DANIEL TURNER

Continual Learning
for
Object and Scene Classification

UNIVERSIDADE DO ALGARVE
Instituto Superior de Engenharia
2020

DANIEL TURNER

Continual Learning
for
Object and Scene Classification

Master's Thesis in Electrical and Electronic Engineering
Specialisation in Information Technologies and Telecommunications

Work done under the supervision of:
Professor Doutor João Miguel Fernandes Rodrigues
Professor Doutor Pedro Jorge Sequeira Cardoso

UNIVERSIDADE DO ALGARVE
Instituto Superior de Engenharia
2020

Continual Learning for Object and Scene Classification

Declaração de autoria de trabalho

Declaro ser o autor deste trabalho, que é original e inédito. Autores e trabalhos consultados estão devidamente citados no texto e constam da listagem de referências incluída.

I hereby declare to be the author of this work, which is original and unpublished. Authors and works consulted are properly cited in the text and included in the reference list.

_____

(Daniel Turner)

Abstract

Since their existence, computers have been a great asset to mankind, primarily because of their ability to perform specific tasks at speeds humans could never compete with. However, there are many tasks that humans consider easy which are quite difficult for computers to perform. For instance, a human can be shown a picture of an automobile and a bicycle and then be able to easily discriminate between future automobiles and bicycles. For a computer to perform such a task using current algorithms, typically, it must first be shown a large number of images of the two classes, with varying features and positions, and then spend a great deal of time learning to extract and identify features so that it can successfully distinguish between the two. Nevertheless, it is still able to perform the task (eventually) and, after the computational training is complete, would be able to classify images of automobiles and bicycles faster, and sometimes better, than the human. Nonetheless, the real out-performance displayed by the human is when another class is added to the mix, e.g., “aeroplane”. The human can immediately add aeroplanes to its set of known objects, whereas a computer would typically have to go almost back to the start and re-learn all the classes from scratch. The reason the network requires to be retrained is because of a phenomenon named Catastrophic Forgetting, where the changes made to the system during the acquisition of new knowledge bring about the loss of previous knowledge. In this dissertation, we explore Continual Learning, where we propose a way to deal with Catastrophic Forgetting by making a framework capable of learning new information without having to start from scratch and even “improving” its knowledge on what it already knows. With the above in mind, we implemented a Modular Dynamic Neural Network (MDNN) framework, which is primarily made up of modular sub-networks and progressively grows and re-arranges itself as it learns continuously. The network is structured in such a way that its internal components function independently from one another so that when new information is learned, only specific sub-networks are altered in a way that most of the old information is not forgotten. The network is divided into two main blocks, the feature extraction component which is based on a ResNet50 and the modular dynamic classification sub-networks. We have, so far, achieved results below those of the state of the art using ImageNet and CIFAR10, nevertheless, we demonstrate that the framework can meet its initial purpose, which is learning new information without having to start from scratch.

Resumo

Desde a sua invenção que os computadores têm sido uma excecional ferramenta para a humanidade, principalmente dada a sua capacidade de realizar tarefas específicas, a velocidades que os humanos nunca poderão atingir. Embora os computadores atuais possam superar os humanos de muitas formas, o oposto também é verdade. Existem muitas tarefas que os humanos consideram fáceis de executar, mas que para os computadores são bastante difíceis. Um ser humano, por exemplo, pode ver uma imagem de um automóvel e outra de uma bicicleta e ser instantaneamente capaz de distinguir (praticamente sem erros) entre futuros automóveis e futuras bicicletas com que venha a ser apresentado. Por outro lado, para um computador realizar tal tarefa é tipicamente necessário que este seja apresentado a um grande número de imagens das duas classes, com características e posições variadas e, a seguir, passar bastante tempo a aprender a extrair e identificar atributos que depois são utilizados para distinguir entre as duas classes. Eventualmente, o computador será capaz de realizar a tarefa e, após terminada a aprendizagem, consegue classificar grandes números de imagens de automóveis e bicicletas mais rapidamente (e às vezes melhor) do que o próprio ser humano.

No entanto, é quando uma nova classe é adicionada ao conjunto de classes a conhecer, como por exemplo “avião”, que os humanos mostram uma capacidade superior. O humano pode adicionar aviões imediatamente ao seu conjunto de objetos conhecidos, enquanto que um computador, usando os métodos de aprendizagem, teria tipicamente de voltar quase ao início e reaprender todas as classes do zero. Nestes métodos incluímos principalmente o uso de redes neuronais artificiais (*Artificial Neural Networks*), que têm demonstrado ser dos métodos de aprendizagem de máquina com melhor performance para estes tipos de problemas. Considerando, pois, o uso destas redes, a razão pela qual os métodos de aprendizagem tipicamente necessitam que uma rede seja treinada novamente perante novas classes, é um fenómeno usualmente designado de *Catastrophic Forgetting*. Neste fenómeno, as mudanças feitas ao sistema durante a aquisição de novos conhecimentos resultam numa perda de conhecimentos anteriores.

É neste contexto que nesta dissertação exploramos métodos usualmente designados de *Continual Learning*, onde se investigam maneiras de lidar com o referido fenómeno, desenvolvendo um sistema capaz de aprender novas informações sem a necessidade de começar do zero.

Continual Learning é na literatura ainda um tópico recente. Dada, pois, a novidade destes métodos, tanto quanto nos foi possível aferir, ainda não existe uma categorização bem definida, o que leva a que múltiplos estudos façam as categorizações dos métodos de formas distintas. Neste trabalho consideramos as arquiteturas dinâmicas como uma forma de resolver o problema de *Catastrophic Forgetting*. Neste caso, a forma destes métodos não perderem informação adquirida consiste no "crescimento" e adaptação da arquitetura implementada. Ou seja, de um modo geral, as adições de novas classes levam à inclusão de novos componentes no sistema que serão responsáveis pela nova informação. Um possível problema neste tipo de solução, que dá origem a não serem consideradas viáveis por alguns estudos, é que alguns dos sistemas não serão escaláveis, pois se estão sempre a crescer eventualmente atingirão tamanhos que resultam em processamentos extremamente lentos.

Como referido, neste trabalho decidimos propor uma arquitetura dinâmica, com o objetivo de não só conseguir aprender sequencialmente nova informação, mas também combater o problema da escalabilidade que vem com este tipo de arquitetura.

A arquitetura proposta baseia-se em dois blocos principais: (i) a extração de atributos e (ii) a classificação modular dinâmica. A ideia é ter um primeiro componente estático que efetua a extração de atributos de baixo-nível, que serão depois passados para um segundo componente que é composto por vários módulos e submódulos que efetuam a classificação. Em mais detalhe, para a primeira parte da arquitetura, (i), resolvemos utilizar um método com resultados comprovados para este tipo de tarefa, nomeadamente o ResNet50. No que toca à segunda parte da arquitetura, (ii), esta é composta por uma série de módulos e submódulos estruturados/ligados em forma de árvore. Esses módulos podem ser constituídos por conjuntos de classificadores binários, implementados usando redes neuronais, ou por apenas dados das classes conhecidas, onde os dados armazenados são na forma dos atributos extraídos e não no seu formato original.

Na arquitetura proposta, cada módulo é independente, ou seja, a informação que entra nos módulos finais é igual à informação que entra nos módulos iniciais, i.e., não existe transformação de dados intermédio. Deste modo garantimos que havendo alterações a um módulo não haverá consequências nos outros e que a sua posição na árvore serve apenas para ditar quando e quais módulos devem ser utilizados numa determinada situação.

Além disso, a arquitetura proposta agrupa automaticamente classes com base nas suas semelhanças e cria classificadores especialistas para fazer distinções entre as classes que são consideradas semelhantes. Recursivamente, estes grupos podem ter mais subgrupos,

subsubgrupos etc. A distinção de quais classes são ou não semelhantes é feita através dos próprios classificadores binários já existentes, i.e., probabilisticamente falando, se uma classe nova faz com que um classificador binário devolva valores de "probabilidade" acima de um dado limite então as duas classes são consideradas semelhantes. Processos específicos para cada um dos vários casos possíveis em relação às quantidades de classes semelhantes, de modo a garantir uma formação ideal dos grupos, foram ainda desenvolvidos neste trabalho.

A estrutura proposta permite-nos efetuar um treino à rede por secções e não com a rede inteira, onde classes novas são adicionadas com mais classificadores binários e alguns outros classificadores específicos são retreinados utilizando partes específicas dos dados armazenados. Neste contexto, foi ainda desenvolvido um algoritmo para fazer a seleção otimizada dos dados armazenados a incluir no treino.

O processo da classificação beneficia também desta estrutura pois, com ela, existe a possibilidade de efetuar classificações utilizando apenas as partes necessárias da rede.

Relativamente aos resultados, a arquitetura foi testada com dois conjuntos de dados, nomeadamente: ImageNet (usando apenas um grupo de classes) e CIFAR10. Para alguns desses testes, mostramos a evolução da estrutura da árvore com a adição de cada uma das classes usadas e, em alguns casos, avaliamos a precisão da classificação de cada classe depois de cada adição de uma classe nova, sendo possível assim mostrar o comportamento do sistema ao longo do tempo.

Dos resultados atingidos, tendo em conta que se trata de uma prova de conceito, consideramos que a arquitetura proposta é um método viável de *Continual Learning.* Neste sentido, a estrutura, os vários componentes e respetivas as afinações deverão ser mais estudadas e melhoradas para que a arquitetura possa ser utilizada numa maior variedade de aplicações com melhores precisões e velocidades.

Acknowledgements

I would like to start by thanking my advisors, Professor *João Rodrigues* and Professor *Pedro Cardoso*, for their constant support and availability, for their valuable feedback and guidance, for having the patience to always listen to my ideas, for always being accessible (even during a global pandemic), and, most importantly, for allowing me to work on something I'm passionate about and showing a genuine interest in all of the developments along the way.

I owe an enormous thanks to my parents for providing me with the opportunity to attend university and for setting the example of what it means to work hard to achieve your goals.

And finally, I owe a special thanks to my fiancé Ana, for all of her love and support, for letting me fill the house up with computers and monitors, for bringing me the occasional snack, for always inspiring me to the best that I can be, and for being patient with me through all my madness.

Table of Contents

1 Introduction

ABSTRACT

Technological advancements are being made in Artificial Intelligence at an outstanding rate. With vast amounts of research being made in Machine Learning, this chapter introduces the reader to the context and goals of the dissertation and explains the appeal of a system that can learn continuously.

1.1 INTRODUCTION

In his classic paper on computing machinery and intelligence, Turing (1950) states the difficulty of writing an intelligent program in the conventional way: "*At my present rate of working I produce about a thousand digits of programme a day, so that about sixty workers, working steadily through the fifty years might accomplish the job, if nothing went into the wastepaper basket. Some more expeditious method seems desirable*".

Turing did propose a more expeditious method, i.e., program an imitation of a child's mind ("simple program") and then educate it into adulthood ("more complex program"). He assumed that the amount of effort in bringing up the child machine would be similar to that of educating a human (Hall, 2007): "*Presumably the child brain is something like a notebook as one buys it from the stationer's. Rather little mechanism, and lots of blank sheets. (Mechanism and writing are from our point of view almost synonymous). Our hope is that there is so little mechanism in the child brain that something like it can be easily programmed*" (Turing, 1950).

The main goal of this dissertation is to study and develop a framework that can learn in a way that it "educates" itself. More specifically, the focus is to adapt or create a classification framework, that can learn new classes automatically and consistently, which would be a holy grail if done without any human intervention. We can recall a McCarthy *et al.* (1955) definition "*Probably a truly intelligent machine will carry out activities which may best be described as self-improvement*", complemented in (McCarthy, 1969) with "*Our ultimate objective is to make programs that learn from experience as effectively as humans do. It may not be realized how far we are presently from this objective*". These words are still true, despite enormous advances in the Artificial Intelligence (AI) field since then.

The dissertation focusses on the application of this concept to object and scene classification, intending to create a method or framework capable of (easily) learning new classes. For instance, in the case of objects, if the framework already knows apples and pineapples, and it is presented with strawberries, it should learn this new class without any human intervention in the teaching of the framework (except, possibly, for the validation and the final labelling of the class "strawberries").

