1. We use a one-shot model to approximate the relative accuracy between potential multipath models with different latency measures.
2. We use a pruning technique to modularize an FC network into a set of balanced multipath networks.
3. We evaluate a small set of sampled architectures and use the results to predict the balance between latency and accuracy for the full spectrum of potential models to help the modeler make a rational decision given specific latency-accuracy constraints.

1.10.3 Reducing Complexity and Maintaining Accuracy

As discussed, an MNN is generally more sparse than an equivalent model, other things being equal. Hence, a good potential for exploiting modularity is introducing sparsity in an existing less modular model to decrease its complexity in terms of parameter count. However, blindly introducing sparsity can lead to degradation in accuracy due to decreased capacity and underfitting. Modularity opens the way for modular changes that we can use to balance the introduced sparsity and hence maintain accuracy. In Chapter 4, we investigate the application of this idea to CapsNet (Sabour, Frosst, and Hinton, 2017). CapsNet achieved a reasonable accuracy on different datasets and its operating concept is different from conventional CNNs. However, CapsNet is a shallow network that has a large proportion of its parameters concentrated in two very wide convolutional layers. Capsules, the main architectural addition defining CapsNet, are built subsequently on top of these layers. Generating these capsules by deep independent paths can significantly cut the number of parameters, while the expressiveness added by depth and other suitable adjustments can compensate for the reduced capacity. To investigate this, we aim to achieve the following ROs:

1. RO: Generate capsules by deep independent paths with balanced width, instead of wide shallow layers, to decrease the network complexity.
2. RO: Reduce the intermediate representations by introducing max-pooling layers in the paths to cut out more parameters.
3. RO: Exploit a multipath-aware regularization technique to maintain network performance.

4. RO: Investigate alternatives to the routing process that are more compatible with the multipath modifications adopted.

We address the following RQs:

1. RQ: Can the expressiveness introduced by depth compensate for the reduced capacity in terms of accuracy?
2. RQ: How will reducing the internal representation by max-pooling affect the network accuracy?
3. RQ: Will DropCircuit, a multipath regularization technique, help maintain the network accuracy?
4. RQ: What will be the effect of modifying the original routing by agreement to be in a fan-in direction? Which variant is more compatible with our modifications?
5. RQ: What is the effect of the introduced modifications on the correlation between capsules? Can this contribute to the multipath variant's accuracy?

Our contributions are:

1. We introduce a deep modular version of CapsNet that is significantly cheaper in terms of parameters, nonetheless with similar or better accuracy compared to the original variant.
2. We show that the coordination of depth, max-pooling, fan-in routing and regularization by DropCircuit made possible by the modular structure can adequately compensate for the reduced absolute capacity to maintain model accuracy.
3. Through our analysis using Representational Similarity Analysis (RSA), we show that modularity can have interesting effects on the internal representations of ANNs and the study of these dynamics can have a significant impact on the study of neural networks.
4. We open the door for more critical investigation regarding routing techniques in CapsNet, their importance and their interaction with data and architectures.

1.10.4 Reducing Sensitivity to Noise and Adversarial Attacks

ANNs in general, and CNNs being no exception, are sensitive to noise. Random noise exists in any realistic dataset and any ML model has to filter out this noise in order to have good generalization. An adversarial attack is an artificially engineered noise that can disrupt the performance of a neural network significantly, while being imperceptible to humans. These kinds of attacks are usually targeted at visual data and hence are mostly applied to CNNs. Adversarial attacks can also be applied to other modalities such as audio data (Takahashi, Inoue, and Mitsufuji, 2021).

CNNs are specialized NNs that are designed to be more effective at processing visual data. As discussed before, instead of the fully connected arrangement from a given layer to the next, CNNs depend on local connectivity and shared weights. From a signal processing perspective, a convolutional layer is a set of linear filters convolved with the layer's input and followed by a non-linearity. A filter can be generalized to any module with arbitrary structure like what has been done in Lin, Chen, and Yan (2013). Hence, a generalized convolutional layer is a modular layer.

In Chapter 5, we aim to increase CNNs robustness to noise and adversarial attacks. We build on the modular structure of convolutional layers to introduce a similar modular layer, called Weight Map (WM) layer that increases the network resistance to noise and adversarial attacks. A WM layer operates in two steps. First, a grid of weights, with the same dimensionality as the input, is elementwise multiplied by the input. Then, a reduction operation is applied which utilizes constant-value non-learnable 2D filters. We have the following ROs:

1. RO: Assess the effect of WM layer and different reduction operations on the accuracy as measured on the raw noise-free data and on the robustness of different architectures to uniform noise and adversarial attacks.

2. RO: Investigate the possible ways of integrating WM layer in a convolutional-based network and the effect on the baseline performance.

3. RO: Analyse the observed effect of the WM layer and its possible working mechanism.

We address the following RQs:

1. RQ: What is the effect of the WM layer on noise robustness when applied to different architectures and different datasets at different noise intensities?

2. RQ: What kind of reduction operations can achieve noise robustness while maintaining the baseline accuracy as much as possible?

3. RQ: What contributes to the specific performance of a reduction operation?

4. RQ: How can we introduce a WM layer in a convolutional based network with minimal disruption to its performance?

5. RQ: How do the features extracted by the network vary from the baseline and how do they relate to the layer's dynamics?

Our contributions are:

1. We show that the WM layer can be integrated in convolutional-based NNs to increase their robustness to noise and adversarial attacks with little disruption to their baseline noise-free performance.

2. We introduce two reduction operations: smoothing and unsharp. We show that while both can increase robustness, unsharp can outperform smoothing by mitigating the latter's over-smoothing effect.

3. We propose activation-variance amplification as the working principle of this layer and discuss its similarity to explicit adversarial training. This further opens the possibility for more future work in similar directions.

1.11 Experimental Design and Implementation

The core neural network algorithms were implemented in PyTorch (Paszke et al., 2019). We used docker containers for dependency management. The main experiments were always automated using scripts that took care of hyperparameter persistence to ensure reproducibility. Raw experimental data were later analysed using Jupyter notebooks. We used Pandas for data analysis along with Matplotlib for visualization. Experiments were accelerated on different Nvidia GPUs including Geforce GTX: 1060, 1080 and 1080Ti and Tesla: K80, P40, P100 and V100.

When assessing a performance measure of a NN, we take the average of three trials with different random network initializations. This is a common methodology in the field (Webb et al., 2020; Omar et al., 2013; Xiao et al., 2018; Sohoni et al., 2019). Unless otherwise specified, we train a given model on the target dataset for 300 epochs (Wang et al., 2016; Mehta et al., 2019; Guo et al., 2019; Bello et al., 2017). We benchmark using three datasets, namely MNIST, CIFAR10 and iWildCam2019 which we describe next.

1.11.1 MNIST

The MNIST dataset is a collection of images of handwritten digits with 10 classes representing the digits from 0 to 9. The images are grayscale with spatial

dimensions of 28x28. MNIST consists of a training dataset of 60k samples and a testing dataset of 10k samples. We divide the training dataset into 90% (54k) for training and 10% (6k) for validation. Unless we specify otherwise, no augmentation was used for the training data. The test dataset was used as it is.

1.11.2 CIFAR10

The CIFAR10 dataset is a collection of colored (3-channels) images comprising 10 classes representing animals and vehicles. The images have 32x32 spatial dimensions. CIFAR10 consists of 50k training images and 10k testing images. We divide the training set into 90% (45k) for training and 10% (5k) for validation. No augmentation was used for the training data. The test dataset was used as it is.

1.11.3 iWildCam2019

The iWildCam2019 dataset is a collection of colored animal images captured in the wild by camera traps. The dataset is imbalanced and there is only a partial overlap between the classes found in the training and test datasets. To balance the dataset and fix the classes between training and testing, we used only the training set by splitting it into train, validation and test sets. This was done by first choosing 10 classes (deer, squirrel, rodent, fox, coyote, raccoon, skunk, cat, dog, opossum), then balancing all the classes by choosing only 1000 samples from each class. We then split the data into 70% (7k) train, 20% (2k) validation and 10% (1k) test. This was done class-wise to maintain the balance. We preprocessed all the sets by converting the images to grayscale and downsampling to 23x32 spatial dimensions. No augmentation was used.

1.12 Thesis Guide

In this chapter we have discussed the theoretical foundation for ANNs and modularity in the artificial domain. We have also discussed the main problems and challenges we tackle in the following chapters and the contributions we offer for solving these problems. In Chapter 2, we present our extensive review of the modularity literature in deep learning. We propose a systematic classification of MNNs and a modularization framework that captures the essence of designing MNNs. In Chapter 3, we investigate the problem of balancing latency and accuracy in ANNs. Through modularizing an FC network into a multipath architecture, we can approximate the relation between the efficiency gained by multiple paths and the accuracy lost by the decreased capacity. Based on this predictive relation, a modeler can make a rational decision to meet specific restrictions of latency and accuracy.

In Chapter 4, we modularize CapsNet by replacing its shallow wide layers by deep balanced paths to reduce its complexity. We investigate using max-pooling, a different routing technique and DropCircuit regularization to further reduce its complexity while maintaining the accuracy as much as possible. In Chapter 5, we use the general modular structure of convolutional layers to introduce the WM layer. We show that the WM layer can increase ANN robustness to noise and adversarial attacks. In Chapter 6, we give a general discussion of how modularity can be used to enhance some of the essential performance measures of ANNs, what conclusions can be drawn regarding the impact of modularity and what are the potential future research opportunities made possible by our contributions.

Chapter 2

A Review of Modularization Techniques in Artificial Neural Networks

An adaptation of this material has been published in Amer and Maul (2019).

2.1 Preface

The adaptation of NNs as ML models has diverged significantly from their biological counterparts. Due to the nature of the engineering problems and the characteristics and limitations of computational hardware, many simplifications were made for the sake of feasibility and efficiency. However, cross-fertilization of ideas between the biological study of neural circuitry and the engineering practices of ANNs never stopped throughout the turbulent history of neural networks. Insights of new architectures, optimization algorithms and internal dynamics coming from ANNs inspired new ideas for studying and understanding biological systems (Barrett, Morcos, and Macke, 2019). The feedback in the other direction consisted mainly of adoptions of more concepts, structural ideas and mechanisms from biological neural systems into ANNs.

Modularity is a well known property in the study of the brain and related neural structures. Since the emergence of Neuroscience as the formal study of the nervous system in different organisms, it was recognized that different brain areas are specialized in the processing of different sensory, motor and integration functionalities (Purves et al., 2004). While functional modularity is a qualitative property that can be understood using methods like ablation and lesion studies, topological modularity, on the other hand, can be quantified using formal measures. Graph theory has a pivotal role in shaping and studying these measures. A neural network can be considered a graph with neurons corresponding to vertices and synapses corresponding to edges, and so neural networks lend themselves naturally to the formal analysis of Graph theory. We explore some of these measures later in this chapter.

The desire to bring modularity into the realm of ANNs has existed for a long time in the field (Happel and Murre, 1994; Caelli, Guan, and Wen, 1999; Sharkey, 1996; Xu, Krzyzak, and Suen, 1992; Auda and Kamel, 1998; Auda and Kamel, 1999). The problem domains that ANNs excel in contain disparate problems differing in input structure, learning requirements, inductive bias and output properties, to name a few. With the advancement of the field, different architectures and techniques excelled in different niches. For example, CNNs (Fukushima and Miyake, 1980; Lecun et al., 1998) were one of the early examples of such architectures developed as specialized networks for the visual modality based on the structure and nature of the corresponding input. Similarly, many specialized architectures exist for audio and speech (Oord et al., 2016), natural language processing (Vaswani et al., 2017) and generative models (Goodfellow et al., 2014), among others. A way of integrating different architectures, or more generally, topologically distinct and functionally specialized modules, into a MNN is necessary to solve multimodal tasks and design more autonomous AI agents. Later and through out the following chapters, we show that modularity is not only useful as a way of integrating modules that specialize in different tasks, but also can be used to enhance different performance measures and solve some problems tightly related to ANNs.

As our aim in this work is to enhance different ANN performance measures, we needed a general framework that can give a bird's-eye view of modular practices in the field and capture the defining patterns in MNNs. Hence, in this chapter, we have performed an extensive review of modularity in ANN literature and established a systematic taxonomy of modular techniques and a general methodology capturing the modularization practices used in MNNs.

We identified that the modularization, i.e. the process of designing MNNs, of ANNs happens at multiple levels of the ANN design process. The first level is modularizing the domain of the problem. Some problems have a structure that lends itself to natural decomposition, like the need to perform specific well defined tasks to solve the problem. In this case, different modules can operate on different partitions, either explicitly or implicitly. This is an optional level of modularization. A neural network can still be modular without explicit domain modularization. At the second level comes the network topology. Here, the architecture of the network is specified in terms of how many modules are used, how they are arranged and connected and what is the sub-architecture of each module, among other structural properties. Having determined the details of the architecture, the next level deals with the formation of modules. Given the problem domain and architecture, is there an obvious way of constructing and arranging these modules? Does the sub-architecture of each module and the arrangement of modules into a full network need an automated optimization process to search the space of possible modules or is there a clear structure that is better enforced manually? At the final level, the question of how the different modules will be integrated to produce the system's final output is addressed. For

example, in some systems the output arises naturally from the learning process, while for other systems the output can be calculated in a more algorithmically defined way. At the end of the chapter, we present some concrete use cases from the literature to further solidify the presented concepts.

In the upcoming sections, we first discuss modularity, its origins in biological systems and its formal treatment and quantification. We then present our systematic taxonomy and the general modularization methodology capturing the different techniques in the literature. During the discussion of these techniques, we present a recent extensive review of the different MNNs found in the literature and further establish a general practical framework of modularity. Finally, we present detailed use cases of some prominent MNNs in the literature to help portray the full picture of the methodology. This chapter will act as a contextual foundation for the subsequent chapters; a frame of reference that relates the different modular ideas that we investigate later.

2.2 Introduction

Modularity is the property of a system whereby it can be broken down into a number of relatively independent, replicable, and composable subsystems (or modules). Although modularity usually adds overhead to system design and formation, it is often the case that a modular system is more desirable than a monolithic system that consists of one tightly coupled structure.

Each subsystem or module can be regarded as targeting an isolated subproblem that can be handled separately from other subproblems. This facilitates collaboration, parallelism and integrating different disciplines of expertise into the design process. As each module is concerned with a certain subtask, the modules can be designed to be loosely coupled, which enhances its fault tolerance. Also, a modular design with well defined interfaces makes it easier to scale and add more functionality without disrupting existing functions or the need for redesigning the whole system. Moreover, as modules correspond to different functions, error localization and fixing tend to be easier.

An MNN is a neural network that embodies the concepts and practices of modularity. Essentially, an MNN can be decomposed into a number of subnetworks or modules. The criteria for this decomposition may differ from one system to another, based on the level at which modularity is applied and the perspective of decomposition.

Since the inception of artificial neural networks and throughout their development, many of their design principles, including modularity, have been adapted from biology. Biological design principles have been shaped and explored by evolution for billions of years and this contributes to their stability and robustness. Evolutionary solutions are often innovative and exhibit unexpected shortcuts or trade-offs that, even if not directly implementable, often provide useful insights. Mapping from biological principles to in-silico realizations is

not a linear one-to-one process. However, there are several common steps that can be shared by different such realizations. Ideally, the process of adapting a biological design principle to an artificial neural network implementation starts with identifying the key function(s) underlying the design principle. Usually, there are several complex biological details that are irrelevant to the functional essence of the principle, which can thus be abstracted away. This may be followed by some enhancement of the identified function(s) in the artificial domain. Finally, the abstracted and enhanced principle is mapped to an artificial neural network construct using a flexible platform. CNNs (Lecun et al., 1998; Fukushima and Miyake, 1980) are a poignant success story of the adoption of some of the key design principles of the visual cortex. The model of the visual cortex was greatly simplified by CNNs by eliminating complexities like the existence of different cortical areas (e.g. areas V1 and V2) and pathways (e.g. ventral and dorsal streams) and focusing on receptive field, pattern specific regions and a hierarchy of extracted features. These were realized using linear filters, weight sharing between neurons and deep composition of layers. More examples include RNNs, which are inspired by the brain's recurrent circuits (Douglas and Martin, 2007), and Parallel Circuit (PC) neural networks (Kien Tuong Phan, Maul, and Tuong Thuy Vu, 2015), which take their inspiration from retinal microcircuits (Gollisch and Meister, 2010).

Biological nervous systems, the early inspiration behind neural networks, exhibit highly modular structure at different levels, from synapses (Kastellakis et al., 2015), to cortical columnar structures (Mountcastle, 1997), to anatomical (Chen et al., 2008) and functional (Schwarz, Gozzi, and Bifone, 2008) areas at the macro level . It has been proposed that natural selection for evolvability would promote modular structure formation (Clune, Mouret, and Lipson, 2013). Modularity is evolvable as it allows for further evolutionary changes without disrupting the existing functionality. It also facilitates exaptation, where existing structures and mechanisms can be reassigned to new tasks (Kashtan and Alon, 2005). Recognition of this prevalence of modularity in biological neural systems has left an indelible albeit somewhat irregular mark on the history of ANNs, mostly under the guise of biologically plausible models. However, with the divergence between the fields of ANNs and Neuroscience, the ANN approach has tended to become more engineering oriented, getting most of its inspiration and breakthroughs from statistics and optimization.

Many researchers have long been aware of the importance and necessity of modularity. For example, the ANN community has for many years recognized the importance of constraining neural network architectures in order to decrease system entropy, lower the number of free parameters to be optimized by learning, and consequently, have good generalization performance. In Happel and Murre (1994), it was argued that constraining neural architectures through modularity, facilitates learning by excluding undesirable input/output mappings, by using prior knowledge to narrow the learning search space. In Caelli,

Guan, and Wen (1999), modularity is considered a crucial design principle if neural networks are to be applied to large scale problems. In Sharkey (1996) and Xu, Krzyzak, and Suen (1992), some of the early techniques of integrating different architectures or modules to build MNNs are discussed. In Caelli, Guan, and Wen (1999), six different MNN models were analytically dissected, in an attempt to provide several modeling practices. On the other hand, in another comparative study (Auda and Kamel, 1998), ten different MNN models were empirically compared using two different datasets. In Auda and Kamel (1999), a survey was done about MNNs, where the MNN design process was broken down into three stages, starting by task decomposition, then training and finally, decision making.

Despite this early interest in neural network modularity, previous research (Waibel, 1989; Happel and Murre, 1994; Fritsch, 1996; Ronco and Gawthrop, 1995; Auda and Kamel, 1998; Auda and Kamel, 1999; Sharkey, 1996; Chris Tseng and Almogahed, 2009; Kacprzyk and Pedrycz, 2015; LeCun, Bengio, and Hinton, 2015) has generally focused on particular MNN models and has lacked systematic principles and a broad general perspective on the topic. Previous research has also been lacking in terms of a systematic analysis of the advantages and disadvantages of different approaches (Chris Tseng and Almogahed, 2009), with an increased focus on empirical comparisons of very specific models and applications (Waibel, 1989; Auda and Kamel, 1998; Fritsch, 1996). Even for theoretically focused reviews, the taxonomy is sparse and fails to capture important properties and abstractions (Ronco and Gawthrop, 1995; Auda and Kamel, 1999; Sharkey, 1996; Kacprzyk and Pedrycz, 2015). Moreover, the scope of modularity focused on is very narrow, ignoring important forms of modularity and focusing mainly on ensembles and simple combinations of models (Sharkey, 1996; Happel and Murre, 1994; Fritsch, 1996). These limitations need to be addressed if modularity is to be applied more generally. More general insights and a toolbox of modularity-related techniques are needed for consistently implementing successful MNNs. Fortunately, recent MNN techniques have been devised and revisited, specially in the last decade after the revival of the ANN field in the form of deep learning.

In this chapter, we aim to expand previous reviews by introducing and analysing modularization techniques in the neural networks literature in an attempt to provide best practices to harness the advantages of modular neural networks. We reviewed prominent modular neural networks throughout the literature, inspected the different levels at which modularity is implemented and how this affects neural network behaviour. We then systematically grouped these techniques according to the aspect of neural networks they exploit in order to achieve modularity. Unlike previous reviews, our focus is the general systematic principles that governs applying modularity to artificial neural networks and the advantages and disadvantages of the different techniques. We

produced a general taxonomy that captures the major traits of different modular neural networks at different levels and for various modularity forms and a framework that captures the essentials of the process of building a modular neural network.

From our study of modular neural networks in the literature, we classified modularization techniques into four major classes, where each class represents the neural network attribute manipulated by the technique to achieve modularity. We thus categorized MNN operations into the following four classes:

1. Domain: this is the input space or the data an MNN operates on, which in turn defines and constrains the problem we are trying to address.

2. Topology: this corresponds to an MNN's architecture, which reflects the family of models that an MNN expresses.

3. Formation: this is how an MNN is constructed and what process is used to form its components.

4. Integration: this is how the different components of an MNN are composed and glued together to form a full network.

So, modularization techniques operating on the domain tend to act by finding a good partitioning of the input data, to which different modules can be assigned. This is the only modular level that is optional in the sense that you may have an MNN that does not have an explicit modularization of the domain, however, any neural network that is modular must use at least one technique from each successive level, which includes selecting a certain modular topology, a formation technique for building the modular architecture, and an integration scheme for combining the different modules. So, as mentioned, topological modularization is the next level at which modularity is achieved, where the technique is essentially a specification of modular topology. Every topological technique is a blueprint for the structure of the MNN, and therefore defines how nodes and modules are connected. Although the topological technique specifies how the MNN as a whole should be at the end, it does not specify how this architecture can be built. This is what formational techniques try to address. Formational techniques are the processes by which modular topologies can be constructed. Finally, while formational techniques focus on the building of modularity, integration techniques specify how different modules can be integrated together to achieve the desired system outputs. So, every modular neural network realization can be seen as chain of modularization techniques applied to each level or aspect of the network.

In section 2.3, we discuss modularity in the context of both ANNs and biological neural circuits in general, along with the different approaches for detecting and quantifying it, its evolutionary context and the practical importance and challenges in applying to engineering problems. In section 2.4, we discuss

the different modularization techniques applied to the different levels of designing an ANN and how different modularization chains can produce a variety of MNNs. In section 2.5, we analyse different state-of-the-art MNNs, applying our conceptual framework to show its explanatory power and emphasise its practical applicability.

2.3 Modularity

In the domain of neural networks, modularity is the property of a network that makes it decomposable into multiple subnetworks based on connectivity patterns. It can be argued that the shift of thinking towards functional modularity in the brain and biological neural networks, is one of the greatest leaps in Neuroscience since the neuron doctrine (López-Muñoz, Boya, and Alamo, 2006). The concept of emphasising the importance of relative connections between neurons and that functionality emerges from intra-modular and inter-modular interactions revolutionized the way we research nervous systems and transformed the idea of a connectome (Bullmore and Bassett, 2011; Sporns, 2011) into a key area of brain research.

As already mentioned, the brain has been shown to be modular at different spatial scales, from the micro level of synapses to the macro level of brain regions. At the level of synapses, it has been suggested (Kastellakis et al., 2015) that synapses show both anatomical and functional clustering on dendritic branches, and this plays a central role in memory formation. At a larger spatial scale, cortical minicolumns (Buxhoeveden, 2002) have been suggested to be the basic building unit of the cortex, largely supported by the claim that they have all of the elements of the cortex represented within them. Lesion studies, brain imaging using fMRI and several other techniques have shown strong evidence of brain modularity, where different areas and regions of the brain are specialized into certain cognitive or physiological functions. More recently, the pioneering work by Sporns, Bullmore and others and the introduction of graph theory into the study of brain networks have shed light on the small world nature of brain connectivity (Bullmore and Sporns, 2009; Sporns and Zwi, 2004). A small world network is a network characterised mainly by clusters which are groups of neurons having more interconnections within the cluster than would be expected by chance, while there is still sparse connectivity between different such clusters. Moreover, despite large networks and sparse inter-cluster connectivity, there is still a short average path length between neurons. In Sporns and Zwi (2004), it was observed that there is a direct correlation between clustering and path length, and between these two measures and brain area functionality. It was suggested that areas with short average path length and low clustering tend to be integrative multimodal association areas, while those with long average path length and high clustering tend to be specialised unimodal processing areas.

The graph theoretical approach to studying neural networks considers the network as a connected graph, where the neurons are represented by nodes (or vertices) and the synaptic connections between neurons as edges. Although, in practise, neural networks are directed graphs, i.e. edges have directionality from pre to postsynaptic neurons, for simplicity and tractability, most researchers in this area treat neural networks as undirected graphs. Central to quantifying the small world properties of biological neural networks is how to cluster or partition the nodes into modules, where each module has dense connectivity between its nodes and sparse connectivity with nodes in other modules. There is no single most efficient algorithm for solving this problem, and indeed it was proven to be an NP-complete problem (Brandes et al., 2008), however, similar problems have long been studied in Computer science and Sociology (Newman, 2004; Newman, 2006).

In the field of computer science, graph partitioning is a well studied problem, where given a certain graph and pre-specified number of groups, the problem is to equally partition the vertices into the specified number of groups, whilst minimizing the number of edges between groups. The problem was motivated by other applications before the interest in partitioning neural networks, like partitioning tasks between parallel processors whilst minimizing inter-processor communication. The main approach in computer science is a collection of algorithms known as iterative bisection, such as spectral bisection and Kernighan-Lin algorithm. In iterative bisection, first the graph is partitioned into the best two groups, then subdivisions are iteratively made until the desired number of groups is reached. The problem with these methods is that the number and sizes of groups are not known a priori when partitioning neural networks. Moreover, a lack of good partitioning measures leads these algorithms to deterministically partition the graph into the desired number of groups, even if the partitions do not reflect the real structure of the graph.

On the other hand, sociological approaches have focused more on the problem of community structure detection. Community structure detection consists of the analysis of a network in an attempt to detect communities or modules, where the algorithm does not pre-specify the number or size of groups. In other words, it is an exploratory approach, where the algorithm may detect subgraphs or may signal that the graph is not decomposable. This flexibility makes community structure detection more suitable for neural network research. The main technique used so far in sociological studies is hierarchical clustering. Based on a metric called similarity measure, hierarchical clustering constructs a tree-like structure of network components called a dendrogram. The horizontal section of this dendrogram at any level gives the network components produced by the algorithm. The algorithm does not require pre-specification of the number or sizes of groups, but it does not necessarily guarantee the best division.

In more recent approaches (Newman, 2004; Tyler, Wilkinson, and Huberman, 2005; Radicchi et al., 2004), a modularity measure (Equation 2.1) was

used to either guide the detection process towards the best division or evaluate the quality of the resulting partitioning. The intuitive notion of modularity as defined by Newman (2004) and Newman (2016) is that a good network division is one that places most of the network edges within groups, whilst minimizing the number of edges between groups. Network connectivity is assumed to be described by a real symmetric matrix called adjacency matrix $\mathbf{A}$, with dimensions $n \times n$, where n is the number of nodes in the network. Each element $\mathbf{A}_{ij}$ is 1 if there is an edge between node i and j and 0 otherwise. If we assume dividing the network into q number of groups, where g_i refers to the group to which node i was assigned, then the sum of edges within groups (i.e. between nodes of the same group) is $\frac{1}{2}\sum_{ij}\mathbf{A}_{ij}\delta_{g_i g_j}$, where $\delta_{g_i g_j}$ is the Kronecker delta. Maximizing this quantity alone is no guide towards a good division, because assigning all the nodes to one big group would maximize this measure whilst completely avoiding any partitions. To remedy this, modularity is taken to be the difference between this quantity (i.e. the actual sum of edges within groups) and the expected number of this sum if edges were placed randomly, whilst keeping the same partition. If the probability of node i connecting to node j after randomization is P_{ij}, then this expected sum is $\frac{1}{2}\sum_{ij}P_{ij}\delta_{g_i g_j}$, and the modularity measure is then

$$Q = \frac{1}{2m}\sum_{ij}(\mathbf{A}_{ij} - P_{ij})\delta_{g_i g_j} \tag{2.1}$$

where m is the total number of edges, which is used as a normalization factor. The most used randomization scheme is one that preserves node degree (i.e. number of edges attached to each node), and the probability of connecting node i and j under this scheme is $\frac{k_i k_j}{2m}$, where k_i is the degree of node i. Please refer to Figure 2.1 for an illustration of different neural topologies with corresponding modularity measures.

On the evolutionary side, multiple hypotheses have been proposed for explaining the origin of modularity in brain organization. The issue is important from the ANN perspective, as it provides inspiration for guiding evolutionary computational algorithms towards generating modular architectures. It was suggested that evolution in an environment with Modularly Varying Goals (MVGs) leads to modular networks (Kashtan and Alon, 2005). The MVG environment consists of varying goals with common subgoals. As the modularity of the solution obtained was usually limited, it was argued that the failure might be explained by the fact that the EA was directed more towards optimal solutions, which was sufficient to solve simple problems, but due to lack of evolvability, failed to scale up to more complex problems. In other studies, the competition between efficient information transfer and wiring cost of the brain have been suggested as sufficient evolutionary pressures for modularity (Clune, Mouret, and Lipson, 2013; Bullmore and Sporns, 2009). It was also

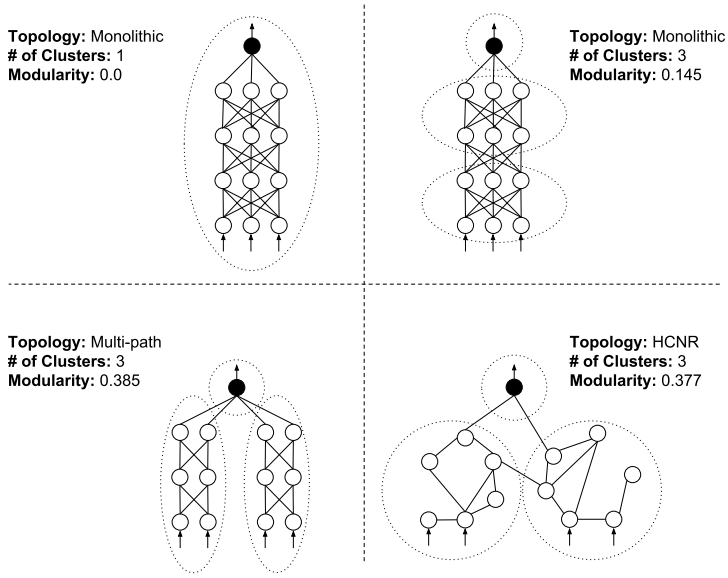


Figure 2.1. Calculated modularity measure (Newman, 2004; Newman, 2016) for different architectures. The partitions, marked as circles, used for calculations may not be optimal.

suggested that selection for minimal connection cost bootstrapped modularity, while MVG helped in maintaining it (Clune, Mouret, and Lipson, 2013).

Using multiple modules in practice is partly motivated by the existence of different subproblems, that may have different characteristics, promoting problem decomposition and functional separation of tasks, that typically contribute towards maintainability and ease of debugging. There are, however, difficulties that surround applying modular neural networks to practical problems. First of all, domain decomposition into meaningful subproblems is usually difficult as the problems tackled by neural networks are usually poorly understood. Moreover, adding modularity to a neural network tends to add a number of new hyperparameters that need optimizing, such as the number of modules, the size of each module, and the pattern of connectivity between modules. Another problem that arises with multiple modules is how to integrate the output of different modules and how to resolve any decision conflicts that might arise. We address these different problems throughout this study. We discuss the issues surrounding domain decomposition in Section 2.4.1, where we show that problem decomposition can be done implicitly or explicitly, and indicate how the process can be automated. Hyperparameter selection and associated issues are discussed in Section 2.4.3, where the different techniques for MNNs formation are presented. Integrating different modules to solve the task at hand is further investigated in Section 2.4.4.

While the study of modularity has focused mainly on topology, which is

indeed the main property of modular structure, we expand our study of modularization techniques to different levels of neural network design that can be exploited to produce modular networks. In the following section, we discuss the different levels of modular neural networks, and how different chains of techniques applied to such levels can produce different MNN variants.

2.4 Modularization Techniques

The modularization of neural networks can be realized with different techniques, acting at different levels of abstraction. A modularization technique is a technique applied to one of these levels in order to introduce modularity to the network topology. We present a taxonomy of such techniques that are categorized based on the abstraction level they exploit to achieve modularity. We analyse each technique, explaining the main rationale behind it, presenting its advantages and disadvantages (Table 2.1) relative to other techniques and providing prominent use cases from the literature. The main levels at which the modularization techniques act are complementary. Consequently, to produce a modular neural network, a chain of techniques, or chain of modularization, is used. A modularization chain (Figure 2.2) consists of a set of techniques (each one corresponding to a different level of the neural network environment) used to produce a modular neural network. So, every modularization chain corresponds to a particular type of MNN.

A modularization chain starts with partitioning the domain, however this is optional as was mentioned earlier. Then, a modular topological structure is selected for the model. After that, formation and integration techniques are selected to build the model and integrate the different modules, respectively. So, for example, if we need to develop an MNN for enhanced MNIST classification, then a modularization chain would look like the following:

1. Domain: we may choose to augment the MNIST dataset by applying a certain image processing function to a copy of each image to extract specific information, and then consider the original and processed images as different subdomains.

2. Topology: here we may select a multi-path topology, where one path of the network has the original image as input and the others have the processed ones.

3. Formation: we may use an EA to build the multi-path topology, constraining it to have exactly two paths.

4. Integration: here we may integrate the outputs of each path into the final system output, either through the evolutionary process itself, or as a post-formation learning (or fine-tuning) algorithm.

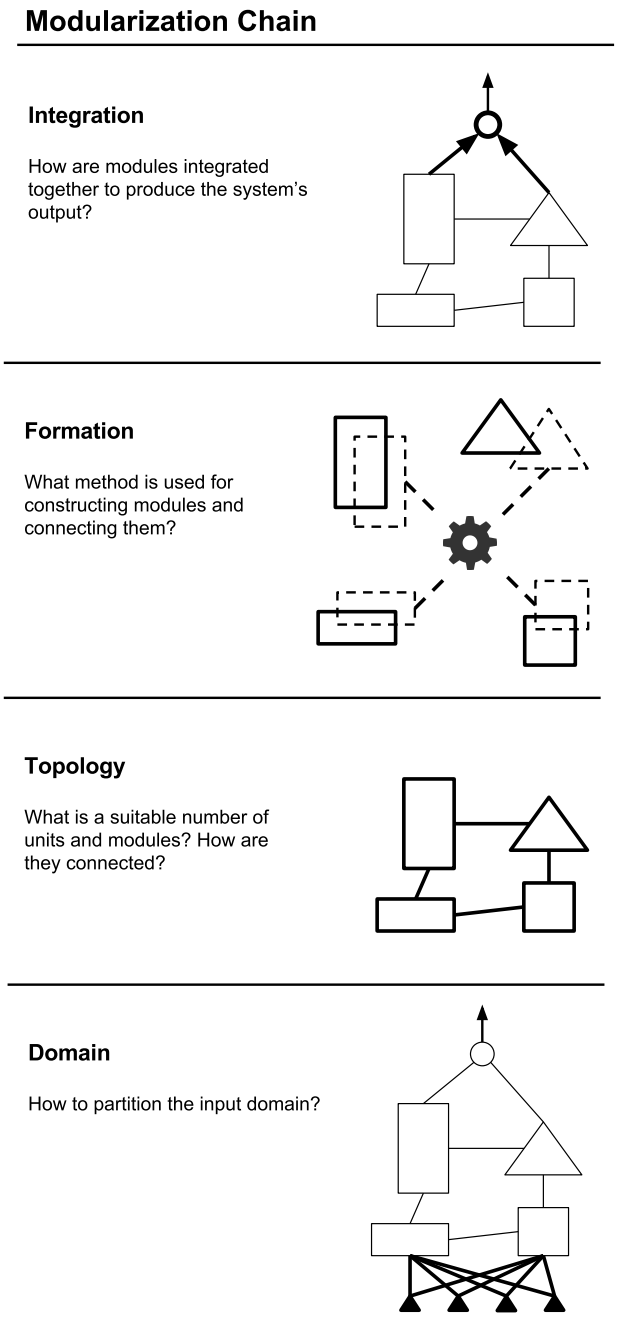


Figure 2.2. Modularization chain acting on the different levels of a neural network.

The underlying concept in the example above is to use MNNs to integrate different sources of information (i.e. source and processed images) to improve

classification performance. A similar concrete MNN application was investigated in Ciregan, Meier, and Schmidhuber (2012) and Wang (2015).

2.4.1 Domain

The domain refers to all of the information that is relevant to the problem and is accessible to the neural network learning model. In other words, it consists of the inputs and outputs that the system relies on for learning how to generalize to unseen inputs. Focusing on the input side, one of the rationales behind domain modularization, is that some functions can be defined piecewise, with each subfunction acting on a different domain subspace. So, instead of learning or applying the neural network model on all of the input space, domain modularization aims to partition this space into a set of subspaces. Then, the modules of an MNN, constructed by applying techniques at different modularization levels, can be readily applied to each subdomain. So, for example, we may choose to partition temporal data according to the time intervals in which they were collected, or partition spatial data according to the places in which they occurred like in Vlahogianni, Karlaftis, and Golias (2007). We refer to this kind of domain partitioning as subspatial domain partitioning, because the individual data items are clustered into multiple subspaces. Curriculum learning (Bengio et al., 2009; Bengio et al., 2015) is a particular form of this partitioning, where the neural network is successively trained on a sequence of subspaces with increasing complexity. Another kind is what we call feature or dimensional domain partitioning. In feature domain partitioning, partitioning occurs at the level of a data instance, such that different subsets of features or dimensions or transformations of these get assigned to different partitions. Examples of this approach include the application of different filters to the original images and processing each with different modules like in Ciregan, Meier, and Schmidhuber (2012), and the partitioning of an image to enhance object detection accuracy (Zhang et al., 2014).

The domain is, conceptually, the most natural and straightforward level of modularization. This is essentially because the domain defines the problem and its constraints, so, a good modularization of the domain corresponds directly to good problem decomposition. Decomposition of complex problems greatly simplifies reaching solutions, facilitates the design and makes it more parallelizable in both conception and implementation. In Nardi et al. (2006), the problem of replacing a manually designed helicopter control system by a neural network could not be tackled when a single MLP was trained to replace the whole system. However, it was feasible by replacing the system components gradually. Moreover, as it holds all the available information about the problem and its structure, it acts as a very good hook for integrating prior knowledge, through the implementation of modularity, that may be useful in facilitating problem solving. Prior knowledge at the domain level is mainly a basis for problem

decomposition, be it an analytical solution, some heuristic or even a learning algorithm. For example, in Babaei, Geranmayeh, and Seyyedsalehi (2010), the problem domain of predicting protein secondary structure was decomposed into two main groups of factors, namely: strong correlations between adjacent secondary structure elements and distant interactions between amino acids. Two different RNNs were used to model each group of factors before integrating both to produce the final prediction. It is also interesting to note that the domain is the only level of modularization that can be absent, at least explicitly, from a modular neural network. In other words, you can have a modular neural network that does not involve any explicit modularization at the domain level, but this is not possible for the other levels. This is mainly because domain decomposition is a kind of priming technique for modularity, in other words, it promotes modularity but is not a necessary condition. Note that domain decomposition can still happen implicitly without intentional intervention. As a simple example, it is well established in the ML literature that Radial Basis Function (RBF) networks, and also non-linearities in feedforward networks, are able to transform inputs that may be non-linearly separable into linearly separable ones (Haykin, 1994; Montufar et al., 2014).

2.4.1.1 Manual

Manual domain modularization is usually done by partitioning the data into either overlapping or disjoint subspaces, based on some heuristic, expert knowledge or analytical solution. Theses partitions are then translated into a full modular solution via different approaches throughout the modularization process.

The manual partitioning of input space allows for the integration of prior knowledge for problems that are easily decomposable based on some rationale. This knowledge-based integration can be done by defining partitions that correspond to simple subproblems that can be addressed separately. Moreover, it gives fine control over the partitioning process that can be exploited to enhance performance. This is contrary to automatic decomposition, which may be adaptive and efficient, but as the rationale is latent and not directly observed, it is hard to tweak manually for further enhancements. On the other hand, it raises the question of what defines a good partition, a partition which corresponds to a well defined subproblem, that has isolated constraints and can be solved separately. Although the domain is what characterises a problem's solution, usually the relation between decomposing the domain and obtaining a solution is not that straightforward. For example, you may think of decomposing some input image to facilitate face recognition. However, since the process of face recognition is not well understood, it is not clear what decomposition is suitable. Is it segmentation of face parts or maybe some filter transformation? (Chihaoui et al., 2016) Figuring this out analytically is not feasible. The data generating process is often very complex and contains many latent factors of

variations, which makes the separation and identification of those factors hard. A good partitioning requires a good prior understanding of the problem and its constraints, which is rarely the case for ML tasks, which generally rely on large datasets for the automatic extraction of the underlying causal factors of the data.