The current state-of-the-art shows that the methods most effectively used for image classification are neural network-based but with the standard versions of these, while very effective and widely used, if we were to decide to add new classes to them, i.e., a new label, and continue the training process from where we left off, this would cause *Catastrophic Forgetting* (Ratcliff, 1990; French, 1999; Parisi *et al.*, 2019). This means that after training new classes the networks ability to recognize old/known classes would be severely reduced, e.g., the network was originally trained to recognize apples and pineapples and able to tell them apart with great success, but by adding a new class, like "strawberries", into the mix and continuing to train the network without retraining the original two classes, it would start learning the new class but damage the network's capability of recognizing the original two in the process.

So, the usual approach for adding a new class to this kind of method would be to simply retrain the (complete) network from scratch, using all the classes that we want the network to recognize. The problem here is that the training process is usually very time consuming, even with powerful machines. While this may not be a problem for some situations, what if we wanted to be able to add new classes constantly and consistently? Given that with currently accessible hardware the training process can take anywhere between hours and days, what if we wanted our network to learn multiple classes per day?

What we intend to investigate in this dissertation is a way to deal with exactly that problem, we want a network that can adapt to new information without needing to retrain the entire network in a way that destroys previous knowledge.

Initial research on our part led us to discover that there were, in fact, already some methods somewhat along these lines that fell under a category referred to as *Continual Learning* (CL) (Ring, 1997; Ebrahimi *et al.*, 2020). Within the focus of AI (Nilsson, 2014), the definition of CL can be considered (in this instance) as an area of Machine Learning (ML) (Alpaydin, 2020).

Much recent work has been done in the domain of CL (see Sec. 2.3), that aims to ease the catastrophic forgetting problem when learning new classes. It is important to note that there are a few other names used to describe Continual Learning such as Lifelong Learning (Parisi *et al.*, 2019), Sequential Learning (Aljundi *et al.*, 2018) and Incremental

Learning (Chen *et al.*, 2017), all with slight and unclear differences, nevertheless, the most widely used one is Continual Learning.

1.2 OBJECTIVES

The main objectives of the dissertation are:

- Analyse, study, and write a state-of-the-art over CL;
- Implement a selected object and scene detection framework;
- Adapt/propose a new object and scene detection framework, with CL;
- Test the framework with real images and/or videos (or camera stream).

We would like our object and scene classification framework to be capable of using streams of labelled data to learn new classes and contribute to its precision for already known classes. To achieve the above, we will naturally focus the initial research on existing CL methods and then analyse and adapt them, so that we can establish which ones are most suited to help us meet our goals.
The main contribution will be a framework that learns new classes without needing to retrain its entire network of classifiers.

1.3 CONTENTS

The present chapter introduced the reader to the context and goals of the dissertation, Chapter 2 includes a state of the art for Continual Learning and some background concepts to aid the reader's understanding of the subject. Chapter 3 presents the architecture of the implemented CL framework, Chapter 4 presents the test and results and, finally, in Chapter 5 we have the conclusion as well as an explanation of what we plan to do in future work.

2 State of the Art

ABSTRACT

As with all areas of Artificial Intelligence, Continual Learning is of fundamental importance. It has seen several years of study but is still only advancing in baby steps. In this chapter, we present a state-of-the-art on the topic and introduce the reader to the main concepts necessary for a better understanding of this dissertation.

2.1 INTRODUCTION

The idea of computers being able to learn has taken many forms and lead to the development of various theories and techniques over the years. This concept has been named *Machine Learning* (Alpaydin, 2020). A simpler definition can be, ML is the study of computer algorithms that improve automatically through experience.

Machine Learning is normally divided into three types of learning: (a) supervised learning, (b) unsupervised learning and (c) reinforcement learning. In a nutshell, *supervised learning* means that the data being analysed comes with additional attributes that give the machine "examples" of correct responses. In *unsupervised learning*, the data is presented without additional information to tell the machine what is correct, making it act on the feature data alone. This type of learning is a good tool for sorting data samples into groups based on discovered similarities. Finally, for *reinforcement learning*, the data is not initially presented with additional information, but the predicted output is evaluated in such a way that correct/incorrect predictions result in rewards/punishments for the machine (Dangeti, 2017).

In Computer Vision, most of the current object recognition methods are supervised and based on Artificial Neural Networks, where they typically consist of the use of samples of known data to try and predict properties of unknown data. The implementation of these methods is generally done by "feeding" a network with large quantities of labelled images and training the network over a long period of time until it achieves a satisfactory accuracy. There are, of course, many factors to consider while doing this and many different approaches, but this is the general process.

2.2 CONTEXTUALIZATION

Artificial Neural Networks (ANN) are currently receiving vast amounts of attention and being applied to all kinds of real-world problems (Abiodun *et al.*, 2018). They are currently the natural choice for dealing with images when it comes to detection, recognition and classification of persons, objects and/or scenes (Nielsen, 2015).

The term *Artificial Neural Networks* was chosen because they are inspired by biological systems and are an attempt at mimicking functionalities similar to those of the human brain. Simply put, an ANN is composed of multiple artificial neurons where each neuron performs a weighted sum on all its inputs to calculate a single output. These artificial neurons are sorted into separate layers where, in many architectures, the outputs of one layer are the inputs of another. The initial input will "travel" through the network's layers until reaching the last one, where the final output is calculated, but many alternative architectures exist like *Recurrent Neural Networks* (Giles *et al*., 1994) where some layers are re-used. A typical ANN is trained by feeding in large amounts of data and adjusting the weights of each neuron to achieve the desired output, then, when future data is presented, the network is expected to produce the desired output for that new input.

As computational capacity has evolved, it has opened up possibilities for huge advancements in ANN. By adding more and more layers to neural networks and increasing their complexity, we end up with what are known as *Deep Neural Networks* (Nielsen, 2015) which lead us to a new, more advanced, branch of Machine Learning called *Deep Learning* (DL). While it is true that DL has recently been displaying far superior results compared to other approaches, it is also true that for it to be so successful it generally requires very large quantities of labelled data and plenty of processing time.

More related to Computer Vision tasks, the *Convolutional Neural Networks* (CNN) are essentially deep neural networks based on their shared-weights architecture and translation invariance characteristics, i.e., CNNs are networks that employ a mathematical operation called convolution in place of general matrix multiplication, in at least one of their layers. O'Shea *et al*. (2015) explain CNN and demonstrate a few different examples of their architectures. Wu (2017) explains CNN at a more in-depth and mathematical level, often using examples of image classification where they go over the various types of layers, including the convolution layer, and explain how it essentially functions as a kernel matrix that "sweeps" over the input image resulting in a more compact version of the original data. For a deeper understanding of CNNs please see also (Wang *et al*., 2020).

Another concept already addressed in the Introduction chapter is *Catastrophic Forgetting* (Parisi *et al.*, 2019). Catastrophic Forgetting is a core notion for this dissertation, as it is one of the main problems Artificial Neural Networks must deal with when they are applied to a CL scenario. The phenomenon occurs specifically when an artificial neural

network is trained sequentially on multiple tasks, because the weights in the ANN that are important for one task are changed to meet the objectives of another. If we think about how ANN work and, once trained, how sensitive their weights can be, it's clear that by bringing in new classes and re-applying the same training techniques to the existing network that it will have hugely detrimental effects to its delicate balance (Ratcliff, 1990; French, 1999; Parisi *et al.*, 2019).

Another important concept is *Transfer Learning* that, when mentioned in this context, is easily mistaken as a name for a full-on ML technique but, in reality, is more of a general concept that some ML techniques can take into consideration. Transfer Learning (Pan *et al*., 2009) is very frequently mentioned in CL methods, and it essentially consists of using knowledge acquired from one task to improve performance on another. In this context, *Forward Transfer* refers to knowledge from older tasks contributing to future tasks and *Backward Transfer* refers to using new knowledge to contribute to older tasks.

2.3 CONTINUAL LEARNING

Many studies about CL use humans as an ideal example of continual learners (De Lange *et al.*, 2019; Parisi *et al.*, 2019; She *et al*., 2019; Aljundi, 2019). The reason for this is that, as is the case with Artificial Neural Networks, many of the ideas behind CL are inspired by how investigators presume our brains work.

In the context of ML, CL means being able to update the prediction model for new tasks while still being able to re-use and retain knowledge from previous tasks. CL problems assume an incremental setting, where tasks are received one at a time and most studies on the matter also consider the non-storage of data to be an essential characteristic of a continual learner. Ring (1997) stated that a continual learner is an autonomous agent, able to learn context-dependent tasks, learn "skills" while solving its tasks, learn incrementally, learn hierarchically, and function as a black box that has no ultimate final task.

Lomonaco *et al*. (2017) proposed CORe50, a dataset and benchmark that is more appropriate for testing CL methods when compared with normal image datasets by taking different factors into account, like the order of image capture and different levels of illumination and occlusion. She *et al.* (2019) then tested a variety of CL methods on this

dataset to get an idea of their behaviour in real-world environments and concluded that the current algorithms are far from ready to face such complex problems.

Requeima *et al.* (2019) approach multi-task classification using *Conditional Neural Adaptive Processes*, and while this may not directly be considered a CL method, it can be applied to CL scenarios. De Lange *et al*. (2019) studied a variety of CL methods and organized them in the form of a tree diagram (Fig. 1), where they sorted 29 different methods into categories and sub-categories. The authors concluded that iCaRL (Rebuffi *et al.*, 2017) was the leading performer for replay-based methods, MAS (Aljundi *et al.*, 2018) was the leading performer for regularization-based methods, and PackNet (Mallya *et al.*, 2018) was the leading performer for parameter isolation methods. However, all the methods featured their advantages and limitations with regards to each other, e.g., Packnet showed the most promising accuracy but, although it can learn a large number of tasks, it does have a limit based on the size of the model. For a detailed explanation of each method please see (De Lange *et al*., 2019).

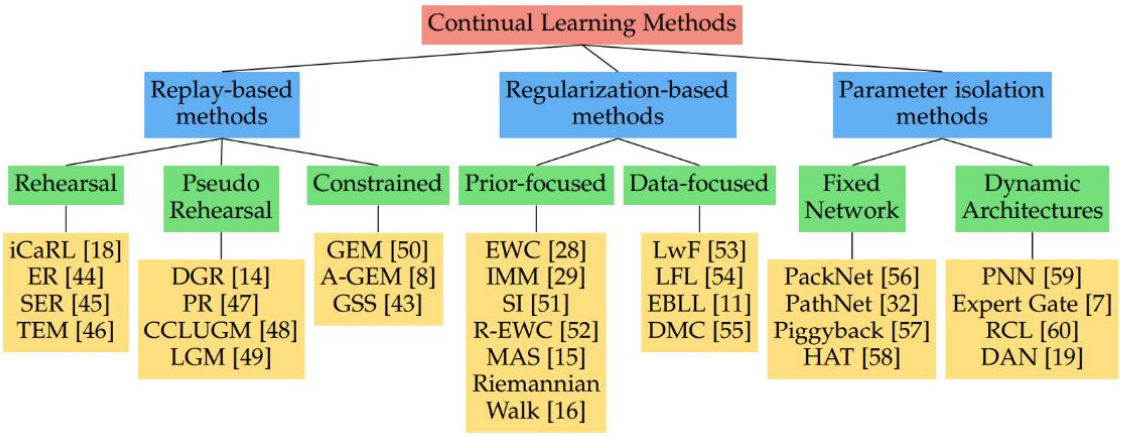


Figure 1: Tree diagram that shows CL methods organized into different categories (adapted from (De Lange *et al*., 2019)).

Van de Ven *et al*. (2019) apply a range of Continual Learning methods to three different scenarios of increasing difficulty, as a way of comparing their performance in different situations. In the first scenario, the models are informed about the identity of the task to be performed, meaning that the model can choose to use specific components to perform the task at hand. In the second scenario, the identity of the task is no longer available, but the model is only required to solve the given task and not necessarily identify what task

it has performed. In the third scenario, the model must be able to solve a given task and identify what task it was presented with.

Parisi *et al.* (2019) discuss three different approaches to CL (they refer to it as lifelong learning). The first approach they introduce is based on retraining the entire network with regularization, meaning that they deal with catastrophic forgetting by applying constraints to the update of the neural network weights. The second approach selectively trains sections of the network and expands it as needed, acting as a type of dynamic architecture that adds new neurons dedicated to new information. And the third approach consists of methods that model complementary learning systems for memory consolidation, where they distinguish between learning and memorizing.

Pellegrini *et al.* (2020) present latent replay for real-time CL. In their work, the authors make a division between the low-level feature extraction and high-level feature extraction. Furthermore, they control the rate at which each one of the levels are trained in such a way that, when trained on new data, the low-level feature extraction is altered very little or not at all, making it possible for them to store intermediate data to be retrained alongside new data when new classes are learned, without having to re-perform low-level feature extraction to the stored data.

2.4 DISCUSSION

While there are mentions of CL as early as the 1990s, where concepts and ideas are mentioned, almost all research with practical testing and applications has only been made very recently, within the previous year or so concerning the time of this dissertation. The fact that many of the researched papers use different terms for CL, and the fact that there are no standard categorizations for CL methods or consistent descriptions of CL are indicators of just how new this area of study is in terms of practical applications.

Our state-of-the-art shows, especially the most recent papers, that there are many existing CL methodological approaches with many differences between them. They vary to such an extent that it is hard to find a situation where they could all be applied that would allow us to easily compare them. This variation between methods has created the need for categorization, where many papers came up with different ways to separate them. Looking at the conclusions from each of the papers that make comparisons between CL

methods, although each one uses a different set of categories to divide them, the more complex ones appear to be the ones that show the most promising results of all the variations.

Nevertheless, some of the papers mentioned in this chapter show similarities to the framework that will be proposed in the next chapter, with a comparative discussion being presented in there and also in the conclusions (Chapter 5), where differences and similarities will be highlighted.

3 Modular Dynamic Neural Network

ABSTRACT

Continual Learning still has a lot of room for improvement and many obstacles to overcome before it can be reliably applied to real-world problems. This dissertation proposes an early-stage architecture that aims to overcome Continual Learning's mains barriers. We demonstrate a proof of concept of the architecture.

3.1 INTRODUCTION

There are several different categories of Continual Learning methods (see Sec. 2.3), where those that show the most promising results for real-world problems tend to be the more complex ones (De Lange *et al*., 2019). In the hopes of achieving better results and scalability, we would like our solution to be more in line with the methods that appear to show the most promise, while including proposals and adaptations of our own.

A very interesting type of architecture for CL is the *Dynamic Architecture* (De Lange *et al*., 2019). With the Catastrophic Forgetting problem being one of the main obstacles that CL methods have to face (as already mentioned), the way dynamic architectures deal with this is by simply adding more and more layers/neurons/blocks to the network, to work with the new information, while not touching the old layers/neurons/blocks, so that no previous "knowledge" is lost.

Many studies on CL immediately dismiss the idea of dynamic architectures because of scalability problems. This is understandable because, despite the fact that ANN can successfully learn new information by adding more and more neurons or layers (as required), the network's speed will decrease every time anything is added, eventually rendering it unusable. Of course, in a situation where the number of classes to be learned is low, scalability may not be an issue, but this kind of situation generally does not need the implementation of a CL method anyway.

The solutions that constrain themselves to a limited "space" are eventually always going to have limitations for their results. Some methods, like PackNet (Mallya & Lazebnik, 2018), re-use inactive neurons for learning new information. But naturally, for this type of approach, the number of "free" (inactive) neurons will steadily decrease as more tasks are required of the network, eventually resulting in some kind of saturation or decrease in performance, as the most inactive neurons stop being so inactive.

Going back to our leading example of a continual learner, one could argue that the human brain learns "infinite" tasks without the need to expand and it is commonly said that people don't "run out of memory", but it is also a fact that a finite amount of space cannot contain an infinite amount of information so, essentially, we do have a limit, but it is just considered to be "almost" infinite.

When thinking about the "limit" of knowledge that a network can achieve, the actual capacity of knowledge is essentially impossible to calculate, as there are so many factors to take into account. Considering that their results are achieved via a combination of processes, we can reason that adding one more neuron/layer to a standard ANN (in real-time) will not simply increase the capacity by one more task. The additional capacity will be relative to the previous and current sizes of the network, meaning that, theoretically, the capacity will grow exponentially with every increase in complexity. This exponential increase in capacity concerning size makes the "almost" infinite capacity of human brains sound more realistic, when we think about the vast scale of their internal connections.

So, following this principle, for an ANN to be capable of learning indefinitely, size seems to be the way to go. This means having a network so large that its capacity tends towards infinite or having a dynamic network that grows in complexity as needed. But, as we mentioned before, this kind of solution is faced with an obvious problem: scalability. This train of thought led us to decide that a large/dynamic network would be an ideal solution, just so long as we can find ways to deal with the scalability problem.

So, we aim to design a method/framework that fits into this category and find ways to deal with scalability, which leads us to our next line of thinking.

Continuing our analogy with the human brain, why do human brains not suffer from huge speed-related problems when their networks of neurons are so vast? The first main reason is that they do not rely on a single processor (or group of processors) to calculate the result of each and every neuron one at a time. In the human brain, each neuron functions on its own in an asynchronous fashion (Shlens *et al*., 2006), meaning that there is a far more complex and "chaotic" process occurring compared to what you would find in a typical ANN implementation.

While there is a great deal of study in the area of applying asynchronous methods to ANN, and that may eventually become a part of our study, at the moment we are more interested in the second main reason that human brains are capable of such high processing speeds (in relation to their vast complexity): not all areas of the brain need to function at the same time to achieve a result. Many studies have shown that different tasks result in different levels of activity in different areas of the brain (Kwong *et al*., 1992). This phenomenon has inspired a possible solution/improvement to the scalability problem that our network will face, which we will describe next.

3.2 THE MODULAR DYNAMIC CLASSIFICATION FRAMEWORK

With all the above in mind, we present the *Modular Dynamic Neural Network* (MDNN). The MDNN is a network primarily made up of modular sub-networks that progressively grows and re-arranges itself as it learns continuously.

The network is structured in such a way that its internal components function independently from one another, so that when new information is learned only specific sub-networks are altered in a way that old information is not forgotten. The network is divided into two main blocks: (a) feature extraction and (b) modular dynamic classification (Fig. 2).

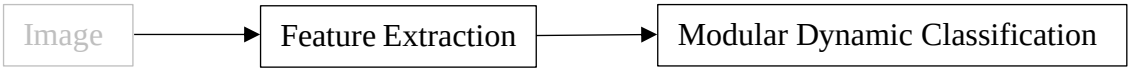


Figure 2: A global overview of the proposed architecture.

The first part of the network, *feature extraction*, as the name suggests, performs an extraction of features where the input data (in our case images) is reduced down to a set of feature values, which vary based on the input but are similar for similar inputs. In our case, we used the feature extraction component of a pre-trained ResNet50 (He *et al*., 2016), which is a CNN with 50 layers shown to yield exceptional results in image classification (Sharma *et al*., 2018). This first static general feature extraction is used only to extract generic "low-level" features, while class-specific features will be extracted by much smaller and simpler (modular) networks at a later point in the architecture, in the dynamic classification block. The feature extraction part of our network is the only part that is never altered when learning new classes, because the components in the next block depend on it to be consistent.

The second part of the network, *modular dynamic classification*, is composed of a tree of components, which are in turn composed of classifiers or data, as we will see next.

These sub-components are independent of one another, but they do depend on the main feature extraction component because they adapt and evolve based on its results. This dependency implies that altering the feature extraction module would invalidate the functionality of the modular dynamic classification blocks.

In more detail, the modular dynamic classification block is comprised of multiple small modules. These modules are responsible for classifying specific classes or groups of classes that, as new information is learned, are automatically divided into groups of modules and sub-modules based on their class's similarities. Figure 3 depicts an example

showing how the sub-modules fit in to the proposed architecture, where the modules containing information inside brackets (e.g., $[X_1, X_2, \dots, X_{n_x}]$) are modules that contain sub-modules or groups of sub-modules in the case of nested brackets (e.g., $[[X1, X2, \dots, X_{n_x}], \dots]$) and A_1 to A_{n_a} are examples of modules with no children, with n_a being the number of sub-modules belonging to their parent.

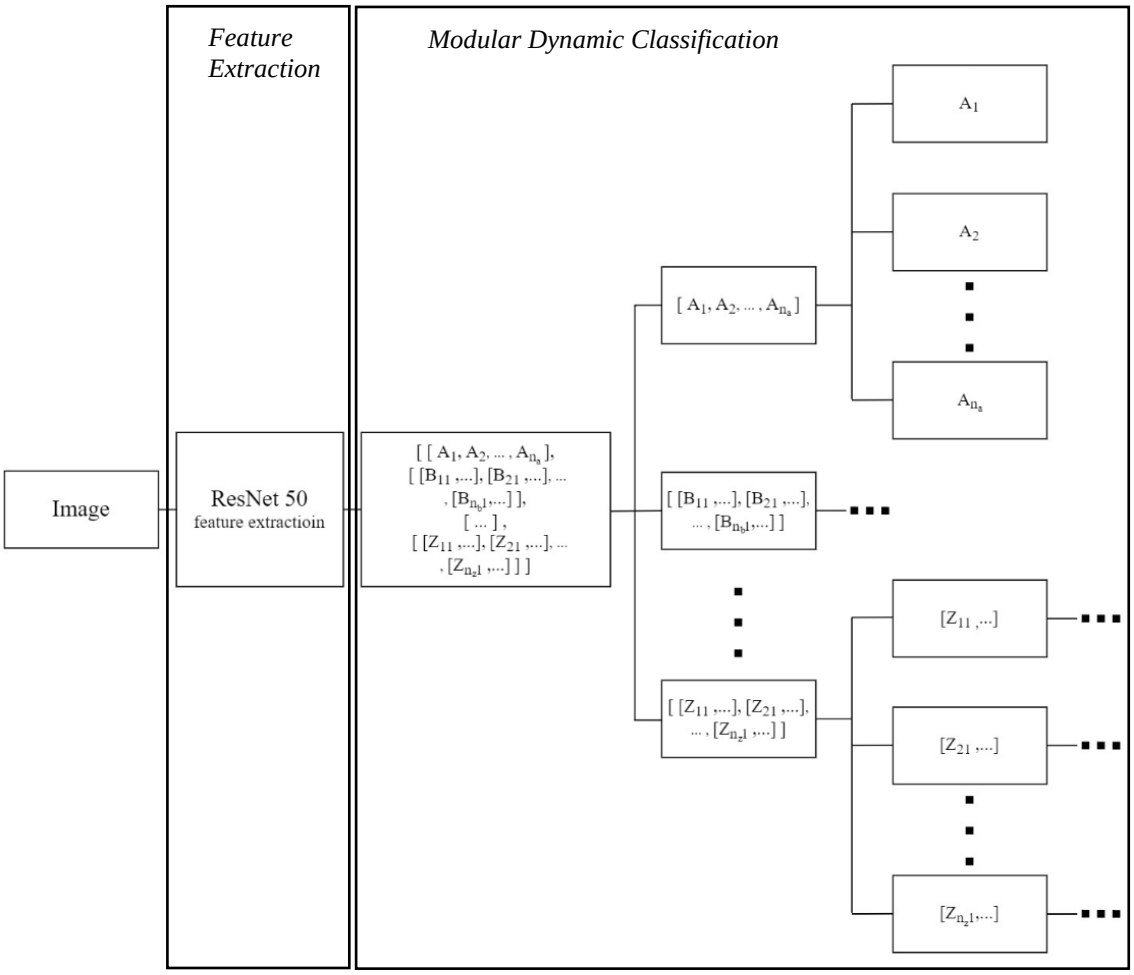


Figure 3: Example of how the Modular Dynamic Classification component fits in to the proposed architecture.

The modules which contain their own sub-modules are designated here as *node modules* (see Sec. 3.4.1) and they contain one binary classifier for each direct child module. The modules with no children are designated here as *endpoint modules* (see Sec. 3.4.1) and contain only data obtained during the training process. When the network is learning new information, the data obtained by the feature extraction component is stored within the *endpoint modules* for later use, specifically for when some classifiers have to be retrained to avoid confusion with a new class (see Sec. 3.4.3). It is important to emphasize that the data preserved from each class is not the class data in its raw format (in our case images),

but the features extracted from the inputs which should have a smaller dimension (Sec. 3.4.2). In the case of the ResNet50 adopted, after being re-sized using nearest-neighbour interpolation (Parker *et al.*, 1983), each sample image with a dimension of 224×224 pixels (px) and 3 colour channels ($224 \times 224 \times 3$) is reduced to a 1×2048 array, corresponding to an approx. 98.6% savings in size. And more importantly, saving the features instead of the image itself allows us to avoid running the images through the feature extractor again and again.