One of the simplest subspatial partitioning schemas that arises naturally in classification tasks is class partitioning. Class partitioning is the partitioning of the domain based on the target classes of the problem. This is a straightforward approach which is built on the assumption that different classes define good partitions. There are three main class partitioning schemes, namely One-Against-All (OAA), One-Against-One (OAO) and P-Against-Q (PAQ) (Ou and Murphey, 2007). In the OAA (Anand et al., 1995; Oh and Suen, 2002) scheme, the domain of K classes is partitioned into K subproblems, where each subproblem is concerned with how to differentiate a particular class A from its complement, that is, all of the remaining classes which are not A. OAO (Rudasi and Zahorian, 1991) partitions the domain into $\binom{K}{2}$ subproblems, with each subproblem concerned with differentiating one class from only one other class. A compromise between the two previous schemes is PAQ (Subirats et al., 2010), where each subproblem aims to differentiate P number of classes from Q number of classes. OAO is the most divisive of the three, which makes it the most computationally expensive, assuming that each classifier's complexity is the same. Whatever the scheme used, the output of each module trained on a different subproblem can then be combined with an integration technique to embody a particular MNN. More generally, different modularization chains can be applied to the different subproblems, thus resulting in different MNNs.

Class partitioning reduces classification complexity and can be seen as a divide-and-conquer approach. If the partitioning results in several smaller datasets, and assuming that the partitioning accurately reflects the problem's underlying structure, then not only should the learning problem be easier, but the overall representation learned by the MNN should be more faithful to the underlying causes of the data. In Bhende, Mishra, and Panigrahi (2008), classification of power quality into 11 classes was done by class partitioning using the OAA technique. The MNNs were applied after feature extraction using the S-transform, and the different modules were integrated using a max activation function to produce the final output.

In Vlahogianni, Karlaftis, and Golias (2007), a subspatial domain partitioning, that is not class based, is used to train different modules on forecasting traffic volume at different road locations. In Aminian and Aminian (2007), an electronic circuit is decomposed into multiple subcircuits to facilitate fault detection by several neural network modules. Feature domain partitioning is another form of manual partitioning, and is seen in Mendoza, Melin, and Licea (2009a) and Mendoza, Melín, and Castillo (2009b) where edge detection using a fuzzy inference system is done on target images to obtain edge vectors.

Then different neural networks are trained on parts of these vectors, and the network outputs are combined using the Sugeno integral to produce the final classification results. Sometimes, the feature domain partitions are just different transformations of the data, with each transformation revealing different perspectives of the data. This is realised in Ciregan, Meier, and Schmidhuber (2012) and Wang (2015) where the input image together with its transformations via different image processing functions are used as inputs into a modular CNN, in order to enhance classification performance.

2.4.1.2 Learned

Learned decomposition is the partitioning of the domain using a learning algorithm. Problem domains are often not easily separable. This is closely related to the problem of representation. If the domain can be manually decomposed into an optimal set of subdomains, that is a set of subspaces that capture all of the constraints of the problem compactly, this will significantly facilitate learning. However, usually the data generating process of the domain involves many interacting factors that are not readily observed. Problems like these typically require learning algorithms to be applied both to the partitioning (explicitly or implicitly) and the overall classification problem.

Learned decomposition facilitates the capturing of useful clustering patterns, especially complex ones that are not tractable by human designers. This intractability may stem from different sources like mathematical complexity or poorly understood problems. For example, the prediction of protein secondary and tertiary structure is often a complex and poorly understood process, that may take tremendous amounts of computational resources to simulate (Freddolino et al., 2008; Allen et al., 2001). However, learning algorithms often add computational cost to the overall process, since they typically involve adding an extra step for optimizing the model responsible for the decomposition.

Yuan and Lin (2006) propose modifications to three factor selection methods, namely lasso, LARS and non-negative garrotte, to allow for selecting groups of factors. Effectively, the proposed generalization aims at reaching a good decomposition of the underlying factor space. Because of the clustering nature of learned decomposition, the mainstream approach involves applying unsupervised learning. In Ronen, Shabtai, and Guterman (2002), fuzzy clustering is used to dynamically partition the domain into regions, then a different MLP is trained on each region, and finally, the MLP outputs are integrated using a Sugeno integral like method. Also, in Fu et al. (2001), a technique called Divide-and-Conquer Learning (DCL) is used to partition the domain whenever learning stalls. DCL acts by dividing training regions into easy and hard sets using Error Correlation Partitioning (ECP) (Chiang and Fu, 1994), which is based on optimizing a projection vector that separates data points according to their training error, then different modules are trained on each region and finally integrated using a gated network.

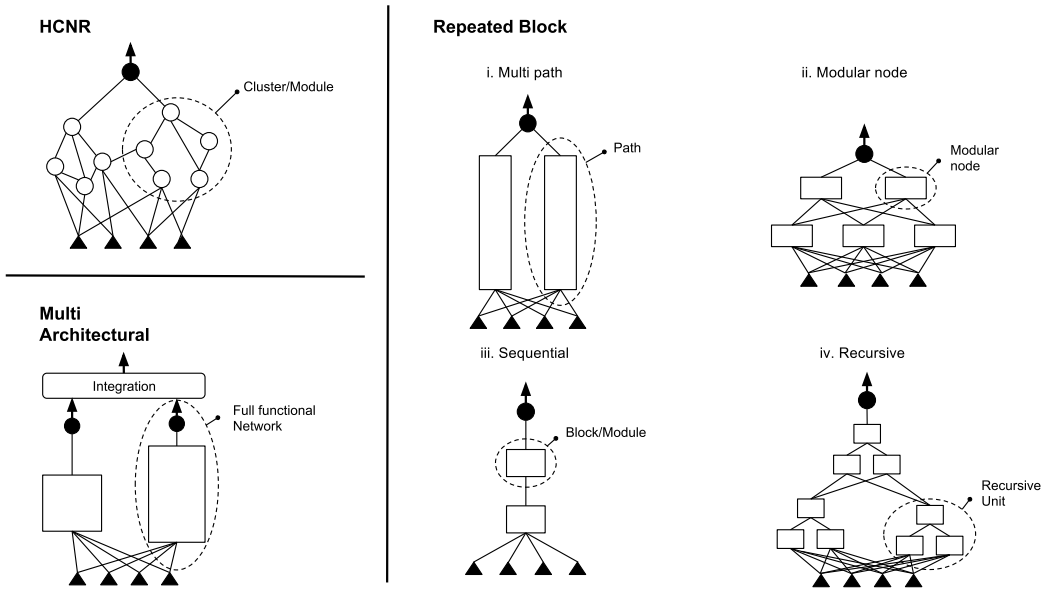


Figure 2.3. Different modular neural network topologies.

2.4.2 Topology

The topology of a network refers to how different nodes and modules within the network connect with each other, in order to produce the overall structure of the model. A neural network with a modular topology (Figure 2.3) exhibits a structure whereby nodes within a module are densely connected to each other, with sparse connectivity between modules. This is topological modularity, whereas functional modularity emerges when each topological module can be assigned a sub function of the whole task addressed by the neural network model. Topological modularity is a necessary, but not sufficient, condition of functional modularity. Without a learning algorithm that promotes functional modularity, topological modularity is not guaranteed to give rise to functional specialisation.

Neuroscience research sheds light on the modular topology of the nervous system. Neural circuitry in the brain is organized into modules at different levels of granularity, from cortical columns and neuronal nuclei to the whole anatomical areas of the brain's macro structure. It has been suggested in different works that the modularity of the brain arises from selection for evolvability and minimization of connection cost (Bullmore and Sporns, 2009; Clune, Mouret, and Lipson, 2013).

Although the early inspiration for neural networks was the brain, artificial neural network research has mostly deviated from biological research. However, there are still occasional insights taken from biology (e.g. deep networks and convolutional structures, to name a few impactful examples) with the usual caveat that the aim is not to closely mimic the brain, but to solve real world

problems in effective ways regardless of the source of the core ideas (Goodfellow et al., 2016). So, although topological modularity is inspired by the brain's modular structure, it has metamorphosed into different forms that better suit the ANN domain.

The formation and learning of monolithic neural networks are hard problems. This is especially true with very deep neural networks. Deep learning faces several problems like overfitting, vanishing gradients and spatial crosstalk. Good topological modularization acts as a kind of regularization relative to highly connected monolithic networks. For example, Watanabe, Hiramatsu, and Kashino (2018) provided a method for detecting topological modular structure in trained monolithic neural networks and empirically showed an inverse correlation between the modularity of the detected structure and the network generalization error. Some forms of modular topologies (Larsson, Maire, and Shakhnarovich, 2016; He et al., 2016; Srivastava, Greff, and Schmidhuber, 2015) provide shortcut paths for gradient flow which help to alleviate vanishing gradients. Moreover, the sparse connectivity of modules reduces spatial crosstalk.

One of the main problems with monolithic networks arises when something wrong occurs. In the vast majority of cases, neural networks are considered black box models. As such, it is usually unrealistically hard to decipher how a neural network makes its predictions. This stems mainly from a neural network's distributed representations, where nodes are tightly coupled, making separation of functions infeasible even in theory. This makes debugging and fixing deviations in behaviour very difficult. Topological modularity, especially if accompanied by functional modularity, can be exploited to localize functional errors so that more investigations may reveal possible solutions. In the case of full functional modularity, there are still distributed representations associated with different modules, however, since modules themselves are loosely coupled, this separation of concerns makes localizing deviation in some sense realistic.

2.4.2.1 HCNR

HCNR topology is a modular topology with non-regular and dense within-module connections, and sparse connectivity between different modules. Non-regularity here roughly means that the overall topology can not be described by a template with repeating structures. This makes the topology generally hard to compress. Elements from graph theory can be used to formalize this notion using measures like characteristic path length and clustering coefficient (Watts, 1999). Aside from high clustering, HCNR does not have to exhibit properties such as the short average path length of Small World (SW) networks. Thus, the broader category of HCNR includes small world topologies as special cases.

Biology has shown significant interest in small world networks as increasing evidence suggests that many biological systems including genetic pathways, cellular signalling and brain wiring, exhibit small world topology. The evolutionary origins of the brain's modular structure is still controversial, but some

hypotheses have been suggested. In Kashtan and Alon (2005), it was suggested that evolution under a MVG environment yields modular structure. An MVG environment changes modularly, in the sense that the environmental goal varies through time, but each goal is comprised of the same common set of subgoals. A biological example is chemotaxis towards nutrients. The process of chemotaxis involves the same set of intermediate goals, like sensing, computing motion direction and moving, that are independent of the target nutrient. In another hypothesis (Clune, Mouret, and Lipson, 2013) it was suggested that modularity is bootstrapped by a pressure for minimizing costs pertaining to neuronal connections, and is then maintained by selection for evolvability. Random networks tend to have high connection cost due to dense connectivity, which is not the case in the brain's neural circuitry, which exhibits economical functional networks (Achard and Bullmore, 2007). Having a modular structure promotes evolvability, as accumulative evolutionary changes tend to be local in effect without disrupting other functions.

In the context of artificial neural networks, the sparse connectivity of HCNR and average short path of its special case SW, reduce computational complexity compared to monolithic networks, whilst maintaining information transfer efficiency. However, due to their structural complexity, analysing and adapting these types of networks to real world problems is hard and raises several technical difficulties. How many nodes should be in each module? Should module node counts vary? How much connectivity is allowed within a module? And what connection sparsity between modules is sparse enough? Also, formation of this network type is done either by modifying a regular lattice (Bohland and Minai, 2001), which is hard to adapt to all ML tasks, or via EAs (Huizinga, Mouret, and Clune, 2014; Verbancsics and Stanley, 2011; Garcia-Pedrajas, Hervas-Martinez, and Munoz-Perez, 2003; Mouret and Doncieux, 2009; Mouret and Doncieux, 2008), which are lengthy and computationally expensive. The adoption of these two approaches, especially EAs, is a direct consequence of the previously mentioned difficulties and lack of good general engineering practices for HCNR.

The work done in Bohland and Minai (2001) shows that an SW topology can approach the performance of random networks for associative memory tasks, with less connectivity than random networks, implying that associative memories can benefit from modularity. This network was constructed by rewiring a regular lattice randomly. The work by Bohland and Minai (2001) shows that performance is not only about the quantitative nature of connectivity (i.e. number of connections), but also about its qualitative nature (i.e. how these connections are placed).

Evolutionary approaches for HCNR are either based on direct connection cost regularization, or the coevolution of modules. Both approaches tend to be biologically inspired, with connection cost regularization corresponding to the pressure of minimizing brain wiring, and coevolution inspired by species coevolution, where in this case each module is considered a different species. The

evolutionary approach in Huizinga, Mouret, and Clune (2014) and Verbancsics and Stanley (2011) made use of the connection regularization studied in biological neural networks (Clune, Mouret, and Lipson, 2013; Bullmore and Sporns, 2009) to promote HCNR modularity in the resulting model, which led to better performance on modular regular problems such as the retina problem. Another evolutionary approach relies on the cooperative coevolution model where modules are dependently co-evolved and their fitness is evaluated based on each module's performance and how well each module cooperates with other modules. COVNET (Garcia-Pedrajas, Hervas-Martinez, and Munoz-Perez, 2003), for example, achieved better results in some classification problems, like the Prima Indian and Cleveland Clinic Foundation Heart Disease datasets. COVNET also showed robustness to damage in some network parts.

Currently, there is an increasing interest in learning and manipulating structured representations using GNNs (Battaglia et al., 2018). For example, Santoro et al. (2017) introduced a module to enhance solving relational reasoning problems. Message-Passing Neural Networks (MPNNs) (Gilmer et al., 2017) are a type of GNN that were applied to the prediction of properties of molecular structures. Another area of application was recovering textual representations from Abstract Meaning Representations (AMR) (Song et al., 2018). One application which is very relevant to architectures and topologies is Graph HyperNetworks (Zhang, Ren, and Urtasun, 2018). This is a GNN for predicting a good set of weights for a neural network by analyzing its graph. We believe that GNNs can facilitate the analysis, manipulation and building of complex graphs, like HCNRs, and help understand their properties.

2.4.2.2 Repeated Block

This topology of modular neural networks is essentially a structure of repeated units or building blocks connected in a certain configuration. The building blocks do not have to be exact clones of each other, but they are assumed to share a general blueprint. The idea of global wiring schema in neural networks has its roots in biological studies and it is the underlying principle of the famous neuroevolution algorithm, HyperNEAT (Stanley, D'Ambrosio, and Gauci, 2009). In Angelucci et al. (1997) it was shown that retinal projections in ferrets, normally relayed to the Lateral Geniculate Nucleus (LGN), when rewired to the Medial Geniculate Nucleus (MGN), normally a thalamic relay in the auditory pathway, led the MGN to develop eye-specific regions. Also, mammalian cortex is considered to be composed of repeating columnar structure (Lodato and Arlotta, 2015). These and other lines of evidence support the notion of a global mechanism of wiring and learning in the brain.

In the artificial realm, repeated block structure allows for easier analysis and extensibility of neural networks. On the theoretical level and due to the high regularity of these topologies, a very large structure can be described by a few equations. For example, a recursive structure like FractalNet (Larsson,

Maire, and Shakhnarovich, 2016), can be described by a simple expansion rule. Also, due to regularity, scaling the capacity of these topologies tends to be very natural. We provide a taxonomy of repeated block topologies based on how the repeated units are wired together.

2.4.2.2.1 Multi-Path

Multipath topology refers to neural networks with multiple semi-independent subnetworks connecting network inputs to outputs. In Kien Tuong Phan, Maul, and Tuong Thuy Vu (2015), Phan et al. (2016), and Phan et al. (2017) this topology is named PCs, which are inspired by the microcircuits in the retina (Gollisch and Meister, 2010). The retina is believed to be of significant computational importance to the visual pathway, not just a simple informational relay. In other words, it has been shown that the retina does perform complex computational tasks, such as motion analysis and contrast modulation, and delivers the results explicitly to downstream areas. Moreover, these microcircuits have been shown to exhibit some sort of multipath parallelism, embodied by semi-independent pathways involving different combinations of photoreceptor, horizontal, bipolar, amacrine and retinal ganglion cells.

The separation of multiple paths allows for overall network parallelization, contrary to network expansion in terms of depth, where deeper layers depend on shallower ones, which makes parallelization problematic. Also, as in Ortín et al. (2005) and Wang (2015), each path can be assigned to a different input modality which allows for modal integration. This resembles brain organization where different cortical areas process different modalities, and then different modalities get integrated by association areas. However, the introduction of multiple paths adds uncertainty in the form of new hyperparameters (e.g. numbers and widths of paths), which if to be determined empirically, often requires a phase of pre-optimization. Moreover, aside from obvious links to ensemble theory, as of yet there is no detailed theoretical justification for multiple paths. Why do empirical experiments show improved generalization performance (Phan et al., 2016) of multipath over monolithic topologies? Does width (in terms of number of paths) promote problem decomposition just like depth promotes concept composition? To date there are no mature gradient-based learning algorithms to fully exploit the parallel circuit architecture, and which are likely to explicitly promote automatic task decomposition across paths.

In Kien Tuong Phan, Maul, and Tuong Thuy Vu (2015), Phan et al. (2016), and Phan et al. (2017), a multipath approach with shared inputs and outputs is shown to often exhibit better generalization than monolithic neural networks. Crucial to this improvement, was the development of a special dropout approach called DropCircuit, where whole circuits are probabilistically dropped during training. In another approach (Guan and Li, 2002), called output parallelism, the inputs are shared between paths, while each path has a separate output layer. This technique can be applied when the output is easily decomposable.

A very related approach can be found in Goltsev and Gritsenko (2015), where a central common layer is connected to multiple paths, each used for a different output class. On the other hand, the work in Wang (2015) enhances CNNs by allowing for two paths, one with the source image as input, and the other with a bilateral filtered version of it. In bilateral filtering, each pixel value is replaced by the average of its neighbours, taking into account the similarity between pixels, so that high frequency components can be suppressed while edges are preserved. The integration of images preprocessed in different ways facilitates the capturing of more useful features. This is motivated by the observation that convolution and pooling operations extract high frequency components, which causes simple shapes and less textured objects to gradually disappear. Bilateral filtering of one of the input images tends to suppress high frequency components, which allows the network to capture both simple and complex objects. The multi-path concept can be integrated with other topologies, as in Xie et al. (2016) where ResNetXt enhances ResNet's sequential topology by introducing modules that have multi-path structure.

2.4.2.2.2 Modular Node

A modular node topology can be viewed as a normal monolithic feedforward neural network, where each node is replaced by a module consisting of multiple neurons. This expansion is computationally justified by replacing a single activation function depending on one weight vector, by a collection of functions or a function depending on multiple weight vectors. This has the effect of increasing the computational capability of the network, while maintaining a relatively small number of model parameters. Moreover, the regularity and sparsity of such a structure, combined sometimes with restricting weights to integer values, can be suitable for hardware realizations (Sang-Woo Moon and Seong-Gon Kong, 2001). On the other hand, this requires additional engineering decisions, like choosing the number of module neurons, how they are interconnected and what activation functions to use.

A special case of this topology is hierarchical modular topology, which consists of modules at different topological scales, where a higher level module is composed of submodules, each of which is composed of submodules, and so on. Hierarchical modularity is known to exist in brain networks (Wang, Hilgetag, and Zhou, 2011; Kaiser and Hilgetag, 2010), other biological networks and Very Large-Scale Integration (VLSI) electronic chips (Meunier, Lambiotte, and Bullmore, 2010). It has been argued that this form of modularity allows for embedding a complex topology in a low dimensional physical space.

In Sang-Woo Moon and Seong-Gon Kong (2001), Wei Jiang and Seong Kong (2007), and Phyo Phyo San, Sai Ho Ling, and Nguyen (2011) modular node topology is realized by replacing the nodes of a feedforward network by a two dimensional mesh of modules, with modular units each consisting of four neurons. The four neurons can be connected in four different configurations. This

network, called Block-Based Neural Network (BBNN), was shown to be applicable to multiple tasks including pattern classification and robotic control, even when its weights were restricted to integer values. Another modular network called Modular Cellular Neural Network (MCNN) (Karami, Safabakhsh, and Rahmati, 2013) exhibits similar array like arrangement, where nine modules are arranged in a grid. A module in MCNN is composed of another grid of dynamic cells, where each cell is described by a differential equation. MCNNs were applied to texture segmentation successfully and benchmarked to other algorithms on the problem of edge detection. Another realization of this topology can be found in the Local Winner-Take-All (LWTA) network (Srivastava et al., 2013), where each node of a feedforward neural network is replaced by a block, each consisting of multiple non interconnected neurons. The network operates by allowing only the neuron with the highest activation in a block to fire, while suppressing other neurons. The block output is

$$y_i = g(h_i^1, h_i^2, ..., h_i^n) \tag{2.2}$$

where $g(.)$ is the local interaction function and h_i^j is the activation of the jth neuron in block i. This is mainly inspired by the study of local competition in biological neural circuits. Network In a Network (NIN) models (Lin, Chen, and Yan, 2013) are the modular node equivalents of CNNs, where each feature map is replaced by a micro MLP network, to allow for high-capacity non-linear feature mapping. The output of a single micro MLP is

$$\boldsymbol{f}_{i,j}^l = max(0, \mathbf{W}_l \boldsymbol{f}_{i,j}^{l-1} + \boldsymbol{b}_l) \tag{2.3}$$

where (i, j) are indices of the central pixel location, l is the index of the MLP layer and $\mathbf{W}$ and $\boldsymbol{b}$ are the weights and bias, respectively. It is also interesting that the LSTM architecture (Hochreiter and Urgen Schmidhuber, 1997), the famous RNN, has a modular node structure, in which each node is an LSTM block. LSTM has shown state-of-the-art results in sequence modeling and different real-world problems (Stollenga et al., 2015; Eyben et al., 2013; Soutner and Müller, 2013). Moreover, Hierarchical Recurrent Neural Networks (HRNNs), which are typically realised using LSTM or its simplification, the GRU, implements hierarchical modular topology, where the first hidden layer is applied to input sequentially and the layer output is generated every n number of inputs, which is then propagated as input to the next layer and so on. Hence, the main difference between HRNNs and classic RNNs is that, for the former, hidden layer outputs are generated at evenly spaced time intervals larger than one. HRNNs have been used for captioning videos with a single sentence (Pan et al., 2016) and with a multi-sentence paragraph (Yu et al., 2016), and for building end-to-end dialogue systems (Serban et al., 2016).

CapsNet was introduced in Sabour, Frosst, and Hinton (2017), which is mainly a vision-centric neural network that attempts to overcome the limitations

of CNNs. The main rationale behind CapsNet is representing objects using a vector of instantiation parameters that ensures equivariance with different object poses. CapsNet can be thought of as an ordinary CNN, in which each node is replaced by a vector-output module. The output of such a modular node $\boldsymbol{v}_j$ in CapsNet is calculated as

$$\boldsymbol{v}_j = \frac{||\boldsymbol{s}_j||^2}{1+||\boldsymbol{s}_j||^2}\frac{\boldsymbol{s}_j}{||\boldsymbol{s}_j||} \tag{2.4}$$

where $\boldsymbol{s}_j$ is the node input and is calculated as

$$\boldsymbol{s}_j = \sum_i c_{ij}\hat{\boldsymbol{u}}_{j|i},\ \hat{\boldsymbol{u}}_{j|i} = \mathbf{W}_{ij}\boldsymbol{u}_i \tag{2.5}$$

where $\boldsymbol{u}_i$ is the output of a unit i from the previous layer, $\mathbf{W}_{ij}$ is a transformation matrix and c_{ij} is a coupling coefficient. Coupling coefficients are calculated through a routing softmax as

$$c_{ij} = \frac{e^{b_{ij}}}{\sum_k e^{b_{ik}}} \tag{2.6}$$

where b_{ij} is a log prior probability which is initialized to zero and updated following the rule

$$b_{ij} \leftarrow b_{ij} + \hat{\boldsymbol{u}}_{j|i}.\boldsymbol{v}_j \tag{2.7}$$

This is called routing-by-agreement and acts to increase contributions from lower layer capsules that make good predictions regarding the state of a higher level capsule.

2.4.2.2.3 Sequential

Sequential topology consists of several similar units connected in series. The idea of composition of units has its roots in deep learning. Deep networks arise when multiple layers are connected in series. This allows for deep composition of concepts, where higher level representations are composed from lower level ones. The difference here is that the composed units consist of whole modules. But with added depth, convergence and generalization can become increasingly difficult, and one must therefore resort to tricks like dropout and batch normalization to make learning feasible. Moreover, there has been recent criticism of very deep networks based on the question of whether this extreme depth is really necessary (Ba and Caruana, 2014; Veit, Wilber, and Belongie, 2016), specially given that the brain can do more elaborate tasks with far fewer layers.

Inception networks (Szegedy et al., 2015a; Szegedy et al., 2016) and Xception networks (Chollet, 2016) (built from an extreme version of an inception module), are essentially a sequential composition of multi-path convolutional

modules. Highway networks were introduced in Srivastava, Greff, and Schmidhuber (2015) and can be seen as a sequentially connected block of modules, where each module output consists of the superposition of the layer output and layer input, weighted by two learning functions called the transform gate and carry gate respectively. The output of a single layer in a highway network can be modelled as

$$\boldsymbol{y} = H(\boldsymbol{x}, \mathbf{W}_H).T(\boldsymbol{x}, \mathbf{W}_T) + \boldsymbol{x}.C(\boldsymbol{x}, \mathbf{W}_C) \tag{2.8}$$

where H is the layer activation function, T is the transform gate and C is the carry gate. These two gates learn to adaptively mix the input and output of each module, acting as a regulator of information flow. A similar idea can be found in He et al. (2016) where a special case of highway networks called residual networks consists of the same structural unit but with both gates set to the identity function. This makes the residual layer output

$$\boldsymbol{y} = H(\boldsymbol{x}, \mathbf{W}_H) + \boldsymbol{x} \tag{2.9}$$

This is motivated by simplifying the learning problem and enforcing residual function learning. Interestingly, the LSTM network, whose modular node topology was discussed above, exhibits a temporal form of sequential topology, where each LSTM block feeds its output to itself through time. So, expanding the LSTM block in time results in a temporal sequential topology, where the output of the LSTM block from the previous time step is considered as an input to the LSTM block in the current time step.

2.4.2.2.4 Recursive

Networks with recursive topology exhibit nested levels of units, where each unit is defined by earlier units in the recursion. Usually, all of the units are defined by the same template of components wiring. Recursion has a long history and is considered to be a key concept in computer science. Although recursive problems can be solved without explicit recursion, the recursive solution is a more natural one. In theory an infinite structure can be defined in one analytical equation or using a simple expansion rule. Due to their recursive structure, networks with recursive topology are readily adaptable to recursive problems (Franco and Cannas, 2001). Recursion also allows for very deep nesting while still permitting short paths, sometimes called information highways (Larsson, Maire, and Shakhnarovich, 2016) that facilitate gradient back-propagation and learning. However, as mentioned earlier, excessive depth is criticised by some researchers and its necessity is becoming increasingly debatable.

FractalNet introduced in Larsson, Maire, and Shakhnarovich (2016) exploits recursive topology to allow for very deeply nested structure that is relatively easy to learn despite its significant depth. It is inspired by the mathematically

beautiful self-similar fractal, where going shallower or deeper in structure yields the same topology schema. It is defined as

$$f_{C+1}(\boldsymbol{x}) = [(f_C \circ f_C)(\boldsymbol{x})] \oplus [conv(\boldsymbol{x})] \tag{2.10}$$

where C is the fractal index, $\circ$ means composition, *conv* is a convolutional layer and $\oplus$ is a join operation which was chosen to be an elementwise average of its inputs. It is supposed that the effectively shorter paths for gradient propagation facilitate learning and protect against vanishing gradients. In Franco and Cannas (2001) the parity problem was decomposed recursively and a recursive modular structure was adapted for its solution. Also in this work on the parity problem, it was shown that generalization was systematically improved by degree of modularity, however, it was not obvious if that was a general conclusion applying to all problems.

2.4.2.3 Multi-Architectural

A multi-architectural topology consists of a combination of full network architectures, integrated together via a high-level and usually simple algorithm. Frequently, it is characterized by each component network having its separate output. The different architectures used may be similar (i.e. homogeneous) or different (i.e. heterogeneous). Architectural differences include, but are not limited to, differences in wiring scheme and activation functions. As different network architectures have different strengths and weaknesses (and make different errors), the integration is usually trying to exploit this diversity in order to achieve a more robust collective performance. Even when networks are similar, diversity can still be achieved since random initialization and stochastic learning makes each network converge differently. However, this usually entails training multiple architectures, which is time consuming and computationally expensive.

One of the early attempts for combining multiple architectures was the mixture of experts systems (Jacobs et al., 1991; Azam, 2000), where a gating network chooses which one of multiple networks should respond to a given input. In another approach (Ciregan, Meier, and Schmidhuber, 2012), a homogeneous model is used where similar CNNs are trained on different types of pre-processing of the same image and their outputs are integrated by averaging. In Yu et al. (2018b), a referring expression is decomposed into three components, subject, location and relationship, where each component is processed using a separate visual attention module, which is essentially a CNN, and then the outputs of the different modules are combined. In Babaei, Geranmayeh, and Seyyedsalehi (2010), a heterogeneous model consisting of two different RNNs, each modeling different protein structural information, is applied to predicting protein secondary structure. In Zhang et al. (2016a), a modular deep Q network is proposed to facilitate transferring of a learned robotic control network from

simulation to real environment. By modularising the network into three components, namely perception, bottleneck and control modules, the perception training can be done independently from the control task, while maintaining consistency through the bottleneck module acting as an interface between the two other modules. In Shetty and Laaksonen (2015), Yu et al. (2016), and Pan et al. (2016) heterogeneous models of CNNs and RNNs are used for video captioning, where one or more CNNs are used for extracting features, which are used as inputs to RNNs for generating video captions. GANs and their variants (Goodfellow et al., 2014; Kim et al., 2017) are also multi-architectural in nature. Another interesting example is the memory network (Weston, Chopra, and Bordes, 2014), where multiple networks are composed end-to-end around a memory module to allow for the utilisation of past experiences. Essentially, a memory network is composed of a memory and four components, namely, I, G, O and R. I is the input network that translates the raw input into an internal representation $I(\boldsymbol{x})$. G stands for generalization and it is responsible for updating memory based on the new input

$$\boldsymbol{m}_i = G(\boldsymbol{m}_i, I(\boldsymbol{x}), \boldsymbol{m}) \tag{2.11}$$

where i is the index of the memory cell. After the memory is updated, another module, O, computes the output features $\boldsymbol{o}$ based on the new input and the memory

$$\boldsymbol{o} = O(I(\boldsymbol{x}), \boldsymbol{m}) \tag{2.12}$$

and finally, the R module converts the output features into the desired format

$$\boldsymbol{r} = R(\boldsymbol{o}) \tag{2.13}$$

2.4.3 Formation

Formation refers to the technique used to construct the topology of the neural network. Manual formation involves expert design and trial and error. In manual formation, the human designer is guided by analytical knowledge, several heuristics and even crude intuition. Because of the difficulty and unreliability of manual formation, and a general lack of understanding of the relation between problems and the models they require, automatic techniques have been devised. Arguably the most popular automatic techniques are EAs, where the structure of the network is evolved over multiple generations, based on a fitness function that evaluates which individuals are more adapted. Another set of automatic formation algorithms constitute the learned formation category, where a learning algorithm is used not only for parameter (e.g. connection weight) optimization, but also for structure selection. Learned formation can be categorized into constructive and destructive algorithms (Garcia-Pedrajas, Hervas-Martinez, and Munoz-Perez, 2003). In constructive learned formation,

the algorithm starts with a small model, learns until the performance stalls, adds components to expand capacity and iterates again. Destructive learned formation algorithms start with a big model that overfits, then iteratively remove nodes until the model generalizes well.

In order to form a modular neural network, one of these construction approaches needs to be modified in order to take modularization into account. With manual formation, it is in principle straightforward to modularize, where instead of designing a monolithic network, different modules are designed and combined to build an MNN. On the other hand, while standard EAs can produce modular structure, they are usually modified using techniques like cooperative coevolution, given that the latter are generally seen to be more effective for evolving modular structure. In the case of learned formation, learning algorithms usually take modularity explicitly into account. So, the ML task becomes that of learning both modular structure and the parameters (e.g. weights) of that structure. A variant of learned formation, which we call implicit learned formation, is a learning algorithm that is implicitly sampling or averaging from a set of modules, so that the overall effective structure of the network can be seen as a modular one.

2.4.3.1 Manual

In manual formation, modular networks are built by manual design and composition of different modules. This type of formation provides useful opportunities for integrating good engineering principles and prior knowledge of the target problem into the modular neural network. For example, in Babaei, Geranmayeh, and Seyyedsalehi (2010), the system for predicting protein secondary structure is formed from two RNN modules that model two different aspects of the process, namely, short and long range interactions. Fine control over what to include or exclude from the model can lead to a robust combination of well performing components. However, regardless of how this sounds theoretically plausible, limited understanding of the underlying structures of most real-world problems and limited, to date, research into good neural modularization practices, make this hard in practice.

In Nardi et al. (2006), different modules are manually composed together to implement a helicopter control system, based on the practices of human designed Proportional–Integral–Derivative (PID) controllers. The PID components are replaced progressively by their neural network counterparts, until the whole control is done by the MNN. More formally in Guang-Bin Huang (2003), analytical introduction of modular layers into feedforward neural networks allows for reducing the number of nodes required to learn a target task. This is a case that shows how good engineering could be integrated, through formal analysis, into the formation of modular neural networks.

2.4.3.2 Evolutionary

EAs represent the current state of the art in formation methods for modular neural networks. This is clearly biologically inspired by the neuroevolution of the brain, which is shown to be highly modular both in topology and functionality (Aguirre et al., 2002; Bullmore and Sporns, 2009). Aside from biological inspiration, evolving modular structure has gained momentum as an effective approach to modularity formation, partly because of a lack of fundamental learning principles supporting artificial neural modularity. Adapting evolution to the problem of modularity formation, through connection cost regularization (Huizinga, Mouret, and Clune, 2014) or cooperative coevolution (Garcia-Pedrajas, Hervas-Martinez, and Munoz-Perez, 2003), partly delegates the problem of choosing modularity-related hyperparameters, such as the number and structure of modules, or connection schema, to a fitness function. Furthermore, EAs are the only fitness-based approach to producing HCNR topology, whereas other methods rely on random modifications to regular networks. On the down side, as already mentioned, EAs tend to be computationally expensive.

In Garcia-Pedrajas, Hervas-Martinez, and Munoz-Perez (2003), COVNET was introduced, which is a modular network formed using a cooperative coevolutionary algorithm. Every module is called a nodule, and is defined as a set of neurons that are allowed to connect to each other and to input/output nodes, but are not allowed to connect to neurons in other nodules. Every nodule is selected from a genetically separated population and different nodules are combined together to form individuals of the network population. To achieve cooperative coevolution, it is not sufficient to assign fitness values to networks, but it is also necessary to assign fitness values to nodules. The fitness of a network is straightforward, where obviously it corresponds to how well the network performs on its target task. The fitness assignment of nodules must enforce: (1) competition, so that different subpopulations do not converge to the exact same function, (2) cooperation, so different subpopulations develop complementary features and (3) meaningful contribution of a nodule to network performance, such that poorly contributing nodules are penalized. In COVNET, a combination of different measures is used to satisfy these criteria. Substitution is used to promote competition, where the best k networks are selected and a nodule a is replaced by a nodule b from the same subpopulation; then the networks fitnesses are recalculated, and nodule a is assigned fitness proportional to the average difference between the network fitnesses with nodule a and the network fitnesses with the substitution nodule b. Difference is used to promote cooperation between nodules by promoting competition between nodule subpopulations, so that they do not develop the same behaviour. Difference is done by eliminating a nodule a from all the networks where it is present, then recalculating network fitnesses; then the nodule is assigned fitness proportional to the average difference between fitnesses of the networks with the nodule and

the networks without it. Finally, best k is used to assess the meaningful contribution of a nodule, where nodule fitness is proportional to the mean of the fitnesses of the best k networks. This has the effect of rewarding nodules in the best performing networks, whilst not penalizing a good nodule in a poor performing network.

Promoting modularity through connection cost minimization (Huizinga, Mouret, and Clune, 2014) is biologically inspired as evidence suggests that the evolution of the brain, guided by the minimization of wiring length, besides improving information transfer efficiency, produces modular structure (Clune, Mouret, and Lipson, 2013; Bullmore and Sporns, 2009). In Hüsken, Igel, and Toussaint (2002), modularity emerges through evolution by selection pressure for both fast and accurate learning. In Di Ferdinando, Calabretta, and Parisi (2001), a modular multi-path neural network was evolved for solving the what and where task of identifying and localizing objects using a neural network. In another type of approach, modules are used as substrates for EAs. For example, in Braylan et al. (2015) pre-learned networks (modules) were combined using EAs, in an attempt to implement knowledge transfer. In Calabretta et al. (2000), evolution was implemented using a technique called duplication-based modular architecture, where the architecture can grow in the number of modules by mutating a set of special duplicating genes. In Miikkulainen et al. (2017) a population of blueprints, each represented by a graph of module pointers, was evolved using CoDeepNEAT, alongside another population of modules, evolved using DeepNEAT, an algorithm based on NEAT (Stanley and Miikkulainen, 2002), to develop deep modular neural networks. CoDeepNEAT seems to be a generalization of a previous algorithm called ModularNEAT (Reisinger, Stanley, and Miikkulainen, 2004), where modules are evolved using classic NEAT and blueprints are shallow specifications of how to bind modules to the final network input and output.

In Fernando et al. (2017), an interesting approach to evolutionary formation is introduced, where only some pathways in a large neural network composed of different modules are trained at a given time. The aim is to achieve multitask learning. The pathways are selected through a GA that uses a binary tournament to choose two pathways through the network. These pathways are then trained for a number of epochs to evaluate their fitness. The winner genome overwrites the other one and gets mutated before repeating the tournament. At the end of the training for some task, the fittest pathway is fixed and the process is repeated for the next task.

2.4.3.3 Learned

Learned formation is the usage of learning algorithms to induce modular structure in neural networks. Learned formation attempts to integrate structural learning into the learning phase, such that the learning algorithm affects network topology as well as parameters. We identified two variants of learned

formation in the literature. Explicit learned formation uses ML algorithms to promote modularity, predict the structure of modular neural networks and specify how modules should be wired together. On the other hand, implicit learned formation corresponds to learning algorithms that implicitly sample from multiple modules during training, although during the prediction phase, the network is explicitly monolithic whilst effectively simulating a modular network. Learned formation, just like evolutionary formation, allows for dynamic formation of modules. Moreover, as mentioned above, it can effectively sample from a large set of models, which is why it is often referred to as effectively implementing ensemble averaging (Srivastava et al., 2014; Huang et al., 2016b; Singh, Hoiem, and Forsyth, 2016; Larsson, Maire, and Shakhnarovich, 2016). The main disadvantage for dynamic algorithms like these is added computational overhead. Also, for implicit learned formation, the network is still densely connected and therefore computationally expensive, and modules are generally sampled randomly without any preference for better modules.

In Andreas et al. (2016a), Andreas et al. (2016b), and Hu et al. (2016), which exemplifies recent work on explicit learned formation, the problem of relating natural language to images was addressed. A set of modular blocks, each specialised in a certain function (e.g. attention and classification), were used as building units of a modular neural network, where different dynamic techniques were applied to assemble units together into an MNN that was capable of answering complex questions about images and comprehending referential expressions. For example, in Andreas et al. (2016b), two distributions were defined: (1) layout distributions, defined over possible layouts/structures of the network and, (2) execution model distributions, defined over selected model outputs. The overall training was done end-to-end with reinforcement learning. Another approach (Ferreira et al., 2018) proposes using False Nearest Neighbours (FNN), an adaptive learning algorithm usually used to determine the dimensionality of embedding, to help determine the kernel size and number of units for CNNs.

One of the most well known implicit learned formation techniques is dropout (Srivastava et al., 2014). Dropout acts by dropping random subsets of nodes during learning, as a form of regularization that prevents interdependency between nodes. Dropout is effectively sampling from a large space of available topologies during learning, because each learning iteration acts on a randomly carved sparse topology. In the prediction phase, networks are effectively averaging those random topologies to produce the output. Stochastic depth (Huang et al., 2016b) is another dropping technique used in training residual networks, which acts by dropping the contribution of whole layers. Swapout (Singh, Hoiem, and Forsyth, 2016) generalizes dropout and stochastic depth, such that it is effectively sampling from a larger topological space, where a combination of dropping single units and whole layers is possible. DropCircuit (Phan et al., 2016; Phan et al., 2017) is another related technique, which is an adaptation of

dropout to a particular type of multipath neural network called Parallel Circuits. In this technique, whole paths are randomly dropped, such that learning iterations are acting on random modular topologies. This effectively corresponds to sampling from a topological space of modular neural networks. Blundell et al. (2015) introduced Bayes by Backprop, a learning algorithm that approximates Bayesian inference which is, as applied to a neural network, the sum of the predictions made by different weight configurations, weighted by their posterior probabilities. This is essentially an ensemble of an infinite number of neural networks. As this form of expectation is intractable, it is approximated using variational learning, using different techniques such as Monte Carlo approximation and a scale mixture prior.

2.4.4 Integration

Integration is how different module outputs are combined to produce the final output of the MNN. Integration may be cooperative or competitive. In cooperative integration, all the MNN modules, contribute to the integrated output. On the other hand , competitive integration selects only one module to produce the final output. The perspective of integration is different from that of formation, where the latter is concerned with the processes that gives rise to modular structure, and the former is concerned with the structures and/or algorithms that use different modules in order to produce model outputs. Integration is a biologically inspired theme of brain structure, where hierarchical modular structures work together to solve a continually changing set of complex and interacting environmental goals.

2.4.4.1 Arithmetic-Logic

Arithmetic-Logic (AL) integration corresponds to a set of techniques that combine different modules through a well-defined algorithmic procedure, combining mathematical operators and logic. For problems that can be described using a sequence of algorithmic steps, this is the simplest and most straightforward approach, and is the most natural hook for integrating prior knowledge. It is worth mentioning that while the relation between steps needs to be algorithmically defined, the computation of the steps themselves is not necessarily well-defined. For example, a car control system may want to steer away from an obstacle once it has identified one. The relation between identification and steering away is AL-defined, while the identification of obstacles is not generally algorithmically defined. Moreover, AL integration allows for module decoupling, where each module has its well-defined interpretable output, which further makes debugging easy. However, in ML tasks, due to our limited understanding of problem domains and corresponding data generating processes, it is rarely the case that problems can easily be decomposed into AL steps.

In Gradojevic, Gençay, and Kukolj (2009), multiple neural networks were logically integrated, where each network was trained on only a part of the input space, and the output was integrated at the prediction phase by selecting the network that corresponded to the input subspace. This is a competitive integration type scheme. A more complex integration was done in Nardi et al. (2006), where neural network components for a helicopter control system were cooperatively integrated based on the AL of a hand designed PID controller. In Wang et al. (2012), two CNNs, one being a text detector and the other a character recognizer, were logically integrated, where the detector determined image locations containing text and the recognizer extracted text given these locations. In Eppel (2017), the recognition of the parts of an object was done in two steps, where in the first step, a CNN was used to segment the image to separate the object from its background, then in the second step another CNN was applied to the original image and the segmentation map to identify the object parts.

2.4.4.2 Learned

Learned integration consists of the composition of modules through a learning algorithm. Here, learning is concerned with how to optimally combine modules in order to obtain the best possible performance on the target problem. Composing modules to solve a certain problem is not straightforward, involving complex interactions between modules. Using learning algorithms in modular integration helps to capture useful complex relationships between modules. Even when subproblems are readily composable into a final solution, learning algorithms can find shortcuts that can help formulate more efficient solutions. However, the introduction of learning can result in unnecessary computational overhead, and can give rise to tightly coupled modules, often leading to overfitting and harder debugging. A very common type of learned integration is synaptic integration, where different modules are combined together by converging to a common parametric layer, which determines, through learning, the contribution of each module to the final output.

In Almasri and Kaluarachchi (2005), several neural network outputs were integrated together, to predict nitrate distribution in ground water, using a gating network. A gating network is a very common integration technique, where a specialised network is used to predict a vector of weights, which is used to combine the outputs of different experts (i.e. networks). In Zheng, Lee, and Shi (2006), a Bayesian probability model was used to combine MLP and RBF network predictions based on how well each module performed in previous examples. This Bayesian model tended to give more weight to the module that performed better on previous examples in a certain target period of prediction. Fuzzy logic has also been used as a tool for learned integration (Melin et al., 2007; Hidalgo, Castillo, and Melin, 2009). In Mendoza, Melin, and Licea (2009a), Mendoza, Melín, and Castillo (2009b), and Melin, Mendoza,

and Castillo (2011) an image recognition MNN was proposed where different neural networks were trained on part of the edge vector extracted from the target image. Fuzzy logic was then used to integrate different neural network outputs by assessing the relevance of each module using a fuzzy inference system and integrating using the Sugeno integral. Synaptic integration was done in Anderson et al. (2016), with the aim of achieving transfer learning on a small amount of data. A set of untrained modules were added to a pretrained network and integrated by learning while freezing the original network weights. In another work (Terekhov, Montone, and O'Regan, 2015), a similar approach utilized synaptic integration for multi-task learning, where an initial network was trained on some task, followed by a modular block of neurons that were added and integrated by training on a different task, while again freezing the original network parameters.

Table 2.1
Advantages and disadvantages of different technique

Technique	Advantages	Disadvantages
A. Domain		
1. Manual	• Prior knowledge integration • Fine control over partitions	• Partitions are hard to define • Relation between decomposition and solution is not straightforward • Separation of variation factors is hard
e.g Anand et al. (1995), Oh and Suen (2002), Rudasi and Zahorian (1991), Subirats et al. (2010), Bhende, Mishra, and Panigrahi (2008), Mendoza, Melin, and Licea (2009a), Mendoza, Melín, and Castillo (2009b), Ciregan, Meier, and Schmidhuber (2012), Wang (2015), Vlahogianni, Karlaftis, and Golias (2007), and Aminian and Aminian (2007)		
2. Learned	• Capture useful relations not tractable by human designer	• Computational cost and extra step of learning the decomposing model
e.g Ronen, Shabtai, and Guterman (2002), Fu et al. (2001), and Chiang and Fu (1994)		
B. Topology		
1. HCNR	• Sparse connectivity • Short average path	• Complex structure • Hard to analyse and adapt to problems • Formation difficulty
e.g Bohland and Minai (2001), Huizinga, Mouret, and Clune (2014), Verbancsics and Stanley (2011), Garcia-Pedrajas, Hervas-Martinez, and Munoz-Perez (2003), Mouret and Doncieux (2009), and Mouret and Doncieux (2008)		
2. Repeated Block		
2.1. Multi-Path	• Parallelizable • Suitable for multi-modal integration	• Additional hyperparameters • Currently lacks theoretical justification
e.g Kien Tuong Phan, Maul, and Tuong Thuy Vu (2015), Phan et al. (2017), Ortín et al. (2005), Wang (2015), Phan et al. (2016), Xie et al. (2016), and Guan and Li (2002)		
2.2. Modular Node	• Computational capability with relatively fewer parameters • Can be adapted for hardware implementation	• Additional hyperparameters
e.g Sang-Woo Moon and Seong-Gon Kong (2001), Wei Jiang and Seong Kong (2007), Serban et al. (2016), Soutner and Müller (2013), Phyo Phyo San, Sai Ho Ling, and Nguyen (2011), Karami, Safabakhsh, and Rahmati (2013), Pan et al. (2016), Srivastava et al. (2013), Lin, Chen, and Yan (2013), Eyben et al. (2013), Yu et al. (2016), Hochreiter and Urgen Schmidhuber (1997), Stollenga et al. (2015), Wang, Hilgetag, and Zhou (2011), and Kaiser and Hilgetag (2010)		
2.3. Sequential	• Deep composition	• Hard training • Excessive depth is arguably unnecessary
e.g Szegedy et al. (2015a), Szegedy et al. (2016), Chollet (2016), Srivastava, Greff, and Schmidhuber (2015), and He et al. (2016)		
2.4. Recursive	• Readily Adaptable to recursive problems • Deep nesting with short paths	• Excessive depth is arguably unnecessary
e.g Franco and Cannas (2001) and Larsson, Maire, and Shakhnarovich (2016)		
3. Multi-Architectural	• Better collective performance • Error tolerance	• Computationally complex
e.g Ciregan, Meier, and Schmidhuber (2012), Babaei, Geranmayeh, and Seyyedsalehi (2010), Shetty and Laaksonen (2015), Yu et al. (2016), Pan et al. (2016), Kim et al. (2017), and Weston, Chopra, and Bordes (2014)		
C. Formation		
1. Manual	• Prior knowledge integration • Fine control over components	• Hard in practice
e.g Nardi et al. (2006) and Guang-Bin Huang (2003)		
2. Evolutionary	• Adaptable way for modularity formation • Suitable for HCNR formation	• Lengthy and computationally complex
e.g Huizinga, Mouret, and Clune (2014), Garcia-Pedrajas, Hervas-Martinez, and Munoz-Perez (2003), Braylan et al. (2015), Miikkulainen et al. (2017), Reisinger, Stanley, and Miikkulainen (2004), Hüsken, Igel, and Toussaint (2002), Calabretta et al. (2000), and Di Ferdinando, Calabretta, and Parisi (2001)		
3. Learned	• Dynamic formation of modularity • Sample from large set of models	• Computational complexity • In implicit learned variant, networks are densely connected
e.g Srivastava et al. (2014), Huang et al. (2016b), Singh, Hoiem, and Forsyth (2016), Larsson, Maire, and Shakhnarovich (2016), Andreas et al. (2016a), Andreas et al. (2016b), Hu et al. (2016), Phan et al. (2016), Phan et al. (2017), and Blundell et al. (2015)		
D. Integration		
1. Arithmetic-Logic	• Prior knowledge integration • Loosely coupled modules	• Difficult in practice

continued on next page

continued from previous page

Technique	Advantages	Disadvantages
e.g Gradojevic, Gençay, and Kukolj (2009), Nardi et al. (2006), and Wang et al. (2012)		
2. Learned	• Captures complex relations	• Computationally complex • Tightly coupled modules
e.g Zheng, Lee, and Shi (2006), Mendoza, Melín, and Castillo (2009b), Mendoza, Melin, and Licea (2009a), Almasri and Kaluarachchi (2005), Melin et al. (2007), Melin, Mendoza, and Castillo (2011), and Hidalgo, Castillo, and Melin (2009)		

In the next section, we apply the discussed modular framework to some of the state-of-the-art MNNs, where we analyse their modular composition and show how their general design is captured by our abstracted modular concepts, in a proof-of-concept of the practical applicability of these modularization principles.

2.5 Case Studies

We will present some case studies of state-of-the-art MNNs, for which we will emphasize the modular techniques applied. We will be concerned with the three main levels of modularization, i.e. topology, formation and integration. As discussed earlier, domain modularization is an optional component. In the Inception architecture (Szegedy et al., 2015a), the architecture which set the state-of-the-art in ILSVRC14, the topology is mainly a repeated block architecture, which combines both multi-path and sequential structures. The main skeleton is a sequence of repeated blocks, where each block is a multi-path subnetwork. Formation was manually engineered. The main guiding heuristic for the manual formation was approximating a sparse architecture using dense modules, such that it can still exploit the hardware optimized for dense computations. Integration is a learned integration.

FractalNet (Larsson, Maire, and Shakhnarovich, 2016) topology is a combination of different modular techniques. The main skeleton is a sequential repeated block structure. Each block is a combination of recursive fractal structure, sequential chaining and multi-path branching. The manual formation is based on making the network robust to the choice of overall depth, through the recursive nesting of subnetworks of various depths, such that the learning algorithm can carve out the efficient paths. Integration is a learned integration.

CapsNet (Sabour, Frosst, and Hinton, 2017) can be considered a modular node topology, where each single activation is replaced by a pose vector capturing different instantiation parameters of the underlying object. Formation is manually engineered, where the main guiding principle is to achieve equivariance through pose vectors, transformation matrices and dynamic routing by agreement. Integration is AL based on dynamic routing by agreement. Dynamic routing by agreement acts by combining predictions of the lower layer based on their relative agreement. In that way, it has no learned state.

PCNet++ (Phaye et al., 2018), a variant of CapsNet (Sabour, Frosst, and Hinton, 2017) has a topology of a sequential and mutli-architectural nature. The main skeleton is a sequential stack of three simpler networks called DCNet. The

output of each subnetwork is routed into two branches, one contributes to the final output and the other is used as the input to the next subnetwork in the stack. This way, each subnetwork builds on the previous one and at the same time contributes to the final output as a standalone architecture. The formation is manual based on diversifying the PrimaryCapsules, such that different capsules carry information of various scales of the image. Integration is a simple AL, where the DigitsCapsules from different output levels are concatenated to form the final output.

In their NAS technique, Liu et al. (2017) used a search space of hierarchical modular topology, a special case of modular node topology as discussed earlier. Each searched architecture is a hierarchical structure of building blocks of increasing complexity starting from a pre-defined set of primitives. The techniques for formation and integration were evolutionary search with tournament selection and learned integration, respectively. Another NAS technique was introduced by Bender et al. (2018). This search technique is based on training an exponential number of modular architectures through training a large model, called one-shot model, that includes these architectures as subnetworks. The main topological structure of the one-shot model is composed of a sequential skeleton of multi-path blocks. Formation is a hybrid of implicit and explicit learned formation, where a path-dropping technique is used for regularization such that the one-shot model can be used as a proxy for the performance measure of the different implicit subnetworks. After convergence of the one-shot model, explicit modular networks are sampled based on the performance measure provided by the one-shot model. Integration is a learned integration.

2.6 Conclusion

This chapter aimed at introducing and analysing the main modularization techniques used in the field of neural networks so far, in order to establish a generalized framework for how to systematically implement neural modularity in order to harness its advantages. We devised a taxonomy of modularization techniques with four main categories, based on the target of the modularization process, i.e.: domain, topology, formation and integration. We further divided each category into subcategories based on the nature of the techniques involved. We discussed the advantages and disadvantages of the techniques, and how they are used to solve real-world problems. Analysis and empirical results show that modularity can be advantageous over monolithic networks in many situations.

The review has shown that a wide variety of algorithms for modularization exists, acting on different parts of the MNN life cycle. We have shown that advances in MNNs are not restricted to biologically inspired topological modularity. The quest for modularity in ANNs is far from being a mere case of enforcing networks to be partial replicas of the brain. Even topological modularity is often a vague imitation of brain structure. As the ANN literature has

increasingly diverged from its early biological roots, so has modularity metamorphosed into different shapes and techniques, ranging from biologically-inspired to purely engineering practices.

The techniques reviewed here have ranged from explicit expert-knowledge based techniques to fully automated implicit modularization techniques, each having its specific set of pros and cons and suitability for particular problems. Some techniques were found to be tailored to satisfy the specific constraints of particular problems, while others were found to be generic, trading specialization performance for full automation and generalizability. Neural modularization was shown to be a sequential application of techniques, which we called modularization chain, where each technique acts on a different aspect of the neural network.

Although modularity has many advantages over monolithic deep networks, the main trend is still oriented towards monolithic deep neural networks. This is mainly due to the many successes of monolithic deep learning in different areas throughout the last decade. Also, the lack of extensive research into learning and formation techniques for neural modularity makes it hard for practitioners to efficiently deploy the approach. Contrary to this, monolithic networks have attracted extensive research that has generated a critical mass of theoretical insights and practical tricks, which facilitate their deployment. EAs are currently the main actors in complex modular neural network construction. However, the debate of whether EAs are the best approach for MNN formation and if they harness the full power of modularization and problem decomposition is still open. Also, there is still a significant gap on how to stimulate problem decomposition in modular networks, so that their topological modularity may also become a full functional modularity.

We tentatively predict that as the challenges facing deep learning become increasingly hard, a saturation phase will eventually be reached where depth and learning techniques may not be enough to fuel progress in deep learning. We do not view modularity as a replacement for depth, but as a complementary and integrable approach to deep learning, especially given that excessive depth is becoming increasingly criticized for reasons of computational cost and extraneousness. The dilemma is similar to the software quality problem, where exponential growth in hardware efficiency is masking poor algorithmic optimization. We believe that as deep learning becomes increasingly applied to more challenging and general problems, the need for robust Artificial General Intelligence practices will sooner or later promote the modularization of neural networks.

Based on the context introduced in this chapter, we use modularity in the subsequent chapters to improve different performance measures of ANNs. We make use of different modular tools like multipath and modular node topologies (Section 2.4.2), manual and learned formation (Section 2.4.3) and learned and AL integration (Section 2.4.4).

Chapter 3

Balancing Accuracy and Latency in Multipath Neural Networks

3.1 Preface

Despite the extraordinary success achieved by deep learning and ANNs techniques in complex AI tasks that defied classical AI and ML for decades, they are becoming more and more computationally demanding over time. Increasing capacity in a coordinated way has led to a remarkable increase in model accuracy. This gave rise to the compute-centric approach which utilizes centralized data repositories and computational resources to train high-capacity models. Since this approach is not suitable for applications with limited resources, the data-centric approach which utilizes distributed data and local computations was actively developed to meet the requirements of these use cases.

Because of these requirements in the data-centric approach, a new breed of models optimized for constrained resources was needed. Usually, as mentioned, the capacity/complexity of a model has a strong relation with its accuracy. This implies that miniaturizing models needs to be done while taking this delicate balance into account. Many techniques have been developed to realize this potential like factorization (Jaderberg, Vedaldi, and Zisserman, 2014; Howard et al., 2017), quantization (Han, Mao, and Dally, 2015) , hashing techniques (Chen et al., 2015), pruning techniques (Han, Mao, and Dally, 2015) and knowledge distillation (Hinton, Vinyals, and Dean, 2015). NAS is a general technique that optimizes an ANN architecture by searching for potential candidates in a given search space. Some NAS techniques were adjusted to search for potential architectures while taking the accuracy-latency balance into consideration (Tan et al., 2019; Tan and Le, 2019; Cai, Zhu, and Han, 2018).

Multipath Neural Network (MpathNN) is an MNN with branched structure that carries computations in parallel through multiple network paths (Kien Tuong Phan, Maul, and Tuong Thuy Vu, 2015; Phan et al., 2016; Phan et al., 2017; Ortín et al., 2005; Wang, 2015). Compared to an MLP, with all other architectural hyperparameters like width and depth being equal, MpathNN has a sparser parameter set and, hence, is less computationally expensive. However,

deriving an MpathNN from an equivalent MLP implies introducing two more extra hyperparameters representing how many paths are needed and where to divide the network to get the extra paths.

In this chapter, we aim to harness the potential of deriving less computationally expensive MpathNN that may be used in limited resource conditions. MpathNN is one of the MNN topologies we discussed in Section 2.4.2. Since this process requires choosing how many paths are needed and where to place the divisions, we use a NAS technique, mainly a one-shot model, to optimize for these architectural hyperparameters. A NAS technique is a variant of the MNN learned formation techniques discussed in Section 2.4.3 and the integration in this case is learned integration as discussed in Section 2.4.4. By implicitly ranking the relative effects of different divisions, we prune the network in a coordinated way to introduce sparsity while preserving accuracy as much as possible. This is a common pruning technique that was chosen because of its simplicity and implementation efficiency. Also, by sampling different architectural candidates from the search space, we can establish a relation between latency and accuracy across the different potential MpathNN candidates that allows a modeler to balance the accuracy-latency requirements without the need for exhaustive training of models. Our results show that our method can find MpathNN candidates with a good balance between accuracy and latency and can reliably predict their performance with high precision.

In Section 1.5, we discussed the problem of efficiently implementing a modular architecture. While we did use optimization techniques in our modular work in Chapter 4 and Chapter 5, in this chapter we used a conventional dense zero-entries matrix. That is, the sparsity was incorporated into the network's feedforward pass by replacing the pruned connections by zero entries. This, however, was still efficient since we needed sparsity only at inference time. However, finding efficient generic techniques for accelerating modular architectures is crucial to the artificial modularity field as we discuss more in Section 3.7 and Section 3.8.

3.2 Introduction

Deep learning has gained momentum as an ML approach that surpassed classical techniques in a broad category of complex tasks by introducing multiple layers of feature composition. As a side effect, the computational complexity of its models grew substantially and they are becoming exceedingly demanding with the accelerating research in the field. The field has seen a trend of strong correlation between increasing model sizes, in a coordinated way, and the accuracy gain in different tasks. Applications that utilize a compute-centric approach, i.e. central repository of accumulated data and high computational resources that can be used for training and enhancing models, could scale their

computational resources and data size sufficiently to accommodate for the exploding demand of the bigger-size data-greedy models.

However, not every application lends itself to central computation. Handheld devices and IoT are pervasive technologies with rich data. For many concerns like privacy, security and bandwidth, these data can not be centralized and a different approach is needed. The data-centric approach, hence, depends on distributed computing and learning across many terminal devices that are generally limited in resources.

Latency refers to the increased processing runtime of a neural network, usually as measured at inference time. While there are implementation and hardware factors that can contribute to a network's latency and cause the same architecture's latency to vary across devices (Yang, Lu, and Ren, 2020), the parameter count of a network is a very important proxy measure to a network's latency, specially when benchmarking similar architectures (Howard et al., 2017; Sandler et al., 2018; Iandola et al., 2016; Hinton, Vinyals, and Dean, 2015). Also, since the parameter count offers a good proxy measure of the model's memory complexity, it is considered an overall good measure of the suitability of a model to limited-resource applications. In this work, we similarly rely on the parameter count as a proxy to the model latency at inference time.

Since deep learning is computationally demanding, ways of reducing its complexity, while maintaining its performance are needed to harness the potential of the passive data available in peripheral and limited resource devices. Several methods were researched in the quest to achieve this goal. Separable convolutions were used in a set of techniques (Howard et al., 2017; Sandler et al., 2018; Iandola et al., 2016) to reduce network size and complexity. Quantization, hashing and pruning techniques (Han, Mao, and Dally, 2015) aim at reducing parameter precision, and allow sharing of parameters based on activations and sparsifying parameters, respectively. Factorization (Jaderberg, Vedaldi, and Zisserman, 2014) reduces the number of parameters by approximating complex computations by another smaller set of factored computations. Distillation (Hinton, Vinyals, and Dean, 2015) effectively compresses knowledge from a large network into another smaller network.

NAS is a set of related techniques for optimizing neural architecture by searching in a prespecified search space. Vanilla NAS techniques depend on sampling architectures from the search space. These architectural samples are then trained and evaluated in order to guide the search process towards more promising architectures (Kyriakides and Margaritis, 2020). This sampling process can happen through techniques like RL (Zoph and Le, 2016; Zoph et al., 2018), EAs (Liu et al., 2017) and random search (Liu et al., 2017; Zoph et al., 2018).

Training and evaluating a large number of sampled architectures are computationally expensive processes that need a serious scale of distributed computing.

One-shot NAS techniques (Liu, Simonyan, and Yang, 2018; Zhao et al., 2020) were developed to avoid these heavy computational needs. A supernet is a large network that spans multiple subnetwork architectures with overlapped weight sharing. By single-time training of a supernet that is designed to span some search space, one-shot NAS can approximate the evaluation of an exponential number of architectures without the need for exhaustive training and evaluation of individual networks. Furthermore, specialized NAS techniques were developed with search spaces aimed at limited resource conditions (Tan et al., 2019; Tan and Le, 2019; Cai, Zhu, and Han, 2018).

MpathNNs are NNs that have multiple, usually independent, paths. Compared to another model having the same width and depth, MpathNNs have a sparser set of parameters. MpathNNs have been used in several previous studies for improving generalization (Kien Tuong Phan, Maul, and Tuong Thuy Vu, 2015; Cireşan, Meier, and Schmidhuber, 2012; Wang, 2015), image captioning (Yu et al., 2019a), feature extraction (Yu et al., 2019b), cross modal learning (Hong et al., 2015; Hong et al., 2019), and dimensionality reduction (Zhang, Yu, and Tao, 2018).

We can think of MpathNNs as a way of sparsifying a FC model with similar width and depth (in general, the same architectural hyperparameters excluding the number of learned parameters). However, it is not clear how to divide such an FC model into multiple paths specially given that the number of possible MpathNNs corresponding to an FC model is intractable to be evaluated thoroughly. We utilize a one-shot model to evaluate the relative importance between such divisions. Then, we can prune the network using this information to get an MpathNN with decent balance between latency and accuracy. We further construct an approximate relation between complexity and accuracy by fine-tuning and validating a small sample of possible models. This allows a modeler to satisfy a given accuracy-latency requirement ahead of any specific MpathNN model training. Finally, the model can be fine-tuned for a given complexity to yield the final desired model. Our contributions are:

- Using a one-shot model to approximate the relative accuracy across all possible divisions of paths of an FC model.
- We utilize the information gained from the search process to get MpathNNs models with a desired path count through a pruning process.
- We use a predictive model learned from a small architectural sample to predict potential model accuracy ahead of any specific model training/fine-tuning.

3.3 Multipath Neural Networks

Biological nervous systems exhibit multipath branching and parallel computations in their structure (Gollisch and Meister, 2010; Otsuna, Shinomiya, and Ito, 2014). This inspired ANNs which adopt similar structures. Multipath CNN was used by Cireşan, Meier, and Schmidhuber (2012) where different paths have a different preprocessed version of the same input. The outputs of different columns were averaged to produce the final output. In a similar approach by Wang (2015), different paths were fed with differently filtered versions of the input and the output was consolidated from the multipaths by a fully connected subnetwork. In image captioning, Yu et al. (2019a) used several detectors applied in parallel for multi-view feature learning. For unsupervised dimensionality reduction, Zhang, Yu, and Tao (2018) used the same idea by applying local contractive autoencoders to extract local features and then applied affine transformations to align with a global coordinate system. Yu et al. (2019b) extracted hierarchical features of different granularity using a parallel structure applied to both image and text data. For Human Pose Recovery (HPR), Hong et al. (2015) extracted 2D and 3D features from different modalities and learned to map 2D to 3D features. Similarly for HPR, Hong et al. (2019) fed inputs from different modalities into different branches, each with its own reconstruction task.

Inception-v1, the ANN that won ILSVRC-14, was introduced by Szegedy et al. (2015b) and it exhibits highly branched structure. Inception-v2/v3 (Szegedy et al., 2016) and Xception (Chollet, 2016) improved over inception-v1 by exploiting more multibranching. ResNetXt (Xie et al., 2016) and Residual Inception (Zhang et al., 2018) are variants of ResNet (He et al., 2016) where the modular block has multipath structure. FractalNet, proposed by Larsson, Maire, and Shakhnarovich (2016), has a recursive highly branched structure. The parallel circuit network by Phan et al. (2016) is based on a multipath structure and showed generalization improvement over FC networks when regularized by DropCircuit (Phan et al., 2018). PathNet (Fernando et al., 2017) is a super NN (a large NN that is supposed to contain multiple subnetworks) that is targeted at learning sequential tasks efficiently. The network is highly branched and a GA is used to select which subnetwork to learn. After the convergence of each task, the fittest path is frozen before moving to the next task.

3.4 Neural Network Compression

Due to the explosion of NN sizes and the need to run on limited resource devices, several techniques have been investigated to reduce network sizes while maintaining as much accuracy as possible. Howard et al. (2017) used a factorization

technique of separable convolutions and 1x1 convolutions in MobileNet to reduce complexity. More fine control over architecture size was done by width and resolution multipliers, two hyperparameters that control network width and representation resolution, respectively. MobileNetv2 (Sandler et al., 2018) enhanced over MobileNet by introducing inverted residuals and bottlenecks in order to reduce computations and promote the extraction of useful representations. A similar approach was adopted by Iandola et al. (2016) where they introduced two different types of cells, namely squeeze and expansions cells, containing either 1x1 filters or a mix of 1x1 and 3x3 filters, respectively. Chen et al. (2015) used the hashing trick (Weinberger et al., 2009; Shi et al., 2009) to share weights based on feature activation. Han, Mao, and Dally (2015) used a combination of pruning, quantization/Huffman-coding and fine tuning. After training, the network was pruned based on the weight magnitudes. The sparse weights were then quantized and Huffman-coded and, finally, fine-tuned. The idea by Jaderberg, Vedaldi, and Zisserman (2014) was to factor convolutional filters into rank-1 horizontal and vertical vectors. The separation of the kernels was based on gradient-guided optimization. Knowledge distillation (Hinton, Vinyals, and Dean, 2015) is a knowledge transfer technique that trains a smaller student network using soft targets generated from a larger teacher network. The soft targets are generated from the teacher using a softmax with high-temperature (i.e. higher than 1). Two objectives are used for the training, the first is based on the cross entropy of the soft targets with the same high-temperature used for generating the soft targets and the other is based on the actual labels with a softmax temperature of 1.

3.5 Neural Architecture Search

NAS is the set of techniques targeted towards the automated optimization of ANN architectures. Zoph and Le (2016) used an RNN to sample an ANN architecture. The sampled architecture was then trained and evaluated to provide the necessary reward that would be used to train the RNN through RL. Zoph et al. (2018) used a similar RL technique, however, instead of searching for a complete ANN architecture, they restricted their search to finding a cell/block that would be repeated later to build a full architecture. Zhang et al. (2020) proposed a NAS technique for DenseNet (Huang et al., 2016a). The core idea was to prune unnecessary skip connections by formulating the pruning process as a Markov Decision Process (MDP). An RL agent was then trained to suggest the potential connections for pruning.

As discussed, one-shot NAS depends on training a supernet in order to implicitly evaluate the performance of many architectures in parallel without the need for exhaustive evaluation of individual candidates. Saxena and Verbeek (2016) used a supernet that spans a CNN search space by arranging the network architectural hyperparameters across three axes: (1) a layer axis, which

is analogous to the usual depth axis, (2) a scale axis, which spans feature maps with different resolutions and (3) a channel axis, across which the number of channels vary. DARTS (Liu, Simonyan, and Yang, 2018) is another differential NAS technique that uses a set of stacked modular multipath cells, each having a branched structure with different operations. The operations are combined by a parameterized weighting function and the whole model is optimized end-to-end for architectures with higher accuracy. PC-DARTS (Xu et al., 2019) mitigated the high memory need of DARTS by downsampling the number of channels of each node input. To avoid fluctuations in the architecture optimization due to this random sampling process, node outputs were combined in a normalized learnable way, instead of concatenation like in DARTS. A technique similar to DARTS is used by Cai et al. (2019) where the weighting factors of the different operations were produced by a secondary network. A highly branched network was used in a NAS technique by Bender et al. (2018) to evaluate multiple branched NNs in parallel. Random paths were dropped during training as a form of regularization and subnetworks are evaluated by sampling from the trained supernet.

Some NAS techniques were developed with the aim of searching architectural spaces that suit limited resource applications in mind. Tan et al. (2019) used a search space based on diverse, instead of homogeneous, blocks. The architectural search optimizes for accuracy subject to a constraint of maximum allowed latency. The latency was measured as the runtime on actual mobile devices, instead of FLOPS. Tan and Le (2019) analysed the effect of scaling the width, depth and resolution of the network, and based on that, modified the work by Tan et al. (2019) to enforce uniform scaling. Cai, Zhu, and Han (2018) used a NAS technique that samples different potential operations from the search space, in contrast to techniques like DARTS (Liu, Simonyan, and Yang, 2018) that use a linear mixture of possible operations.

The approach most similar to our technique is slimmable NNs (Yu et al., 2018a). Slimmable NNs aim to balance network complexity and accuracy by activating different fractions of the total network width. The network is trained by accumulating gradients from a predefined set of width fractions.

In this work we use a one-shot model to evaluate different possible divisions of a fully connected network into independent paths and, then, guide the pruning process to get potential MpathNN models. Our work is different from Yu et al. (2018a) in that the whole width of the network is active since we aim at evaluating all possible divisions. The one-shot model, in contrast to previous work, is used to draw a relation between parameter count and validation error, which can aid the modeler to realize the required accuracy-latency balance beforehand, without the need for training multiple models from scratch.

3.6 Methodology

For our purposes, we formally define a MpathNN as a NN having two or more independent paths (i.e. having no interconnections). We also define an MpathNN-equivalent MLP, which we will refer to by an equivalent MLP for convenience, as an NN having the same number of layers (i.e. depth) and number of nodes per layer (i.e. width). In the same way, we will call any MpathNN that has an equivalence to a given MLP as a derived MpathNN. An equivalent MLP has certainly more capacity in terms of its parameter count than any corresponding derived MpathNN. Pruning an MLP into a number of paths is done by dropping a set of connections in the way described later. Thus, a derived MpathNN is more computationally and memory efficient, at the expense of losing capacity, which may lead to accuracy degradation. For some applications, like models targeted at devices with limited resources, some degree of accuracy degradation can be tolerated in order to achieve a more efficient runtime and less memory consumption.

Due to the limited resources in these kinds of applications, usually a model is trained on more powerful machines, and used later for inference on the resource-constrained device. At one extreme, one can imagine training every possible derived model and choosing the one with the optimal balance between accuracy and complexity. This is, however, not practically possible for almost any realistic MLP due to combinatorial explosion[1].

Our aim in this chapter, is to use a method similar to one-shot models (Liu, Simonyan, and Yang, 2018; Xu et al., 2019; Cai et al., 2019; Bender et al., 2018; Yu et al., 2018a; Saxena and Verbeek, 2016) to approximate this kind of accuracy-latency relation, without the need to train every possible derived model. Fig. 3.1 shows a sketch of the one-shot NAS and the pruning steps of our method. In step (a), a one-shot NAS model based on an equivalent MLP is initialized. Then, this model is fully-trained to estimate the relative importance of different path divisions as shown in step (b). A sample of derived models with different numbers of paths can be generated from the trained model by pruning. Step (c) shows the pruned models with 2 paths (i) and 3 paths (ii). The derived models are fine-tuned for a small number of epochs and, then, validated to establish a relation between capacity and validation accuracy. Based on

[1]Calculating the number of unique derived MpathNN models is not a trivial problem and is beyond conventional combinatorial techniques and the scope of this work. The problem is related to the integer partitions problem which can be solved using generating functions or through an approximation formula (Andrews, 1998; Andrews and Eriksson, 2004). In a more recent work, Bruinier and Ono (2013) found a closed-form formula which is, nonetheless, not trivial. For our purposes, it is sufficient to state that for a model following the simplified scheme shown in Fig. 3.2 with width W, the number of unique architectures grows sub-exponentially in W (Andrews and Eriksson, 2004) and that for widths as small as 64 and 128, the number of unique architectures is on the order of 1.7M and 4.4B, respectively.

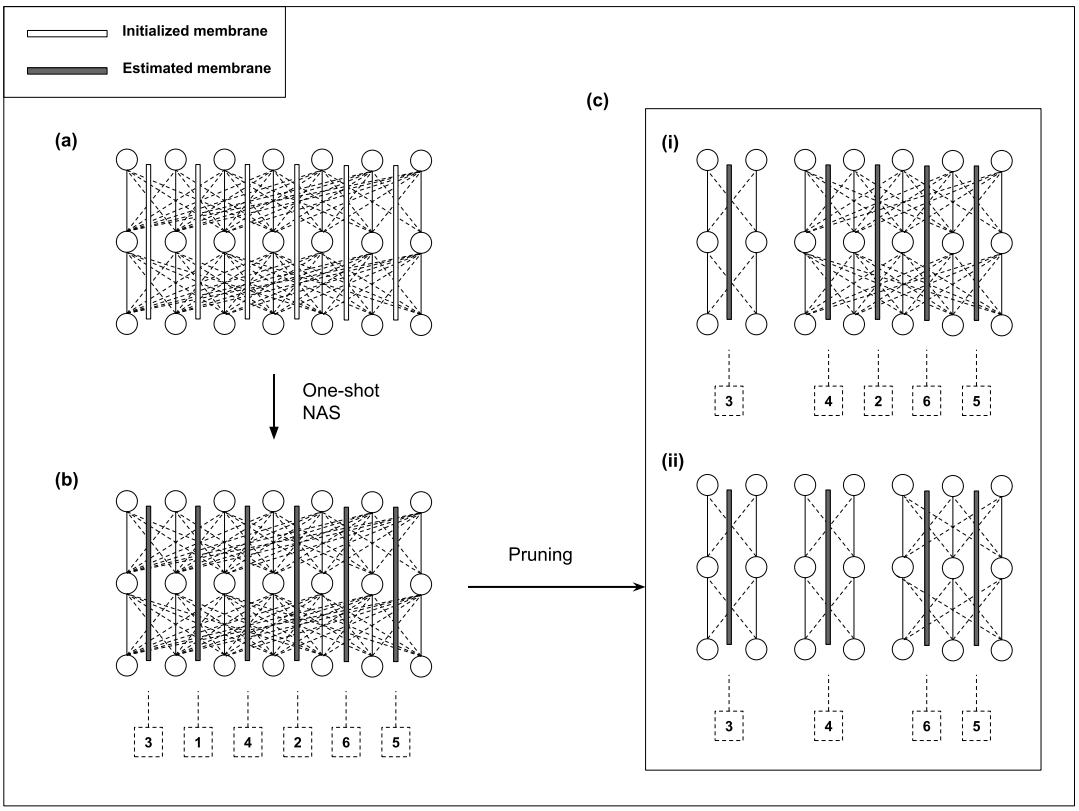


Figure 3.1. One-shot NAS of an MpathNN with 3 hidden layers and a width of 7 nodes. The input connections and the output layer were omitted for clarity. (a) An equivalent MLP is turned into a one-shot model by inserting a set of randomly initialized membranes. (b) The estimated membrane permeabilties are ranked in ascending order. (c) Pruning can be used to get an MpathNN with a given path count, e.g MpathNN with 2 paths (i) or 3 paths (ii).

this relation, a modeler can select the model with the desired balance between accuracy and complexity and fine-tune it to get the final inference-ready model.

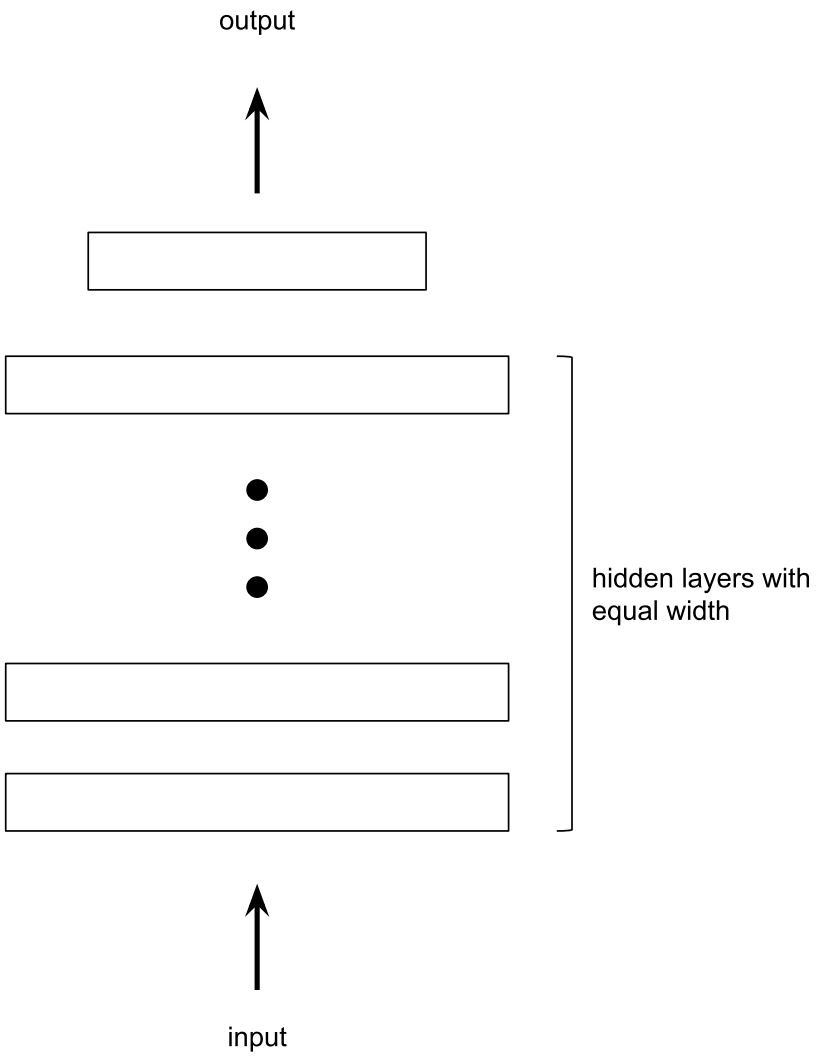


Figure 3.2. Simplified Equivalent MLP Architecture

For simplicity of analysis and efficiency of implementation purposes, we consider only MpathNNs which follow the structure of the equivalent MLP depicted in Fig. 3.2. The set of models following this general architecture consist of a number of hidden layers, all having the same width (i.e. number of nodes). The first hidden layer (i.e. the layer receiving the input) is a FC layer. The remaining hidden layers are divided into a number of independent paths, each having the same width for all of their layers. The output layer is an FC layer much like the first hidden layer, receiving all path outputs as a concatenated vector.