This module-based structure allows our network to have a dynamic growth that fits well with our objective to develop a dynamic architecture capable of learning new information without forgetting the old, since new classes are learned by adding new, smaller classifiers to the existing network (see Sec. 3.5). Furthermore, our other main objective of coping with the scalability problem, which normally accompanies this type of approach, is achieved because our modular structure permits us to make classifications without using the entirety of the network. This functionality will be explained in detail in Sec. 3.4.5.

Again, it is important to stress that our modular-based network can learn new classes by adding new modules and making alterations to some modules but leaving other modules untouched. For the un-touched modules to still function after other modules are altered though, they need to be independent of one another. The way we achieve this is by ensuring that they all receive the same input, no matter where they are located within the network (the significance of the modules positions in the network will be explained in Sec. 3.4). For all the modules to receive the same input this could mean receiving the data samples in their original form, but then that would imply that each module would have to perform its own feature extraction from the raw data, which would be a costly process to train especially as the network grew to have numerous modules. It would also mean that the data stored by the modules for future corrections would be in its original format which would cause serious memory implications.

As already mentioned, our solution is to have a generic feature extraction component (addressed in Sec. 3.3) for low-level features to allow the modular sub-networks to be drastically smaller and to significantly reduce the dimension of the stored data. The modules do have their own smaller feature extraction components for determining higher-level features, but these are in-comparable to the main feature extraction component in terms of size and complexity. The feature extraction component needs to be appropriate for the data type we are dealing with. This means it needs to be prepared to receive data

samples of the type you want the network to be able to predict (e.g., images or sounds). It also needs to be capable of extracting a reasonable number of low-level features and to be pre-trained with a dataset comprised of many different classes so that when shown new classes it can still detect relevant features.

3.3 FEATURE EXTRACTION

In an ANN, feature extraction is the process that takes the input data and transforms it into a set of values where each one represents the weight of a given feature. After being trained with a dataset, the feature extraction learns to output similar feature values when data samples from the same classes are presented. These similar feature values then make it easier for the last layers of the network to learn how to make predictions by analysing which features are predominant in new data.

As our current focus is the application of the network to images of objects and scenes, we decided to use a RestNet50 that was pre-trained with the ImageNet dataset (Deng *et al.*, 2009), because of its proven capabilities for feature extraction over a great variety of classes (it was trained with 1,000 classes). Other viable options that could substitute this component for our application would be, for instance, VGG16, Inception, or EfficientNet (Simonyan *et al.*, 2014; Szegedy *et al.*, 2016; Tan *et al.*, 2019). In the future, comparative tests will be done using feature extraction sections from different networks.

Naturally, whichever feature extraction network is used, the final classification layers must be excluded as these are very specific to the network's original application and usually normalize their values to a probability distribution with a function like SoftMax (Liu *et al.*, 2016). However, we do not want a probability distribution or anything specific to the mature network's original use-case, we instead want the raw, low-level, feature values to maximize our chances at getting useful information out of new data.

As the modular sub-classifiers receive these features as an input, it is also necessary to make sure that their first layer is of the same dimension as the extracted features. In our case, the RestNet50's feature extraction gives us an output array of 1×2048 feature values, so our sub-networks are accordingly structured to receive an array of that dimension as an input.

Figure 4 illustrates the original ResNet architectures proposed by (He *et al.*, 2016) including ResNet50. And Fig. 5 illustrates the ResNet50 network with a block diagram

(adapted from (Fang *et al.*, 2019)), making it easier to explain which blocks are removed in the latter figure, namely: the *average pool*, the *2-d full connected* and the *softmax* blocks.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 4: ResNet architectures proposed by He *et al*. (2016).

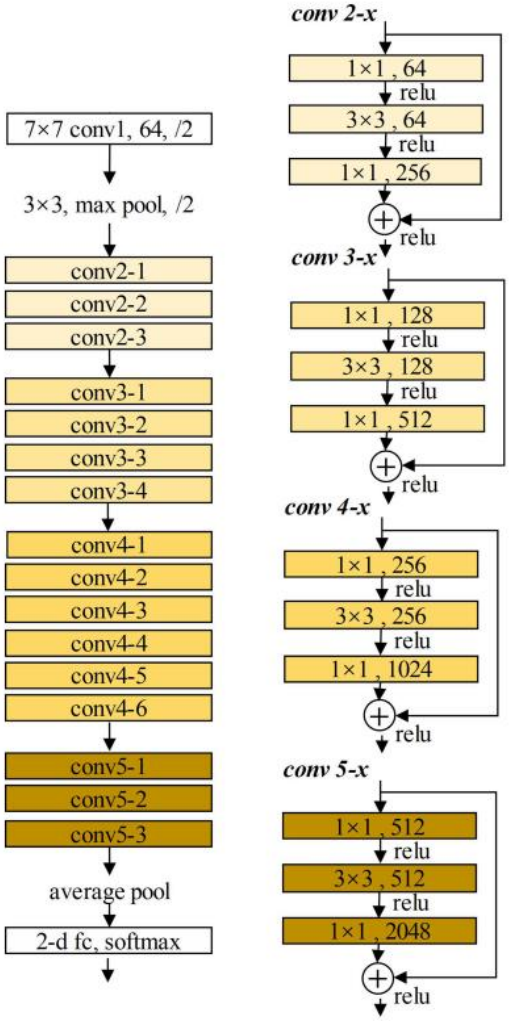


Figure 5: ResNet50 architecture, adapted from (Fang *et al.*, 2019).

3.4 MODULAR DYNAMIC CLASSIFICATION

One of the most important aspects of the proposed architecture is its modularity. Ensuring that certain sections of the network can work independently from one another makes it possible for us to alter or add modules without affecting others. This means that by treating sections of the network as modules, where each one is responsible for a given class or group of classes, we can safely add, remove or alter parts of our networks' knowledge base without affecting the rest.

Another great advantage to a modular approach is its flexibility, as it allows us to make changes to modules or even manually re-structure parts of the trained network, which in a ("standard") neural network would yield destructive results. Although not yet applied in our work, this kind of approach also allows for the sharing of knowledge, e.g., multiple networks of this type can share knowledge with each other.

3.4.1 Nodes and Endpoints

The modules present in our network are of two different types, designated here as *endpoints* and *nodes*. An *endpoint* is responsible for the storage of the feature data extracted from a single class during training. A *node* contains references to a group of two or more sub-modules and a *binary classifier* for each one of those sub-modules (see Sec. 3.4.3). Each of these sub-modules can then also be *endpoints* and/or *nodes*.

Figure 6 illustrates the difference between *nodes* and *endpoints* with a basic demonstration of a possible network with three classes.

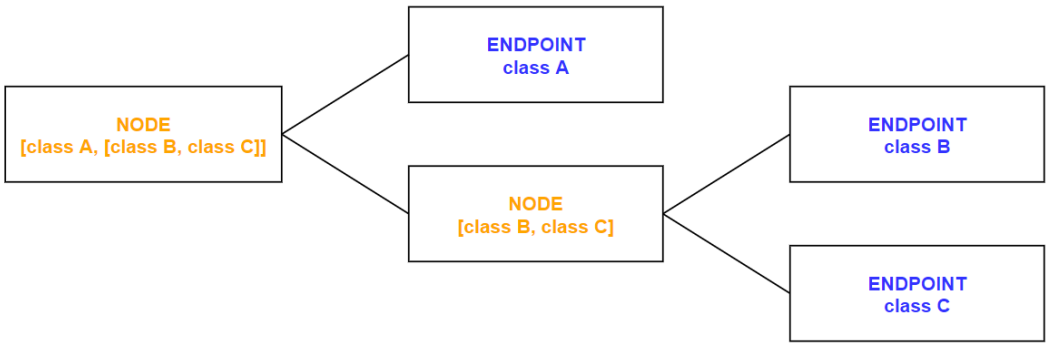


Figure 6: Illustration showing the positional difference between *nodes* and *endpoints*.

All the modules of the *node* type have their own set of binary classifiers (see Sec. 3.4.3), which, in this case, are ANN that output two values: a confidence value for "true" between 0 and 1, and a confidence value for "false" between 0 and 1, where the definition of what is true and what is false depends on the *node* and its position in the network, as explained next.

Each binary classifier represents one of a nodes' sub-modules. In the first case we have the ones that represent *endpoints*, which define "true" as the one class they are responsible for, and "false" as the classes present in all the other modules and sub-modules in parallel with that *endpoint*. In the second case we have the classifiers that represent *nodes*, which consider all their own sub-module classes as "true" and all the classes present in the other parallel modules and sub-module as "false". This selection of what is true or false for a given classifier will be explained in more detail in Sec. 3.5 and the structure of these classifiers will be explained in Sec. 3.4.3.

During classification, these binary classifiers are used to determine which sub-module to continue through and define a kind of path that eventually leads to a final prediction. During training, they are used in a similar fashion to determine the best location in which to "insert" new modules.

3.4.2 Data Storage

The binary classifiers can be called upon to retrain themselves if necessary (see Sec. 3.4.3). For this to be possible, the classifiers need data to learn from. So, when new classes are added to the network, their data is stored in the final *endpoint* module for that class. *Endpoint* modules are the only place where data is stored and each one only stores the data of the class it is responsible for. When and how much of this data is used, is decided when it comes to training or retraining a classifier, and will vary based on its position in the network and how much data from the other classes exists (see Sec. 3.6).

The format of the stored data is the same as the output of the feature extraction section. As already mentioned, in our case, all input images are re-sized to a $224 \times 224 \times 3$ tensor via nearest-neighbour interpolation (Parker *et al.*, 1983) and then reduced to a 1×2048 feature vector via feature extraction, which is what is stored.

3.4.3 Binary Classifiers

Each *Binary Classifier* (BC) has the task of classifying their inputs into two groups. BC present in the network have the exact same structure, but of course different trained weights. In other words, the number of layers, the number of neurons in each layer, the connections, the activation functions etc. are all the same, and the only difference between them is their trained weights.

The main reason for this is to maintain consistency and give all the classifiers an equal chance of success and to avoid "favouritism". This is important because there are points during classification and training processes where comparisons are made between the predictions of the different classifiers. The process of selecting which classifiers are compared with each other will be better explained when we go over the classification and training processes in Sec. 3.4.5 and 3.5.

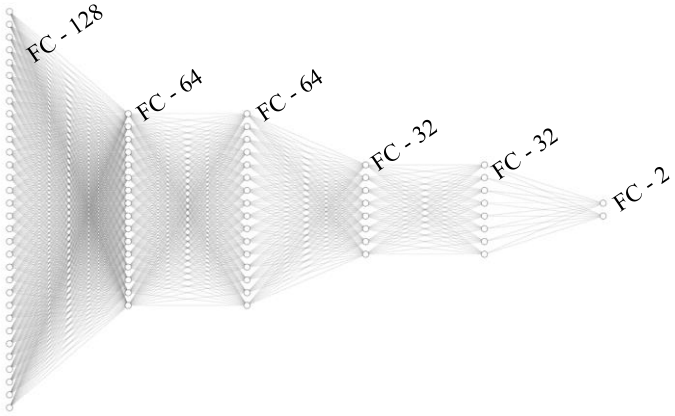


Figure 7: Implemented binary classifier ANN architecture – BC block. Made up of six fully connected layers.

The structure we used for our binary classifiers is shown in Fig. 7, where the idea is to have a small ANN to extract some higher-level features specific to whatever classes the network is being trained with and then finally end up with a classification layer that has 2 neurons which represent "true" and "false". The ANN uses six fully connected layers with numbers of neurons shown in Fig. 7. The final layers' activation function is not a SoftMax function, as would usually be seen in regular ANN's, but a *sigmoid* activation function. This choice was made because the algorithms that make use of the output of this classifier only make use of the true value, so there is no point in letting the false value interfere with the true value which is what would happen with an activation function like SoftMax. The reason we even have a neuron that represents the false value, when it is not

used, is that it is much simpler to implement and train a binary classifier this way, as we can just use standard back-propagation for the false class, the same as we do for the true one. Nevertheless, we do not plan to abandon these false values forever, as some of our future work includes developing more intelligent algorithms that take the false values into account as well as the true ones, but for now, this is the reason why the final activation function is a sigmoid function.