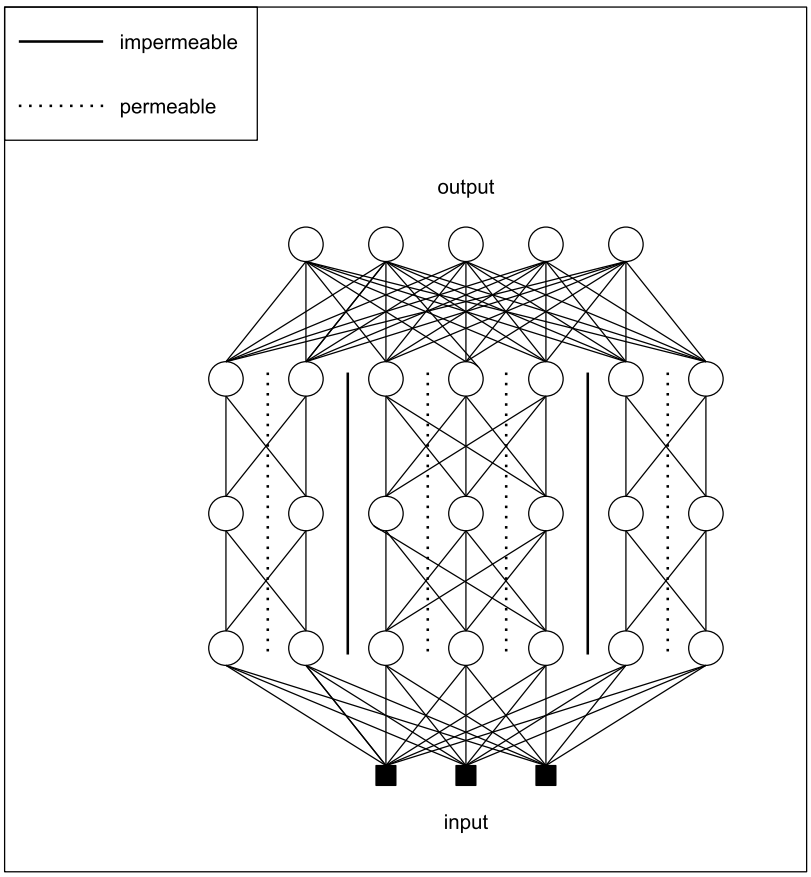


Figure 3.3. MpathNN Example with 3 paths

We can think of an MpathNN as a variation of an FC network, where we draw an imaginary line, which we will call a membrane, between each two adjacent neurons across all the hidden layers starting from the second hidden layer up to the final hidden layer. This membrane has a permeability that ranges from impermeable (0) to fully permeable (1). Then, we allow connections based on this permeability, with full connection strength if the membrane is fully permeable, no connection if it is impermeable and a graded connection strength in between. This means that for an MpathNN of width W, if an impermeable membrane is located between the two adjacent neurons n_i and n_{i+1} with indices i and $i+1$, then for any two successive hidden layers, no connections going into neuron n_{i+1} from its previous layer are allowed from the set of neurons $\{n_k \mid k \leq i\}$ and, similarly, no connections are allowed into a neuron n_i from the set of neurons $\{n_l \mid l \geq i+1\}$. An example diagram is shown in Fig. 3.3.

We think of each of the described membranes between any two adjacent neurons as having a permeability $p(m_i) \in [0,1]$ described by

$$p(m_i \mid \mathbf{\Omega}_i) = \sigma(\mathbf{\Omega}_i) \mid \forall m_i \in \mathbb{M} \tag{3.1}$$

where $\mathbb{M}$ is the set of membranes with cardinality $|\mathbb{M}|$ equal to $W-1$ (where W is the network width), $m_i \in \mathbb{M}$ is the membrane between neurons n_i and n_{i+1}, σ is the sigmoid nonlinearity and $\mathbf{\Omega}$ is the set of weights parameterizing the membrane permeabilities. Note that $\mathbf{\Omega}$ is shared between all hidden layers. Also, note that at inference time, we need to derive an MpathNN, hence, a membrane is either permeable, so that it adds no paths to the architecture, or impermeable and so it contributes one additional path to the modular architecture. However, in order to perform a differential search on our one-shot model, we need the permeability function to be differentiable. We will discuss later how we use pruning to discretize these permeabilities to obtain an MpathNN.

As we have a set of membranes spanning the whole network between each two adjacent neurons, we need to model the general connectivity between any two neurons with connections crossing multiple membranes. In order to model a connection between any two neurons n_i and n_j, we generalize equation Eq. (3.1) by considering the set of membranes that will be crossed by the given connection to connect the two neurons

$$p(c_{i\to j} \mid \mathbf{\Omega}_{i:j-1}) = \min_{k\in[i,j-1]} \sigma(\mathbf{\Omega}_k) \tag{3.2}$$

where $c_{i\to j}$ is the connection from neuron n_i in the hidden layer $l-1$ into neuron n_j in the following hidden layer l and $i < j$. Note that due to symmetry, $p(c_{j\to i}) = p(c_{i\to j})$. In other words, when a connection crosses multiple membranes, we model it by the lowest membrane permeability it crosses. The reasoning is that a single impermeable membrane is sufficient to hinder a connection.

To create our one-shot model, we need to integrate these permeabilities into our equivalent MLP feedforward pass so that we can perform an end-to-end optimization. We do this by multiplying each weight by its corresponding permeability in the feedforward pass. In general,

$$\widehat{w}_{i,j}^{(l)} = w_{i,j}^{(l)} * p(c_{i\to j}) \tag{3.3}$$

where $w_{i,j}^{(l)}$ is the weight connecting neuron n_i in layer $l-1$ to neuron n_j in layer l. The resulting modified weight matrix is then used for carrying on the feedforward in the usual way. We, then, train the model parameters (including membranes parameters) till convergence. We refer to this phase as the one-shot training.

After the model convergence, we will have the estimations of the membrane permeabilities, with each permeability $p(m_i) \in [0,1]$. To have an MpathNN with a given path count from the converged model, we need a way to select a set of membranes to prune (i.e. to make impermeable), based on their corresponding permeabilities. We note that adding one more impermeable membrane to an equivalent MLP will add one more path to the resulting MpathNN. This

means that the range of possible path counts M in a derived MpathNN with width W is $[2, W]$. For each possible path count, there will be many possible ways to make the additional division (i.e. to place the additional impermeable membrane) and each division will give rise to an MpathNN with a different capacity (i.e. parameter count), and, hence, with a different accuracy in general. To choose a good division, we follow the following pruning technique. We select which membranes to make impermeable by ordering the membrane permeabilities in ascending order. Then, to have an MpathNN with a given path count b, we set to zero the ordered membrane permeabilties with indices in the range $[1, b)$. This is motivated by the intuition that weights multiplied by a small permeability will be less important for the MpathNN accuracy. We set the permeabilities of the remaining permeable membranes to one. The described pruning technique based on membrane permeability magnitude was chosen because of its simplicity and implementation efficiency.

While each path count can have many corresponding MpathNNs with different parameter counts, under the described pruning technique, each path count will be mapped to a specific MpathNN with a unique parameter count. To establish a relation between the path count, and hence the parameter count, and accuracy, we need to obtain the validation accuracies for a sample of MpathNNs with path counts that sufficiently cover our search space. Given some architecture sample size, we choose to sample a set of path counts $\mathbb{B} = \{b : b \in [2, W]\}$ that are evenly spaced from each other over the specified interval $[2, W]$. For example, for $W = 10$ and a sample size of five, we choose $\mathbb{B} = \{2, 4, 6, 8, 10\}$. Then, for each path count $b \in \mathbb{B}$, we obtain the corresponding MpathNN with b number of paths from the one-shot model using the described pruning strategy. We, then, fine-tune each of these MpathNNs for a small number of epochs and calculate their validation accuracies. We refer to this phase as sample fine-tuning.

At the end, we are left with a set of path counts $\mathbb{B}$ and their corresponding validation accuracies $\boldsymbol{v}$. As we discussed, path counts can be uniquely mapped to parameter counts under the defined pruning technique. Despite the fact that we are ultimately interested in the parameter counts, we use the path counts as the input to the regression model in order to make the regression more stable. The reason is that path counts are a well-behaving arithmetic sequence of integers, while their corresponding parameter counts are a sequence of integers with irregular gaps. Hence, we fit a simple linear regression model using path counts as an input and the corresponding validation accuracies as the target. To do that, we first order the path counts in an ascending order and reorder the corresponding accuracies $\boldsymbol{v}$ to maintain the alignment between each path count and its associated accuracy. With an overload of notation, we will refer to this ordered array also by $\mathbb{B}$. Then, the regression model is trained to predict the accuracy array from the path count array. This means that the regression model is a map $g : \mathbb{N}^{|\mathbb{B}|} \to [0, 1]^{|\mathbb{B}|}$ with input and output sizes $|\mathbb{B}|$

and the training data is just a single point consisting of the array pair $(\mathbb{B}, \boldsymbol{v})$. The reason for not treating the path counts and their associated accuracies as multiple data points of a scalar dimensionality is that the mapping of the accuracy would be reduced to a linear function of the path count. This would limit the prediction accuracy significantly. While fitting the model to the path counts and the accuracies as a single point in a higher dimensional space is still a linear transformation, this will increase the prediction accuracy by utilizing more information from the relation between single values. Alternate methods, like polynomial fitting or non-linear models, can be used, however at the cost of introducing more complexity or additional hyperparameters.

We use this regression model to predict the validation accuracy for any path count not in the regression data by the following method. We insert the path count that we need to predict the accuracy of in the ordered path count array $\mathbb{B}$ in a correctly ordered position. For example, if our path count array is $\mathbb{B} = \{2, 4, 6, 8\}$ and we want to predict the accuracy of MpathNN with 5 paths, then we have $\mathbb{B}' = \{2, 4, 5, 6, 8\}$ where we have inserted 5 at index 3 (indices start from 1). To maintain the correct input size of our regression model (4 for this example), we need to remove one of the old elements in $\mathbb{B}'$. We do that by removing the first element of the ordered array if the new path count was inserted as the last element or by removing the last element if it was inserted anywhere else. For our example, since the new path count 5 was not inserted as the last element, we remove the last element in the new array which becomes $\mathbb{B}' = \{2, 4, 5, 6\}$. After that, we can use our trained regression model to map the new array $\mathbb{B}'$ into a predicted accuracy array $\boldsymbol{v}'$. The accuracy prediction for our new path count is then acquired from the same index in $\boldsymbol{v}'$ (i.e. the insertion index used in $\mathbb{B}'$), which for our example would be index 3.

Using the described inference process, a modeler, having a range of required accuracies and an upper limit on resources, can make the required trade off between the two quantities by estimating the validation accuracy, and hence an approximate test accuracy, for any path count and its corresponding parameter count, without the need to train the intractable number of all possible MpathNN models. As we discussed, while we use the path counts for regression, they are uniquely convertible to parameter counts since the pruning process produces a unique MpathNN for each path count.

3.7 Experiments

To assess our method, we tested three different architectures, following the simplified general architecture described earlier, on three different datasets. We will refer to each architecture by the abbreviation [mlp or mpath]-[hidden layers number]-[hidden features]. The prefix is mlp if it is an equivalent MLP or mpath if it is an MpathNN model. We follow the prefix by the number of hidden layers and, then, the number of hidden features in each hidden layer. Our

three architectures have 4, 4 and 6 layers and 500, 200 and 200 hidden features, respectively. For each of the used architectures, we train an equivalent MLP to act as a baseline.

We test on three datasets, namely, MNIST, CIFAR10 and iWildCam2019 as described in Section 1.11. We use 300 epochs for the equivalent MLP training and the one-shot training of MpathNN. After convergence, we report the test error of both the equivalent MLP and MpathNN. The test error for MpathNN at this point is calculated without any divisions into independent paths by the pruning technique, i.e. according to Eq. (3.3), and we will refer to it by the one-shot MpathNN test error. For the sample fine-tuning phase, we use 10 evenly spaced path counts. We fine-tune the corresponding MpathNNs for 10 epochs and then report the error on the validation dataset. After that, we sample a new set of path counts that were not used in the sample fine-tuning phase, fine-tune their corresponding MpathNNs for 30 epochs and calculate the test error. After we fit the regression model to the validation error data, we use it to predict the validation error for the new set of path counts and calculate the Pearson correlation and the Mean Absolute Error (MAE) between the test error and the predicted validation error. The Pearson correlation reflects how much the predictions can model the relative relation between the test error of different derived MpathNN models, while the MAE measures how accurate the predictions are.

For all of our experiments, we do three trials per condition, each with a different random initialization. The weights parameterizing the membrane permeabilities are initialized according to

$$\mathbf{\Omega}_i \sim U(-1, 1) \tag{3.4}$$

where U is the uniform distribution. We report the test error of the equivalent MLP and the one-shot MpathNN test error as an average of the three trials. We, then, randomly choose an MpathNN model from one of these three trials to conduct the sample fine-tuning phase. Pruning and sampled architecture fine-tuning will be based on this selected model. For the optimization, we use Adam optimizer.

From each dataset, we use a small percentage of the training dataset as a validation dataset (Section 1.11). The validation dataset is used for two purposes. First, calculating the validation error after each epoch during the training of the MLP and the one-shot training of the MpathNN to select the best model. Second, calculating the validation error during the sample fine-tuning phase.

As we discussed in Section 1.5, implementation efficiency is one of the main problems in MNNs research. We have used optimization techniques to enhance the modular architecture efficiency in the work we have done in Chapter 4 and Chapter 5 and we elaborate more on the used technique in each corresponding chapter. Due to the non-regularity of the search process in this chapter, we had

to rely on a conventional dense matrix with zero entries at inference time. This means that we carry on the forward computation in the usual way, however, we zero out the matrix entries corresponding to the connections that were pruned due to the introduced path divisions. Since inference is not very computationally demanding except for models with extreme capacity, experimentation was smooth. We elaborate more on this choice in Section 3.8.

3.7.1 Results

The test performance results of the equivalent MLP models and MpathNN models after the one-shot training (i.e. as per equation Eq. (3.3)) are listed in Table 3.1, Table 3.2 and Table 3.3. Relative difference in performance is consistent across the three different architectures. For MNIST and iWildCam2019, MLPs perform consistently better than MpathNNs, while the reverse is true for CIFAR10.

Model	# hidden layers	# hidden features	Test error (%)
mlp-4-500	4	500	1.49±0.06
mpath-4-500	4	500	1.64±0.05
mlp-4-200	4	200	1.79±0.1
mpath-4-200	4	200	1.87±0.05
mlp-6-200	6	200	1.7±0.1
mpath-6-200	6	200	1.82±0.1

Table 3.1
MNIST results

Model	# hidden layers	# hidden features	Test error (%)
mlp-4-500	4	500	49.08±0.5
mpath-4-500	4	500	46.94±0.2
mlp-4-200	4	200	48.6±0.2
mpath-4-200	4	200	47.71±0.2
mlp-6-200	6	200	49.75±0.3
mpath-6-200	6	200	48.65±0.1

Table 3.2
CIFAR10 results

Model	# hidden layers	# hidden features	Test error (%)
mlp-4-500	4	500	22.57±0.3
mpath-4-500	4	500	23.27±1.2
mlp-4-200	4	200	23.7±0.5
mpath-4-200	4	200	26.1±0.5
mlp-6-200	6	200	23.1±0.9
mpath-6-200	6	200	26.83±0.9

Table 3.3
iWildCam2019 results

Fig. 3.4, Fig. 3.5 and Fig. 3.6 show different plots of the sample fine-tuning phase for the different architectures benchmarked on the three datasets. The upper left plot of each figure shows the validation error as a function of the path count. The upper right plot shows the validation error as a function of the percentage of parameters in the corresponding MpathNN with a given division of paths. The percentage of parameters here is relative to the parameter count of the equivalent MLP, i.e. $\frac{N_d}{N}\%$, where N_d is the parameter count of a given division of an MpathNN and N is the parameter count in an equivalent MLP. Note that, as we discussed, the mapping from a path count to its corresponding parameter count is unique under the previously described pruning technique. Hence, the path count and parameter percentage scales are unique maps from each others, however, in reverse direction, i.e. larger path count means smaller parameter percentage.

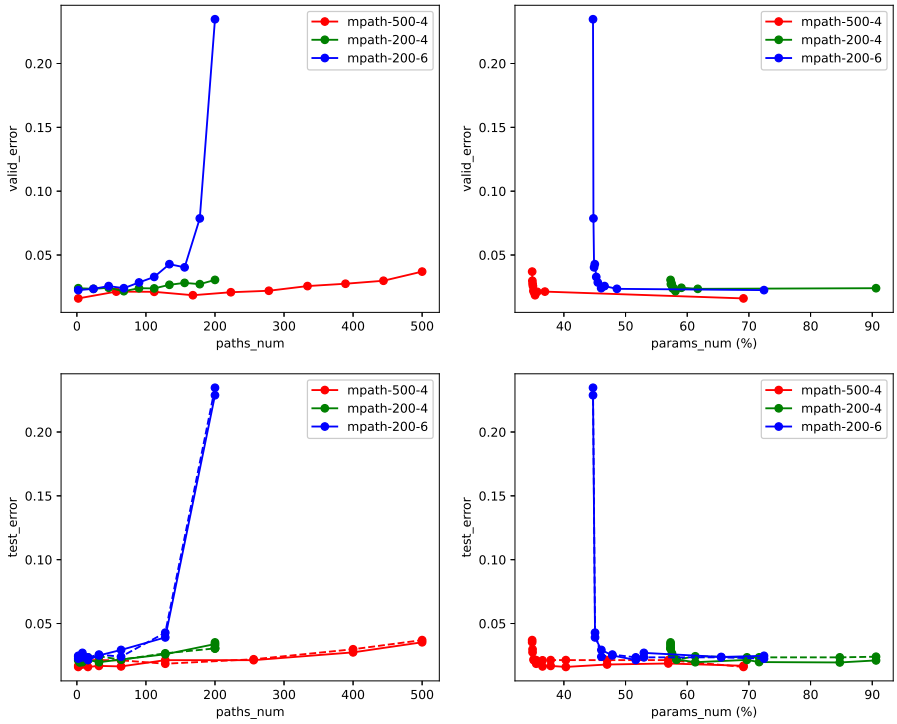


Figure 3.4. MNIST Sample Fine-tuning Phase. (Dotted curves represent the predicted validation error)

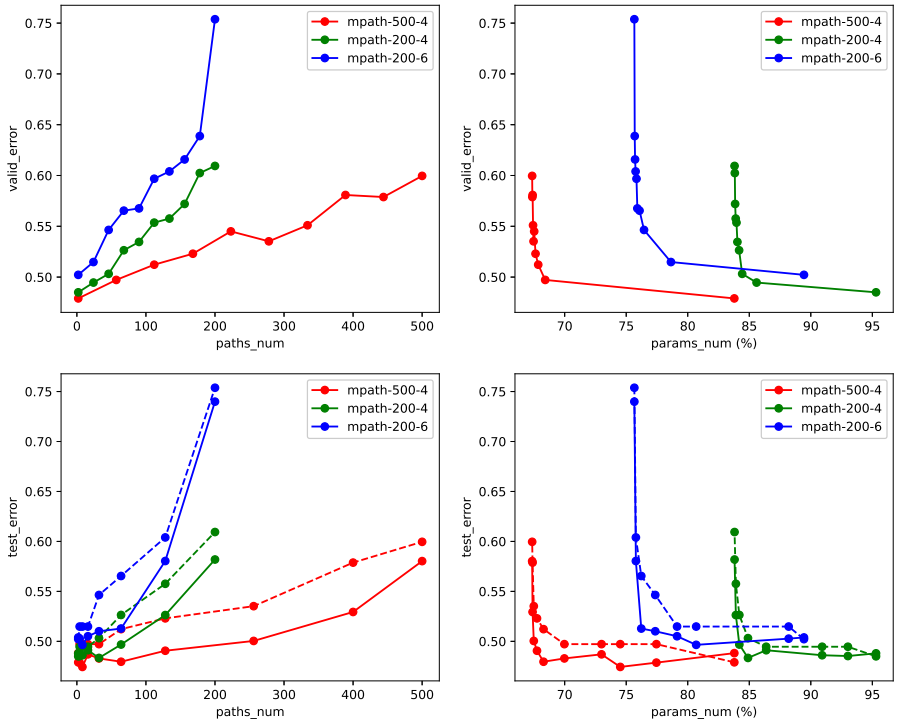


Figure 3.5. CIFAR10 Sample Fine-tuning Phase. (Dotted curves represent the predicted validation error)

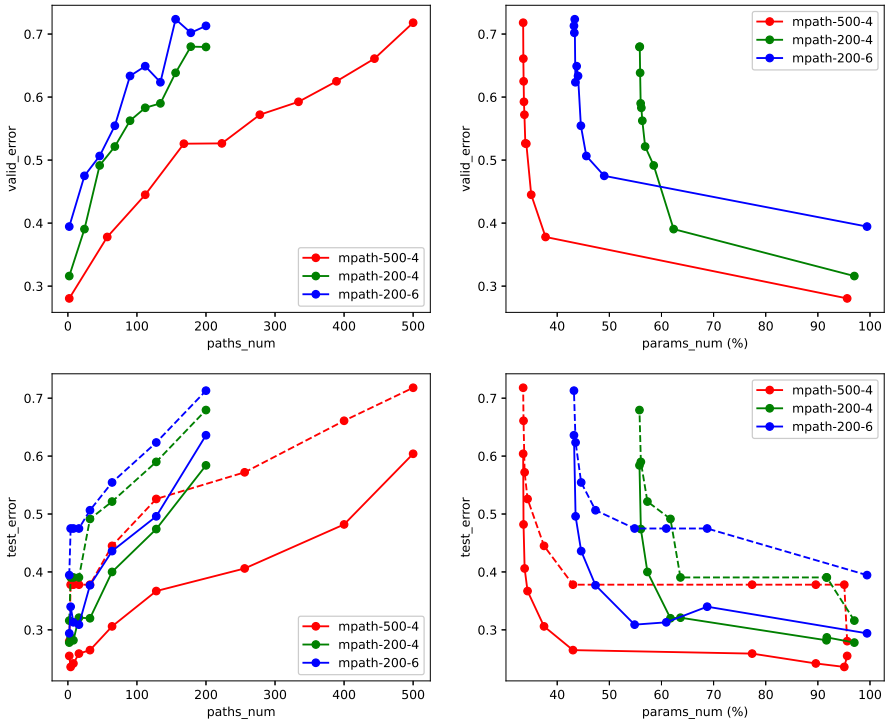


Figure 3.6. iWildCam2019 Sample Fine-tuning Phase.
(Dotted curves represent the predicted validation error)

The bottom left plot shows the actual test error as solid curves and the corresponding predicted validation error as dotted curves, both as a function of the path count. Note that we predict only the validation error of the path counts which we are going to calculate their test error. This means that while the big dots on the dotted curve are actual predictions from the regression model, the dotted line segments are just a linear interpolation. The bottom right plot shows the same results but as a function of the percentage of parameters. The MAE and correlation between the actual test error and the predicted validation error are shown in Table 3.4. The MAE results show an overall small deviation between the predictions and the actual errors. The Pearson correlation results as well show a consistent high correlation between the same two sets. The correlation level of mpath-4-500 for CIFAR10 is relatively lower than the other correlation results but still a reasonable value for a good correlation (> 0.5).

Model	MAE			Pearson		
	MNIST	CIFAR10	iWild	MNIST	CIFAR10	iWild
mpath-4-500	0.003	0.02	0.1	0.91	0.57	0.92
mpath-4-200	0.003	0.02	0.1	0.91	0.97	0.95
mpath-6-200	0.003	0.02	0.1	1.0	0.98	0.98

Table 3.4
MAE and Pearson correlation

3.8 Discussion

The main idea behind our approach is using differential architectural search to rank the different divisions of an MpathNN. By subsequent pruning, a relation between parameter count and accuracy can be established and divisions can be made to meet the desired accuracy-latency requirements. Given a permissible range of latency and accuracy, a modeler can, without the need for training every potential model, meet the desired balance. Some trade-offs can not be realised due to limitations of the model and the capacity-accuracy relationship. In this case, the modeler can know beforehand and ranges can be adjusted to more realistic values.

The results of the one-shot training in Table 3.1, Table 3.2 and Table 3.3 show performance close to equivalent MLPs. The results of equivalent MLPs for MNIST and iWildCam2019 are slightly better than MpathNN. For CIFAR10, the performance of MpathNN is significantly better than MLP. This confirms that no serious under/overfitting is happening from introducing the permeability factors.

The MAE of the sample fine-tuning phase is small and consistent, showing decent approximation for different architectures and datasets and reliability as a proxy for balancing accuracy-latency requirements. Pearson correlation also shows consistency and decent high correlation for different architectures and datasets. This supports our conclusion of the ability of the method to approximate the relation between latency and accuracy based on the introduced permeability factors and the pruning based on the relative magnitude of factors.

We have used compact matrices with zero-entries in all the inference phases. Unlike training, inference is usually not very computationally demanding except for extremely large models like Transformers (Vaswani et al., 2017). However, many situations in modular research will need sparsity at training time. Most of the modular NAS techniques will avoid any irregularity by sticking to a regular search space; by limiting their search space, a lot of potential for discovering more innovative and diverse architectures is wasted. Hence, finding efficient optimizations for sparse matrices is a line of research that we think is very important to opening the potential of research in modular architectures.

3.9 Conclusion

In this chapter, we investigated using NAS to modularize a fully-connected network into sparser MpathNN architectures targeted at use cases with limitations of resources. By modeling the search using the metaphoric membrane permeability, we could use a one-shot model to find divisions of multiple paths that could be used efficiently to predict accuracy-latency relationships. Hence, an efficient balancing between the available resources and the desired accuracy threshold can be realised. We showed that the proposed method has good predictive power and can replace exhaustive search.

3.10 Chapter Acknowledgements

This work was partially supported by a grant from Microsoft's AI for Earth program.

Chapter 4

Path Capsule Networks

An adaptation of this material has been published in Amer and Maul (2020).

4.1 Preface

The CNN (Fukushima and Miyake, 1980; LeCun and Bengio, 1995) was a successful architectural innovation that played an important rule in popularizing ANNs after big data and cheap powerful computing became available (Krizhevsky, Sutskever, and Hinton, 2012; Goodfellow et al., 2016). Building on the structure of the visual domain and previous research on animal visual cortex (Hubel and Wiesel, 1968), convolutions were introduced as shared weight kernels that can be trained to identify recurring structures and primitive shapes in images and compose them in a hierarchical way to represent higher-level abstract visual concepts. Augmented with pooling operations, CNNs could achieve translation-invariant representations of images and visual data. That is, shifting an object to a different location in the image still leads to similar higher-level representations.

Despite CNNs wide success in practical applications and becoming the state-of-the-art in visual processing, translation invariance meant by definition that location data are lost in the higher layers of the CNN. This limitation stimulated a line of research in deep learning aimed at overcoming this problem using different location data augmenting techniques (Wang and Veksler, 2018; Tang et al., 2015; Ghafoorian et al., 2017). This approach to the location agnosticism problem, however, was focused on avoiding the core problem in CNNs by using different auxiliary tricks to alleviate the limitation. CapsNet (Sabour, Frosst, and Hinton, 2017) took a different approach to the problem targeted at replacing translation invariance by translation equivariance. The core of the solution is to replace point-wise activations by pose vectors, named capsules, such that even if the view point of the object changes, the pose vectors would change in a coordinated manner and, since the vector magnitudes are used to signify object presence, the network should still signal the object's existence.

The original variant of CapsNet in Sabour, Frosst, and Hinton (2017) is composed of 3 layers. The first layer is a normal convolutional layer with ReLU

activation. The second layer is the PrimaryCaps layer. It is basically a convolutional layer but with a specific rearrangement of the activations into a specific vector/capsule configuration and a special kind of squashing non-linearity applied to each capsule. These capsules are aggregated into larger sets called PrimaryCapsules. The final layer is the DigitCaps layer, which produces the output vectors signaling the existence or absence of different classes. The calculation of the DigitCaps layer involves the routing-by-agreement algorithm. In this routing mechanism, each capsule from the previous layer votes for each DigitCapsule and the output is calculated by a weighted sum of the votes, which is iteratively fine-tuned based on the agreement between the votes and the output. Despite the fact that CapsNet achieved decent performance, specially on datasets with explicit varying view points (LeCun, Huang, and Bottou, 2004), we identified two potential opportunities for enhancement that can be achieved using modularity. First, the first layer of CapsNet is a very wide layer that contains most of the network parameters. Second, the CapsNet architecture is very shallow when compared to the usual depth used in state-of-the-art CNNs. We hypothesized that we can exploit modularity to add depth to CapsNet such that we enhance the representation capacity of the network, while at the same time reducing the number of parameters significantly.

Since the second layer of CapsNet is composed of a given number of PrimaryCapsules, a natural choice of a modular architecture that we could exploit to add depth consists of a multipath architecture, one of the topologies discussed in Section 2.4.2. So, instead of producing the PrimaryCaps layer through rearrangement of a large number of feature maps obtained from the previous layer, we produced each PrimaryCapsule as the output of an independent deep path of convolutional layers. This is a manual formation similar to the manual techniques discussed in Section 2.4.3. The integration of capsules into a final output is an AL integration similar to the original paper and like what was discussed in Section 2.4.4 and Section 2.5. Since these paths collectively are sparser than a fully connected set of layers, we could cut parameters number significantly. Also, since these paths have deeper architecture, and by exploiting max-pooling to make the computations more efficient and using Dropcircuit (Phan et al., 2018) as a regularizer and a modified variant of the routing algorithm, we could balance the decreased absolute capacity as measured by the number of parameters using the additional effective representational capacity. We did a RSA (Mehrer et al., 2020) and the results suggest that the increased representational capacity can be partially explained by the low correlation between PrimaryCapsules produced by the path independence.

In this chapter, we present and discuss how we modularized CapsNet to reduce the number of parameters and at the same time maintain the network accuracy. Besides the deep independent paths and the combined usage of max-pooling and Dropcircuit, we introduced another variant of the routing-by-agreement algorithm, namely fan-in routing. We show that fan-in routing

combined with the previously mentioned modular modifications to CapsNet could achieve better or similar results to CapsNet with a significant reduction in the number of parameters. Our modular multipath architecture can be readily parallelized and computationally distributed. However, the naive implementation as a set of independent modules/subnetworks is not efficient to run on a single GPU due to sparsity. Later in this chapter, we discuss in Section 4.6.1 and Section 4.7 how we could achieve an efficient single-GPU implementation.

4.2 Introduction

CNNs (Fukushima and Miyake, 1980; LeCun and Bengio, 1995) have remained state-of-the-art in image processing and computer vision tasks since their successful large scale training by Krizhevsky, Sutskever, and Hinton (2012). CNNs were biologically inspired by the visual cortex (Hubel and Wiesel, 1968) and were built on the principle of translation invariance, achieved through local receptive fields, weight sharing and pooling operations. Despite their success, CNNs suffer from inherent limitations, most significantly the fact that translation invariance by definition causes loss of location information. This limitation has stimulated a lot of research in the direction of augmenting learning with location data (Wang and Veksler, 2018; Tang et al., 2015; Ghafoorian et al., 2017).

Sabour, Frosst, and Hinton (2017) argued that the main limitation of CNNs is the focus on achieving translation invariance, and that equivariance should also be targeted. Hence, the authors proposed CapsNet as a step towards achieving equivariance. The philosophy of CapsNet is that a single activation/feature should be replaced by a pose vector, named capsule, representing the different properties of an object's viewpoint. CapsNet has two main components, which are PrimaryCapsule and DigitCaps layers. PrimaryCapsules represent the different parts of the underlying objects, which are then multiplied by translation matrices to get prediction vectors, representing the votes of each PrimaryCapsule with respect to each DigitCaps, which are then routed using routing by agreement to compute DigitCaps activations, which can then be used to signify the presence of an object. The philosophy is that with changing the viewpoint of an object, the change in pose matrices should be coordinated, such that the voting agreement is maintained. We consider using another form of routing by agreement, fan-in routing in contrast to fan-out routing used by Sabour, Frosst, and Hinton (2017), which we show can have better performance under some conditions.

CapsNet was shown to achieve very good results with a shallow architecture and decent parameter savings, compared with deep CNNs. However, the lack of depth can be limiting to the expressiveness of the network. Moreover, the first convolutional layer in CapsNet is large and contributes to increasing the number of CapsNet parameters significantly. We believe that a coordinated inclusion of

depth and multiple pathways can help increase the network performance and simultaneously help save more parameters.

We consider a multipath architecture for including more depth into CapsNet. Multiple paths in neural networks are biologically plausible and biological neural networks have been shown to exhibit multipath parallel processing (Gollisch and Meister, 2010; Otsuna, Shinomiya, and Ito, 2014). Aside from biological inspiration, we think that using different paths for generating PrimaryCapsules can be exploited to enhance performance while saving parameters significantly. A PrimaryCapsule generated by a deep path can be considered a deep version of the original CapsNet capsule, which we believe can exhibit more expressiveness and more abstraction, similar to other deep structures in the deep learning paradigm.

The universal approximation theorem by Hornik, Stinchcombe, and White (1990) showed that any Borel measurable function can be approximated by a sufficiently wide single layer MLP. Empirically, however, this is infeasible due to optimization limitations, and is rarely desirable due to the problem of overfitting. On the other hand, making use of depth is statistically motivated by composition of functions and empirically can lead to better generalization. Moreover, as we show, depth can be added judiciously to save parameters without sacrificing performance.

Our contributions in this paper are:

1. We propose PathCapsNet, a multipath deep version of CapsNet.
2. We enrich the routing by agreement methodology by a new variant, fan-in routing.
3. By carefully adding depth and max-pooling, along with a multi-path structure, fan-in routing and DropCircuit, we achieved better or comparable results to CapsNet with significant parameter savings.
4. We open the possibility of leveraging significant model parallelism in the context a capsule networks.

In the next section we discuss the previous work done around capsule networks and multipath architectures, and how we enhance by building on these concepts.

4.3 Capsule Network

CapsNet (Sabour, Frosst, and Hinton, 2017) was introduced as an architecture that builds up on the conventional CNN (Fukushima and Miyake, 1980; LeCun et al., 1989) trying to overcome its limitations. The main motivation behind CapsNet is achieving equivariance, in addition to the invariance properties

already implemented by CNN. CapsNet could achieve a good generalization using relatively fewer parameters than deep CNNs (only 8.2M parameters for the MNIST model with reconstruction). Different variants have been introduced since the original CapsNet. Phaye et al. (2018) introduced DCNet as a dense version of capsule networks and DCNet++ by stacking multiple DCNets. In DCNet++, each DCNet in the stack produces its version of the PrimaryCapsule layer, which is then fed to the next DCNet in the stack. The final output is calculated based on both the output of each subnetwork and their concatenation. They also made some modifications to the decoder (reconstruction) subnetwork. DCNet++ achieved good generalization in relatively few epochs at the cost of using more parameters (13.4M).

Another variant is MS-CapsNet (Xiang et al., 2018). MS-CapsNet is composed of three successive modules. The first module is the feature extractor and it has two convolutional paths of depths 1 and 2 and a third path which is just a skip connection. Each path produces a PrimaryCapsule of different dimension. The second module is a capsule encoding and it is responsible for projecting the PrimaryCapsules to a common dimension and concatenating them. The third module, capsule dropout, is applied before routing and it is responsible for dropping random capsules as a way of regularization in a manner similar to dropout (Srivastava et al., 2014) and other similar techniques. Capsule dropout showed enhancement in performance relative to the non-dropout condition. MS-CapsNet could achieve better performance than the original CapsNet on FashionMNIST and CIFAR-10 with fewer parameters ($\sim$11M).

SECaps (He et al., 2018) is an adaptation of CapsNet to sequential tasks, specifically NLP. The word embeddings of single words are treated as PrimaryCapsules. Since the dynamic routing is not sequential in nature and does not respect order, the seq-caps layer is introduced. This layer is basically composed of a LSTM layer that is applied to a given sequence of the data as a series encoding, and then the output is dynamically routed in the conventional way to produce the output of the next layer. Multiple seq-caps layers can be stacked. Another module, the attention module, transforms the word embeddings, which are then concatenated with the seq-layer output. The final output is produced by an MLP subnetwork. SECaps was evaluated on multiple charge prediction datasets, achieving better performance than the state-of-the-art.

Siamese Capsule Network (SCN) (Neill, 2018) is the capsule version of the conventional siamese network. Neill (2018) introduced SCN as a face verification approach similar to DeepFace (Taigman et al., 2014). SCN is very similar in architecture to the original CapsNet. It has a convolutional layer, followed by the PrimaryCapsules layer and then a layer called Face Capsule layer, which is essentially similar to the DigitCaps layer. The final output is produced by a fully connected layer on top of the Face Capsule layer. SCN achieved good performance on different datasets with a smaller model, little preprocessing and less data.

Matrix capsules network was proposed by Hinton, Sabour, and Frosst (2018) as a generalization of the original CapsNet for more efficient pose estimation. Each capsule is represented by a matrix and a sigmoid unit that controls the probability of activating the capsule. Every pose matrix is multiplied by a transformation matrix to get the votes which will be used for routing to the next layer. Routing is done using Expectation Maximization (EM) that takes as input the votes and activation probabilities of the previous layer. Matrix capsules network achieved a very good accuracy improvement on the smallNORB dataset, a dataset that is highly viewpoint variant, but it seems that it does not have the same advantage on MNIST.

4.4 Multipath Architectures

The ideas of branching, parallel computation and multiple paths are well established in the deep learning literature and have their supporting biological plausibility (Gollisch and Meister, 2010; Otsuna, Shinomiya, and Ito, 2014). In Cireşan, Meier, and Schmidhuber (2012), each path in a multi-path CNN is trained on a different preprocessing/distortion of the input image and the columns outputs are averaged to produce the final output. A similar approach is used in Wang (2015), but with different types of inputs which are the source image and a bilateral filtered version of it, and the outputs of the paths are integrated using fully connected layers. Similarly, in Yu et al. (2019a) and in the context of image captioning, multi-view feature learning is achieved using several detectors applied in parallel. The same idea of parallel detectors is also used in Zhang, Yu, and Tao (2018) in the context of unsupervised dimensionality reduction. Zhang, Yu, and Tao (2018) use a set of local contractive autoencoders to extract representations of the neighbourhood of each sample, and then apply local affine transforms to align the local features with a global coordinate system. For extracting hierarchical features in the context of predicting the click feature vector of images, Yu et al. (2019b) apply a parallel structure to extract features of different granularity from the word and image modalities. In the context of HPR, Hong et al. (2015) and Hong et al. (2019) utilize parallel structures in different ways. Hong et al. (2015) use two parallel autoencoders to extract 2D and 3D from different modalities and then learn the mapping from 2D to 3D features. On the other hand, Hong et al. (2019) use a model that takes different modalities as inputs and then produce different outputs using a branching structure, where each output corresponds to a different reconstruction task of a specific modality.

Szegedy et al. (2015b) proposed the Inception-v1 model , which was responsible for winning ILSVRC-14, and is composed of a highly branched multipath architecture. Szegedy et al. (2016) further improved the design of Inception-v1 to produce Inception-v2&3 which exploit large scale branching and multiple

paths even more. The Xception architecture (Chollet, 2016) is a further extension to the Inception family, that uses more branching based on separable convolutions. ResNetXt (Xie et al., 2016) and Residual Inception (Zhang et al., 2018) are extensions of ResNet (He et al., 2016) where the modular block is multipath instead of single path. FractalNet (Larsson, Maire, and Shakhnarovich, 2016) is another type of architecture that has a recursive self-similar, highly branched structure. Parallel circuit networks, introduced by Phan et al. (2016), adopt an extensively multipath architecture, and have demonstrated generalization improvements using a dropping technique called DropCircuit (Phan et al., 2018). Related to the DropCircuit technique is the path dropout used by Bender et al. (2018) to regularize the training of a one-shot model, which is an implicit form of a multipath network, where a whole space of possible branches is trained simultaneously.

We build on previous work by:

1. Adding representational power to PrimaryCapsules by generating each capsule using a deep path.

2. Enriching dynamic routing by agreement with a new fan-in variant.

3. Combining depth, a multipath architecture, DropCircuit, max-pooling and fan-in routing to obtain a level of performance congruent with the original CapsNet, with significant parameter savings.

4. Showing that max-pooling is not inherently contradictory with the CapsNet philosophy, and that it can be used to save parameters significantly without sacrificing neither performance nor pose awareness.

5. Opening a new illuminating perspective on the potential diversity of the representations learned by multipath architectures using RSA-analysis (Mehrer et al., 2020).

In the next section, we explain the general PathCapsNet architecture and the different pieces that contribute to its performance.

4.5 Methods

The original CapsNet (Sabour, Frosst, and Hinton, 2017) has two main capsule types, namely the PrimaryCapsules and the DigitCaps. PrimaryCapsules are formed by applying an initial convolution layer to produce 256 channels, then another set of convolutions, which are then rearranged into 32 8D PrimaryCapsules. PrimaryCapsules are then routed to the next DigitCaps layer using dynamic routing by agreement. In one variant of CapsNet, namely CapsNet with reconstruction, a reconstruction layer is learned on top of the DigitCaps layer

to facilitate the learning of instantiation (or transformation) parameters and therefore enhance generalization.

PathCapsNet Fig. 4.1 shares the upper part of CapsNet, starting from the PrimaryCapsules layer, through the DigitCaps layer and ending with a reconstruction layer if needed. However, PathCapsNet is fundamentally different in how the PrimaryCapsules are constructed. In PathCapsNet, each PrimaryCapsule is formed by a deep CNN, named a path. So, the input is fed into different CNNs (paths) and the output of each path comprises one PrimaryCapsule.

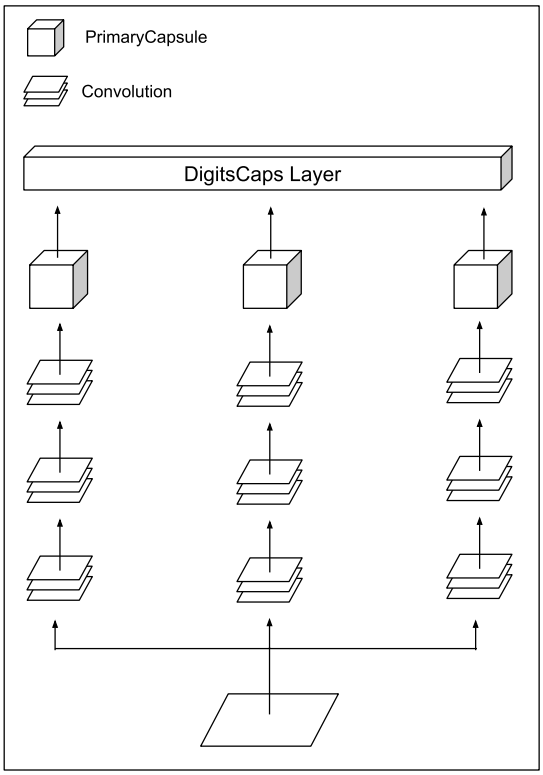


Figure 4.1. PathCapsNet architecture

The experiments done by Phan et al. (2018) demonstrated enhanced generalization in multipath MLPs, named parallel circuits in their work, using a drop technique called DropCircuit. DropCircuit is an adaptation of dropout to multipath architectures, where different paths are dropped during training, using a pre-specified probability. This is believed to enhance generalization by promoting independence between paths, hence allowing for problem decomposition and learning more useful representations, similar to dropout (Srivastava et al., 2014) and related techniques.

Dynamic routing is the mechanism by which PrimaryCapsules are routed to DigitCaps capsules, such that similar votes from PrimaryCapsules contribute more strongly to the target DigitCaps. The dynamic routing by agreement algorithm used in (Sabour, Frosst, and Hinton, 2017) updates the contribution

of votes based on the similarity between the output DigitCaps and the prediction vector, representing the vote, using dot product as a measure of similarity. So, given the prediction vectors (votes) from the previous layer of capsules (PrimaryCapsule layer) $\hat{\boldsymbol{u}}_{j|i}$, where j is the index of the DigitCaps capsule and i is the index of a single capsule in the PrimaryCapsule layer, the output vector (DigitCaps) is calculated as,

$$\boldsymbol{s}_j = \sum_i c_{ij} \hat{\boldsymbol{u}}_{j|i} \tag{4.1}$$

where c_{ij} are the coupling coefficients weighting the contributions of different prediction vectors,

$$c_{ij}^{(fout)} = \frac{\exp(b_{ij})}{\sum_k \exp(b_{ik})} \tag{4.2}$$

and b_{ij} is the log probability (logits) that the ith PrimaryCapsule should be coupled to the jth DigitCaps capsule. We call this fan-out (fout) routing, since the weights of the contributions of the ith PrimaryCapsule to each DigitCaps capsule in the next layer are normalized probabilities that sum to 1.0. For PathCapsNet, we used a different form of dynamic routing by agreement, named fan-in (fin) routing, where logits are normalized such that the weights of the contributions to the jth DigitCaps capsule from all the PrimaryCapsules are normalized probabilities that sum to 1.0. Coupling coefficients for fan-in routing are calculated as,

$$c_{ij}^{(fin)} = \frac{\exp(b_{ij})}{\sum_k \exp(b_{ki})} \tag{4.3}$$

Figure 4.2 shows a simplified diagram highlighting the difference in Softmax calculation direction between fan-out and fan-in variants. All connections with similar line pattern are inputs to the same Softmax. Note how the Softmax is applied across connections fanning into the same DigitsCaps in the fan-in variant, while it is applied across connections fanning out from a single PrimaryCapsule in the fan-out variant.

The accuracy and reconstruction losses are calculated the same way as (Sabour, Frosst, and Hinton, 2017), using margin loss and sum of squared errors loss, respectively.

In the next section, we present the details of our experimental design and the results we obtained.

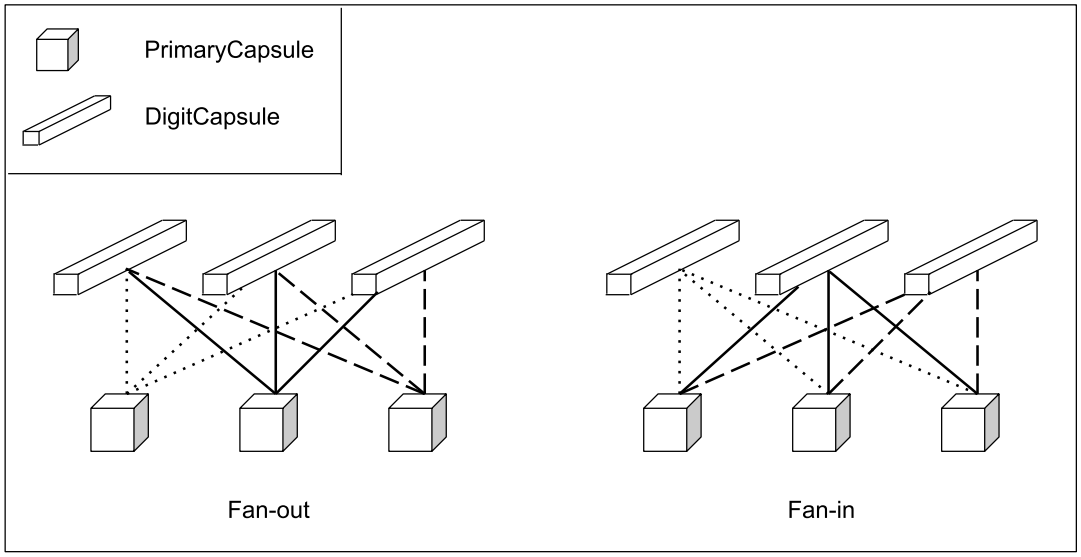


Figure 4.2. A simplified diagram showing the main difference in Softmax direction between fan-out and fan-in routing. Connections with similar lines are inputs to the same Softmax. Note that the other operations between PrimaryCapsules and DigitsCaps layers are abridged for clarity.

4.6 Results

4.6.1 PathCapsNet Architecture

For all of our experiments, each path had the same architecture as in Table 4.1. The number of paths in each experiment varied and will be clarified for each set of experimental results. We will use the notation PathCapsNet-[num], where [num] is replaced by the number of paths, so PathCapsNet-5 is PathCapsNet with 5 paths. All PrimaryCapsules were 8D with spatial dimensions 7x7. As each path produces one PrimaryCapsule, the number of PrimaryCapsules is equal to the number of paths. The DigitCaps layer was exactly the same as Sabour, Frosst, and Hinton (2017). We used 3 routing iterations in all the experiments and whenever we used fan-in routing, we initialized the transformation matrices of the DigitCaps layer randomly from a standard normal distribution. The Adam optimizer was used in all of the experiments using the default parameters and learning rate. When DropCircuit was used, the probability of path dropping was 0.5. Our benchmark was the original CapsNet (Sabour, Frosst, and Hinton, 2017) with and without reconstruction and using 3 routing iterations. The benchmark was implemented using the same architecture as reported in the original paper without any modifications, unless otherwise specified. All reported results are based on an average of three trials. Each trial is 300 epochs of training which is in the range of the reasonable values used in the literature

(Section 1.11) as compared to the number used by Sabour, Frosst, and Hinton (2017), which seems to be more than 1000 epochs.

Layer	Type	Kernel	Padding	Stride	Output Channels
1	Conv	9	4	1	16
2	Conv	9	4	1	16
3	Maxpool	2	0	2	16
4	Conv	9	4	1	16
5	Conv	9	4	1	8
6	Maxpool	2	0	2	8

Table 4.1
Single path architecture

When we discussed the challenges that face the research in artificial modularity in Section 1.5, we discussed the problem of implementation efficiency. Accelerating ANNs using GPUs assumes dense matrices. Since modularity introduces considerable sparsity into the network architecture, accelerating MNNs using GPUs can be tricky. For the work in this chapter, we assume regular paths, i.e. having the same width and depth. This way we could feedforward through the paths in parallel by reducing the calculation of the convolutional layers at the same level in each path into a single dense convolutional operation.

To achieve that, we utilized the optimized depthwise-separable convolution operator that exists in most tensor libraries. In regular convolution, a single convolutional filter has a depth equal to the input channels count. This constraint is relaxed in depthwise-separable convolution and, hence a filter can span only a fraction of the input channels. We concatenated the channels of the inputs to the convolutional layers at the same level across the different paths as a single input. Similarly, we concatenated the filters at the same level of layers into a single group of filters. Then, by repeating input channels so that they align with the right set of filters, we could carry on the whole operation at each level of layers as a single depthwise-separable convolution.

A formal description of this idea for the convolutional case would need extra details about how a convolutional operation can be reduced to a dense matrix-matrix operation. Since the core idea is the same, we will avoid the extra complications that might be introduced by this and we will instead formally describe the analogous process for fully-connected paths, i.e. paths with fully-connected layers instead of convolutional layers. Extending to the convolutional case can be considered a special case of the fully-connected one. We denote the weight matrix that maps from layer $l-1$ to layer l in any path (b) by $\mathbf{M}_b \in R^{WxW}$, where W is the width of any path. Despite the fact that under the regularity assumption the input and output dimensions are both equal to W, we emphasize that the first dimension of $\mathbf{M}_b$ is the output dimension and the second is the input dimension to avoid any confusion later. Note that each

layer in each path will have its own weight matrix, however we omit any layer indexing for convenience and assume that all the matrices $\mathbf{M}_b$ are at the same layer level l. We also denote the input to layer l in path b by $\boldsymbol{x}_b \in R^{1xW}$. We first concatenate the weight matrices into the block matrix $\mathbf{M}$,

$$\mathbf{M} = \begin{bmatrix} \mathbf{M}_1 \\ \mathbf{M}_2 \\ \cdots \\ \mathbf{M}_B \end{bmatrix}$$

where B is the number of paths. Then, we define a block matrix $\mathbf{X}_b$ for each input $\boldsymbol{x}_b$ by repeating the vector $\boldsymbol{x}_b$ W times,

$$\mathbf{X}_b = \begin{bmatrix} \boldsymbol{x}_b^{(1)} \\ \\ \boldsymbol{x}_b^{(2)} \\ \\ \cdots \\ \\ \boldsymbol{x}_b^{(W)} \end{bmatrix}$$

Using the $\mathbf{X}_b$ matrices, we define the block matrix $\mathbf{X}$ as,

$$\mathbf{X} = \begin{bmatrix} \mathbf{X}_1 \\ \mathbf{X}_2 \\ \cdots \\ \mathbf{X}_B \end{bmatrix}$$

Then, we can carry out the feedforward computation by

$$\begin{aligned} \widehat{\mathbf{O}} &= \mathbf{M} \odot \mathbf{X} \\ \mathbf{O} &= \sum_j \widehat{\mathbf{O}}_{:,j} \end{aligned} \tag{4.4}$$

where $\odot$ is the elementwise multiplication operator and the indexing operator (:) means selecting all the elements across the indexed dimension. The $\mathbf{O}$ output will be a block matrix with the following format

$$\mathbf{O} = \begin{bmatrix} \boldsymbol{o}_1 \\ \boldsymbol{o}_2 \\ \cdots \\ \boldsymbol{o}_B \end{bmatrix}$$

where $\mathbf{O} \in R^{(W*B)x1}$, $*$ is the scalar multiplication operator and $\boldsymbol{o}_b \in R^{Wx1}$ is the output of layer l in path b. The previous calculation can be easily extended to be in a minibatch form.

The following three sub-sections will describe the results of our experiments, applying the mentioned model architecture on 3 different datasets, namely, MNIST, CIFAR10 and iWildCam2019. In the tables summarizing the results, the best test performance across all conditions was shown in bold and we marked the best test performance across PathCapsNet conditions by an asterisk. The fourth and last sub-section in the results, will present an analytical perspective on the learning characteristics of the proposed PathCapsNet model compared to CapsNet.

4.6.2 MNIST

We use the MNIST dataset as described in Section 1.11. Following Sabour, Frosst, and Hinton (2017), the only augmentation used during training was padding by 2 and random cropping using a 28x28 patch. Our performance results on MNIST are summarized in Table 4.2.

For the no-reconstruction setting, fan-in routing improved CapsNet test performance from 0.48% to 0.42%. A similar improvement was observed for PathCapsNet-5, where test performance improved from 0.54% to 0.47%. With DropCircuit, we observed no improvement for a small number of paths, i.e. PathCapsNet-5, while a significant improvement was observed for PathCapsNet-10, where the error improved from 0.52% to 0.42%, which had the same performance as the standard CapsNet with only 21% of the parameters. A regularization effect can be observed from the validation curves Fig. 4.3. For the reconstruction setting, both of CapsNet and PathCapsNet-16 with fan-in and DropCircuit had the best test performance, with PathCapsNet-16 having only 44% of the parameters. Combining DropCircuit and fan-out had the same or worse performance compared to the corresponding no-DropCircuit condition, everything else being equal. The reverse seemed to hold for DropCircuit when combined with fan-in, where the performance was better or the same.

4.6.3 CIFAR10

We benchmark on CIFAR10 following the description in Section 1.11. All the PathCapsNet variants for CIFAR10 had 16 paths. CIFAR10 results are summarized in Table 4.3.

For both the no-reconstruction and reconstruction conditions, fan-in routing improved CapsNet performance. All PathCapsNet conditions, except PathCapsNet with fan-out and DropCircuit, were better than CapsNet. The best performance was achieved by PathCapsNet with fan-in and DropCircuit, which was better than CapsNet by around 5%, using only 28% and 51% of the CapsNet parameters in the no-reconstruction and reconstruction settings, respectively. For

Architecture	Routing	Paths	DC	#params	params(%)	Test error (%)
No Reconstruction						
CapsNet	Fan-out	N/A	N/A	6.8M	100%	0.48±0.02
CapsNet	Fan-in	N/A	N/A	6.8M	100%	**0.42**±0.03
PathCapsNet	Fan-out	5	Yes	683K	10%	0.54±0.05
PathCapsNet	Fan-in	5	No	683K	10%	0.48±0.07
PathCapsNet	Fan-in	5	Yes	683K	10%	0.47±0.04
PathCapsNet	Fan-in	10	No	1.4M	21%	0.52±0.03
PathCapsNet	Fan-in	10	Yes	1.4M	21%	***0.42**±0.05
Reconstruction						
CapsNet	Fan-out	N/A	N/A	8.2M	100%	**0.35**±0.04
CapsNet	Fan-in	N/A	N/A	8.2M	100%	0.47±0.03
PathCapsNet	Fan-out	10	No	2.8M	34%	0.44±0.06
PathCapsNet	Fan-in	10	No	2.8M	34%	0.47±0.02
PathCapsNet	Fan-out	10	Yes	2.8M	34%	0.49±0.02
PathCapsNet	Fan-in	10	Yes	2.8M	34%	0.42±0.05
PathCapsNet	Fan-in	16	Yes	3.6M	44%	***0.38**±0.02

Table 4.2
MNIST results. DC: DropCircuit. #params: parameters count. params(%): percentage of parameters (relative to the baseline).

all conditions, everything else being equal, adding DropCircuit to PathCapsNet fan-out models worsened the performance, while it improved the performance for fan-in models.

4.6.4 iWildCam2019

We use iWildCam2019 as described in Section 1.11. All the PathCapsNet models had 16 paths. The results are summarized in Table 4.4.

For both of the no-reconstruction and reconstruction settings, and in contrast to the other datasets, CapsNet with fan-out was better than the corresponding fan-in condition. Combining DropCircuit with fan-out had the same or better performance compared to the corresponding no-DropCircuit condition. The reverse was true for fan-in. Both CapsNet with fan-out and PathCapsNet with fan-out and DropCircuit had the best test performance with PathCapsNet having only 26% and 38% of the CapsNet parameters in the no-reconstruction and reconstruction settings, respectively.

4.6.5 RSA Analysis

The main crucial difference between CapsNet and PathCapsNet is how the PrimaryCapsules are generated. In order to peek into the characteristics of the learned features at the PrimaryCapsules layer, we followed the RSA-based methods used in Mehrer et al. (2020). For a given model, either CapsNet or

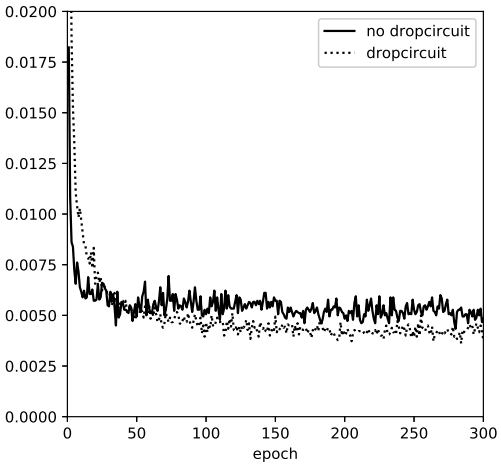


Figure 4.3. Average validation curves for PathCapsNet-10 with fan-in routing (MNIST)

PathCapsNet, we calculate the Representational Dissimilarity Matrices (RDMs) for each PrimaryCapsule. RDMs are calculated by presenting pairs of different samples to a given model (or a part of a model) and then calculating a measure of distance between the corresponding outputs. We use Euclidean distance as our distance measure. We then calculate the squared Pearson correlation (r^2) between each pair of the PrimaryCapsules using the RDMs. After taking the average of r^2 across PrimaryCapsules pairs, we use it as a measure of the correlation between PrimaryCapsules in the given model. We use a subset of 500 samples from the test set of each dataset for calculating the RDMs and the reported correlation for each model architecture is an average of three independently trained models.

Architecture	Routing	Paths	DC	#params	params(%)	Test error (%)
No Reconstruction						
CapsNet	Fan-out	N/A	N/A	8.0M	100%	35.6±0.4
CapsNet	Fan-in	N/A	N/A	8.0M	100%	32.7±1.1
PathCapsNet	Fan-out	16	No	2.2M	28%	31.3±1.2
PathCapsNet	Fan-in	16	No	2.2M	28%	31.7±0.3
PathCapsNet	Fan-out	16	Yes	2.2M	28%	35.6±1.4
PathCapsNet	Fan-in	16	Yes	2.2M	28%	***30.6**±0.7
Reconstruction						
CapsNet	Fan-out	N/A	N/A	11.7M	100%	35.5±0.8
CapsNet	Fan-in	N/A	N/A	11.7M	100%	32.9±1.2
PathCapsNet	Fan-out	16	No	6.0M	51%	30.9±0.4
PathCapsNet	Fan-in	16	No	6.0M	51%	32.2±0.9
PathCapsNet	Fan-out	16	Yes	6.0M	51%	35.2±1.5
PathCapsNet	Fan-in	16	Yes	6.0M	51%	***30.7**±0.7

Table 4.3
CIFAR10 results. DC: DropCircuit. #params: parameters count. params(%): percentage of parameters (relative to the baseline).

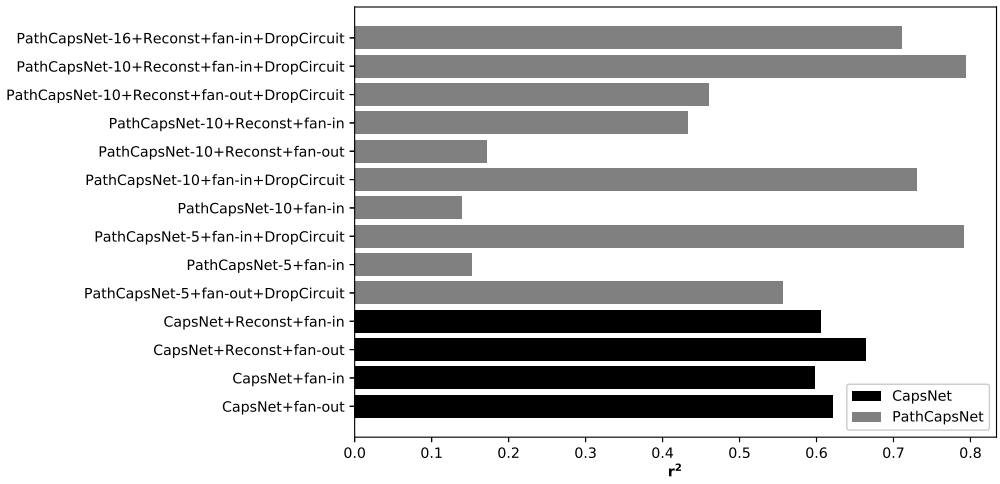


Figure 4.4. Squared Pearson Correlation (MNIST)

Architecture	Routing	Paths	DC	#params	params(%)	Test error (%)
No Reconstruction						
CapsNet	Fan-out	N/A	N/A	6.6M	100%	**21.6** ±0.8
CapsNet	Fan-in	N/A	N/A	6.6M	100%	23.8±0.3
PathCapsNet	Fan-out	16	No	1.7M	26%	22.5±0.5
PathCapsNet	Fan-in	16	No	1.7M	26%	24.3±1.6
PathCapsNet	Fan-out	16	Yes	1.7M	26%	*21.8(11)
PathCapsNet	Fan-in	16	Yes	1.7M	26%	24.3±0.5
Reconstruction						
CapsNet	Fan-out	N/A	N/A	8.0M	100%	**21.6**±0.6
CapsNet	Fan-in	N/A	N/A	8.0M	100%	23.1±0.3
PathCapsNet	Fan-out	16	No	3.0M	38%	22.1±1.0
PathCapsNet	Fan-in	16	No	3.0M	38%	22.3±0.6
PathCapsNet	Fan-out	16	Yes	3.0M	38%	*21.9(9)
PathCapsNet	Fan-in	16	Yes	3.0M	38%	24.1±1.0

Table 4.4
iWildCam2019 results. DC: DropCircuit. #params: parameters count. params(%): percentage of parameters (relative to the baseline).

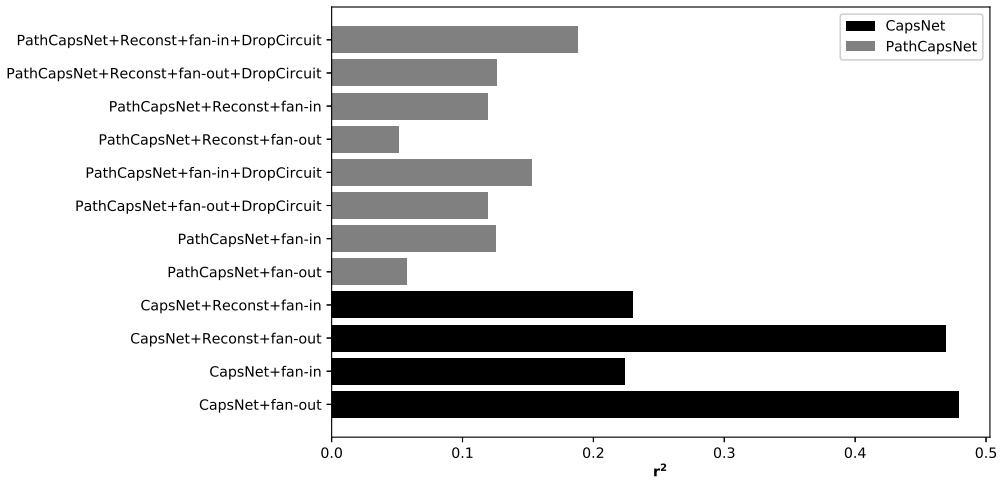


Figure 4.5. Squared Pearson Correlation (CIFAR10)

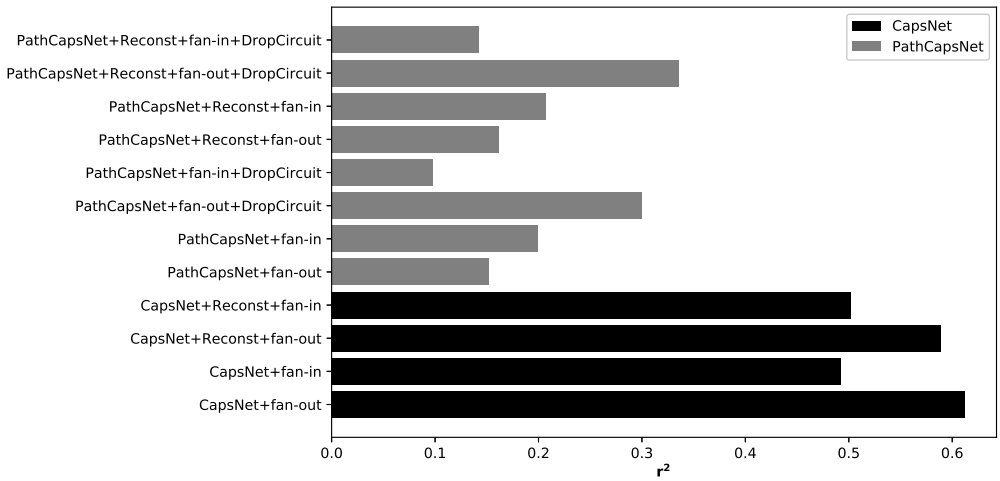


Figure 4.6. Squared Pearson Correlation (iWildCam2019)

For the MNIST dataset Fig. 4.4, CapsNet models showed a consistent high correlation between PrimaryCapsules. In contrast, PathCapsNet conditions did not show a consistent pattern of correlation. Some PathCapsNet conditions showed very low correlation, while others showed high correlation, sometimes higher than CapsNet. For CIFAR10 and iWildCam2019 Figs. 4.5 and 4.6, CapsNet still showed a consistent high correlation. However, in contrast to MNIST, all the PathCapsNet conditions showed a consistent low correlation.

In the next section we discuss our interpretations and hypotheses explaining the different techniques that contributed to these results and why we think that they enhance the current methodologies.

4.7 Discussion

Three main components, we believe, contributed to the performance of PathCapsNet, namely deep PrimaryCapsules, fan-in routing and DropCircuit. Deep paths, even without fan-in routing and no DropCircuit, could achieve a better or near test performance. We attribute this to the increased representational power of each PrimaryCapsule.

The main rationale behind fan-out routing was that each detected part of an object should contribute more strongly to a single object category rather than to multiple object categories. Fan-in routing, on the other hand, is expressing the idea that for a given object, different detected parts should contribute differently. Both philosophies can be seen to have different pros and cons, making each one optimal for a different set of contexts. This was supported empirically since fan-in achieved better performance on average in MNIST and CIFAR10, while fan-out was mostly better in iWildCam2019.

DropCircuit showed performance enhancement for conditions with large numbers of paths, and no significant effect for small numbers of paths. We believe DropCircuit, being a form of regularization, needs a sufficiently large number of paths to show a positive effect. With a small number of paths, dropping becomes too destructive, specially with a high dropping rate, to show any significant improvement. We believe DropCircuit, like other drop techniques, is introducing independence between paths and promoting the extraction of more useful PrimaryCapsule representations. This is partly supported by the RSA-analysis, which is further discussed later in this section.

A mutual interaction seems to exist between the routing mode (i.e. fan-out and fan-in) and DropCircuit. For MNIST and CIFAR10, where fan-in seems to improve performance over fan-out, DropCircuit enhances the performance of fan-in models, while it worsens fan-out models. For iWildCam2019, where fan-out is mostly better than fan-in, DropCircuit seems to endorse the same effect by enhancing the better performing mode (fan-out) and worsening the lower performing mode (fan-in).

We hypothesize that a contributing factor to PathCapsNet performance is the ability of the independent paths to extract diverse uncorrelated representations. This, however, seems to depend not only on the model architecture, but also on the dataset structure. RSA-analysis showed that while CapsNet had a consistent high correlation between its PrimaryCapsules in all the datasets, PathCapsNet showed a consistent low correlation in CIFAR10 and iWildCam2019. However, as mentioned this seems to be dataset dependent since in MNIST, and despite the fact that some PathCapsNet models showed low correlation, others showed higher correlation than CapsNet. Since this high PathCapsNet correlation seems to happen with DropCircuit, one potential explanation is that due to the similarity between MNIST classes, learning diverse features is difficult and enforcing independent learning through DropCircuit leads only to convergence of the learned features.

The modular redesign of CapsNet into PathCapsNet, combined with max-pooling, introduced a strong sparsity and large reduction in the parameter count. We could achieve a performance comparable to CapsNet by exploiting the modular multipath architecture to increase the network depth. Essentially, we substituted the wide convolutional layer of CapsNet with deeper narrower paths regularized by DropCircuit, a modular oriented regularization technique. Fan-in routing alone enhanced the performance in two out of the three datasets used. Moreover, it enabled the effective utilization of DropCircuit regularization in these cases, since, as discussed, the DropCircuit effect depends on the routing mode.

We showed that max-pooling, which may be considered incompatible with CapsNet and its equivariance aim, can be used to increase parameter savings without sacrificing performance. Moreover, experiments perturbing different dimensions in the DigitCaps layer learned on MNIST confirmed that, even when

using max-pooling, different pose parameters can be successfully learned. For example, as shown in Fig. 4.7 perturbing the first three dimensions of the DigitCaps layer of one of the models suggested that the first dimension in the model was controlling multiple pose parameters, like the elongation of the circular and linear regions and stroke thickness. The second dimension seemed to affect the circle axes orientations and also stroke thickness, while the third dimension resulted in a combination of vertical translation, vertical axis inclination and elongation of the circular part.

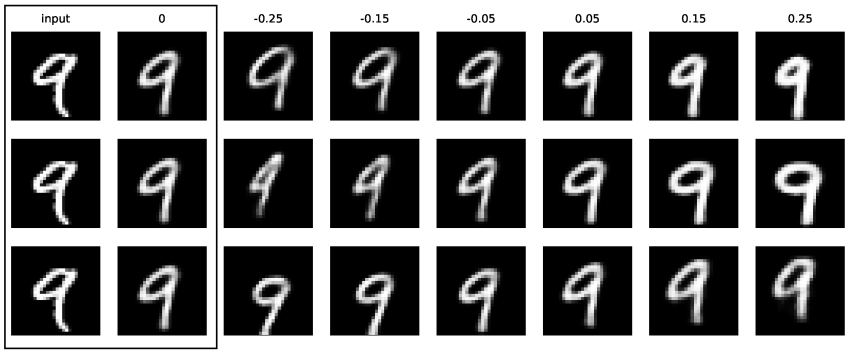


Figure 4.7. Perturbing different dimensions of PathCapsNet-10 (DropCircuit and fan-in) learned on MNIST. The images in the box are, from the left, the input and the unperturbed reconstruction, respectively.

Despite of the sparsity of PathCapsNet, we could achieve efficient implementation since the paths have the same width and depth. As we discussed, we did that by reducing the operations in layers that are located at the same level of each path into a single compact matrix operation. However, this technique is not feasible in the general case of heterogeneous paths having different depths and widths. Finding new ways of generalizing this efficiency to the general case of multipath architecture and further to arbitrary modular architectures would be of significant impact to the research in modularity.

4.8 Conclusion

We have introduced PathCapsNet, a modular sparse multipath version of CapsNet that can achieve better or comparable performance to its baseline with significant reduction in complexity. In order to achieve this, we used regularization by DropCircuit along with a new variant of dynamic routing by agreement, fan-in routing. The careful coordination of depth, max-pooling, routing mode

and DropCircuit allowed for maintaining, and sometimes even improving, CapsNet performance, while cutting down the parameter count considerably. RSA-analysis suggested that the independent multipath architecture of PathCapsNet has a diversifying effect on the learned representations. Reconstructions with perturbations showed that the use of max-pooling is not necessarily in conflict with retaining location information and pose estimation. The independence of paths renders the model suitable for model parallelism, a property which we did not investigate in detail and we leave for future work. We think there is still more space for enhancing PathCapsNet, specially in the reconstruction setting where we believe there is a complex interaction between routing, DropCircuit and reconstruction.

4.9 Chapter Acknowledgements

- We acknowledge the use of Athena at HPC Midlands+, which was funded by the EPSRC on grant EP/P020232/1, in this research, as part of the HPC Midlands+ consortium.
- This work was partially supported by a grant from Microsoft's AI for Earth program.

Chapter 5

Weight Map Layer for Noise and Adversarial Attack Robustness

5.1 Preface

Since their early days, it was recognized that neural networks NNs, like many other ML models in general, are susceptible to noise. Although artificial noise can be used as a method of regularization, generally noise in real-world data is an inevitable annoyance that arises from many sources in the stages of data acquisition and pre-processing. The sensitivity of NNs to noisy data is taken to its extreme through adversarial attack (Szegedy et al., 2013), an engineered noise that can severely disrupt the performance of NNs.

Since the discovery of adversarial attacks, there has been an ongoing research-oriented arms race between finding techniques to desensitize NNs to adversarial attacks, and potentiating adversarial attacks to overcome these techniques. Techniques for overcoming adversarial attacks, or adversarial defences, fall into two main categories: learning approaches and architectural approaches. Learning approaches aim to modify and enhance the learning process such that the trained network is less sensitive to adversarial attacks. Techniques such as augmenting the dataset with adversarial examples (Goodfellow, Shlens, and Szegedy, 2014; Jin, Dundar, and Culurciello, 2015; Seltzer, Yu, and Wang, 2013), distillation (Hinton, Vinyals, and Dean, 2015; Papernot et al., 2016; Papernot and McDaniel, 2017), feature squeezing (Xu, Evans, and Qi, 2017), NULL labeling (Hosseini et al., 2017), utilizing gradient and loss (Sinha et al., 2018) and using generative models (Pontes-Filho and Liwicki, 2018) fall into this category. Architectural techniques, on the other hand, are based on modifying the network architecture to introduce robustness to the attacks. Techniques based on autoencoders (Lamb et al., 2018; Ghosh, Losalka, and Black, 2018), sparsity and masking (Gao et al., 2017) and special layers (Sun, Ozay, and Okatani, 2017) fall into this category.

In principle, adversarial attacks can be applied to any kind of data. However, data that are readily perceptible by humans, like applications to visual and acoustic data (Seltzer, Yu, and Wang, 2013), are of special interest on the

theoretical and practical level. Hence, when it comes to the visual domain, desensitizing CNNs, being the SOTA models, to adversarial attacks is of a special importance. While the following point may not be frequently expressed, CNNs are in fact modular adaptations of feedforward networks, that were designed to exploit the inherent structure in visual data through introducing an inductive bias that is based on translation invariance. In its original formulation, a convolutional layer applies a set of independent filters to an input at different spatial locations. Hence, different filters can be considered different modules, even if they are simple modules of a single linear operation followed by a non-linearity.

In fact, this concept of modularity was generalized more by Lin, Chen, and Yan (2013) by replacing the filters by small feedforward networks. So, in principle, the most generalized form of a convolutional layer is composed of a set of modules applied at different spatial locations of the input to produce feature maps. In this chapter, we introduce the WM layer, a layer that is similar to and inspired by the convolutional layer. Both layers are a variant of the modular node architecture discussed in Section 2.4.2. The formation in both layers is a manual formation as discussed in Section 2.4.3 and the integration is AL as discussed in Section 2.4.4, since the module outputs are concatenated spatially to generate the feature map. We show that by inserting the WM layer in different SOTA NN architectures, we can increase their robustness to random noise, as well as adversarial attacks. The closest technique to ours is HyperNetworks with statistical filtering (Sun, Ozay, and Okatani, 2017). It depends on utilizing the channel-wise mean and standard deviation statistics to filter the weights of a traditional convolutional layer in order to render it more resistant to adversarial attacks. Our technique is significantly different in the methodology, and while it conceptually manipulates the input statistics, it performs that in an implicit way as we show in our discussion and interpretation.

In the following sections, we present our contributions to the problems of noise and adversarial attacks by introducing the WM layer. We benchmark different architectures and discuss our in depth analysis regarding the WM layer's dynamics and principle of action. The implementation efficiency issue that was discussed in Section 1.5 did not need a specific treatment in this architecture. Due to the relatedness to convolutional layers, the optimization needed is in fact reducible to the corresponding one needed in convolutional layers, which is well studied and supported in every modern tensor library.

5.2 Introduction

Despite their wide adoption in vision tasks and practical applications, CNNs (Fukushima and Miyake, 1980; LeCun et al., 1989; Krizhevsky, Sutskever, and Hinton, 2012) suffer from the same noise susceptibility problems manifested in the majority of neural network models. Noise is an integral component of any input signal that can arise from different sources, from sensors and data

acquisition to data preparation and pre-processing. Szegedy et al. (2013) opened the door to an extreme set of procedures that can manipulate this susceptibility by applying an engineered noise to confuse a neural network to misclassify its inputs.

The core principle in this set of techniques, called adversarial attacks, is to apply the least possible noise perturbation to the neural network input, such that the noisy input is not visually distinguishable from the original and yet it still disrupts the neural network output. Generally, adversarial attacks are composed of two main steps:

- **Direction sensitivity estimation:** In this step, the attacker estimates which directions in the input are the most sensitive to perturbation. In other words, the attacker finds which input features will cause the most degradation of the network performance when perturbed. The gradient of the loss with respect to the input can be used as a proxy of this estimate.
- **Perturbation selection:** Based on the sensitivity estimate, some perturbation is selected to balance the two competing objectives of being minimal and yet making the most disruption to the network output.

The above general technique implies having access to the attacked model and thus is termed a whitebox attack. Blackbox attacks on the other hand assume no access to the target model and usually entail training a substitute model to approximate the target model and then applying the usual whitebox attack (Chakraborty et al., 2018). The effectiveness of this approach mainly depends on the assumption of the transferability between ML models (Papernot, McDaniel, and Goodfellow, 2016). Transferability means that adversarial attacks engineered on a given model can affect other models trained on the same or similar datasets, even when the models are of different architectures.

Since their introduction, a lot of research have been done to guard against these attacks. An adversarial defence is any technique that is aimed at reducing the effect of adversarial attacks on neural networks. This can be through detection, modification to the learning process, architectural modifications or a combination of these techniques. Our approach consists of an architectural modification that aims to be easily integrated into any existing convolutional neural network architecture.

The core hypothesis we base our approach on starts from the premise that the noise in an input is unavoidable and in practise is very difficult to separate from the signal effectively. Instead, if the network can adaptively amplify the features activation variance selectively in its representations based on their importance, then it can absorb the variation introduced by the noise and map the representations to the correct output. This means that if a feature is very important to the output calculation, then its activation and intrinsic noise should be adequately amplified at training time to allow the classification layers to be

robust to this feature's noisiness at inference time, since it is crucial to performance. In the context of CNNs, this kind of feature-wise amplification can be achieved by an adaptive elementwise scaling of feature maps.

We introduce the WM layer, which is an easy to implement layer composed of two main operations: elementwise scaling of feature maps by a learned weight grid of the same size, followed by a non-adaptive convolution reduction operation. We use two related operations in the two WM variants we introduce. The first variant, smoothing WM, uses a non-adaptive smoothing convolution filter of ones. The other variant, unsharp WM, adds an extra step to exploit the smoothed intermediate output of the first variant to implement an operation similar to unsharp mask filtering (Gonzalez and Woods, 2002). The motivation for the second variant was to decrease the over-smoothing effect produced by stacking multiple WM layers. Smoothing is known to reduce adversarial susceptibility (Xu, Evans, and Qi, 2017), however if done excessively this can negatively impact accuracy, which motivates the unsharp operation as a counter-measure to help control the trade-off between noise robustness and overall accuracy. We show and argue that the weight map component can increase robustness to noise by amplifying the noise during the training phase in an adaptive way based on feature importance and hence can help networks absorb noise more effectively. In a way, this can be thought of as implicit adversarial training (Goodfellow, Shlens, and Szegedy, 2014; Lyu, Huang, and Liang, 2015; Shaham, Yamada, and Negahban, 2015). We show that the two components, weight map and reduction operations, can give rise to robust CNNs that are resistant to uniform and adversarial noise.