3.4.4 The Dynamic Network Structure

The dynamic part of our network (Fig. 8 bottom row) comes after the feature extraction component (Fig. 8 top row), as we have already mentioned. It is made up of various modules which can be altered (e.g., by retraining) or added to the network for it to grow as it learns new classes.

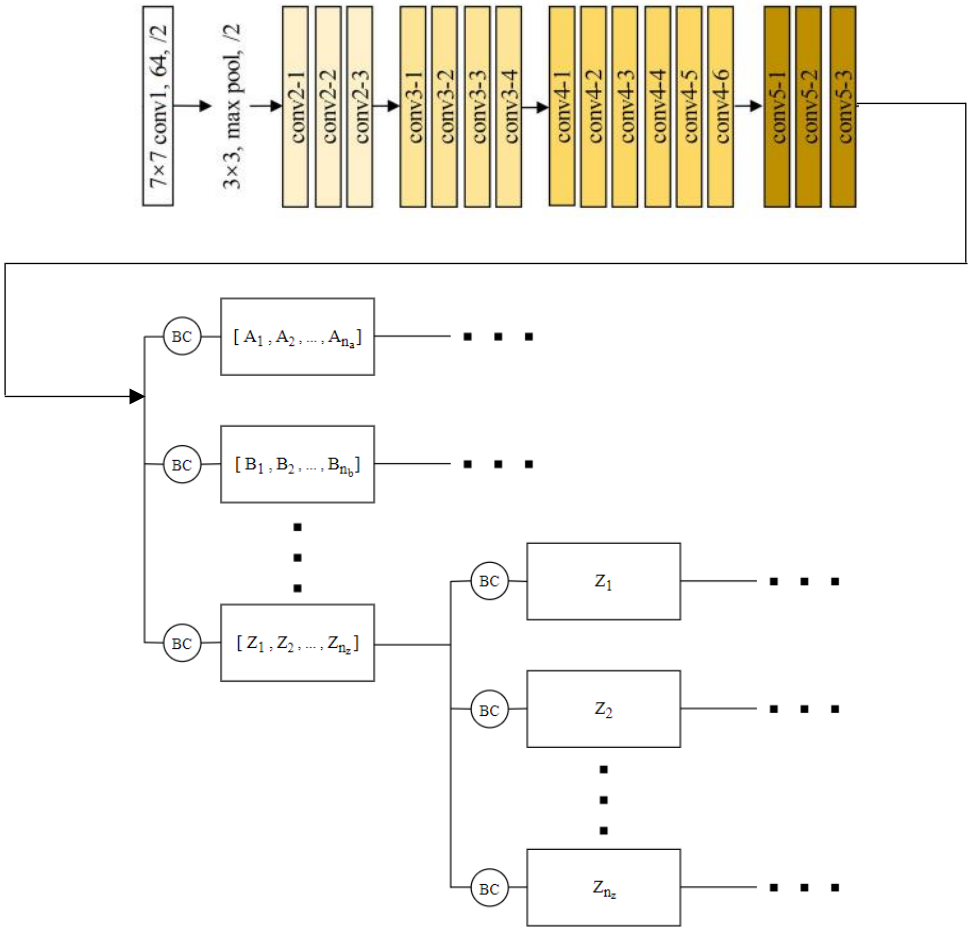


Figure 8: Full architecture of MDNN: at the top, ResNet50 architecture (Fang *et al*., 2019) – feature extraction block; at the bottom, a generic demonstration of the modular dynamic classification block (using a simplified notation of Fig. 3).

Figure 8 bottom shows how the network's structure is extremely flexible, as there is no limit to how many sub-modules a *node* can have or to how many sub-modules those sub-modules can have. In Fig. 8 bottom, we are using a simplified notation of what is shown in Fig. 3. The top section of the figure represents the feature extraction block whose output is the input of the modular dynamic classification block (bottom section), and the BC represent the binary classifiers associated with their respective modules.

For our network to meet its purpose of distinguishing classes from other classes, the minimum number of classes the network can initialize with is two. Therefore, the root module will always be a *node*, as *endpoints* only ever represent one class and *nodes* are the only modules which can contain more modules.

As already mentioned, each *node* represents a collection of sub-modules and, for the sake of clarity, their names are displayed as arrays of their children's names, and nested arrays when the sub-modules also include *nodes* of their own.

Figure 9 top illustrates a basic example of a network that is familiar with 3 classes (A, B, and C). The figure depicts an alternative representation of the example shown in Fig. 6, which was used to explain the positional difference between *nodes* and *endpoints*. In this example, classes B and C are joined as children of a *node* (*node* [B, C]) and class A is on its own. This indicates that when this network was built, the training process considered classes B and C to be similar to each other, and decided to group them and train classifiers to distinguish between A and B ∪ C. And then, some sub-classifiers were trained to specialize in distinguishing between classes B and C. To clarify, in the example, there are 4 binary classifiers, namely to distinguish: A (against [B, C]), [B, C] (against A), B (against C), and C (against B).

<table>
<tr><td rowspan="3">[A, [B, C]]</td><td colspan="2">A</td></tr>
<tr><td rowspan="2">[B, C]</td><td>B</td></tr>
<tr><td>C</td></tr>
</table>

<table>
<tr><td rowspan="3">[rose, [cat, dog]]</td><td colspan="2">rose</td></tr>
<tr><td rowspan="2">[cat, dog]</td><td>cat</td></tr>
<tr><td>dog</td></tr>
</table>

Figure 9: Top, an alternative representational method of the network presented in Fig. 6. Bottom, the same example with examples of object class names.

As a more realistic example, classes B and C could be considered similar if they were both animals (e.g., cat and dog) and class A could be considered dissimilar from the other two if it was a plant (e.g., rose). Figure 9 bottom illustrates this example.

The process of deciding which classes are similar enough to join and how they are joined is explained in further detail in Sec. 3.5.

3.4.5 Classification

The classification process is the main purpose of the entire network. The objective is to present the network with a data sample (in our case an image) belonging to one of the learned classes and it successfully predict which class the sample belongs to.

It is important to remember that every time we call upon a binary classifier, no matter where it is located in the network, the input values are always the same feature values extracted once from the data sample being classified. The positioning of modules within other modules is simply used to decide if and which other modules should be used, but the feature data itself is never altered. It is important to clarify this because, when thinking of a tree-like structure that uses ANN one might easily confuse it with a single network in a tree format, like that which is seen in (Wan *et.al*, 2020) where everything is backward-dependent. Therefore, it must be noted that our "tree" is merely a representation of the order in which things are done and which data is used by each binary classifier. All in all, the data is not transformed intermediately, i.e., the input data for the last modules is the same as it was for the first modules.

Because of the way our dynamic network is structured, the classification process can be recursive because when it is applied to different *nodes* of the network, those results will determine whether it should be re-applied to certain *sub-nodes*. I.e., the classification process is (i) first applied to the root *node* and then, possibly, (ii) recursively applied to other *nodes* depending on the results.

The first step, see Fig. 10 (1st), is to apply our extracted features to all the classifiers in the current *node* and analyse their output values. Remembering that these classifiers tell us how certain they are that the presented sample data belongs to the module they represent, by applying our input sample's features to each of the binary classifiers (represented in the figure with the formula $BC(X) \in [0,1]$), we will obtain a group of values between 0 and 1 (because of the sigmoid activation function) that represent the likeliness of our input sample belonging to each of the modules in the current *node*. Then, we seek out the maximum value from these results, which will tell us which module is most likely to include the correct class. This module can either be an *endpoint* or a *node*

and that will dictate the next step. If the selected module is an *endpoint*, then the classification process is complete, and the predicted class is the class that belongs to that *endpoint*. If the selected module is a *node* then we enter that *node* and repeat this process (Fig. 10, 2nd) until the selected module is eventually an *endpoint* resulting in a final classification.

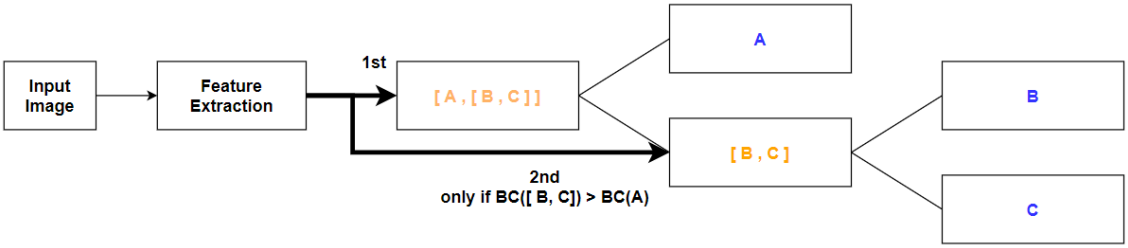


Figure 10: A demonstration of how the input data is the same for all sub-classifiers and not altered by their parent *nodes* (see also Figs. 6, 8 and 9).

This implementation is something that makes us able to meet one of our main objectives where we aim to avoid scalability issues. We manage this because only the sub-classifiers in the highest-scoring *nodes* are used, meaning that, in a lot of cases, we can make classifications using only a small percentage of the network. This means that as the network grows our classification speed will only be slowed down for similar classes that get grouped together, because a bit more time is required to make distinctions between them. In this context, in the future, we will explore solutions to balance the tree in such a way as to optimize both the classification process and the addition of new classes (see Sec. 3.5).

Figure 11 illustrates how the binary classifier results are compared with each other and how the maximum is used to decide which submodule to continue the process with.

Figure 12 shows a numeric example of the classification process being applied to a network whose modules are distributed in such a way that, depending on the class, a different percentage of the network is required to reach it. The predicted class in the example is one of the worst-cases in terms of network usage. The example network includes a total of 14 binary classifiers (1 for each *nodes'* sub-modules), but in the example, only 8/14 of the classifiers were used to make the final prediction (approx. 57% of the network) and this is the maximum possible amount of the network that can be used in this case.

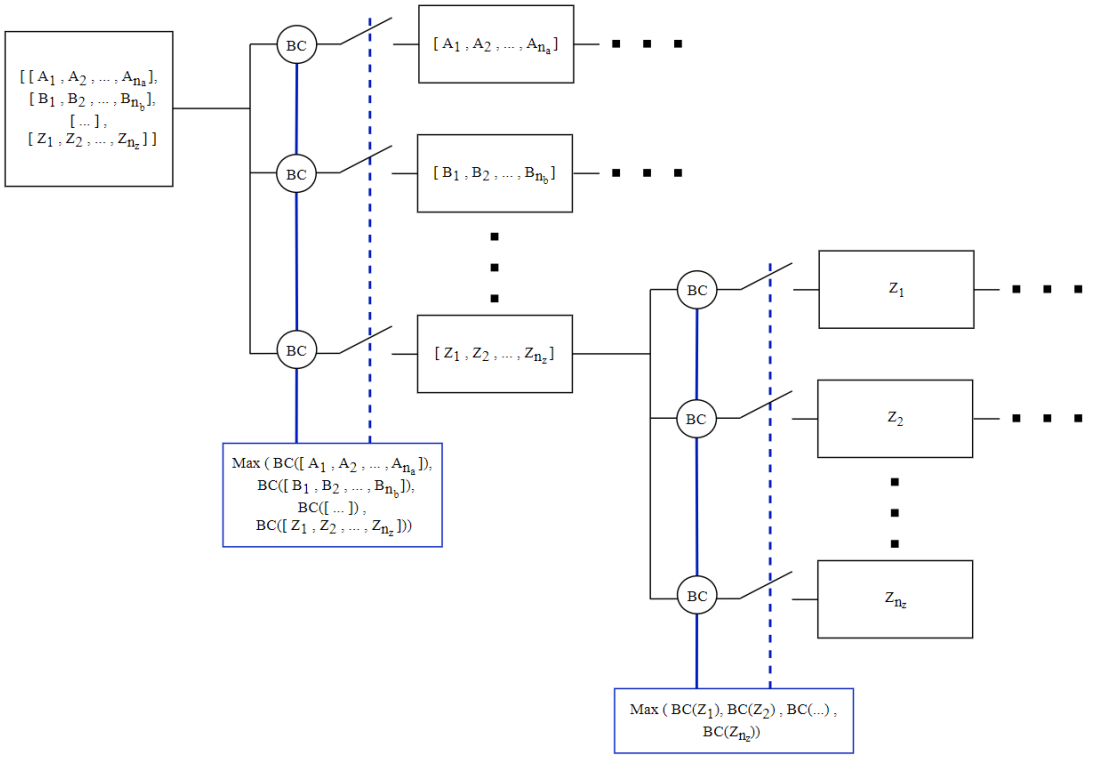


Figure 11: An illustration of the classification process. The process evaluates all the binary classifiers in a *node* and recursively re-applies itself to the highest scoring module until an *endpoint* is reached.