5.3 Adversarial Attack

Since the intriguing discovery by Szegedy et al. (2013) that neural networks can be easily forced to misclassify their input by applying an imperceptible perturbation, many attempts have been made to fortify them against such attacks. These techniques are generally applied to either learning or architectural aspects of networks. Learning techniques modify the learning process to make the learned model resistant to adversarial attacks, and are usually architecture agnostic. Architectural techniques, on the other hand, make modifications to the architecture or use a specific form of architecture engineered to exhibit robustness to such attacks.

Goodfellow, Shlens, and Szegedy (2014) suggested adversarial training, where the neural network model is exposed to crafted adversarial examples during the training phase to allow the network to map adversarial examples to the right class. Tramèr et al. (2017) showed that this can be bypassed by a two step-attack, where a random step is applied before perturbation. Jin, Dundar, and Culurciello (2015) used a similar approach of training using noisy inputs, with some modifications to network operators to increase robustness to adversarial

attacks. Seltzer, Yu, and Wang (2013) also applied a similar technique in the audio domain, namely, multi-condition speech, where the network is trained on samples with different noise levels. They also benchmarked against training on pre-processed noise-suppressed features and noise-aware training, a technique where the input is augmented with noise estimates.

Distillation (Hinton, Vinyals, and Dean, 2015) was proposed initially as a way of transferring knowledge from a larger teacher network to a smaller student network. One of the tricks used to make distillation feasible was the usage of softmax with a temperature hyperparameter. Training the teacher network with a higher temperature has the effect of producing softer targets that can be utilized for training the student network. Papernot et al. (2016) and Papernot and McDaniel (2017) showed that distillation with a high temperature hyperparameter can render the network resistant to adversarial attacks. Feature squeezing (Xu, Evans, and Qi, 2017) corresponds to another set of techniques that rely on desensitizing the model to input, e.g. through smoothing images, so that it is more robust to adversarial attacks. This, however, decreases the model's accuracy. Hosseini et al. (2017) proposed NULL labeling, where the neural network is trained to reject inputs that are suspected to be adversarials.

Sinha et al. (2018) proposed using adversarial networks to train the target network using gradient reversal (Ganin et al., 2015). The adversarial network is trained to classify based on the loss derived gradient, so that the confusion between classes with similar gradients is decreased. Pontes-Filho and Liwicki (2018) proposed bidirectional learning, where the network is trained as a classifier and a generator, with an associated adversarial network, in two different directions and found that it renders the trained classifier more robust to adversarial attacks.

From the architectural family, Lamb et al. (2018) proposed inserting Denoising Autoencoders (DAEs) between hidden layers. They act as regularizers for different hidden layers, effectively correcting representations that deviate from the expected distribution. A related approach was proposed by Ghosh, Losalka, and Black (2018), where a Variational Autoencoder (VAE) was used with a mixture of Gaussians prior. The adversarial examples could be detected at inference time based on their high reconstruction errors and could then be correctly reclassified by optimizing for the latent vector that minimized the reconstruction error with respect to the input. DeepCloak (Gao et al., 2017) is another approach that accumulates the difference in activations between the adversarials and the seeds used to generate them at inference time and, based on this, a binary mask is inserted between hidden layers to zero out the features with the highest contribution to the adversarial problem. The nearest to our approach, is the method proposed by Sun, Ozay, and Okatani (2017). This work made use of a HyperNetwork (Ha, Dai, and Le, 2016) that receives the mean and standard deviation of the convolution layer and outputs a map that is multiplied elementwise with the convolution weights to produce the final weights used to

filter the input. The dependency of the weights on the statistics of the data renders the network robust to adversarial attacks.

We introduce the WM layer, an adversarial defence which requires a minimal architectural modification since it can be inserted between normal convolutional layers. We propose that the adaptive activation-variance amplification achieved by the layer, which can be considered as a form of dynamic implicit adversarial training, can render CNNs robust to different forms of noise. Finally, we show that the WM layer can be integrated into scaled up models to achieve noise robustness with the same or similar accuracy in many cases across different datasets.

5.4 Methods

The main operation involved in any variant of a weight map layer (Fig. 5.1 and Fig. 5.2) is an elementwise multiplication of the layer input with a map of weights. For a layer l with an input $\boldsymbol{x}_l \in R^{C_i \times D_i \times D_i}$ with C_i input channels and spatial dimension D_i and an output $\boldsymbol{o}_l \in R^{C_o \times D_o \times D_o}$ with C_o output channels and D_o spatial dimension, the channel map of the c_ith input channel contributing to the c_oth output channel is calculated as

$$\boldsymbol{m}_l^{(c_i,c_o)} = \mathbf{W}_l^{(c_i,c_o)} \odot \boldsymbol{x}_l^{(c_i)} \tag{5.1}$$

where $\mathbf{W}_l^{(c_i,c_o)} \in R^{D_i \times D_i}$ is the weight mapping between c_i and c_o, $\boldsymbol{x}_l^{(c_i)}$ is the c_ith input channel and $\odot$ is the elementwise multiplication operator. We used two techniques for producing the pre-nonlinearity output of the weight map layer. The first variant, smoothing weight map layer, produces the c_oth output channel $\boldsymbol{o}_l^{(c_o)}$ by convolving the maps with a kernel $\boldsymbol{k} \in R^{C_i \times D_k \times D_k}$ of ones with D_k spatial dimension as follow,

$$\boldsymbol{o}_l^{(c_o)} = \boldsymbol{m}_l^{(c_o)} * \boldsymbol{k} + \boldsymbol{b}_l^{(c_o)} \tag{5.2}$$

where $\boldsymbol{m}_l^{(c_o)}$ is the set of intermediate maps contributing to output channel c_o, $\boldsymbol{b}_l^{(c_o)} \in R^{D_o \times D_o}$ is a bias term and $*$ is the convolution operator. The other variant, unsharp weight map layer, produces the output by an operation similar to unsharp filtering as follow,

$$\boldsymbol{s}_l^{(c_i,c_o)} = 2\boldsymbol{m}_l^{(c_i,c_o)} - \boldsymbol{m}_l^{(c_i,c_o)} * \boldsymbol{k} \tag{5.3}$$

$$\boldsymbol{o}_l^{(c_o)} = \sum_{c_i} \boldsymbol{s}_l^{(c_i,c_o)} + \boldsymbol{b}_l^{(c_o)} \tag{5.4}$$

where $\boldsymbol{k} \in R^{D_k \times D_k}$ is a kernel of ones applied with a suitable padding element to ensure similar spatial dimensions between the convolution input and output.

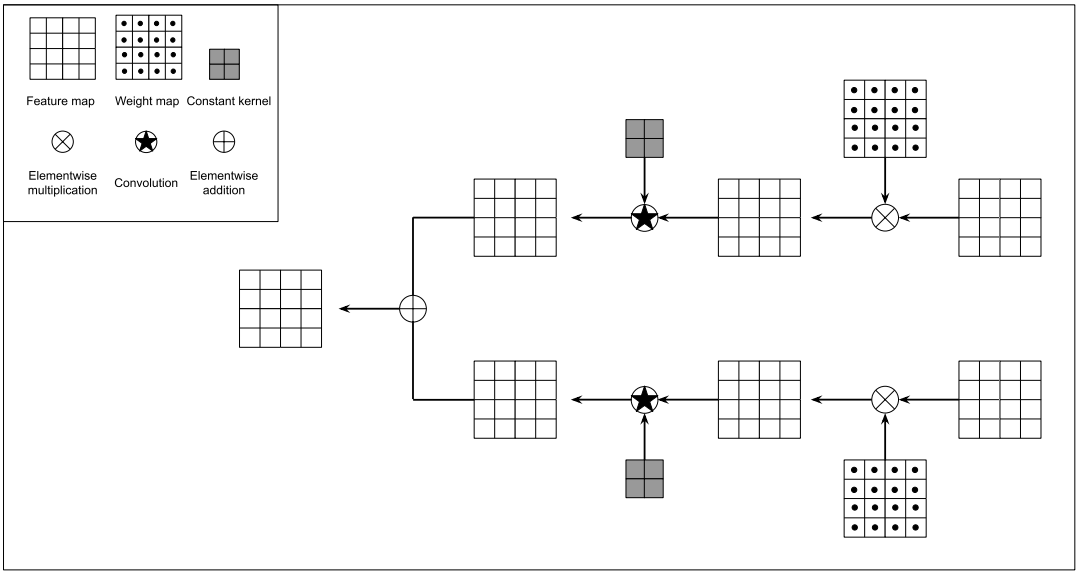


Figure 5.1. Smoothing weight map layer.

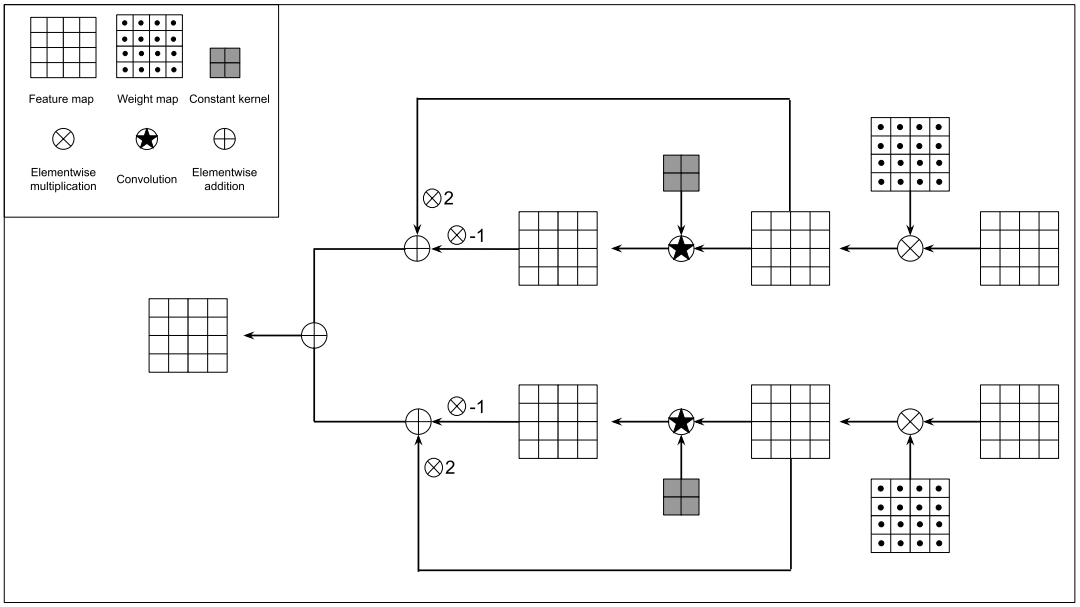


Figure 5.2. Unsharp weight map layer.

5.5 Experiments

We tested our method on three architectures and three datasets, namely MNIST, CIFAR10 and iWildCam2019. For every dataset, we benchmark a number of

weight-map layer variants against a baseline. The baseline and the variants share the same skeleton, but the layers in the baseline are convolutional layers, while in the different weight map variants some or all of the convolutional layers are replaced with a variant of the weight map layer. We train each model for 300 epochs using the Adam optimizer. The test error reported on the non-noisy dataset is an average of three trials. For the weight map layer, we initialize the weights and biases according to

$$w_{ij} \sim U(-s, s)$$
$$s = \frac{1}{\sqrt{a_{in}}}$$

where U is the uniform distribution and a_{in} is the fan-in.

Layer	CNN	CNN (wide)	WM
1	Conv(33 channels)	Conv(200 channels)	WM (32 channels)
2	Conv(33 channels)	Conv(500 channels)	WM (32 channels)
3	Conv(8 channels)	Conv(8 channels)	WM (8 channels)
4	FC(64 nodes)	FC(64 nodes)	FC(64 nodes)
5	FC(10 nodes)	FC(10 nodes)	FC(10 nodes)

Table 5.1
CNN variants basic skeletons

Layer	Out dimension	Repeat
ResBlock	8	3
ResBlock	16	4
ResBlock	32	6
ResBlock	64	4
Global average pool	64	1
Fully connected	10	1

Table 5.2
ResNet skeleton

Layer	Hyperparams	Repeat
Conv	channels: 16	1
ReLU	channels: 16	1
Dense	Growth rate: 8	2
Max pool	size:2, stride:2	1
Dense	Growth rate: 8	4
Max pool	size:2, stride:2	1
Dense	Growth rate: 8	8
Max pool	size:2, stride:2	1
Dense	Growth rate: 8	16
Global average pool	out: 256	1
Fully connected	out: 10	1

Table 5.3
DenseNet skeleton

In the experiments on the MNIST dataset, we tested three different architectures: CNN, ResNet (He et al., 2015) and DenseNet (Huang et al., 2016b) and for the two other datasets (CIFAR10 and iWildCam2019) we tested ResNet and DenseNet. The CNN skeleton is a stack of three layers, where the first two layers have either 32 channels, if it is a weight map network variant, or 33 channels if it is a normal CNN (Table 5.1). This difference was adopted to maintain approximately the same number of FLOPS between the two architectures. In one of the CNN variants, we increased the channels in the first two layers to 200 and 500, respectively, to compare with the weight map network having the same number of parameters. We will refer to this scaled up variant as "wide" in the results. The final layer in the skeleton body has 8 channels. Classification output is made by a 2 layer fully connected MLP, where the first layer has 64 nodes followed by an output layer. We fixed the kernel size across all the layers. We compared two kernel sizes, 3 and 9, and we included batchnorm (Ioffe and Szegedy, 2015) layers in some of the variants to test the interaction with the proposed layer. When batchnorm was included, it was inserted in the convolutional or WM layers just before the nonlinearity. In one of the variants, we multiplied the input with a learned weight map elementwise to probe the effect of the input weight map on noise robustness.

To assess the scalability of the proposed weight map layer, we integrated it into two popular CNN skeletons: ResNet and DenseNet. Table 5.2 shows the skeleton of the ResNet variant. ResBlock was composed of two layers of 3x3 convolutions with ReLU activations. At layer transitions characterized by doubling of the number of channels, downsampling to half of the spatial dimension was done by the first layer of the first block. Residual connections were established from the input to each ResBlock to its output, following the pattern used in the original paper (He et al., 2015), where projections using 1x1

convolutions were applied when there was a mismatch of the number of channels or spatial dimensions. Table 5.3 shows the skeleton of DenseNet. Each Dense layer is assumed to be followed by a ReLU nonlinearity. For integrating WM layers into the architectures, we either replace all the layers by one of the WM layer variants or replace half of the layers by skipping one layer and replacing the next. We will refer to the former by the non-alternating WM model and to the latter by the alternating WM model.

We tested the model robustness on two types of noise, uniform noise and adversarial noise. When testing models for uniform noise robustness, we added random uniform noise to the input, which always had a lower boundary of zero. We varied the upper boundary from values close to zero towards 1 in order to assess the degree of robustness. After the addition of the noise, the input was linearly renormalized to be within the range $[0, 1]$. The robustness measure is reported as the average test error achieved by the model on the noisy test dataset averaged over three trials. For testing the models against adversarial attacks, we followed the Fast Gradient Sign Method (FGSM) (Goodfellow, Shlens, and Szegedy, 2014) approach. Let the cost function used to train the network be $J(\boldsymbol{\theta}, \boldsymbol{x}, y)$ where $\boldsymbol{\theta}$ is the network parameters, $\boldsymbol{x}$ is the input and y is the input's label. Then following FGSM, the perturbation $\boldsymbol{\eta}$ to be added to the input can be obtained as follows,

$$\boldsymbol{\eta} = \epsilon * sign(\nabla_x J(\boldsymbol{\theta}, \boldsymbol{x}, y)) \tag{5.5}$$

where $\epsilon > 0$ is a scaling factor controlling the severity of the attack, $*$ is the multiplication operator and ∇ is the gradient operator.

The datasets used for benchmarking are described in Section 1.11. For MNIST, the only augmentation used during the training was padding by 2 and then random cropping a 28x28 patch. The results on MNIST are summarized in Table 5.5, Fig. 5.3 and Fig. 5.4. CIFAR10 results are summarized in Table 5.6 and Fig. 5.5. iWildCam2019 results are summarized in Table 5.7 and Fig. 5.6.

5.5.1 Results

The test errors of the preliminary experiments benchmarking CNN on MNIST are summarized in Table 5.4. The basic weight map network has better performance than the corresponding basic CNN with the same number of FLOPS. The unsharp version is better by a larger margin but with slightly higher FLOPS. Increasing the CNN parameters to the level of the corresponding weight map network results in lowering its performance. Including batchnorm in either the CNN or the weight map network boosted the performance of both variants to nearly the same level. On the other hand, increasing the kernel size to 9 boosted the CNN performance, whilst degrading the weight map network performance.

Arch	**Variant**	**Params**	**GFLOPS**	**Test error** (%)
CNN	wide	1.19M	1.07	1.0±0.09
CNN	33 channels	261K	0.014	0.85±0.11
CNN	33 channels - batchnorm	261K	0.014	0.73±0.10
CNN	33 channels - kernel size 9	361K	0.125	0.7±0.09
CNN	33 channels - input-scale	261K	0.014	0.86±0.01
Smooth WM	32 channels	1.16M	0.013	0.79±0.03
Smooth WM	32 channels - batchnorm	1.16M	0.013	0.7±0.07
Smooth WM	32 channels - kernel size 9	1.16M	0.119	0.88±0.04
Unsharp WM	32 channels	1.49M	0.020	0.73±0.03

Table 5.4
CNN results (MNIST)

Variant	**Test error** (%)
ResNet	
Conv	0.5±0.05
Smoothing WM	0.8±0.09
Unsharp WM	0.91±0.14
Alternating Conv/Smoothing WM	0.65±0.08
Alternating Conv/Unsharp WM	0.71±0.1
DenseNet	
Conv	0.52±0.09
Smoothing WM	0.67±0.05
Unsharp WM	0.6±0.04
Alternating Conv/Smoothing WM	0.55±0.07
Alternating Conv/Unsharp WM	0.54±0.04

Table 5.5
MNIST results

Variant	**Test error** (%)
ResNet	
Conv	32.14±2.2
Alternating Conv/Smoothing WM	39.31±0.7
Alternating Conv/Unsharp WM	38.49±0.9
DenseNet	
Conv	23.18±0.7
Alternating Conv/Smoothing WM	29.04±0.4
Alternating Conv/Unsharp WM	30.15±0.8

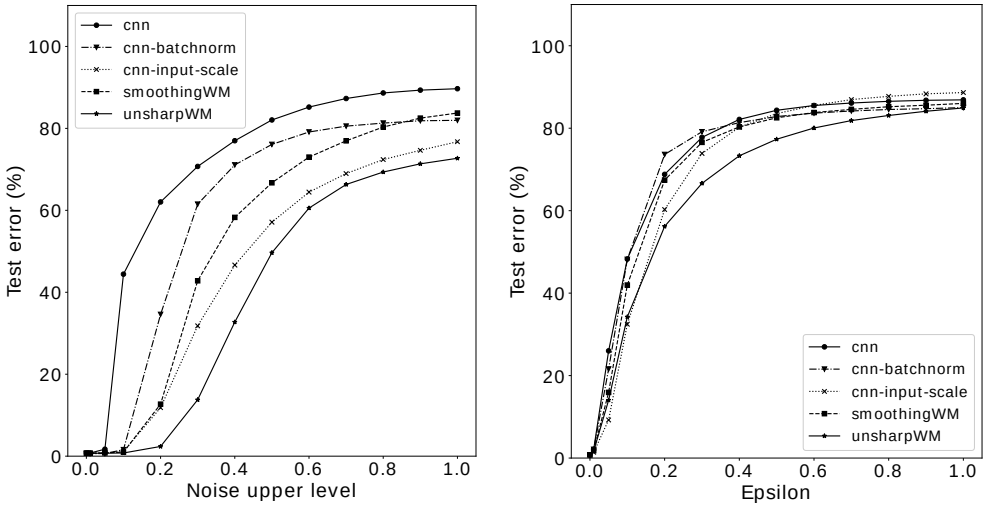
Table 5.6
CIFAR10 results

Variant	**Test error** (%)
ResNet	
Conv	23.3±1.0
Alternating Conv/Smoothing WM	20.43±0.8
Alternating Conv/Unsharp WM	20.4±0.4
DenseNet	
Conv	22.53±0.8
Alternating Conv/Smoothing WM	22.47±0.7
Alternating Conv/Unsharp WM	20.7±1.0

Table 5.7
iWildCam2019 results

The test error results of ResNet and DenseNet are summarized in Table 5.5, Table 5.6 and Table 5.7. Baseline ResNet had better performance than the weight map layer variants on MNIST and CIFAR10, while all of the weight map layer variants were better than the baseline on iWildCam2019. For DenseNet tested on MNIST, the baseline had better performance than the non-alternating weight map layer variants, while it had similar performance to the alternating variants. On iWildCam2019, the baseline DenseNet had similar performance to the alternating smoothing variant and lower performance than the alternating unsharp variant. On CIFAR10, the weight map variants had lower performance than the baseline DenseNet.

Robustness to noise results are shown in Fig. 5.3, Fig. 5.4, Fig. 5.5 and Fig. 5.6. For uniform noise, all of the WM variants tested on MNIST had better performance than the baseline, except for the non-alternating smoothing WM variant of ResNet. On CIFAR10, the alternating unsharp WM variant of ResNet showed lower performance than the baseline at lower noise. However, it becomes more robust than the baseline as the noise magnitude increases. All the



(a) Uniform noise **(b)** Adversarial noise

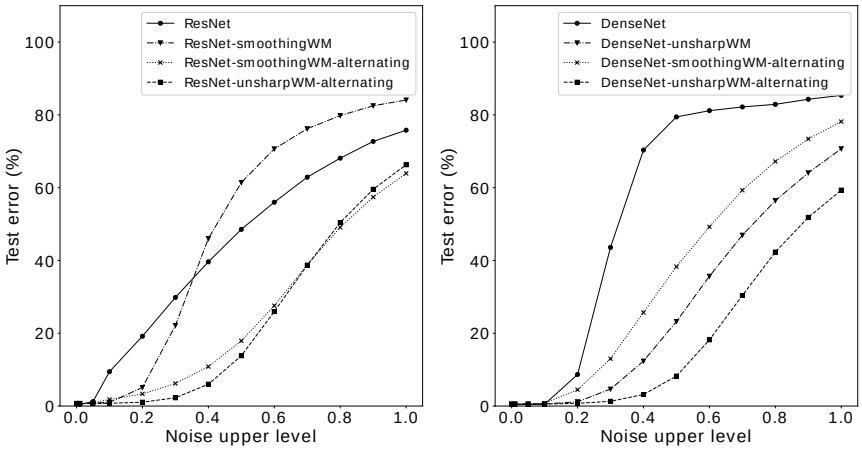
Figure 5.3. MNIST. CNN variants.

alternating WM variants of DenseNet showed a similar behaviour. However, the alternating smoothing WM variants of ResNet had less robustness at all noise levels. On iWildCam2019, all of the WM variants of ResNet were less robust than the baseline, while all the WM variants of DenseNet were more robust than the baseline.

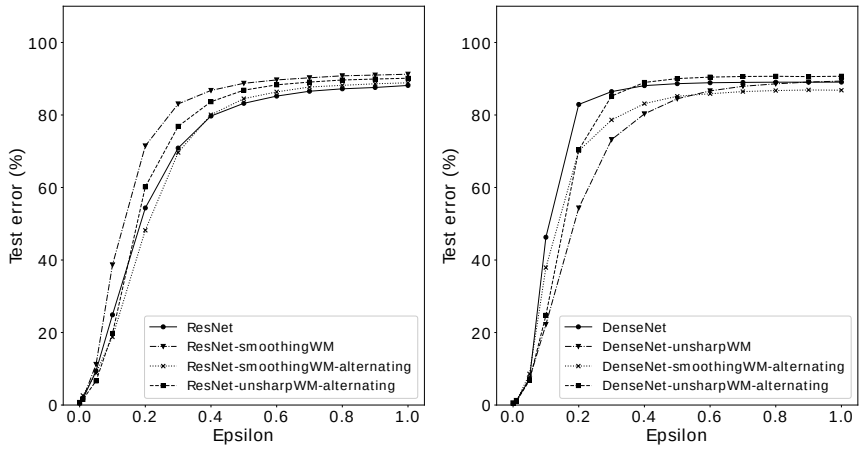
Regarding adversarial noise results on MNIST, the unsharp WM variant of CNN was more robust than the baseline, while the baseline and all the other WM variants of CNN had similar performance. For ResNet, all the WM variants had either less or the same robustness compared to the baseline. All the WM variants of DenseNet were better than the baseline, except the alternating unsharp WM model which had similar performance to the baseline. On CIFAR10, both the baselines and the WM models of ResNet performed poorly. However, on average baseline was better than WM variants of ResNet, while WM variants of DenseNet were better on average than their baseline. iWildCam2019 had similar results to CIFAR10, where all the models, including the baseline, performed poorly. All ResNet models on iWildCam2019 had similar performance, while for DenseNet models, the baseline was on average better than the WM variants.

5.6 Discussion

We have benchmarked two reduction operations in WM layers: smoothing and unsharp. The smoothing operation is effectively a moving average over a window equal to the kernel area. Since this will introduce blurriness into the input which will accumulate further on stacking multiple layers, we benchmarked

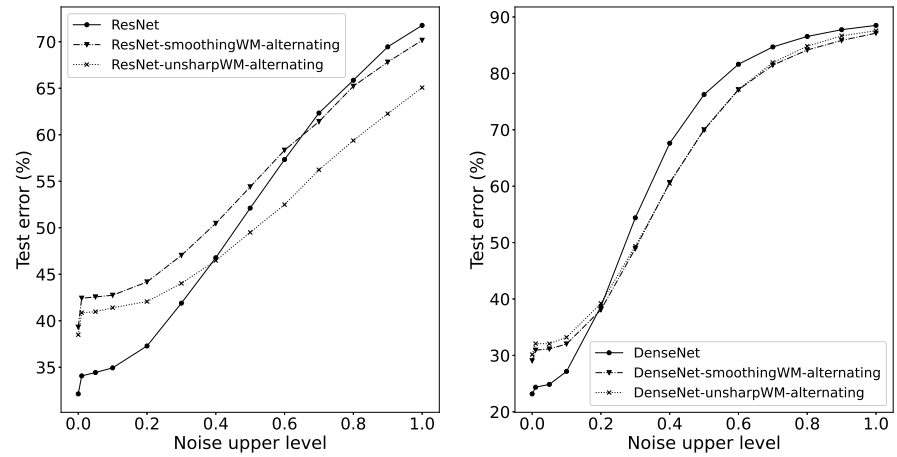


(a) Uniform noise

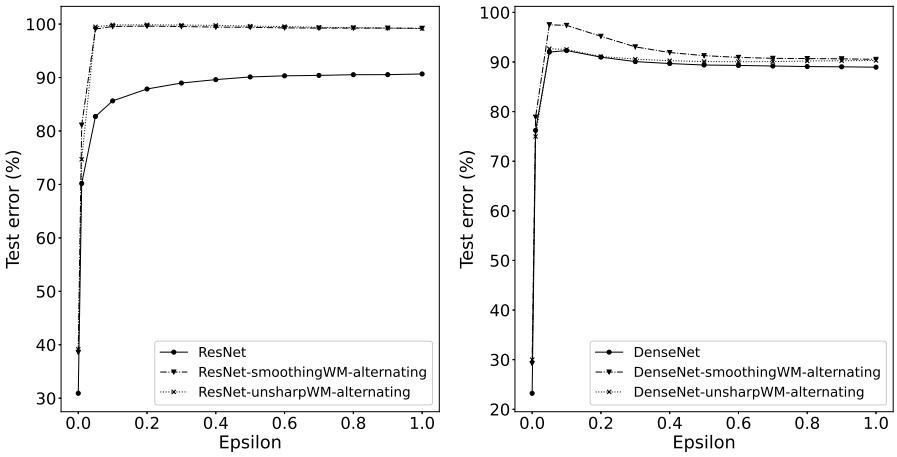


(b) Adversarial noise

Figure 5.4. MNIST. Left: ResNet variants. Right: DenseNet variants.

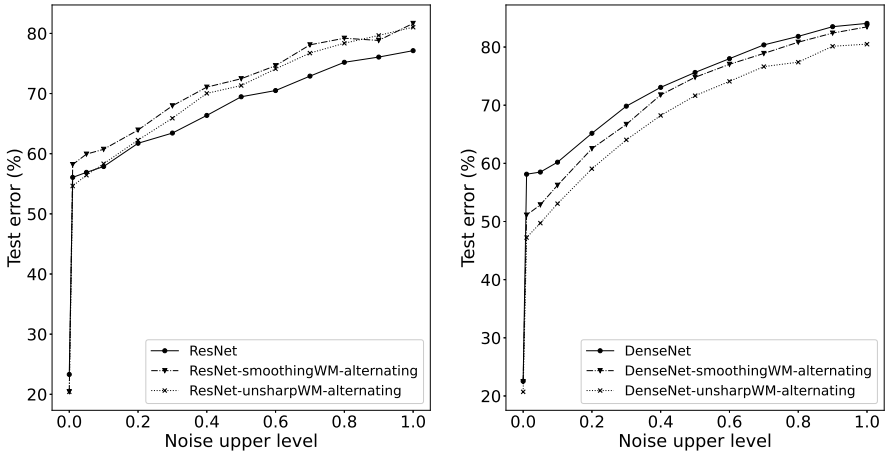


(a) Uniform noise

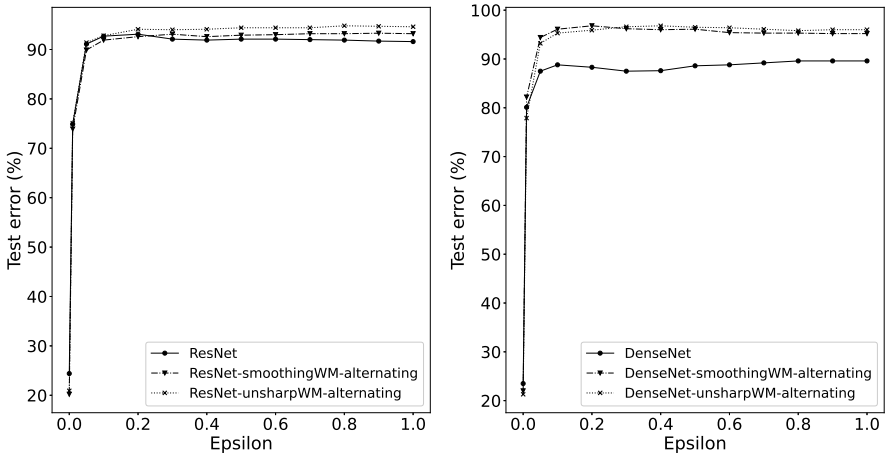


(b) Adversarial noise

Figure 5.5. CIFAR10. Left: ResNet variants. Right: DenseNet variants.



(a) Uniform noise



(b) Adversarial noise

Figure 5.6. iWildCam2019. Left: ResNet variants. Right: DenseNet variants.

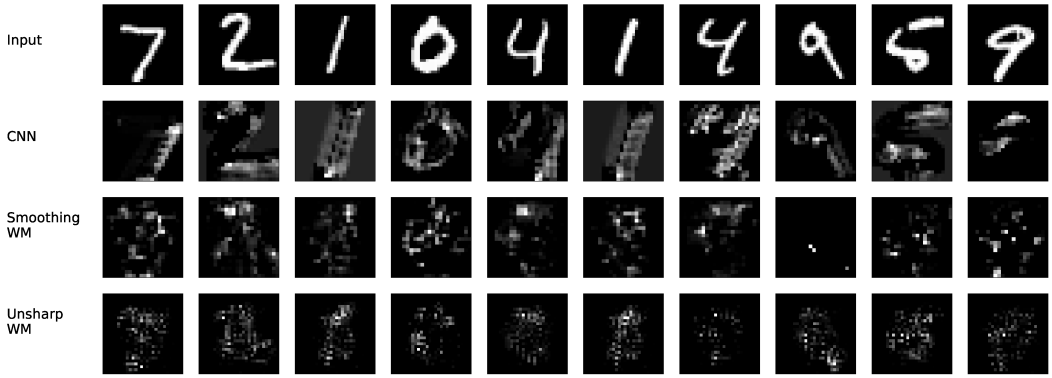


Figure 5.7. Grad-CAM visualization

against another reduction operation that mimics the action of an unsharp filter to reduce the accumulating blurring effect. To further reduce the introduced distortion, we benchmarked models which alternate between having a normal convolutional layer and a WM layer.

For the experiments based on CNN, the WM variant (no batchnorm and kernel size of 3) has a better performance than the corresponding vanilla CNN having the same number of FLOPS. We attribute this to two main factors. First, the WM variant is more expressive since it has larger number of parameters and hence has higher capacity. WM representation does not, however, need to be in the same space as the CNN variant. The Grad-CAM (Selvaraju et al., 2016) visualization of both vanilla CNN and the two WM variants Fig. 5.7 shows a substantial difference. While the CNN CAM is a blurry, diffused distortion of the input and sometimes activating for a large proportion of the background, the WM CAM is sharper, sparser and more localized with almost no diffused background activation, specially for the unsharp WM variant. We attribute this background activation sparsity to the feature selection ability of WM. Much like the way attentional techniques (Bahdanau, Cho, and Bengio, 2014; Vinyals et al., 2014; Xu et al., 2015; Hermann et al., 2015) can draw the network to focus on a subset of features, WM includes an elementwise multiplication by a weight map, that can in principle achieve feature selection on a pixel by pixel basis. On the other hand, normal convolution can not achieve a similar effect because of weight sharing. The second possible reason for better performance consists of the scaling properties of the WM layer. This can in principle act like the normalization done by batchnorm layers. However, applying batchnorm can boost the performance of both CNN and WM variants, which indicates that the two approaches have an orthogonal component between them. Moreover, as we discuss below, batchnorm alone does not protect against uniform noise and adversarial attacks. If we fix the number of parameters, instead of FLOPS, along with depth, we observe a clear advantage for WM variants. The WM variant with the same number of parameters and depth is better in performance by a

large margin, and cheaper in FLOPS by around 100x. We attribute this to the large width compared to depth of the CNN variant, which makes it harder to optimize. On the other hand, WM can pack larger degrees of freedom without growing in width.

Increasing the kernel size enhances the performance of the CNN variant, while it lowers the performance of the WM variant. The enhanced CNN performance is due to increased capacity and a larger context made available by the larger receptive field. In the case of WM, the increased kernel size results in over smoothing and larger overlapping between adjacent receptive fields, effectively sharing more parameters and limiting the model's effective capacity.

For deeper networks like ResNet and DenseNet, we will focus the discussion on the alternating models. The unsharp models show a similar or better test performance compared to the smoothing models across all the tested datasets. This supports our motivation for introducing the unsharp operation to reduce distortion.

For uniform noise, we find that for all the WM models in MNIST and CIFAR10, at least one of the WM variants have better noise robustness than the baseline. In this subset, all of the WM models of CNN and DenseNet in MNIST have better noise robustness than the baseline. For uniform noise in iWildCam2019, the WM variants of DenseNet are more robust than the baseline, while the WM variants of ResNet are less robust than the baseline. For adversarial attacks, the WM variants of CNN and DenseNet have at least one model which is better than the baseline. For ResNet on MNIST, WM variants have the same as or less robustness than the baseline. For CIFAR10 and iWildCam2019, all the models are very sensitive to the adversarial noise, with mixed relative results between the WM variants and the baseline.

The above summary of results shows that the WM layer is very effective against random noise, while it does not have the same efficacy against adversarial attacks. This may be explained by our hypothesis on the mechanism of the WM layer. The elementwise multiplication of the weight map by the input pixels, or any intermediate activation in general, can be considered as a feature selection mechanism. This implies that pixels that have large contribution to the accuracy will have large magnitude. The variance of any activation after applying the elementwise multiplication is then

$$\mathrm{Var}[w_{ij} \odot x_{ij}] = w_{ij}^2 \odot \mathrm{Var}[x_{ij}] \tag{5.6}$$

where x_{ij} is the activation at the index ij, Var is the variance of that activation over the dataset, w_{ij} is the weight corresponding to the activation and $\odot$ is the elementwise multiplication operator.

We attribute the noise resistance introduced by the WM layer to the described amplification of the activation variance. This amplification is proportional to the weight magnitude and, hence, to the importance of the corresponding feature to the model performance. The amplification of variance

increases the variations that the network encounters for the corresponding pixel/activation, which can be considered as an implicit augmentation of the dataset. This hypothesis, in turn, explains why it is performing better on random noise than adversarial attacks. While random noise has arbitrary direction, an adversarial attack can exploit gradient information to account for any differentiable component.

The efficiency of modular implementation that we discussed Section 1.5 was not an issue in this architecture since it is very similar in principle to convolutional layers. The efficient implementation is a matter of applying the usual matrix-matrix elementwise multiplication, followed by a convolution with a constant-values kernel. Optimization of convolution efficiency is well-studied and supported in modern tensor libraries and does not need any special treatment here.

5.7 Conclusion

We have introduced the weight map layer, a modular layer that shares similarity with the architecture of convolutional layers. The two WM layer variants introduced represent a generic architectural modification that can increase the robustness of convolutional neural networks to noise and adversarial attacks. We showed that it can be used to boost performance and increase noise robustness in small convolutional networks. Moreover, we showed that WM layers can be integrated into scaled up networks, ResNet and DenseNet, to increase their noise and adversarial attack robustness, while achieving comparable accuracy. We explained that the adaptive activation-variance amplification exhibited by the WM layer can explain its noise and adversarial attack robustness and the associated experimental observations regarding its dynamics.

5.8 Chapter Acknowledgements

This work was partially supported by a grant from Microsoft's AI for Earth program.

Chapter 6

Discussion and Conclusion

ANNs are ML models that are composed of interconnected nodes, where each node is a mathematical function consisting of a linear transformation followed by a non-linearity. Connections between nodes are specified by weights, which are learned parameters that control the strength by which nodes affect each other. MNNs are a specific type of ANN; an MNN is a neural network that consists of a set of subnetworks (called modules) that are densely connected internally, while sparsely connected between each other.

In this work, we aimed to use modularity to address some of the essential problems that face the research in deep learning and its application to complex ML problems. In Chapter 2, we did an extensive review of the literature that would serve as a contextual framework for the subsequent chapters. We identified common patterns between modular architectures and based on that we formulated our modularization framework that captures the general process of introducing modularity into an ANN. We showed that this framework can provide a broader perspective of modularity when applied to common case studies in the field.

We focused on applying modularity to the relation between the accuracy of a model on one side and its latency and the effect of noise in the data on the other side. ANNs are usually high-capacity models. This high-capacity correlates with high latency and more susceptibility to overfitting the noise in the data. A modeler who aims to apply a deep learning model in a limited resource environment has to sacrifice capacity in order to decrease latency. In Chapter 3, we used differential NAS to explore the space of modular multipath NNs that can be derived from a fully-connected network and by applying pruning and evaluating an architectural sample, we could predict the latency-accuracy relationship with good precision. This allows the modeler to find the required balance between resources and the model performance without the need for exhaustive training of models. In the described work, we used a multipath topology, a learned formation technique and a learned integration technique as described in Section 2.4.2, Section 2.4.3 and Section 2.4.4, respectively.

A related problem with stronger constraints is reducing the complexity while

maintaining accuracy. The target in this type of problem is reducing complexity as much as possible, while maintaining the accuracy of the baseline high-capacity model. In this sense, this problem has a much stronger constraint on the accuracy side and complexity can be reduced as long as we do not depart significantly from the accuracy of the high capacity model. In Chapter 4, we modularized CapsNet in order to introduce sparsity and reduce model complexity. We reduced the intermediate computations by integrating max-pooling into the manually engineered multipath architecture. We further exploited the modular architecture by introducing more depth and by regularizing using DropCircuit to compensate for the reduced capacity. The redesigned modular path capsule network model, combined with the introduced variant of the routing algorithm, was in some conditions as low as 20% of the original model parameters while its accuracy was similar to the original model and on some datasets it had even better accuracy. We used a multipath topology, manual formation and AL integration as described in Section 2.4.2, Section 2.4.3 and Section 2.4.4, respectively.

The third problem we investigated was the sensitivity of ANN accuracy to noise and adversarial attacks. There is a special interest in applying adversarial attacks to visual data and hence CNNs are a major target for these attacks. A convolutional layer is a modular layer that consists of a module being convolved with the input. In Chapter 5, we introduced the WM layer, which is a modular layer that has similarity to convolutional layers. The topology of the WM layer is a modular node topology as described in Section 2.4.2 and we used manual formation and AL integration as discussed in Section 2.4.3 and Section 2.4.4, respectively. A WM layer consists of two operations, elementwise multiplication by a weight grid followed by a reduction convolutional operation. We showed that the WM layer can be integrated into CNNs to increase their robustness to random noise and adversarial attacks and we investigated its working principle.