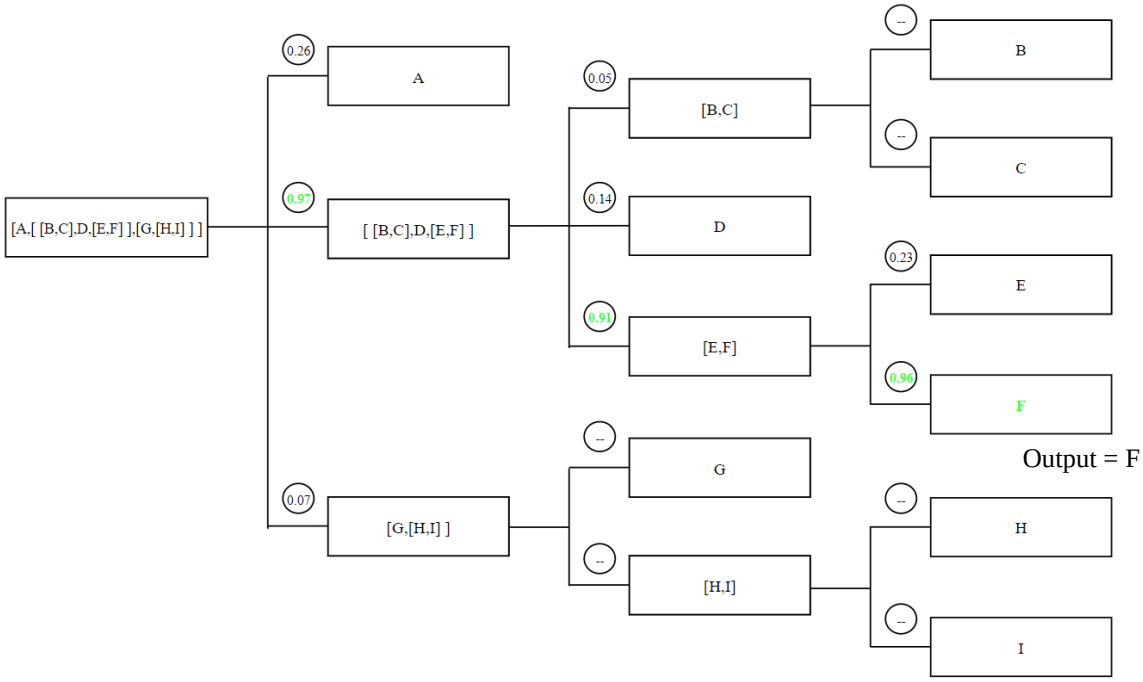


Figure 12: A numeric example of how only part of the network is necessary to make a classification.

The minimum possible amount of the network that could be used for this example would be the case of class A, where only 3/14 classifiers (approx. 21% of the network) would

be called. Table 1 shows the number of classifiers and the respective percentage of the network necessary for a prediction of each class from the example shown in Fig. 12. We should notice that for larger trees, depending on how they are balanced, these percentages could be drastically reduced, e.g., if we had 1024 classes on a perfectly balanced binary tree, we would only need to call 20 binary classifiers to obtain a classification.

Table 1: Number of classifiers necessary for each final classification and how much of the modular network those numbers equate to percentage-wise (in relation to the network shown in Fig. 12).

class name	Number of necessary classifiers	percentage of network
A	3	0.21
B	8	0.57
C	8	0.57
D	6	0.43
E	8	0.57
F	8	0.57
G	5	0.36
H	7	0.50
I	7	0.50
average	6.67	0.48
median	7	0.50

3.5 TRAINING

The addition of a class begins with the network being fed a set of data samples and a label, and with this, the network can: (i) process the data samples, (ii) decide where best to place a new *endpoint*, (iii) make any necessary adjustments to the existing modules, and (iv) train the necessary classifiers so that when the network is presented with data of the same class in future it can identify it.

Figure 13 depicts an example of how the network can change as new classes are added, where it starts off able to classify three classes, A, B and C, where B and C are grouped together. Then four more classes are added one at a time (D, E, F and G) making it able to classify a total of seven classes.

<table>
<tr><td rowspan="4">[A, [B, C]]</td><td colspan="2"></td><td>A</td></tr>
<tr><td colspan="2" rowspan="3">[B, C]</td><td>B</td></tr>
<tr><td>C</td></tr>
<tr><td rowspan="5">[[A, D], [B, C]]</td><td colspan="2" rowspan="2">[A, D]</td><td>A</td></tr>
<tr><td>D</td></tr>
<tr><td colspan="2" rowspan="2">[B, C]</td><td>B</td></tr>
<tr><td>C</td></tr>
<tr><td rowspan="6">[[A, D], [B, C], E]</td><td colspan="2" rowspan="2">[A, D]</td><td>A</td></tr>
<tr><td>D</td></tr>
<tr><td colspan="2" rowspan="2">[B, C]</td><td>B</td></tr>
<tr><td>C</td></tr>
<tr><td colspan="3"></td><td>E</td></tr>
<tr><td rowspan="6">[[A, D, F], [B, C], E]</td><td colspan="2" rowspan="3">[A, D, F]</td><td>A</td></tr>
<tr><td>D</td></tr>
<tr><td>F</td></tr>
<tr><td colspan="2" rowspan="2">[B, C]</td><td>B</td></tr>
<tr><td>C</td></tr>
<tr><td colspan="3"></td><td>E</td></tr>
<tr><td rowspan="6">[[A, D, F], [[B, C], E, G]]</td><td colspan="2" rowspan="3">[A, D, F]</td><td>A</td></tr>
<tr><td>D</td></tr>
<tr><td>F</td></tr>
<tr><td rowspan="3">[[B, C], E, G]</td><td rowspan="2">[B, C]</td><td>B</td></tr>
<tr><td>C</td></tr>
<tr><td colspan="2"></td><td>E</td></tr>
<tr><td colspan="2"></td><td>G</td></tr>
</table>

Figure 13: Demonstration of how the network can grow progressively by adding and joining modules.

The position in which new classes are placed in the network is not random. There are several processes involved in calculating the optimal position for a new class, some of which can be recursive. When we want our network to learn a new class, we are essentially asking it to be able to tell the new class apart from the ones which have already been learned. This brings us to the most important priority of our class placement process: avoiding conflict/confusion between classes. The best way, in our proposal, to avoid confusion between classes is to group them by their similarities and then focus on their differences. This concept of grouping classes refers to the placement of modules in

parallel within a node, as seen in Fig. 13, where, for example, in the first state of the network class B is grouped with class C.

3.5.1 Grouping by Similarities

To find similarities between a new class and existing classes, we make use of the binary classifiers placed within the nodes. The process of identifying similar classes is relatively similar to the classification process in Sec. 3.4.5, but instead of only searching for the single most similar class, here we are searching for any number of partially similar classes.

So, we are looking for somewhere to place our new class. When we enter a *node*, we are faced with *n* possible routes to follow which can be a mix of *endpoints* and/or *nodes*.

The idea is that any of our node or endpoint classifiers that might accidentally mistake the new class for their own, should be grouped with the new class and retrained, so that when presented with samples of the new class in the future they don't make the same mistake again. To achieve this, we must first see which *nodes/endpoints* are at all similar to our new class. So, the first thing we do is classify all the samples of our new class with all the classifiers in the current node, and then we can use these results to decide on how best to proceed.

Once we have all the current *node's* classifier results from all the new samples, we move on to calculate the average certainty per classifier, which gives us a single value for each classifier, representing how likely it is to mistake the new class for its own class (or classes if it represents a *node*). The use of the average was the first natural choice for an initial proof-of-concept, but in the future other solutions will be tested as well.

These average values can now be easily analysed to see which classes are the most like our new class. By establishing a threshold value AV_c (in the initial case, AV_c was empirically set to 0.3) to compare these values with, we can use that comparison to make a final decision on whether a class is similar or not. As this threshold value is what decides if two classes are similar or not, it should not be too high so as to not let any similar classes slip through the net, but also high enough so that only reasonably similar classes are considered. Values between 0.1 and 0.4 were what seemed to work best in our tests. The ideal value will always depend on the accuracy of the binary classifiers and will essentially decide how grouped or separated the final network will be. Future work will

also include a study of the effects of this threshold value and the application of a dynamic threshold value that is calculated as the network grows.

After we have compared the average results from the classifications of all the input samples with the threshold value (AV_c), we will know which, if any, modules are considered to be similar to the input class. This will lead us to one of the following five possible outcomes (summarized in Tab. 2) in function of the number of similar *nodes* and/or *endpoints*:

(a) 0 similar *nodes* or *endpoints*;

(b) 1 similar *node*, 0 similar *endpoints*;

(c) 0 similar *nodes*, 1 similar *endpoint*;

(d) >1 similar *node* and/or *endpoint*;

(e) All *nodes* and *endpoints* are similar.

Table 2: Possible outcomes during the recursive training process. With *0*, *1*, *>1* and *all* being the number of similar *nodes* and/or *endpoints*.

nodes / endpoints	0	1	>1	all
0	a	b	d	d
1	c	d	d	d
>1	d	d	d	d
all	d	d	d	e

Each one of these above outcomes is then dealt with differently (see also Fig. 14, for a draft of the architecture which demonstrates each of the mentioned outcomes):

a) *Place a new endpoint in the current node for the new class*.

   Train the classifier for the new *endpoint* with true data as data from the new class and false as a balanced distribution of data (see Sec. 3.6) from the other *endpoints* and *nodes* present within in the current *node*.

b) *Enter that similar node and repeat the process.*

c) *Create a new node and place inside it a new endpoint for the new class as well as the endpoint that was matched with the new class*.

   Train the classifier for the new *endpoint* with true data as data from the new class and false data as data from the class it was matched with.

Retrain the classifier for the pre-existing *endpoint* that was moved into the new *node* with its true data as the data it was stored with and false data as data from the new class.

d) Create a new node and place inside it a new endpoint for the new class as well as all the endpoints/nodes that were matched with the new class.

Train the classifier for the new *endpoint* with true data as data from the new class and false data as a balanced distribution of the data from all the other *nodes*/*endpoints* it was matched with.

Retrain the classifiers for all the pre-existing *nodes* and *endpoints* which were moved into the new *node* using balanced distributions of their own data as true data and balanced distributions of their sibling's data as false data.

The children modules of the *nodes* that were moved into the new *node* do not need to be touched as they are not backward dependent and only relate to each other.

e) *Place a new endpoint in the current node for the new class*.

Train the classifier for the new endpoint with true data as data from the new class and false data as a balanced distribution of data from the other *endpoints* and *nodes* present in the current *node*.

Retrain the classifiers for all the pre-existing *nodes* and *endpoints* in the current *node* using balanced distributions of their data as true data and balanced distributions of their sibling's data as false data.

Of the above processes, case (b) is the only one which does not involve placing the new class or training any networks, but it does repeat the entire process applied to an identified sub-*node* and eventually, there will be a point where there are no more sub-*nodes* and case (b) will no longer be an option so, the new class is guaranteed to eventually be placed somewhere in the network.

Once the new class has been placed somewhere within the network, it is necessary to retrain the parent *node* to also consider the new class as true data, so as to increase the chance of success of the classification process, by increasing the chance of samples of the new class reaching their respective *endpoint*. This retraining of the parent *node* must then be applied to the parent of the parent, and the parent of the parent of the parent, and so on and so forth, until reaching the root node of the network. This process helps to guarantee that, when presented with samples of the new class in the future, the initial *nodes* will be more likely to send the data down the correct path.

As we are using the same classifiers for the training process as we do in the classification of new data, when we retrain/correct the most problematic parts of the network, we are immediately making drastic improvements to the classification process. This is due to the fact that we are predicting and correcting the modules that will cause the most errors after adding the new class.

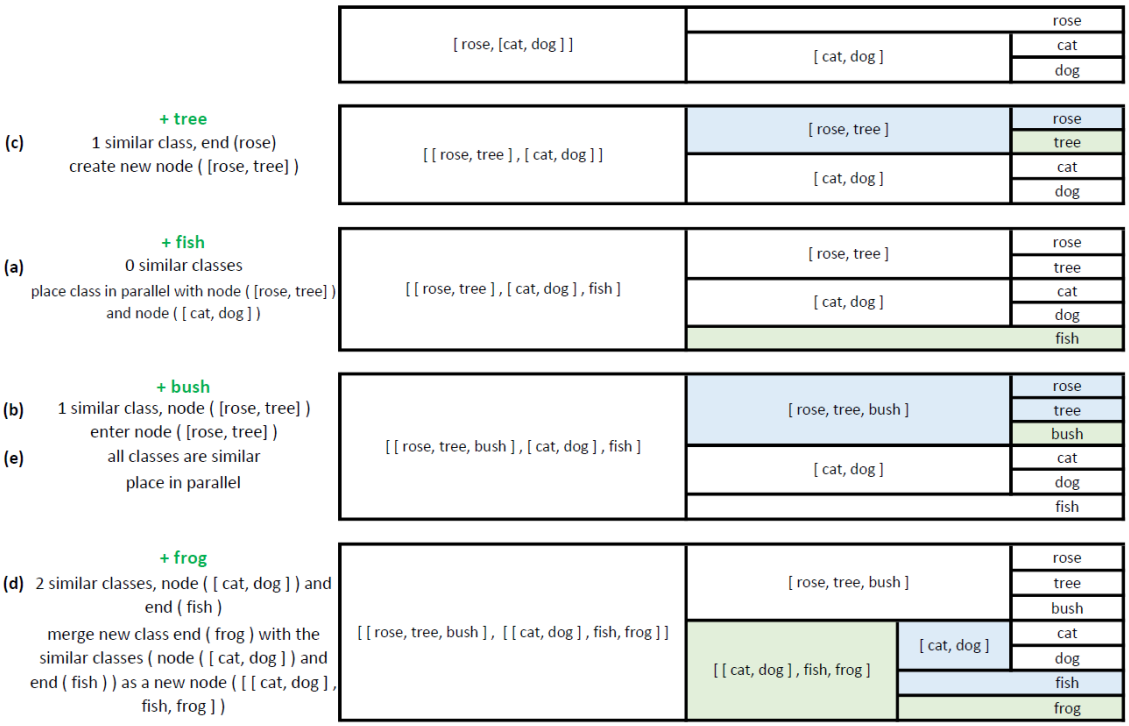


Figure 14: An example of the expansion of the network as classes are added 1 by 1 with a description of each decision referring to the options available in Tab. 2.

In Fig. 14 we re-visit the progression of the network shown in Fig. 13, but here we use more realistic examples of similar classes and show where and how the different situations from Tab. 2 occur, and how their respective procedures are applied when adding new classes to the network. The modules highlighted in green are new modules and the modules highlighted in blue are pre-existing modules that had to be retrained either to avoid confusion or to ensure that the parent nodes are aware of the new class.

3.6 BALANCED TRAINING DATA

Neural networks are proven to have higher success rates when trained on balanced data (Hensman *et al.*, 2015), meaning that they should, in principle, be trained with approximately the same number of samples per class. And as our classifiers only have

two classes (true and false), this means that we should, in principle, aim for an equal number of true samples and false samples leaving us to tackle the problem of what to do when we have different numbers of true or false samples.

In some cases, when the inputs are images, data augmentation can be performed on the samples to generate more samples of a class, synthetic data can be created etc., but the data we store and the data these classifiers are working with are feature values and not images, so data augmentation is not directly applicable here.

It is important to stress that the first block – feature extraction – of the framework does not receive any changes whatever the input or its class may be, i.e., there is (as already mentioned) no training of these initial layers of the network. So, in our case, even though we are working with images, our "input" for the binary classifiers is the output data from the feature extraction, which cannot be augmented.

Considering that true and false can each be their own combinations of classes and groups of sub-classes and that we want the same number of true and false samples, this created the need for us to develop an algorithm for calculating an ideal distribution of data.

The algorithm we implemented for selecting training samples calculates the maximum possible number of samples per class that can maintain an optimal distribution based on their positions in the network. In our situation, where we have various groups of classes with their own sub-groups, an optimal distribution does not mean using the same number of samples for each class. It means that the same number of samples should be used from each of a *nodes'* children, i.e., if one of these children is an *endpoint* and one is a *node* then the same number of samples will be used from each, where the *nodes* samples will be a mix of its children's samples and so on. For example, if we were training a classifier to recognize an *endpoint* A as true and a *node* [B, C] as false, and A, B and C all had N samples each, a balanced distribution for this classifier would be N samples of A and N samples of [B, C]. Then, to maintain an equal distribution, the N samples of [B, C] would consist of $N/2$ samples of B and $N/2$ samples of C.

This example makes the problem seem quite straightforward, but when we have different numbers of samples per class and a more complex network, with various nested modules on both the true and the false side, it is necessary to have an algorithm that can calculate the optimal distribution for any situation and still make use of as many samples as possible.

Maintaining this equal distribution, unfortunately, signifies that the classes with fewer samples will dictate how many samples can be used from the classes with more samples.

So, it is recommended to establish a *minimum number of samples* (*MNS*) to use when a class is added to the network to reduce this effect. In our case we empirically defined *MNS* as 175. More tests will be done in the future to determine the best *MNS* value.

There are two main steps to this process: (i) The first one is to calculate the largest possible number of samples that permits us to use the distribution we just explained. (ii) The second is to recursively divide this number by the number of children in a *node* until all the *endpoints* are reached, leaving us with the number of samples to use per class.

The first step of the algorithm, Fig. 15 (left), calculates the maximum possible number of samples that lets us maintain an equal distribution for a *node* by multiplying the number of sub-modules in the *node* by the number of samples of the sub-module with the smallest number of samples. The number of samples to be used from the sub-modules which are also *nodes* are calculated the same way. This means the process is applied recursively until all the sub-modules that belong to the *node* we are calculating have also been calculated. When this process finishes, we end up with the maximum number of samples that lets us maintain an equal distribution for the *node* requested. In Fig. 15, M_p is the maximum possible number of samples that lets us maintain an equal distribution for the *node* being calculated. $M_{p,i}$ represents the maximum number of samples that can be used by one of the node's sub-modules. In the indexation of $M_{p,i}$, *p* represents the path created by the indexations that lead to the location of the *node* in question, e.g., in the final state of the network shown in Fig. 14, the path to reach the *node* "[cat, dog]" would be the second sub-module of the root *node* and then the first sub-module of that *node*. Meaning that for the *node* "[cat, dog]", the indexations represented by *p* would be "2,1", and the number of samples present in the *endpoint* "dog" would be "$M_{2,1,2}$". The value of n_p represents the number of sub-modules present in the node being calculated.

Following Fig. 15, the number of samples is computed as follows:

$$M = n_p \times \min\left(M_{p,1}, M_{p,2}, \dots, M_{p,n_p}\right).$$

The second step of the algorithm, Fig. 15 (right), is much simpler. We use the value obtained from the first step (the maximum possible number of samples that lets us maintain an equal distribution) and progressively divide it throughout our network. The *node* being calculated distributes its number evenly between its children. The children that are also *nodes* then do the same thing with their values to their own children, resulting in sub-divided values. This is repeated until all sub-modules of the initial *node* are reached and eventually results in a final number of samples to be used for each class (E_p).

With the value of M_p calculated in part one, it is possible to compute $E_{p,i}$, which is essentially the value of M_p evenly distributed between each of the *node*'s sub-modules. The indexations with p and the value of n_p maintain the same logic as described in step one.

While in step one the values of the parents depended on the values of the children, in step two the values of the children depend on the values of the parents. To initialize the process, the *node* being calculated has its value set with $E_p = M_p$ and then it's sub-module's values are calculated as $E_{p,1} = E_{p,2} = \cdots = E_{p,n_p} = \frac{E_p}{n_p}$. Then, each sub-modules' sub-modules' are calculated the same way until all the *endpoints* that are descendants of the *node* being calculated are reached, which will eventually result in a series of final numbers of samples to be used from each endpoint.

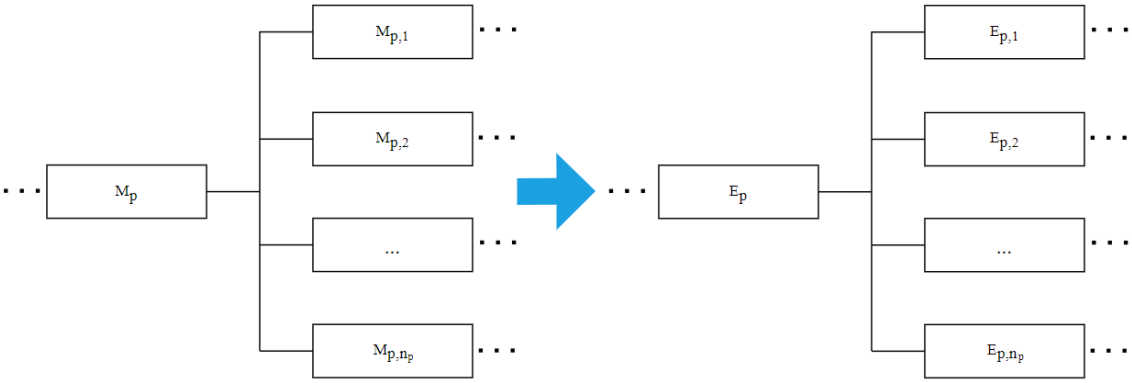


Figure 15: The two mains steps of the optimal data distribution process

This strategy will also be subject to further studies, as it is possible that with large trees and/or classes that have a small number of samples, the number of input images to train some classifiers could be too low to achieve a proper classification accuracy. As mentioned, data augmentation might not be a straightforward solution, but alternatives will be explored.

Figure 16 is an example of the algorithm being applied to the network from Fig. 14 in its initial state, where it only knows three classes. The illustration shows how the samples would be divided for training rose as true and [cat, dog] as false, at the root *node* of the network. The first row simply shows the network with its class names for reference. The second-row shows the maximum possible number of samples that allow for an even distribution per node (800 in this example) calculated using step 1 of the algorithm. And the third row shows the divisions of the final numbers that result in the number of samples

to be used per class, calculated using step 2 of the algorithm. So, for this example, the resulting numbers of samples per class to be used for training the rose classifier would be as follows: 400 rose samples for true, and 200 cat samples plus 200 dog samples for false. This strategy results in a balanced total of 400 samples for true and 400 samples for false, as desired.

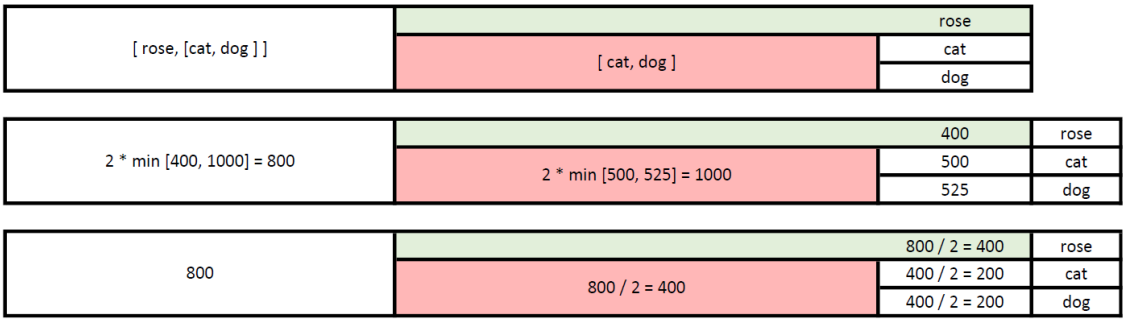


Figure 16: An example application of the optimal data distribution algorithm being applied to the first state of the network shown in Fig. 14.