In implementing our modularity experiments we faced the problem of implementation efficiency. The main executing engine of ANNs is the GPU. The usage of GPU acceleration in ANNs depends on the assumption of dense matrix-matrix multiplications. Hence, in order to realize accelerated performance using a GPU, an operation has to be in a dense format. Since modular architectures have sparsity by definition, they do not readily benefit from GPU acceleration. We could solve the mentioned sparsity problem in the realized implementations in Chapter 5 and Chapter 4. In Chapter 5, the WM layer is similar in its general structure to convolutional layers and, hence, we could benefit from the existing GPU optimization of these operations. In Chapter 4, since one of our assumptions is a homogeneous multipath architecture (i.e. independent paths have the same width and depth), we could reduce the multipath convolutional operations to a dense matrix form. We discussed the general idea of dense transformation of an MNN with fully-connected paths in Section 4.6.1 and extending this to convolutional operators is straightforward and can be implemented with the

optimized depthwise-separable convolution utilities in modern tensor libraries. For Chapter 3, since the paths are not homogeneous, we could not use a similar strategy. We instead used the less efficient option of regular dense matrices where we zero out entries which correspond to connections that are severed due to modularity. However, since for our case sparsity is needed only at inference time which is not very computationally demanding, the experiments were still feasible.

Throughout this work we showed how modularity can be used to tackle different problem in deep learning. Future work holds a high potential for modularity based ideas. The work in Chapter 3 showed how differential search could help in exploring the combinatorial space of latency and accuracy in modular multipath networks in order to achieve a required balance. We used a pruning technique based on weight magnitude, which has logical justification and previous usage in literature besides its simplicity and efficient implementation. However, pruning is a large topic in the field and more advanced techniques may give rise to a more efficient architectural search. We regard NAS, and the general idea of learned formation described in Section 2.4.3, to be a very important idea for MNNs. While designing MNNs manually is suitable for some problems as was discussed in Chapter 2 and as we applied it in Chapter 4, the majority of problems will benefit more from learned and automatic formation of MNNs. In general, current NAS techniques depend on regular homogeneous blocks, mainly for the efficiency of implementation issue that was discussed. Hence, there is a serious missed opportunity for benefiting from modular diversity. In the same sense, the space of modular architectures searched by NAS techniques is very limited and has a very restrictive manually engineered component. EAs, which are another good candidate for modular architectural search, suffer from the same problems. Additionally, EAs are very demanding in terms of parallelization and, hence unlike differential techniques, they are further limited by the lack of an efficient acceleration engine. We regard finding more efficient modular search objectives, flexible modular NAS techniques and efficient modular implementations to be highly impactful future research directions.

In Chapter 4, we showed how manual modularity design can be used to make models more efficient and even increase their performance at the same time. The independence of paths renders the redesigned model suitable for model parallelism, a property which we did not investigate in detail and we leave for future work. We think there is still more space for enhancing PathCapsNet, specially in the reconstruction setting where we believe there is a complex interaction between routing, DropCircuit and reconstruction. Our results showed that modularity does not only hold the potential for sparsity and decrease in latency, but also can be utilized to enhance model accuracy and generalization. Using modularity to enhance generalization of models is still an open ended question and many challenges need to be addressed to achieve its full potential. More research in promoting functional modularity in MNNs and

in finding more objectives that can increase diversity and modular specialization is needed. In Chapter 5, we further proved the utility of modularity for another measure which is noise and adversarial attack robustness. More future work is needed to identify more effective ways to integrate with other architectures and to gain more insights regarding the dynamics of the WM layer. We think that a lot of potential exists in investigating the relation between modularity and many performance measures that may be considered not to be directly related to modularity like catastrophic forgetting, data greediness, few-shot learning and regularization.

As we discussed throughout the preceding chapters, this work was not possible without finding an implementation that is efficient enough to be computable and the same applies to any potential work in modularity, either artificially or biologically inspired. We had to transform our operators into an efficient form and to adapt them to the existing optimized algorithms in the field. Nonetheless, there are still a lot of general use cases that do not have a feasible or straightforward efficient implementation using the current technology and tools. This limitation is extremely crippling to the research in modularity and, hence, the impact of developing this area can not be overemphasized.

Modularity has its deep origins in biology. Regardless of the frequent divergence between the study of deep learning and its biological roots, we believe that many fruitful links between artificial and biological modularity exist. The original inspiration of CNN was based on neuroscientific studies and it culminated in one of the most successful examples of this crossover of ideas. We discussed similar inspirations in Chapter 2 and we regard this direction as very important, at least at the explanatory and illuminating level, if not also at the practical level. We could use modularity to balance accuracy and latency in multipath neural networks, reduce complexity of CapsNet and increase CNN model robustness to noise and adversarial attack. We believe that the research in modularity has just scratched the surface of its potential and by building on previous studies in the artificial domain along with more adoption of inspirations from the biological domain, the reach of modularity applications can be significantly expanded.

Appendix A

Acronyms

AI	Artificial Intelligence
AL	Arithmetic-Logic
AMR	Abstract Meaning Representations
ANN	Artificial Neural Network
BBNN	Block-Based Neural Network
CG	Cyclic Graph
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CV	Computer Vision
DAE	Denoising Autoencoder
DAG	Directed Acyclic Graph
DCL	Divide-and-Conquer Learning
EA	Evolution Algorithm
ECP	Error Correlation Partitioning
EM	Expectation Maximization
ES	Evolution Strategies
EWC	Elastic Weight Consolidation
FC	Fully Connected
FGSM	Fast Gradient Sign Method
FLOPS	Floating Point Operations Per Second

FNN False Nearest Neighbours

GA Genetic Algorithm

GAN Generative Adversarial Network

GMM Gaussian Mixture Model

GNN Graph Neural Network

GP Genetic Programming

GPGPU General-Purpose Graphical Processing Unit

GPU Graphical Processing Unit

GRU Gated Recurrent Unit

HCNR Highly-Clustered Non-Regular

HMM Hierarchical Markov Model

HPR Human Pose Recovery

HRNN Hierarchical Recurrent Neural Network

IoT Internet of Things

LGN Lateral Geniculate Nucleus

LSTM Long Short-Term Memory

LTM Long-Term Memory

LWTA Local Winner-Take-All

MAE Mean Absolute Error

MCNN Modular Cellular Neural Network

MDP Markov Decision Process

MGN Medial Geniculate Nucleus

ML Machine Learning

MLP Multilayer Perceptron

MNN Modular Neural Network

MPNN Message-Passing Neural Network

MVG Modularly Varying Goal

MpathNN Multipath Neural Network

NAS Neural Architecture Search

NIN Network In a Network

NLP Natural Language Processing

NN Neural Network

OAA One-Against-All

OAO One-Against-One

PAQ P-Against-Q

PC Parallel Circuit

PID Proportional–Integral–Derivative

RBF Radial Basis Function

RDM Representational Dissimilarity Matrix

RF Receptive Field

RL Reinforcement Learning

RNN Recurrent Neural Network

RO Research Objective

RQ Research Question

RSA Representational Similarity Analysis

SCN Siamese Capsule Network

SGD Stochastic Gradient Descent

SI Synaptic Intelligence

SOTA state-of-the-art

STM Short-Term Memory

SVM Support Vector Machine

SW Small World

TBPTT Truncated Backpropagation Through Time

UAV Unmanned Aerial Vehicle

VAE Variational Autoencoder

VLSI Very Large-Scale Integration

WM Weight Map

References

Abraham, Wickliffe C and Anthony Robins (2005). "Memory retention–the synaptic stability versus plasticity dilemma". In: *Trends in neurosciences* 28.2, pp. 73–78.

Achard, Sophie and Ed Bullmore (2007). "Efficiency and cost of economical brain functional networks". In: *PLoS Computational Biology* 3.2, pp. 0174–0183. ISSN: 1553734X. DOI: 10.1371/journal.pcbi.0030017.

Aguirre, Carlos, Ramón Huerta, Fernando Corbacho, and Pedro Pascual (2002). "Analysis of biologically inspired small-world networks". In: *International Conference on Artificial Neural Networks*. Springer, pp. 27–32.

Allen, F., G. Almasi, W. Andreoni, D. Beece, B. J. Berne, A. Bright, J. Brunheroto, C. Cascaval, J. Castanos, P. Coteus, P. Crumley, A. Curioni, M. Denneau, W. Donath, M. Eleftheriou, B. Flitch, B. Fleischer, C. J. Georgiou, R. Germain, M. Giampapa, D. Gresh, M. Gupta, R. Haring, H. Ho, P. Hochschild, S. Hummel, T. Jonas, D. Lieber, G. Martyna, K. Maturu, J. Moreira, D. Newns, M. Newton, R. Philhower, T. Picunko, J. Pitera, M. Pitman, R. Rand, A. Royyuru, V. Salapura, A. Sanomiya, R. Shah, Y. Sham, S. Singh, M. Snir, F. Suits, R. Swetz, W. C. Swope, N. Vishnumurthy, T. J. C Ward, H. Warren, and R. Zhou (2001). "Blue Gene: A vision for protein science using a petaflop supercomputer". In: *IBM Systems Journal* 40.2, pp. 310–327. ISSN: 0018-8670. DOI: 10.1147/sj.402.0310. URL: http://ieeexplore.ieee.org/document/5386970/.

Almasri, Mohammad N and Jagath J Kaluarachchi (2005). "Modular neural networks to predict the nitrate distribution in ground water using the on-ground nitrogen loading and recharge data". In: *Environmental Modelling & Software* 20.7, pp. 851–871.

Amer, Mohammed and Tomás Maul (2019). "A review of modularization techniques in artificial neural networks". In: *Artificial Intelligence Review* 52.1, pp. 527–561. ISSN: 15737462. DOI: 10.1007/s10462-019-09706-7. eprint: 1904.12770. URL: https://doi.org/10.1007/s10462-019-09706-7.

— (2020). "Path Capsule Networks". In: *Neural Processing Letters*, pp. 1–15. ISSN: 1573773X. DOI: 10.1007/s11063-020-10273-0. arXiv: 1902.03760. URL: https://doi.org/10.1007/s11063-020-10273-0.

Aminian, Mehran and Farzan Aminian (2007). "A modular fault-diagnostic system for analog electronic circuits using neural networks with wavelet transform as a preprocessor". In: *IEEE Transactions on Instrumentation and Measurement* 56.5, pp. 1546–1554.

Anand, R., K. Mehrotra, C.K. Mohan, and S. Ranka (1995). "Efficient classification for multiclass problems using modular neural networks". In: *IEEE Transactions on Neural Networks* 6.1, pp. 117–124. ISSN: 10459227. DOI: 10.1109/72.363444.

Anderson, Ark, Kyle Shaffer, Artem Yankov, Court D. Corley, and Nathan O. Hodas (2016). "Beyond Fine Tuning: A Modular Approach to Learning on Small Data". In: arXiv: 1611.01714. URL: https://arxiv.org/abs/1611.01714v1.

Andreas, Jacob, Marcus Rohrbach, Trevor Darrell, and Dan Klein (2016a). "Neural module networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 39–48.

— (2016b). "Learning to Compose Neural Networks for Question Answering". In: arXiv: 1601.01705.

Andrews, George E (1998). *The theory of partitions*. 2. Cambridge university press.

Andrews, George E and Kimmo Eriksson (2004). *Integer partitions*. Cambridge University Press.

Angelucci, a, F Clascá, E Bricolo, K S Cramer, and M Sur (1997). "Experimentally induced retinal projections to the ferret auditory thalamus: development of clustered eye-specific patterns in a novel target." In: *The Journal of neuroscience : the official journal of the Society for Neuroscience* 17.6, pp. 2040–2055. ISSN: 0270-6474.

Arik, Sercan O., Jitong Chen, Kainan Peng, Wei Ping, and Yanqi Zhou (2018). *Neural Voice Cloning with a Few Samples*. arXiv: 1802.06006 [cs.CL].

Auda, G and M Kamel (1998). "Modular Neural Network Classifiers: A Comparative Study". In: *Journal of Intelligent and Robotic Systems* 21, pp. 117–129. ISSN: 09210296. DOI: 10.1023/A:1007925203918.

— (1999). "Modular neural networks: a survey." In: *International journal of neural systems* 9.2, pp. 129–51. ISSN: 0129-0657.

Azam, Farooq (2000). "Biologically Inspired Modular Neural Networks". In: URL: https://vtechworks.lib.vt.edu/handle/10919/27998.

Ba, Jimmy and Rich Caruana (2014). "Do deep nets really need to be deep?" In: *Advances in neural information processing systems*, pp. 2654–2662.

Babaei, Sepideh, Amir Geranmayeh, and Seyyed Ali Seyyedsalehi (2010). "Protein secondary structure prediction using modular reciprocal bidirectional recurrent neural networks". In: *Computer Methods and Programs in Biomedicine* 100.3, pp. 237–247. ISSN: 01692607. DOI: 10.1016/j.cmpb.2010.04.005.

Bäck, Thomas, David B Fogel, and Zbigniew Michalewicz (2018). *Evolutionary computation 1: Basic algorithms and operators*. CRC press.

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio (2014). "Neural Machine Translation by Jointly Learning to Align and Translate". In: arXiv: 1409.0473. URL: http://arxiv.org/abs/1409.0473.

Barrett, David GT, Ari S. Morcos, and Jakob H. Macke (2019). *Analyzing biological and artificial neural networks: challenges with opportunities for synergy?* DOI: 10.1016/j.conb.2019.01.007. arXiv: 1810.13373.

Barua, Sukarna, Sarah Monazam Erfani, and James Bailey (2019). *FCC-GAN: A Fully Connected and Convolutional Net Architecture for GANs*. arXiv: 1905.02417 [cs.LG].

Battaglia, Peter W., Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andrew Ballard, Justin Gilmer, George Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu (2018). "Relational inductive biases, deep learning, and graph networks". In: arXiv: 1806.01261. URL: http://arxiv.org/abs/1806.01261.

Bello, Irwan, Barret Zoph, Vijay Vasudevan, and Quoc V Le (2017). "Neural optimizer search with reinforcement learning". In: *International Conference on Machine Learning*. PMLR, pp. 459–468.

Bender, Gabriel, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le (2018). *Understanding and Simplifying One-Shot Architecture Search*. URL: http://proceedings.mlr.press/v80/bender18a.

Bengio, Samy, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer (2015). *Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks*. URL: http://papers.nips.cc/paper/5956-scheduled-sampling-for-sequence-prediction-with-recurrent-neural-networks.

Bengio, Yoshua, Jérôme Louradour, Ronan Collobert, and Jason Weston (2009). "Curriculum learning". In: *Proceedings of the 26th Annual International Conference on Machine Learning - ICML '09*. New York, New York, USA: ACM Press, pp. 1–8. ISBN: 9781605585161. DOI: 10.1145/1553374.1553380. URL: http://portal.acm.org/citation.cfm?doid=1553374.1553380.

Bhende, C.N., S. Mishra, and B.K. Panigrahi (2008). "Detection and classification of power quality disturbances using S-transform and modular neural network". In: *Electric Power Systems Research* 78.1, pp. 122–128. ISSN: 03787796. DOI: 10.1016/j.epsr.2006.12.011.

Blundell, Charles, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra (2015). "Weight uncertainty in neural networks". In: *arXiv:1505.05424*.

Bohland, Jason W. and Ali A. Minai (2001). "Efficient associative memory using small-world architecture". In: *Neurocomputing* 38, pp. 489–496. ISSN: 09252312. DOI: 10.1016/S0925-2312(01)00378-2.

Brandes, Ulrik, Daniel Delling, Marco Gaertler, Robert Gorke, Martin Hoefer, Zoran Nikoloski, and Dorothea Wagner (2008). "On modularity clustering". In: *IEEE Transactions on Knowledge and Data Engineering* 20.2, pp. 172–188. ISSN: 10414347. DOI: 10.1109/TKDE.2007.190689.

Braylan, Alexander, Mark Hollenbeck, Elliot Meyerson, and Risto Miikkulainen (2015). "Reuse of Neural Modules for General Video Game Playing". In: arXiv: 1512.01537.

Bruinier, Jan Hendrik and Ken Ono (2013). "Algebraic formulas for the coefficients of half-integral weight harmonic weak Maass forms". In: *Advances in Mathematics* 246, pp. 198–219.

Bullmore, Ed and Olaf Sporns (2009). "Complex brain networks: graph theoretical analysis of structural and functional systems". In: *Nature Reviews Neuroscience* 10.3, pp. 186–198. ISSN: 1471-003X. DOI: 10.1038/nrn2575.

Bullmore, Edward T. and Danielle S. Bassett (2011). "Brain Graphs: Graphical Models of the Human Brain Connectome". In: *Annual Review of Clinical Psychology* 7.1, pp. 113–140. ISSN: 1548-5943. DOI: 10.1146/annurev-clinpsy-040510-143934.

Buxhoeveden, D. P. (2002). "The minicolumn hypothesis in neuroscience". In: *Brain* 125.5, pp. 935–951. ISSN: 14602156. DOI: 10.1093/brain/awf110.

Caelli, Terry, Ling Guan, and Wilson Wen (1999). "Modularity in neural computing". In: *Proceedings of the IEEE* 87.9, pp. 1497–1518. ISSN: 00189219. DOI: 10.1109/5.784227.

Cai, Han, Ligeng Zhu, and Song Han (2018). "Proxylessnas: Direct neural architecture search on target task and hardware". In: *arXiv:1812.00332*.

Cai, Shaofeng, Yao Shu, Wei Wang, and Beng Chin Ooi (2019). "Isbnet: Instance-aware selective branching network". In: *arXiv preprint arXiv:1905.04849*.

Calabretta, Raffaele, Stefano Nolfi, Domenico Parisi, and Günter P Wagner (2000). "Duplication of modules facilitates the evolution of functional specialization". In: *Artificial life* 6.1, pp. 69–84.

Chakraborty, Anirban, Manaar Alam, Vishal Dey, Anupam Chattopadhyay, and Debdeep Mukhopadhyay (2018). "Adversarial Attacks and Defences: A Survey". In: arXiv: 1810.00069. URL: http://arxiv.org/abs/1810.00069.

Chandna, Pritish, Marius Miron, Jordi Janer, and Emilia Gómez (2017). "Monoaural Audio Source Separation Using Deep Convolutional Neural Networks". In: *Latent Variable Analysis and Signal Separation*. Ed. by Petr Tichavský, Massoud Babaie-Zadeh, Olivier J J Michel, and Nadège Thirion-Moreau. Cham: Springer International Publishing, pp. 258–266. ISBN: 978-3-319-53547-0.

Chen, Beidi, Tharun Medini, James Farwell, Sameh Gobriel, Charlie Tai, and Anshumali Shrivastava (2020). *SLIDE : In Defense of Smart Algorithms over Hardware Acceleration for Large-Scale Deep Learning Systems*. arXiv: 1903.03129 [cs.DC].

Chen, Wenlin, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen (2015). "Compressing neural networks with the hashing trick". In: *International conference on machine learning*, pp. 2285–2294.

Chen, Zhang J., Yong He, Pedro Rosa-Neto, Jurgen Germann, and Alan C. Evans (2008). "Revealing modular architecture of human brain structural

networks by using cortical thickness from MRI". In: *Cerebral Cortex* 18.10, pp. 2374–2381. ISSN: 10473211. DOI: 10.1093/cercor/bhn003.

Chiang, Cheng-Chin and Hsin-Chia Fu (1994). "A divide-and-conquer methodology for modular supervised neural network design". In: *Neural Networks, 1994. IEEE World Congress on Computational Intelligence., 1994 IEEE International Conference on*. Vol. 1. IEEE, pp. 119–124.

Chihaoui, Mejda, Akram Elkefi, Wajdi Bellil, and Chokri Ben Amar (2016). "A Survey of 2D Face Recognition Techniques". In: *Computers* 5.4, p. 21. DOI: 10.3390/computers5040021.

Cho, Kyunghyun, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio (2014). *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*. arXiv: 1406.1078 [cs.CL].

Chollet, François (2016). "Xception: Deep Learning with Depthwise Separable Convolutions". In: arXiv: 1610.02357. URL: http://arxiv.org/abs/1610.02357.

Chris Tseng, H. and Bassam Almogahed (2009). "Modular neural networks with applications to pattern profiling problems". In: *Neurocomputing* 72.10-12, pp. 2093–2100. ISSN: 0925-2312. DOI: 10.1016/J.NEUCOM.2008.10.020. URL: https://www.sciencedirect.com/science/article/pii/S0925231208005444?via{\%}3Dihub.

Ciregan, Dan, Ueli Meier, and Jürgen Schmidhuber (2012). "Multi-column deep neural networks for image classification". In: *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE, pp. 3642–3649.

Cireşan, Dan, Ueli Meier, and Juergen Schmidhuber (2012). "Multi-column Deep Neural Networks for Image Classification". In: arXiv: 1202.2745. URL: http://arxiv.org/abs/1202.2745.

Clune, Jeff, Jean-Baptiste Mouret, and Hod Lipson (2013). "The evolutionary origins of modularity." In: *Proceedings. Biological sciences / The Royal Society* 280.1755, p. 20122863. ISSN: 1471-2954. DOI: 10.1098/rspb.2012.2863. arXiv: 1207.2743v1.

Coop, Robert, Aaron Mishtal, and Itamar Arel (2013). "Ensemble learning in fixed expansion layer networks for mitigating catastrophic forgetting". In: *IEEE Transactions on Neural Networks and Learning Systems* 24.10, pp. 1623–1634. ISSN: 2162237X. DOI: 10.1109/TNNLS.2013.2264952.

Csaky, Richard (2019). *Deep Learning Based Chatbot Models*. arXiv: 1908.08835 [cs.CL].

Dai, Wenyuan, Qiang Yang, Gui-Rong Xue, and Yong Yu (2007). *Boosting for Transfer Learning*. Tech. rep. URL: http://www.cs.berkeley.edu/.

Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova (2019). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. arXiv: 1810.04805 [cs.CL].

Di Ferdinando, Andrea, Raffaele Calabretta, and Domenico Parisi (2001). "Evolving modular architectures for neural networks". In:

Douglas, Rodney J and Kevan A C Martin (2007). "Recurrent neuronal circuits in the neocortex." In: *Current biology : CB* 17.13, R496–500. ISSN: 0960-9822. DOI: 10.1016/j.cub.2007.04.024. URL: http://www.ncbi.nlm.nih.gov/pubmed/17610826.

Duchi, John, Elad Hazan, and Yoram Singer (2011). "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *Journal of Machine Learning Research* 12.61, pp. 2121–2159. URL: http://jmlr.org/papers/v12/duchi11a.html.

Eich, Janet M. (1982). "A composite holographic associative recall model." In: *Psychological Review* 89.6, pp. 627–661. ISSN: 0033-295X. DOI: 10.1037/0033-295X.89.6.627. URL: http://content.apa.org/journals/rev/89/6/627.

Eppel, Sagi (2017). "Hierarchical semantic segmentation using modular convolutional neural networks". In: arXiv: 1710.05126. URL: https://arxiv.org/abs/1710.05126v1.

Eyben, Florian, Felix Weninger, Stefano Squartini, and Bjorn Schuller (2013). "Real-life voice activity detection with LSTM Recurrent Neural Networks and an application to Hollywood movies". In: *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, pp. 483–487. ISBN: 9781479903566. DOI: 10.1109/ICASSP.2013.6637694.

Fan, Wenqi, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin (2019). *Graph Neural Networks for Social Recommendation*. arXiv: 1902.07243 [cs.IR].

Fernando, Chrisantha, Dylan Banarse, Charles Blundell, Yori Zwols, David Ha, Andrei A. Rusu, Alexander Pritzel, and Daan Wierstra (2017). "PathNet: Evolution Channels Gradient Descent in Super Neural Networks". In: arXiv: 1701.08734. URL: http://arxiv.org/abs/1701.08734.

Ferreira, Martha Dais, Débora Cristina Corrêa, Luis Gustavo Nonato, and Rodrigo Fernandes de Mello (2018). "Designing architectures of convolutional neural networks to solve practical problems". In: *Expert Systems with Applications* 94, pp. 205–217. ISSN: 0957-4174. DOI: 10.1016/J.ESWA.2017.10.052. URL: https://www.sciencedirect.com/science/article/pii/S0957417417307340.

Franco, Leonardo and Sergio Alejandro Cannas (2001). "Generalization properties of modular networks: Implementing the parity function". In: *IEEE Transactions on Neural Networks* 12.6, pp. 1306–1313. ISSN: 10459227. DOI: 10.1109/72.963767.

Freddolino, Peter L, Feng Liu, Martin Gruebele, and Klaus Schulten (2008). "Ten-microsecond molecular dynamics simulation of a fast-folding WW domain." In: *Biophysical journal* 94.10, pp. L75–7. ISSN: 1542-0086. DOI: 10.1529/biophysj.108.131565. URL: http://www.ncbi.nlm.nih.gov/

pubmed/18339748http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC2367204.

Fritsch, J urgen (1996). "Modular Neural Networks for Speech Recognition". In:

Fu, Hsin Chia, Yen Po Lee, Cheng Chin Chiang, and Hsiao Tien Pao (2001). "Divide-and-conquer learning and modular perceptron networks". In: *IEEE Transactions on Neural Networks* 12.2, pp. 250–263. ISSN: 10459227. DOI: 10.1109/72.914522.

Fukushima, Kunihiko and Sei Miyake (1980). "Neocognitron: Self-organizing network capable of position-invariant recognition of patterns". In: *Proc. 5th Int. Conf. Pattern Recognition*. Vol. 1, pp. 459–461.

Ganin, Yaroslav, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky (2015). "Domain-Adversarial Training of Neural Networks". In: arXiv: 1505.07818. URL: https://arxiv.org/abs/1505.07818.

Gao, Ji, Beilun Wang, Zeming Lin, Weilin Xu, and Yanjun Qi (2017). "DeepCloak: Masking Deep Neural Network Models for Robustness Against Adversarial Samples". In: arXiv: 1702.06763. URL: http://arxiv.org/abs/1702.06763.

Garcia-Pedrajas, N., C. Hervas-Martinez, and J. Munoz-Perez (2003). "COVNET: a cooperative coevolutionary model for evolving artificial neural networks". In: *IEEE Transactions on Neural Networks* 14.3, pp. 575–596. ISSN: 1045-9227. DOI: 10.1109/TNN.2003.810618.

Gasse, Maxime, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi (2019). *Exact Combinatorial Optimization with Graph Convolutional Neural Networks*. arXiv: 1906.01629 [cs.LG].

Ghafoorian, Mohsen, Nico Karssemeijer, Tom Heskes, Inge W. M. van Uden, Clara I. Sanchez, Geert Litjens, Frank-Erik de Leeuw, Bram van Ginneken, Elena Marchiori, and Bram Platel (2017). "Location Sensitive Deep Convolutional Neural Networks for Segmentation of White Matter Hyperintensities". In: *Scientific Reports* 7.1, p. 5110. ISSN: 2045-2322. DOI: 10.1038/s41598-017-05300-5. URL: http://www.nature.com/articles/s41598-017-05300-5.

Ghosh, Partha, Arpan Losalka, and Michael J Black (2018). "Resisting Adversarial Attacks using Gaussian Mixture Variational Autoencoders". In: arXiv: 1806.00081. URL: http://arxiv.org/abs/1806.00081.

Gilmer, Justin, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl (2017). "Neural Message Passing for Quantum Chemistry". In: arXiv: 1704.01212. URL: http://arxiv.org/abs/1704.01212.

Gollisch, Tim and Markus Meister (2010). *Eye Smarter than Scientists Believed: Neural Computations in Circuits of the Retina*. DOI: 10.1016/j.neuron.2009.12.009. arXiv: NIHMS150003.

Goltsev, Alexander and Vladimir Gritsenko (2015). "Modular neural networks with radial neural columnar architecture". In: *Biologically Inspired Cognitive*

Architectures 13, pp. 63–74. ISSN: 2212-683X. DOI: 10.1016/J.BICA.2015.06.001. URL: https://www.sciencedirect.com/science/article/pii/S2212683X15000286.

Gonzalez, Rafael C and Richard E Woods (2002). "Digital image processing [M]". In: *Publishing house of electronics industry* 141.7, pp. 184–186.

Goodfellow, Ian, Yoshua Bengio, Aaron Courville, and Yoshua Bengio (2016). *Deep learning*. Vol. 1. 2. MIT press Cambridge.

Goodfellow, Ian, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio (2014). *Generative Adversarial Nets*. URL: http://papers.nips.cc/paper/5423-generative-adversarial-nets.

Goodfellow, Ian J., Jonathon Shlens, and Christian Szegedy (2014). "Explaining and Harnessing Adversarial Examples". In: arXiv: 1412.6572. URL: http://arxiv.org/abs/1412.6572.

Gradojevic, Nikola, Ramazan Gençay, and Dragan Kukolj (2009). "Option pricing with modular neural networks." In: *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council* 20.4, pp. 626–637. ISSN: 1941-0093. DOI: 10.1109/TNN.2008.2011130.

Guan, Sheng-Uei and Shanchun Li (2002). "Parallel growing and training of neural networks using output parallelism". In: *IEEE transactions on Neural Networks* 13.3, pp. 542–550.

Guang-Bin Huang (2003). "Learning capability and storage capacity of two-hidden-layer feedforward networks". In: *IEEE Transactions on Neural Networks* 14.2, pp. 274–281. ISSN: 1045-9227. DOI: 10.1109/TNN.2003.809401.

Guo, Guodong and Na Zhang (2019). "A survey on deep learning based face recognition". In: *Computer Vision and Image Understanding* 189, p. 102805. ISSN: 1077-3142. DOI: https://doi.org/10.1016/j.cviu.2019.102805.

Guo, Qiushan, Zhipeng Yu, Yichao Wu, Ding Liang, Haoyu Qin, and Junjie Yan (2019). "Dynamic recursive neural network". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5147–5156.

Ha, David, Andrew Dai, and Quoc V. Le (2016). "HyperNetworks". In: arXiv: 1609.09106. URL: http://arxiv.org/abs/1609.09106.

Han, Song, Huizi Mao, and William J Dally (2015). "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding". In: *arXiv preprint arXiv:1510.00149*.

Hao, Zhongkai, Chengqiang Lu, Zhenya Huang, Hao Wang, Zheyuan Hu, Qi Liu, Enhong Chen, and Cheekong Lee (2020). "ASGN: An Active Semi-supervised Graph Neural Network for Molecular Property Prediction". In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. DOI: 10.1145/3394486.3403117. URL: http://dx.doi.org/10.1145/3394486.3403117.

Happel, Bart L M and Jacob M J Murre (1994). "Design and evolution of modular neural network architectures". In: *Neural Networks* 7.6-7, pp. 985–1004. ISSN: 08936080. DOI: 10.1016/S0893-6080(05)80155-8.

Hassabis, Demis, Dharshan Kumaran, Christopher Summerfield, and Matthew Botvinick (2017). "Neuroscience-inspired artificial intelligence". In: *Neuron* 95.2, pp. 245–258.

Haykin, Simon (1994). *Neural networks: a comprehensive foundation*. Prentice Hall PTR.

He, Congqing, Li Peng, Yuquan Le, and Jiawei He (2018). "SECaps: A Sequence Enhanced Capsule Model for Charge Prediction". In: arXiv: 1810.04465. URL: http://arxiv.org/abs/1810.04465.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2015). "Deep Residual Learning for Image Recognition". In: arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

— (2016). "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

Hermann, Karl Moritz, Tomáš Kočiský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom (2015). "Teaching Machines to Read and Comprehend". In: arXiv: 1506.03340. URL: http://arxiv.org/abs/1506.03340.

Hidalgo, Denisse, Oscar Castillo, and Patricia Melin (2009). "Type-1 and type-2 fuzzy inference systems as integration methods in modular neural networks for multimodal biometry and its optimization with genetic algorithms". In: *Information Sciences* 179.13, pp. 2123–2145.

Hinton, Geoffrey, Oriol Vinyals, and Jeff Dean (2015). "Distilling the Knowledge in a Neural Network". In: arXiv: 1503.02531. URL: http://arxiv.org/abs/1503.02531.

Hinton, Geoffrey E, Sara Sabour, and Nicholas Frosst (2018). "Matrix capsules with {EM} routing". In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=HJWLfGWRb.

Hochreiter, Sepp and J Urgen Schmidhuber (1997). "LONG SHORT-TERM MEMORY". In: *Neural Computation* 9.8, pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. arXiv: 1206.2944.

Hong, Chaoqun, Jun Yu, Jian Wan, Dacheng Tao, and Meng Wang (2015). "Multimodal Deep Autoencoder for Human Pose Recovery". In: *IEEE Transactions on Image Processing* 24.12, pp. 5659–5670. ISSN: 10577149. DOI: 10.1109/TIP.2015.2487860.

Hong, Chaoqun, Jun Yu, Jian Zhang, Xiongnan Jin, and Kyong Ho Lee (2019). "Multimodal Face-Pose Estimation With Multitask Manifold Deep Learning". In: *IEEE Transactions on Industrial Informatics* 15.7, pp. 3952–3961. ISSN: 15513203. DOI: 10.1109/TII.2018.2884211. arXiv: 1712.06467.

Hornik, Kurt, Maxwell Stinchcombe, and Halbert White (1990). "Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks". In: *Neural Networks* 3.5, pp. 551–560. ISSN: 0893-6080. DOI: 10.1016/0893-6080(90)90005-6. URL: https://www.sciencedirect.com/science/article/pii/0893608090900056.

Hosseini, Hossein, Yize Chen, Sreeram Kannan, Baosen Zhang, and Radha Poovendran (2017). "Blocking Transferability of Adversarial Examples in Black-Box Learning Systems". In: arXiv: 1703.04318. URL: https://arxiv.org/abs/1703.04318.

Howard, Andrew G, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam (2017). "Mobilenets: Efficient convolutional neural networks for mobile vision applications". In: *arXiv preprint arXiv:1704.04861*.

Hu, Ronghang, Marcus Rohrbach, Jacob Andreas, Trevor Darrell, and Kate Saenko (2016). "Modeling Relationships in Referential Expressions with Compositional Modular Networks". In: arXiv: 1611.09978.

Huang, Gao, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger (2016a). "Densely Connected Convolutional Networks". In: arXiv: 1608.06993. URL: http://arxiv.org/abs/1608.06993.

Huang, Gao, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger (2016b). "Deep networks with stochastic depth". In: *European Conference on Computer Vision*. Springer, pp. 646–661.

Hubel, DH and TN Wiesel (1968). "Receptive fields and functional architecture of monkey striate cortex". In: *The Journal of physiology*. URL: http://onlinelibrary.wiley.com/doi/10.1113/jphysiol.1968.sp008455/abstract?rss=1.

Huizinga, Joost, Jean-Baptiste Mouret, and Jeff Clune (2014). "Evolving Neural Networks That Are Both Modular and Regular: HyperNeat Plus the Connection Cost Technique". In: *Gecco*, pp. 697–704. DOI: 10.1145/2576768.2598232.

Hüllermeier, Eyke, Thomas Fober, and Marco Mernberger (2013). "Inductive Bias". In: *Encyclopedia of Systems Biology*. Ed. by Werner Dubitzky, Olaf Wolkenhauer, Kwang-Hyun Cho, and Hiroki Yokota. New York, NY: Springer New York, p. 1018. ISBN: 978-1-4419-9863-7. DOI: 10.1007/978-1-4419-9863-7_927. URL: https://doi.org/10.1007/978-1-4419-9863-7_927.

Husic, Brooke E., Nicholas E. Charron, Dominik Lemm, Jiang Wang, Adrià Pérez, Maciej Majewski, Andreas Krämer, Yaoyi Chen, Simon Olsson, Gianni de Fabritiis, Frank Noé, and Cecilia Clementi (2020). *Coarse Graining Molecular Dynamics with Graph Neural Networks*. arXiv: 2007.11412 [physics.comp-ph].

Hüsken, Michael, Christian Igel, and Marc Toussaint (2002). "Task-dependent evolution of modularity in neural networks". In: *Connection Science* 14.3, pp. 219–229.

Iandola, Forrest N, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer (2016). "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size". In: *arXiv:1602.07360*.

Ioffe, Sergey and Christian Szegedy (2015). "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: arXiv: 1502.03167. URL: http://arxiv.org/abs/1502.03167.

Isele, David and Akansel Cosgun (2018). *Selective Experience Replay for Lifelong Learning*. arXiv: 1802.10269 [cs.AI].

Jacobs, Robert A., Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton (1991). "Adaptive Mixtures of Local Experts". In: *Neural Computation* 3.1, pp. 79–87. ISSN: 0899-7667. DOI: 10.1162/neco.1991.3.1.79. URL: http://www.mitpressjournals.org/doi/10.1162/neco.1991.3.1.79.

Jaderberg, Max, Andrea Vedaldi, and Andrew Zisserman (2014). "Speeding up convolutional neural networks with low rank expansions". In: *arXiv preprint arXiv:1405.3866*.

Jin, Jonghoon, Aysegul Dundar, and Eugenio Culurciello (2015). "Robust Convolutional Neural Networks under Adversarial Noise". In: arXiv: 1511.06306. URL: http://arxiv.org/abs/1511.06306.

Jing, Yongcheng, Yezhou Yang, Zunlei Feng, Jingwen Ye, Yizhou Yu, and Mingli Song (2018). *Neural Style Transfer: A Review*. arXiv: 1705.04058 [cs.CV].

Kacprzyk, Janusz and Witold Pedrycz (2015). *Springer handbook of computational intelligence*. Springer.

Kaiser, Marcus and Claus C Hilgetag (2010). "Optimal hierarchical modular topologies for producing limited sustained activation of neural networks". In: *Frontiers in neuroinformatics* 4.

Karami, Mojtaba, Reza Safabakhsh, and Mohammad Rahmati (2013). "Modular cellular neural network structure for wave-computing-based image processing". In: *ETRI Journal* 35.2, pp. 207–217. ISSN: 12256463. DOI: 10.4218/etrij.13.0112.0107.

Kashtan, Nadav and Uri Alon (2005). "Spontaneous evolution of modularity and network motifs." In: *Proceedings of the National Academy of Sciences of the United States of America* 102.39, pp. 13773–8. ISSN: 0027-8424. DOI: 10.1073/pnas.0503610102.

Kastellakis, George, Denise J. Cai, Sara C. Mednick, Alcino J. Silva, and Panayiota Poirazi (2015). *Synaptic clustering within dendrites: An emerging theory of memory formation*. DOI: 10.1016/j.pneurobio.2014.12.002. arXiv: 15334406.

Kien Tuong Phan, Tomas Henrique Maul, and Tuong Thuy Vu (2015). "A parallel circuit approach for improving the speed and generalization properties of neural networks". In: *2015 11th International Conference on Natural Computation (ICNC)*. IEEE, pp. 1–7. ISBN: 978-1-4673-7679-2. DOI: 10.1109/ICNC.2015.7377956.

Kim, Taeksoo, Moonsu Cha, Hyunsoo Kim, Jung Kwon Lee, and Jiwon Kim (2017). "Learning to Discover Cross-Domain Relations with Generative Adversarial Networks". In: arXiv: 1703.05192.

Kingma, Diederik P. and Jimmy Ba (2017). *Adam: A Method for Stochastic Optimization*. arXiv: 1412.6980 [cs.LG].

Kirk, David B and W Hwu Wen-Mei (2016). *Programming massively parallel processors: a hands-on approach*. Morgan kaufmann.

Kirkpatrick, James, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell (2016). "Overcoming catastrophic forgetting in neural networks". In: *arXiv preprint*. ISSN: 0027-8424. DOI: 10.1073/PNAS.1611835114. arXiv: 1612.00796.

Kormushev, Petar, Sylvain Calinon, and Darwin G Caldwell (2013). "Reinforcement Learning in Robotics: Applications and Real-World Challenges". In: *Robotics* 2.3, pp. 122–148. ISSN: 2218-6581. DOI: 10.3390/robotics2030122. URL: https://www.mdpi.com/2218-6581/2/3/122.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton (2012). *ImageNet Classification with Deep Convolutional Neural Networks*. URL: https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.

Kruschke, John K. (1991). "ALCOVE: A Connectionist Model of Human Category Learning". In: *Advances in Neural Information Processing Systems 3*. Ed. by R. P. Lippmann, J. E. Moody, and D. S. Touretzky. Morgan-Kaufmann, pp. 649–655. URL: http://papers.nips.cc/paper/416-alcove-a-connectionist-model-of-human-category-learning.pdf.

Kukačka, Jan, Vladimir Golkov, and Daniel Cremers (2017). *Regularization for Deep Learning: A Taxonomy*. arXiv: 1710.10686 [cs.LG].

Kumaran, Dharshan, Demis Hassabis, and James L. McClelland (2016). "What Learning Systems do Intelligent Agents Need? Complementary Learning Systems Theory Updated". In: *Trends in Cognitive Sciences* 20.7, pp. 512–534. ISSN: 13646613. DOI: 10.1016/j.tics.2016.05.004. URL: http://www.ncbi.nlm.nih.gov/pubmed/27315762https://linkinghub.elsevier.com/retrieve/pii/S1364661316300432.

Kyriakides, George and Konstantinos Margaritis (2020). "An Introduction to Neural Architecture Search for Convolutional Networks". In: *arXiv preprint arXiv:2005.11074*.

Lamb, Alex, Jonathan Binas, Anirudh Goyal, Dmitriy Serdyuk, Sandeep Subramanian, Ioannis Mitliagkas, and Yoshua Bengio (2018). "Fortified Networks: Improving the Robustness of Deep Networks by Modeling the Manifold of Hidden Representations". In: arXiv: 1804.02485. URL: http://arxiv.org/abs/1804.02485.

Larsson, Gustav, Michael Maire, and Gregory Shakhnarovich (2016). "FractalNet: Ultra-Deep Neural Networks without Residuals". In: arXiv: 1605.07648. URL: http://arxiv.org/abs/1605.07648.

LeCun, Y., B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel (1989). "Backpropagation Applied to Handwritten Zip Code Recognition". In: *Neural Computation* 1.4, pp. 541–551. ISSN: 0899-7667. DOI: 10.1162/neco.1989.1.4.541. URL: http://www.mitpressjournals.org/doi/10.1162/neco.1989.1.4.541.

Lecun, Y., L. Bottou, Y. Bengio, and P. Haffner (1998). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11, pp. 2278–2324. ISSN: 00189219. DOI: 10.1109/5.726791. URL: http://ieeexplore.ieee.org/document/726791/.

LeCun, Yann and Yoshua Bengio (1995). "Convolutional networks for images, speech, and time series". In: *The handbook of brain theory and neural networks* 3361.10, p. 1995.

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). "Deep learning". In: *nature* 521.7553, p. 436.

LeCun, Yann, Fu Jie Huang, and Léon Bottou (2004). "Learning methods for generic object recognition with invariance to pose and lighting". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Vol. 2. DOI: 10.1109/cvpr.2004.1315150.

Li, Junying, Deng Cai, and Xiaofei He (2017). *Learning Graph-Level Representation for Drug Discovery*. arXiv: 1709.03741 [cs.LG].

Lin, Min, Qiang Chen, and Shuicheng Yan (2013). "Network In Network". In: *arXiv preprint*, p. 10. ISSN: 03029743. DOI: 10.1109/ASRU.2015.7404828. arXiv: 1312.4400.

Liu, Hanxiao, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu (2017). "Hierarchical Representations for Efficient Architecture Search". In: arXiv: 1711.00436. URL: http://arxiv.org/abs/1711.00436.

Liu, Hanxiao, Karen Simonyan, and Yiming Yang (2018). "Darts: Differentiable architecture search". In: *arXiv preprint arXiv:1806.09055*.

Lodato, Simona and Paola Arlotta (2015). "Generating Neuronal Diversity in the Mammalian Cerebral Cortex". In: *Annual Review of Cell and Developmental Biology* 31.1, pp. 699–720. ISSN: 1081-0706. DOI: 10.1146/annurev-cellbio-100814-125353. arXiv: 15334406.

López-Muñoz, Francisco, Jesús Boya, and Cecilio Alamo (2006). "Neuron theory, the cornerstone of neuroscience, on the centenary of the Nobel Prize award to Santiago Ramón y Cajal". In: *Brain Research Bulletin* 70.4-6, pp. 391–405. ISSN: 03619230. DOI: 10.1016/j.brainresbull.2006.07.010.

Lu, Sidi, Yaoming Zhu, Weinan Zhang, Jun Wang, and Yong Yu (2018). *Neural Text Generation: Past, Present and Beyond*. arXiv: 1803.07133 [cs.CL].

Luong, Minh-Thang, Hieu Pham, and Christopher D. Manning (2015). *Effective Approaches to Attention-based Neural Machine Translation*. arXiv: 1508.04025 [cs.CL].

Lyu, Chunchuan, Kaizhu Huang, and Hai-Ning Liang (2015). "A Unified Gradient Regularization Family for Adversarial Examples". In: *2015 IEEE International Conference on Data Mining*. IEEE, pp. 301–309. ISBN: 978-1-4673-9504-5. DOI: 10.1109/ICDM.2015.84. URL: http://ieeexplore.ieee.org/document/7373334/.

McClelland, James L., Bruce L. McNaughton, and Randall C. O'Reilly (1995). "Why there are complementary learning systems in the hippocampus and neocortex: Insights from the successes and failures of connectionist models of learning and memory." In: *Psychological Review* 102.3, pp. 419–457. ISSN: 1939-1471. DOI: 10.1037/0033-295X.102.3.419. URL: http://www.ncbi.nlm.nih.gov/pubmed/7624455http://doi.apa.org/getdoi.cfm?doi=10.1037/0033-295X.102.3.419.

Mehrer, Johannes, Courtney J. Spoerer, Nikolaus Kriegeskorte, and Tim C. Kietzmann (2020). "Individual differences among deep neural network models". In: *bioRxiv* 3.1, p. 2020.01.08.898288. DOI: 10.1101/2020.01.08.898288.

Mehta, Sachin, Mohammad Rastegari, Linda Shapiro, and Hannaneh Hajishirzi (2019). "Espnetv2: A light-weight, power efficient, and general purpose convolutional neural network". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9190–9200.

Melin, Patricia, Alejandra Mancilla, Miguel Lopez, and Olivia Mendoza (2007). "A hybrid modular neural network architecture with fuzzy Sugeno integration for time series forecasting". In: *Applied Soft Computing* 7.4, pp. 1217–1226.

Melin, Patricia, Olivia Mendoza, and Oscar Castillo (2011). "Face recognition with an improved interval type-2 fuzzy logic sugeno integral and modular neural networks". In: *IEEE Transactions on systems, man, and cybernetics-Part A: systems and humans* 41.5, pp. 1001–1012.

Mendoza, Olivia, Patricia Melín, and Oscar Castillo (2009b). "Interval type-2 fuzzy logic and modular neural networks for face recognition applications". In: *Applied Soft Computing* 9.4, pp. 1377–1387. ISSN: 15684946. DOI: 10.1016/j.asoc.2009.06.007.

Mendoza, Olivia, Patricia Melin, and Guillermo Licea (2009a). "A hybrid approach for image recognition combining type-2 fuzzy logic, modular neural networks and the Sugeno integral". In: *Information Sciences* 179.13, pp. 2078–2101. ISSN: 00200255. DOI: 10.1016/j.ins.2008.11.018.

Meunier, David, Renaud Lambiotte, and Edward T. Bullmore (2010). *Modular and hierarchically modular organization of brain networks*. DOI: 10.3389/fnins.2010.00200.

Miikkulainen, Risto (2010). "Topology of a Neural Network". In: *Encyclopedia of Machine Learning*. Ed. by Claude Sammut and Geoffrey I Webb. Boston,

MA: Springer US, pp. 988–989. ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_837. URL: https://doi.org/10.1007/978-0-387-30164-8_837.

Miikkulainen, Risto, Jason Liang, Elliot Meyerson, Aditya Rawal, Dan Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, and Babak Hodjat (2017). "Evolving Deep Neural Networks". In: arXiv: 1703.00548.

Mikolov, Tomas, Kai Chen, Greg Corrado, and Jeffrey Dean (2013). *Efficient Estimation of Word Representations in Vector Space*. arXiv: 1301.3781 [cs.CL].

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, and Georg Ostrovski (2015). "Human-level control through deep reinforcement learning". In: *nature* 518.7540, pp. 529–533.

Montufar, Guido F., Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio (2014). *On the Number of Linear Regions of Deep Neural Networks*. URL: http://papers.nips.cc/paper/5422-on-the-number-of-linear-regions-of-deep-neural-networks.

Mountcastle, Vernon B. (1997). *The columnar organization of the neocortex*. DOI: 10.1093/brain/120.4.701.

Mouret, Jean-Baptiste and Stéphane Doncieux (2008). "MENNAG: a modular, regular and hierarchical encoding for neural-networks based on attribute grammars". In: *Evolutionary Intelligence* 1.3, pp. 187–207. ISSN: 1864-5909. DOI: 10.1007/s12065-008-0015-7.

— (2009). "Evolving modular neural-networks through exaptation". In: *2009 IEEE Congress on Evolutionary Computation, CEC 2009*, pp. 1570–1577. ISBN: 9781424429592. DOI: 10.1109/CEC.2009.4983129.

Murdock, Bennet B (1983). *A Distributed Memory Model for Serial-Order Information*. Tech. rep. 4, pp. 316–338.

Nardi, R. de, J. Togelius, O.E. Holland, and S.M. Lucas (2006). "Evolution of Neural Networks for Helicopter Control: Why Modularity Matters". In: *2006 IEEE International Conference on Evolutionary Computation*. IEEE, pp. 1799–1806. ISBN: 0-7803-9487-9. DOI: 10.1109/CEC.2006.1688525.

Neill, James O' (2018). "Siamese Capsule Networks". In: arXiv: 1805.07242. URL: http://arxiv.org/abs/1805.07242.

Newatia, S. and R. K. Aggarwal (2018). "Convolutional Neural Network for ASR". In: *2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA)*, pp. 638–642. DOI: 10.1109/ICECA.2018.8474688.

Newman, M E J (2004). "Detecting community structure in networks". In: *Eur. Phys. J. B* 38, pp. 321–330. DOI: 10.1140/epjb/e2004-00124-y.

Newman, M E J (2006). "Modularity and community structure in networks." In: *Proceedings of the National Academy of Sciences of the United States*

of America 103.23, pp. 8577–82. ISSN: 0027-8424. DOI: 10.1073/pnas.0601602103.

Newman, M. E. J. (2016). "Community detection in networks: Modularity optimization and maximum likelihood are equivalent". In: *Arvix* 1, pp. 1–8. ISSN: 2470-0045. DOI: 10.1103/PhysRevE.94.052315. arXiv: 1606.02319.

Nugraha, B. T., S. Su, and Fahmizal (2017). "Towards self-driving car using convolutional neural network and road lane detector". In: *2017 2nd International Conference on Automation, Cognitive Science, Optics, Micro Electro-Mechanical System, and Information Technology (ICACOMIT)*, pp. 65–69. DOI: 10.1109/ICACOMIT.2017.8253388.

Oh, Il-Seok and Ching Y. Suen (2002). "A class-modular feedforward neural network for handwriting recognition". In: *Pattern Recognition* 35.1, pp. 229–244. ISSN: 00313203. DOI: 10.1016/S0031-3203(00)00181-3.

Omar, M Azhar, M Khair Hassan, Azura Che Soh, and MZA Ab Kadir (2013). "Lightning severity classification utilizing the meteorological parameters: A neural network approach". In: *2013 IEEE International Conference on Control System, Computing and Engineering*. IEEE, pp. 111–116.

Oord, Aaron van den, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu (2016). *WaveNet: A Generative Model for Raw Audio*. arXiv: 1609.03499 [cs.SD].

Ortín, S., J.M. Gutiérrez, L. Pesquera, and H. Vasquez (2005). "Nonlinear dynamics extraction for time-delay systems using modular neural networks synchronization and prediction". In: *Physica A: Statistical Mechanics and its Applications* 351.1, pp. 133–141. ISSN: 03784371. DOI: 10.1016/j.physa.2004.12.015.

Otsuna, Hideo, Kazunori Shinomiya, and Kei Ito (2014). "Parallel neural pathways in higher visual centers of the Drosophila brain that mediate wavelength-specific behavior." In: *Frontiers in neural circuits* 8, p. 8. ISSN: 1662-5110. DOI: 10.3389/fncir.2014.00008. URL: http://www.ncbi.nlm.nih.gov/pubmed/24574974http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC3918591.

Ou, Guobin and Yi Lu Murphey (2007). "Multi-class pattern classification using neural networks". In: *Pattern Recognition* 40.1, pp. 4–18. ISSN: 00313203. DOI: 10.1016/j.patcog.2006.04.041.

Padhy, Ram Prasad, Sachin Verma, Shahzad Ahmad, Suman Kumar Choudhury, and Pankaj Kumar Sa (2018). "Deep Neural Network for Autonomous UAV Navigation in Indoor Corridor Environments". In: *Procedia Computer Science* 133, pp. 643–650. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2018.07.099. URL: http://www.sciencedirect.com/science/article/pii/S1877050918310524.

Pan, Pingbo, Zhongwen Xu, Yi Yang, Fei Wu, and Yueting Zhuang (2016). "Hierarchical Recurrent Neural Encoder for Video Representation With Application to Captioning". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Papernot, Nicolas and Patrick McDaniel (2017). "Extending Defensive Distillation". In: arXiv: 1705.05264. URL: http://arxiv.org/abs/1705.05264.

Papernot, Nicolas, Patrick McDaniel, and Ian Goodfellow (2016). "Transferability in Machine Learning: from Phenomena to Black-Box Attacks using Adversarial Samples". In: arXiv: 1605.07277. URL: http://arxiv.org/abs/1605.07277.

Papernot, Nicolas, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami (2016). "Distillation as a Defense to Adversarial Perturbations Against Deep Neural Networks". In: *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, pp. 582–597. ISBN: 978-1-5090-0824-7. DOI: 10.1109/SP.2016.41. URL: http://ieeexplore.ieee.org/document/7546524/.

Paszke, Adam, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala (2019). "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d' Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Paul, Sandeep and Lotika Singh (2015). "A review on advances in deep learning". In: *2015 IEEE Workshop on Computational Intelligence: Theories, Applications and Future Directions (WCI)*. IEEE, pp. 1–6.

Phan, Kien Tuong, Tomas Henrique Maul, Tuong Thuy Vu, and Lai Weng Kin (2016). "Improving Neural Network Generalization by Combining Parallel Circuits with Dropout". In: DOI: 10.1007/978-3-319-46675-0_63. arXiv: 1612.04970. URL: http://arxiv.org/abs/1612.04970http://dx.doi.org/10.1007/978-3-319-46675-0{_}63.

Phan, Kien Tuong, Tomas Henrique Maul, Tuong Thuy Vu, and Weng Kin Lai (2017). "DropCircuit: A Modular Regularizer for Parallel Circuit Networks". In: *Neural Processing Letters*, pp. 1–18.

— (2018). "DropCircuit : A Modular Regularizer for Parallel Circuit Networks". In: *Neural Processing Letters* 47.3, pp. 841–858. ISSN: 1370-4621. DOI: 10.1007/s11063-017-9677-4. URL: http://link.springer.com/10.1007/s11063-017-9677-4.

Phaye, Sai Samarth R, Apoorva Sikka, Abhinav Dhall, and Deepti Bathula (2018). "Dense and Diverse Capsule Networks: Making the Capsules Learn Better". In: arXiv: 1805.04001. URL: http://arxiv.org/abs/1805.04001.

Phyo Phyo San, Sai Ho Ling, and H. T. Nguyen (2011). "Block based neural network for hypoglycemia detection". In: *2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*. IEEE, pp. 5666–5669. ISBN: 978-1-4577-1589-1. DOI: 10.1109/IEMBS.2011.6091371.

Polikar, Robi, Lalita Udpa, Vasant Honavar, Senior Member, and Satish S Udpa (2001). *Learn++: An Incremental Learning Algorithm for Supervised Neural Networks mHealth View project Analysis of RNA-protein interactions View project Learn++: An Incremental Learning Algorithm for Supervised Neural Networks*. Tech. rep. 4. URL: https://www.researchgate.net/publication/2489080.

Pontes-Filho, Sidney and Marcus Liwicki (2018). "Bidirectional Learning for Robust Neural Networks". In: arXiv: 1805.08006. URL: http://arxiv.org/abs/1805.08006.

Purves, D, GJ Augustine, D Fitzpatrick, WC Hall, AS LaMantia, JO McNamara, and SM Williams (2004). "Neuroscience. 3rd". In: *Massachusetts: Sinauer Associates Inc Publishers*.

Radicchi, Filippo, Claudio Castellano, Federico Cecconi, Vittorio Loreto, and Domenico Parisi (2004). "Defining and identifying communities in networks". In: *Proceedings of the National Academy of Sciences of the United States of America* 101.9, pp. 2658–2663.

Ranasinghe, Tharindu, Constantin Orasan, and Ruslan Mitkov (Sept. 2019). "Semantic Textual Similarity with Siamese Neural Networks". In: *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2019)*. Varna, Bulgaria: INCOMA Ltd., pp. 1004–1011. DOI: 10.26615/978-954-452-056-4_116. URL: https://www.aclweb.org/anthology/R19-1116.

Reisinger, Joseph, Kenneth O Stanley, and Risto Miikkulainen (2004). "Evolving reusable neural modules". In: *Genetic and Evolutionary Computation Conference*. Springer, pp. 69–81.

Ronco, Eric and Peter Gawthrop (1995). "Modular neural networks: a state of the art". In: *Rapport Technique CSC-95026, Center of System and Control, University of Glasgow. http://www. mech. gla. ac. uk/control/report. html.*

Ronen, M, Y Shabtai, and H Guterman (2002). "Hybrid model building methodology using unsupervised fuzzy clustering and supervised neural networks". In: *Biotechnol Bioeng* 77.4, pp. 420–429.

Ronneberger, Olaf, Philipp Fischer, and Thomas Brox (2015). *U-Net: Convolutional Networks for Biomedical Image Segmentation*. arXiv: 1505.04597 [cs.CV].

Rosenbaum, Clemens, Ignacio Cases, Matthew Riemer, and Tim Klinger (2019). *Routing Networks and the Challenges of Modular and Compositional Computation*. arXiv: 1904.12774 [cs.LG].

Rosenbaum, Clemens, Tim Klinger, and Matthew Riemer (2017). *Routing Networks: Adaptive Selection of Non-linear Functions for Multi-Task Learning*. arXiv: 1711.01239 [cs.LG].

Rudasi, L. and S.A. Zahorian (1991). "Text-independent talker identification with neural networks". In: *[Proceedings] ICASSP 91: 1991 International Conference on Acoustics, Speech, and Signal Processing*. IEEE, 389–392 vol.1. ISBN: 0-7803-0003-3. DOI: 10.1109/ICASSP.1991.150358.

Ruder, Sebastian (2017). *An overview of gradient descent optimization algorithms*. arXiv: 1609.04747 [cs.LG].

Sabour, Sara, Nicholas Frosst, and Geoffrey E Hinton (2017). "Dynamic Routing Between Capsules". In: arXiv: 1710.09829. URL: http://arxiv.org/abs/1710.09829.

Sandler, Mark, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen (2018). "Mobilenetv2: Inverted residuals and linear bottlenecks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520.

Sang-Woo Moon and Seong-Gon Kong (2001). "Block-based neural networks". In: *IEEE Transactions on Neural Networks* 12.2, pp. 307–317. ISSN: 10459227. DOI: 10.1109/72.914525.

Santoro, Adam, David Raposo, David G. T. Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Timothy Lillicrap (2017). "A simple neural network module for relational reasoning". In: arXiv: 1706.01427. URL: http://arxiv.org/abs/1706.01427.

Saxena, Shreyas and Jakob Verbeek (2016). "Convolutional Neural Fabrics". In: *Advances in Neural Information Processing Systems*, pp. 4060–4068. arXiv: 1606.02492. URL: http://arxiv.org/abs/1606.02492.

Schwarz, Adam J., Alessandro Gozzi, and Angelo Bifone (2008). "Community structure and modularity in networks of correlated brain activity". In: *Magnetic Resonance Imaging* 26.7, pp. 914–920. ISSN: 0730725X. DOI: 10.1016/j.mri.2008.01.048. arXiv: 0701041v2 [arXiv:q-bio.NC].

Seltzer, Michael L., Dong Yu, and Yongqiang Wang (2013). "An investigation of deep neural networks for noise robust speech recognition". In: *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, pp. 7398–7402. ISBN: 978-1-4799-0356-6. DOI: 10.1109/ICASSP.2013.6639100. URL: http://ieeexplore.ieee.org/document/6639100/.

Selvaraju, Ramprasaath R., Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra (2016). "Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization". In: arXiv: 1610.02391. URL: http://arxiv.org/abs/1610.02391.

Serban, Iulian V., Alessandro Sordoni, Yoshua Bengio, Aaron Courville, and Joelle Pineau (2016). "Building End-To-End Dialogue Systems Using Generative Hierarchical Neural Network Models". In: *Aaai*, p. 8. ISSN: 10495258. DOI: 10.1017/CBO9781107415324.004. arXiv: 1507.04808.

Shaham, Uri, Yutaro Yamada, and Sahand Negahban (2015). "Understanding Adversarial Training: Increasing Local Stability of Neural Nets through Robust Optimization". In: DOI: 10.1016/j.neucom.2018.04.027. arXiv: 1511.05432. URL: http://arxiv.org/abs/1511.05432http://dx.doi.org/10.1016/j.neucom.2018.04.027.

Sharkey, A J C (1996). "On Combining Artificial Neural Nets". In: *Connection Science* 8.3-4, pp. 299–313. ISSN: 09540091 (ISSN). DOI: 10.1080/095400996116785.

Shetty, Rakshith and Jorma Laaksonen (2015). "Video captioning with recurrent networks based on frame- and video-level features and visual content classification". In: arXiv: 1512.02949.

Shi, Qinfeng, James Petterson, Gideon Dror, John Langford, Alex Smola, and SVN Vishwanathan (2009). "Hash kernels for structured data". In: *The Journal of Machine Learning Research* 10, pp. 2615–2637.

Shin, Hanul, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim (2017). "Continual learning with deep generative replay". In: *Advances in neural information processing systems*, pp. 2990–2999.

Silver, David, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, and Marc Lanctot (2016). "Mastering the game of Go with deep neural networks and tree search". In: *nature* 529.7587, pp. 484–489.

Silver, David, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, and Thore Graepel (2017a). "Mastering chess and shogi by self-play with a general reinforcement learning algorithm". In: *arXiv:1712.01815*.

Silver, David, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, and Adrian Bolton (2017b). "Mastering the game of go without human knowledge". In: *nature* 550.7676, pp. 354–359.

Singh, Saurabh, Derek Hoiem, and David Forsyth (2016). "Swapout: Learning an ensemble of deep architectures". In: *Advances in Neural Information Processing Systems*, pp. 28–36.

Sinha, Ayan, Zhao Chen, Vijay Badrinarayanan, and Andrew Rabinovich (2018). "Gradient Adversarial Training of Neural Networks". In: arXiv: 1806.08028. URL: http://arxiv.org/abs/1806.08028.

Sohoni, Nimit Sharad, Christopher Richard Aberger, Megan Leszczynski, Jian Zhang, and Christopher Ré (2019). "Low-memory neural network training: A technical report". In: *arXiv preprint arXiv:1904.10631*.

Song, Linfeng, Yue Zhang, Zhiguo Wang, and Daniel Gildea (2018). "A Graph-to-Sequence Model for AMR-to-Text Generation". In: arXiv: 1805.02473. URL: http://arxiv.org/abs/1805.02473.

Soutner, Daniel and Luděk Müller (2013). "Application of LSTM neural networks in language modelling". In: *International Conference on Text, Speech and Dialogue*. Springer, pp. 105–112.

Sporns, Olaf (2011). *The human connectome: A complex network*. DOI: 10.1111/j.1749-6632.2010.05888.x.

Sporns, Olaf and Jonathan D. Zwi (2004). "The Small World of the Cerebral Cortex". In: *Neuroinformatics* 2.2, pp. 145–162. ISSN: 1539-2791. DOI: 10.1385/NI:2:2:145.

Srivastava, Nitish, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov (2014). "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15, pp. 1929–1958. ISSN: 15337928. DOI: 10.1214/12-AOS1000. arXiv: 1102.4807.

Srivastava, Rupesh Kumar, Klaus Greff, and Jürgen Schmidhuber (2015). "Highway Networks". In: *arXiv:1505.00387 [cs]*. arXiv: 1505.00387.

Srivastava, Rupesh Kumar, Jonathan Masci, Sohrob Kazerounian, Faustino Gomez, and Jürgen Schmidhuber (2013). "Compete to Compute". In: *Nips*, pp. 2310–2318. ISSN: 10495258.

Stanley, Kenneth O., David B. D'Ambrosio, and Jason Gauci (2009). "A Hypercube-Based Encoding for Evolving Large-Scale Neural Networks". In: *Artificial Life* 15.2, pp. 185–212. ISSN: 1064-5462. DOI: 10.1162/artl.2009.15.2.15202.

Stanley, Kenneth O. and Risto Miikkulainen (2002). "Evolving Neural Networks through Augmenting Topologies". In: *Evolutionary Computation* 10.2, pp. 99–127. ISSN: 1063-6560. DOI: 10.1162/106365602320169811. arXiv: 1407.0576.

Stollenga, Marijn F, Wonmin Byeon, Marcus Liwicki, and Juergen Schmidhuber (2015). "Parallel Multi-Dimensional LSTM, With Application to Fast Biomedical Volumetric Image Segmentation". In: *Advances in Neural Information Processing Systems 28*. Ed. by C Cortes, N D Lawrence, D D Lee, M Sugiyama, and R Garnett. Curran Associates, Inc., pp. 2998–3006.

Subirats, José L., José M. Jerez, Iván Gómez, and Leonardo Franco (2010). "Multiclass Pattern Recognition Extension for the New C-Mantec Constructive Neural Network Algorithm". In: *Cognitive Computation* 2.4, pp. 285–290. ISSN: 18669956. DOI: 10.1007/s12559-010-9051-6.

Sun, Zhun, Mete Ozay, and Takayuki Okatani (2017). "HyperNetworks with statistical filtering for defending adversarial examples". In: arXiv: 1711.01791. URL: http://arxiv.org/abs/1711.01791.

Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich (2015a). "Going deeper with convolutions". In: *Proceedings of the IEEE*

Computer Society Conference on Computer Vision and Pattern Recognition. Vol. 07-12-June, pp. 1–9. ISBN: 9781467369640. DOI: 10.1109/CVPR.2015.7298594. arXiv: 1409.4842.

Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich (2015b). "Going deeper with convolutions". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Vol. 07-12-June, pp. 1–9. ISBN: 9781467369640. DOI: 10.1109/CVPR.2015.7298594. arXiv: 1409.4842.

Szegedy, Christian, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna (2016). "Rethinking the inception architecture for computer vision". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2818–2826.

Szegedy, Christian, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus (2013). "Intriguing properties of neural networks". In: arXiv: 1312.6199. URL: https://arxiv.org/abs/1312.6199.

Taigman, Yaniv, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf (2014). *DeepFace: Closing the Gap to Human-Level Performance in Face Verification*.

Takahashi, Naoya, Shota Inoue, and Yuki Mitsufuji (2021). "Adversarial attacks on audio source separation". In: *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, pp. 521–525.

Tan, Mingxing, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le (2019). "Mnasnet: Platform-aware neural architecture search for mobile". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2820–2828.

Tan, Mingxing and Quoc V Le (2019). "Efficientnet: Rethinking model scaling for convolutional neural networks". In: *arXiv preprint arXiv:1905.11946*.

Tang, Kevin, Manohar Paluri, Li Fei-Fei, Rob Fergus, and Lubomir Bourdev (2015). "Improving Image Classification with Location Context". In: arXiv: 1505.03873. URL: https://arxiv.org/abs/1505.03873.

Terekhov, Alexander V., Guglielmo Montone, and J. Kevin O'Regan (2015). "Knowledge Transfer in Deep Block-Modular Neural Networks". In: Springer, Cham, pp. 268–279. DOI: 10.1007/978-3-319-22979-9_27.

Tramèr, Florian, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel (2017). "Ensemble Adversarial Training: Attacks and Defenses". In: arXiv: 1705.07204. URL: http://arxiv.org/abs/1705.07204.

Tyler, Joshua R., Dennis M. Wilkinson, and Bernardo A. Huberman (2005). "E-Mail as Spectroscopy: Automated Discovery of Community Structure within

Organizations". In: *The Information Society* 21.2, pp. 143–153. ISSN: 0197-2243. DOI: 10.1080/01972240590925348. arXiv: 0303264 [cond-mat].

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin (2017). *Attention Is All You Need*. arXiv: 1706.03762 [cs.CL].

Veit, Andreas, Michael J Wilber, and Serge Belongie (2016). "Residual networks behave like ensembles of relatively shallow networks". In: *Advances in Neural Information Processing Systems*, pp. 550–558.

Ven, Gido M. van de and Andreas S. Tolias (2018). "Generative replay with feedback connections as a general strategy for continual learning". In: arXiv: 1809.10635. URL: http://arxiv.org/abs/1809.10635.

Verbancsics, Phillip and Kenneth O Stanley (2011). "Constraining connectivity to encourage modularity in HyperNEAT". In: *Proceedings of the 13th annual conference on Genetic and evolutionary computation - GECCO '11*, p. 1483. DOI: 10.1145/2001576.2001776.

Vinyals, Oriol, Lukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton (2014). "Grammar as a Foreign Language". In: arXiv: 1412.7449. URL: http://arxiv.org/abs/1412.7449.

Vitoria, Patricia, Lara Raad, and Coloma Ballester (2020). *ChromaGAN: Adversarial Picture Colorization with Semantic Class Distribution*. arXiv: 1907.09837 [cs.CV].

Vlahogianni, Eleni I, Matthew G Karlaftis, and John C Golias (2007). "Spatio-Temporal Short-Term Urban Traffic Volume Forecasting Using Genetically Optimized Modular Networks". In: *Computer-Aided Civil and Infrastructure Engineering* 22.5, pp. 317–325.

Wadawadagi, Ramesh and Veerappa Pagi (2020). "Sentiment analysis with deep neural networks: comparative study and performance assessment". In: *Artificial Intelligence Review*.

Waibel, Alex (1989). "Modular Construction of Time-Delay Neural Networks for Speech Recognition". In: *Neural Computation* 1.1, pp. 39–46. ISSN: 0899-7667. DOI: 10.1162/neco.1989.1.1.39. URL: http://www.mitpressjournals.org/doi/10.1162/neco.1989.1.1.39.

Wan, Li, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus (2013). "Regularization of neural networks using dropconnect". In: *International conference on machine learning*, pp. 1058–1066.

Wang, Jinjiang, Junfei Zhuang, Lixiang Duan, and Weidong Cheng (2016). "A multi-scale convolution neural network for featureless fault diagnosis". In: *2016 International Symposium on Flexible Automation (ISFA)*. IEEE, pp. 65–70.

Wang, Mingming (2015). "Multi-path Convolutional Neural Networks for Complex Image Classification". In: arXiv: 1506.04701.

Wang, Sheng-Jun, Claus C Hilgetag, and Changsong Zhou (2011). "Sustained activity in hierarchical modular neural networks: self-organized criticality and oscillations". In: *Frontiers in computational neuroscience* 5.

Wang, Tao, David J Wu, Adam Coates, and Andrew Y Ng (2012). "End-to-end text recognition with convolutional neural networks". In: *Pattern Recognition (ICPR), 2012 21st International Conference on*. IEEE, pp. 3304–3308.

Wang, Yaqing, Quanming Yao, James Kwok, and Lionel M. Ni (2020). *Generalizing from a Few Examples: A Survey on Few-Shot Learning*. arXiv: 1904.05046 [cs.LG].

Wang, Zhenyi and Olga Veksler (2018). "Location Augmentation for CNN". In: arXiv: 1807.07044. URL: https://arxiv.org/abs/1807.07044.

Watanabe, Chihiro, Kaoru Hiramatsu, and Kunio Kashino (2018). "Modular representation of layered neural networks". In: *Neural Networks* 97, pp. 62–73. ISSN: 0893-6080. DOI: 10.1016/J.NEUNET.2017.09.017. URL: https://www.sciencedirect.com/science/article/pii/S0893608017302319.

Watts, Duncan J. (1999). "Networks, Dynamics, and the Small-World Phenomenon". In: *American Journal of Sociology* 105.2, pp. 493–527. ISSN: 0002-9602. DOI: 10.1086/210318. arXiv: 9910332 [cond-mat].

Webb, Andrew, Charles Reynolds, Wenlin Chen, Henry Reeve, Dan Iliescu, Mikel Lujan, and Gavin Brown (2020). "To Ensemble or Not Ensemble: When does End-To-End Training Fail?" In: *stat* 1050, p. 6.

Wei Jiang and G. Seong Kong (2007). "Block-Based Neural Networks for Personalized ECG Signal Classification". In: *IEEE Transactions on Neural Networks* 18.6, pp. 1750–1761. ISSN: 1045-9227. DOI: 10.1109/TNN.2007.900239.

Weinberger, Kilian, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg (2009). "Feature hashing for large scale multitask learning". In: *Proceedings of the 26th annual international conference on machine learning*, pp. 1113–1120.

Weston, Jason, Sumit Chopra, and Antoine Bordes (2014). "Memory Networks". In: arXiv: 1410.3916.

Wu, Yonghui, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean (2016). "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation". In: abs/1609.08144. URL: http://arxiv.org/abs/1609.08144.

Xiang, Canqun, Lu Zhang, Yi Tang, Wenbin Zou, and Chen Xu (2018). "MS-CapsNet: A Novel Multi-Scale Capsule Network". In: *IEEE Signal Processing*

Letters 25.12, pp. 1850–1854. ISSN: 1070-9908. DOI: 10.1109/LSP.2018.2873892. URL: https://ieeexplore.ieee.org/document/8481393/.

Xiao, Dengyu, Yixiang Huang, Xudong Zhang, Haotian Shi, Chengliang Liu, and Yanming Li (2018). "Fault diagnosis of asynchronous motors based on LSTM neural network". In: *2018 prognostics and system health management conference (PHM-Chongqing)*. IEEE, pp. 540–545.

Xie, Saining, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He (2016). "Aggregated residual transformations for deep neural networks". In: *arXiv preprint arXiv:1611.05431*.

Xu, Kelvin, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard Zemel, and Yoshua Bengio (2015). "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention". In: arXiv: 1502.03044. URL: http://arxiv.org/abs/1502.03044.

Xu, L., A. Krzyzak, and C.Y. Suen (1992). "Methods of combining multiple classifiers and their applications to handwriting recognition". In: *IEEE Transactions on Systems, Man, and Cybernetics* 22.3, pp. 418–435. ISSN: 00189472. DOI: 10.1109/21.155943.

Xu, Weilin, David Evans, and Yanjun Qi (2017). "Feature Squeezing: Detecting Adversarial Examples in Deep Neural Networks". In: DOI: 10.14722/ndss.2018.23198. arXiv: 1704.01155. URL: http://arxiv.org/abs/1704.01155http://dx.doi.org/10.14722/ndss.2018.23198.

Xu, Yuhui, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong (2019). "Pc-darts: Partial channel connections for memory-efficient differentiable architecture search". In: *arXiv:1907.05737*.

Yang, Luting, Bingqian Lu, and Shaolei Ren (2020). "A Note on Latency Variability of Deep Neural Networks for Mobile Inference". In: *arXiv preprint arXiv:2003.00138*.

Yang, Zichao, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy (June 2016). "Hierarchical Attention Networks for Document Classification". In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. San Diego, California: Association for Computational Linguistics, pp. 1480–1489. DOI: 10.18653/v1/N16-1174. URL: https://www.aclweb.org/anthology/N16-1174.

Yu, Haonan, Jiang Wang, Zhiheng Huang, Yi Yang, and Wei Xu (2016). "Video Paragraph Captioning Using Hierarchical Recurrent Neural Networks". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Yu, Jiahui, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang (2018a). "Slimmable neural networks". In: *arXiv:1812.08928*.

Yu, Jun, Jing Li, Zhou Yu, and Qingming Huang (2019a). "Multimodal Transformer with Multi-View Visual Representation for Image Captioning". In: *IEEE Transactions on Circuits and Systems for Video Technology*, pp. 1–1.

ISSN: 1051-8215. DOI: 10.1109/tcsvt.2019.2947482. arXiv: 1905.07841. URL: http://arxiv.org/abs/1905.07841.

Yu, Jun, Min Tan, Hongyuan Zhang, Dacheng Tao, and Yong Rui (2019b). "Hierarchical Deep Click Feature Prediction for Fine-grained Image Recognition". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1. ISSN: 0162-8828. DOI: 10.1109/tpami.2019.2932058.

Yu, Licheng, Zhe Lin, Xiaohui Shen, Jimei Yang, Xin Lu, Mohit Bansal, and Tamara L. Berg (2018b). "MAttNet: Modular Attention Network for Referring Expression Comprehension". In: arXiv: 1801.08186. URL: https://arxiv.org/abs/1801.08186v2.

Yuan, Ming and Yi Lin (2006). "Model selection and estimation in regression with grouped variables". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 68.1, pp. 49–67. ISSN: 1467-9868.

Zeiler, Matthew D. (2012). *ADADELTA: An Adaptive Learning Rate Method*. arXiv: 1212.5701 [cs.LG].

Zeng, Yu, Zhe Lin, Jimei Yang, Jianming Zhang, Eli Shechtman, and Huchuan Lu (2020). *High-Resolution Image Inpainting with Iterative Confidence Feedback and Guided Upsampling*. arXiv: 2005.11742 [cs.CV].

Zenke, Friedemann, Ben Poole, and Surya Ganguli (2017). "Continual Learning Through Synaptic Intelligence". In: arXiv: 1703.04200. URL: http://arxiv.org/abs/1703.04200.

Zhang, Chris, Mengye Ren, and Raquel Urtasun (2018). "Graph HyperNetworks for Neural Architecture Search". In: arXiv: 1810.05749. URL: http://arxiv.org/abs/1810.05749.

Zhang, Fangyi, Jürgen Leitner, Michael Milford, and Peter Corke (2016a). "Modular Deep Q Networks for Sim-to-real Transfer of Visuo-motor Policies". In: arXiv: 1610.06781. URL: https://arxiv.org/abs/1610.06781v4.

Zhang, Jian, Jun Yu, and Dacheng Tao (2018). "Local Deep-Feature Alignment for Unsupervised Dimension Reduction". In: *IEEE Transactions on Image Processing* 27.5, pp. 2420–2432. ISSN: 10577149. DOI: 10.1109/TIP.2018.2804218. arXiv: 1904.09747.

Zhang, Kaipeng, Zhanpeng Zhang, Zhifeng Li, and Yu Qiao (2016b). "Joint Face Detection and Alignment Using Multitask Cascaded Convolutional Networks". In: *IEEE Signal Processing Letters* 23.10, pp. 1499–1503. ISSN: 1558-2361. DOI: 10.1109/lsp.2016.2603342. URL: http://dx.doi.org/10.1109/LSP.2016.2603342.

Zhang, Ning, Jeff Donahue, Ross Girshick, and Trevor Darrell (2014). "Part-based R-CNNs for fine-grained category detection". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 8689 LNCS. PART 1, pp. 834–849. ISBN: 9783319105895. DOI: 10.1007/978-3-319-10590-1_54. arXiv: 1407.3867.

Zhang, Xingpeng, Sheng Huang, Xiaohong Zhang, Wei Wang, Qiuli Wang, and Dan Yang (2018). "Residual Inception: A New Module Combining Modified Residual with Inception to Improve Network Performance". In: *2018 25th IEEE International Conference on Image Processing (ICIP)*. IEEE, pp. 3039–3043. ISBN: 978-1-4799-7061-2. DOI: 10.1109/ICIP.2018.8451515. URL: https://ieeexplore.ieee.org/document/8451515/.

Zhang, Xingxing, Furu Wei, and Ming Zhou (2019). *HIBERT: Document Level Pre-training of Hierarchical Bidirectional Transformers for Document Summarization*. arXiv: 1905.06566 [cs.CL].

Zhang, Xuanyang, Hao Liu, Zhanxing Zhu, and Zenglin Xu (2020). "Learning to Search Efficient DenseNet with Layer-wise Pruning". In: *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, pp. 1–8.

Zhao, Yiyang, Linnan Wang, Yuandong Tian, Rodrigo Fonseca, and Tian Guo (2020). "Few-shot neural architecture search". In: *arXiv:2006.06863*.

Zheng, Weizhong, Der-Horng Lee, and Qixin Shi (2006). "Short-Term Freeway Traffic Flow Prediction: Bayesian Combined Neural Network Approach". In: *Journal of Transportation Engineering* 132.2, pp. 114–121. ISSN: 0733-947X. DOI: 10.1061/(ASCE)0733-947X(2006)132:2(114).

Zhou, Jie, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun (2019). *Graph Neural Networks: A Review of Methods and Applications*. arXiv: 1812.08434 [cs.LG].

Zoph, Barret and Quoc V. Le (2016). "Neural Architecture Search with Reinforcement Learning". In: *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*. arXiv: 1611.01578. URL: http://arxiv.org/abs/1611.01578.

Zoph, Barret, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le (2018). "Learning transferable architectures for scalable image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8697–8710.