Figure 17 is a slightly more complex example of the algorithm, where it is applied to the network's final state from Fig. 14, where seven classes are known. The context of this example is the calculation of equal data distribution for a classifier that considers the *node* [rose, tree, bush] as true and the *node* [[cat, dog], fish, frog] as false. In this example we consider that our stored data consists of the following samples: 400 roses, 350 trees, 450 bushes, 500 cats, 525 dogs, 475 fish, and 550 frogs.

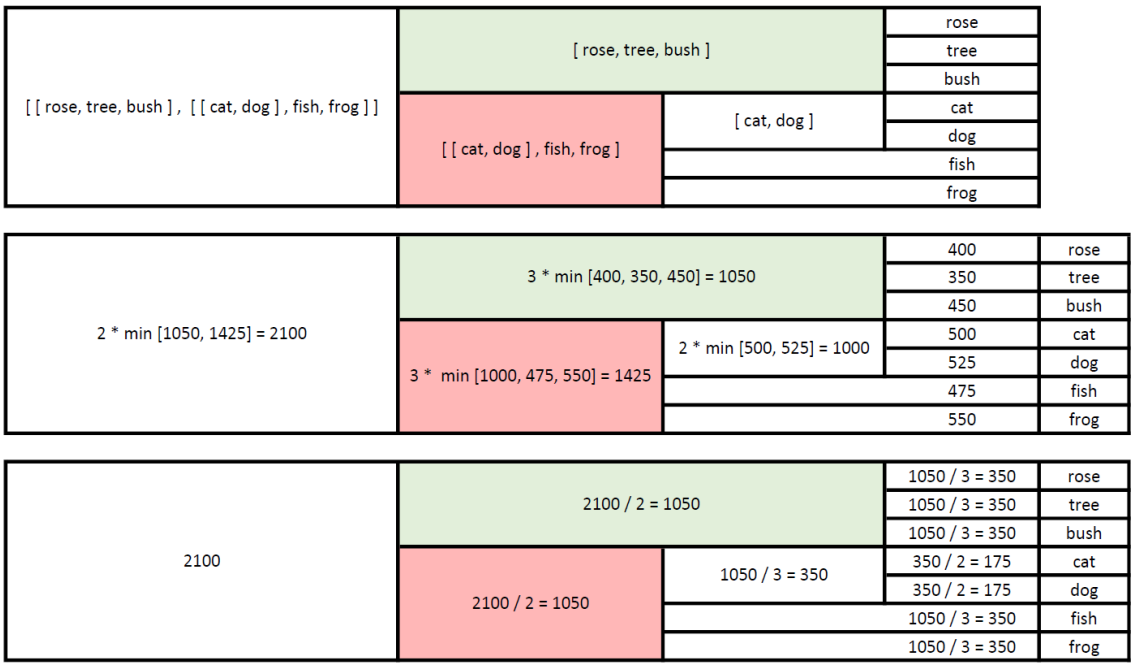


Figure 17: An example application of the optimal data distribution algorithm being applied to the final state of the network shown in Fig. 14.

The resulting numbers of samples per class to be used for training the classifier for the *node* [rose, tree, bush] in Fig. 17 would be as follows: 350 rose samples, 350 tree samples and 350 bush samples for true; and 175 cat samples, 175 dog samples, 350 fish samples and 350 frog samples for false. Again making a balanced total of 1050 samples for true and 1050 samples for false.

3.7 DISCUSSION

We present an ANN-based architecture capable of learning new classes one after another, while retaining previous knowledge. This functionality is made possible by its dynamic behaviour, where the network grows as new classes are added by adding and rearranging the internal neural networks. This type of approach would normally lead to extreme scalability problems, but the architecture is structured in such a way that it functions using only parts of the network, not only during classification but while learning too. The optimal positioning and structuring of the network's internal components is done automatically with no need for human interference, but at the same time, it is flexible enough that humans can easily make adjustments if they so desire.

These types of functionalities are made possible by the use of a modular approach, which makes it much easier to move things around and apply logical algorithms that dictate how things should be structured.

The architecture is still in its initial stages as a proof of concept, but with so many configurable components there are many ways it can be improved and also many ways it can be fine tuned for different applications. In the next chapter some tests and results are presented.

4. Tests and Results

ABSTRACT

Using the methods and algorithms described in the dissertation we present some tests and results to demonstrate the performance of the proposed architecture.

4.1 INTRODUCTION

Image classification networks are usually validated by learning from and performing a series of predictions on one or multiple well-known datasets and measuring their accuracy in order to compare themselves with other networks. Being a recent concept, Continual Learning networks do not have many well-known and reputable datasets for their validation. While some do exist, they do not hold the same reputation as popular datasets like ImageNet.

One of the datasets we mentioned in the state-of-the-art, Core50 (Lomonaco *et al.*, 2017), while created for the purpose of CL validation, is structured in such a way that is not compatible with how our network learns. In Core50, data samples from already known classes are also presented as training data, in order to attempt to improve the knowledge of previously known classes, but our architecture currently learns one class at a time. While it is included in our future work to develop ways to perform that type of learning as well, due to time constraints, we decided to perform our tests on traditional image databases, namely: ImageNet and CIFAR10 (Krizhevsky *et al.*, 2009).

Because our network learns over time, we decided to re-calculate the accuracy values every time we added another class, to show how the accuracy evolved over time/new class additions. We demonstrate the accuracy per class as well as the global accuracy, and we also show the evolution of the modular network over time.

We performed two tests using some classes from the ImageNet dataset, one using 11 classes and one using 25 classes. The ImageNet dataset consists of 1,000 classes with a different number of samples per class and images of different sizes. The reason we did not perform a full set of tests with the ImageNet dataset was that for images of this size, the training and classification processes are very costly in terms of processing power, and we did not have access to hardware that would allow us to do this in reasonable time frame.

In a complementary fashion, to test how the network (MDNN) behaves and not just the results it can achieve, we used CIFAR10. Made up of 10 classes with 6,000 samples each, CIFAR10 has a total of 60,000 samples, where each class has 5,000 training samples and 1,000 test samples. The images are fairly small at 32×32 pixels and 3 colour channels.

As we mentioned before, our network re-sizes images to a $224 \times 224 \times 3$ tensor to prepare them for the ResNet50 feature extractor, but for the sake of simplicity, with the CIFAR10 dataset, we simply altered the ResNet50 feature extractor's input to accept $32 \times 32 \times 3$ tensors.

Figure 18 shows the difference between ImageNet images and CIFAR10 images, with all images in the figure resized to the same height. As expected, the CIFAR images were of very low quality, and this will deeply affect the results. Nevertheless, it is a useful dataset to show how our MDNN behaves without spending several weeks performing tests.

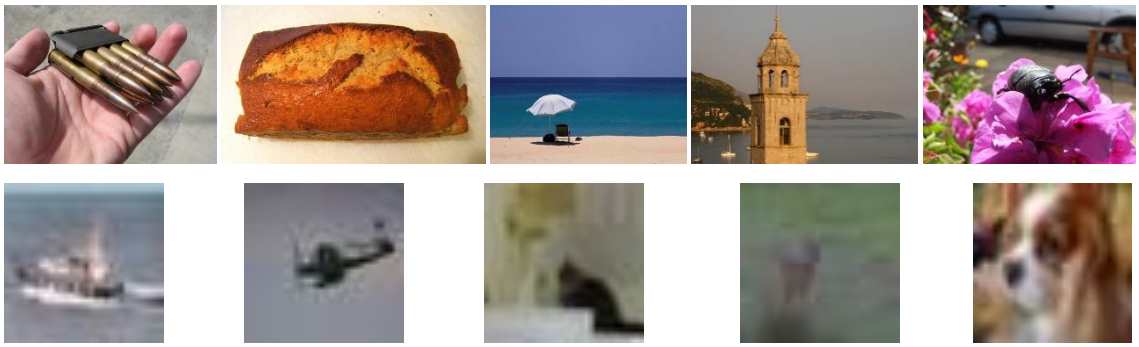


Figure 18: Examples of images from ImageNet (top) and CIFAR10 (bottom).

4.2 IMAGENET TESTS

As mentioned, two tests were performed using data from ImageNet: one with 11 random classes and one with 25 random classes from the original 1,000 classes. All ImageNet tests were performed using 400 images from each class (300 for training and 100 for testing).

The network's final structure for the test with 11 classes can be seen in Fig. 19 top, and at the bottom a simplified representation for the same network. Both representations show how the network grew progressively by adding and joining modules (classes).

Table 3 shows how the accuracy behaves when we classify more/different classes, with accuracy being defined by the number of correct predictions divided by the total number of predictions.

We start with two classes, "airplane propeller" and "almond tree", and after that, as each new class is learned, we present the accuracy result for the new class as well as for the already known classes. It is important to stress that this result shows only "*1 run*", this means that there was no selection of best or worst results, but just one random run.

<table>
<tr><td rowspan="11">[[A, [B, C]],
[D, [E, F], [[G, H], [
I, [J, K]]]]]</td><td rowspan="3">[A, [B, C]]</td><td colspan="4">A. almondtree</td></tr>
<tr><td colspan="3">[B, C]</td><td>B. alyssum</td></tr>
<tr><td>C. astilbe</td></tr>
<tr><td rowspan="8">[D, [E, F],
[[G, H],
[I, [J, K]]]]</td><td colspan="4">D. artillery</td></tr>
<tr><td colspan="3">[E, F]</td><td>E. ammunition</td></tr>
<tr><td>F. baggage</td></tr>
<tr><td rowspan="5">[[G, H],
[I, [J, K
]]]</td><td colspan="2">[G, H]</td><td>G. airplanepropeller</td></tr>
<tr><td>H. backpackingtent</td></tr>
<tr><td rowspan="3">[I, [J, K]]</td><td colspan="2">I. amphibian</td></tr>
<tr><td rowspan="2">[J, K]</td><td>J. anchovipizza</td></tr>
<tr><td>K. bananabread</td></tr>
</table>

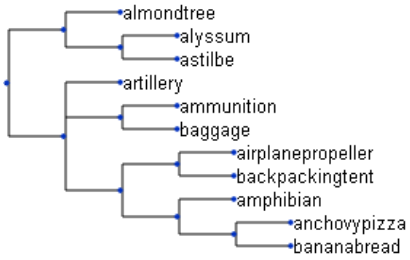


Figure 19: Two different representations of the resulting network structure of a test performed using 11 classes from ImageNet (see text for details).

Table 3: Using 11 classes from ImageNet – accuracy per step/class additions.

Classes \ *Steps*	2 (init)	3	4	5	6	7	8	9	10	11
airplane propeller	1.00	1.000	1.00	1.00	1.00	1.00	1.00	0.98	0.98	**0.94**
almond tree	0.97	0.92	0.92	0.95	0.96	0.96	0.96	0.94	0.96	**0.94**
alyssum		0.79	0.79	0.79	0.80	0.80	0.32	0.32	0.32	**0.32**
ammunition			0.92	0.95	0.92	0.88	0.89	0.88	0.75	**0.72**
amphibian				0.96	0.94	0.94	0.95	0.96	0.98	**0.97**
anchovy pizza					0.98	0.98	0.98	0.99	0.99	**0.89**
artillery						0.85	0.85	0.88	0.91	**0.91**
astilbe							0.68	0.67	0.68	**0.68**
backpacking tent								0.91	0.91	**0.83**
baggage									0.56	**0.80**
banana bread										**0.90**
Mean accuracy	0.99	0.90	0.91	0.93	0.93	0.92	0.83	0.84	0.80	**0.81**

The Column labelled as "11" shows the final accuracy of the framework *per* class. Of the 11 classes, 2 (~18%) of the classes (marked in red) present very poor results, 4 classes (~36%) present accuracy above 0.70 (marked in yellow), and 5 (~45%) present accuracy results above 0.90 (marked in green). The overall mean accuracy is 0.81. Looking at the last line in Tab. 3, it appears, that as more classes are added to the network, the overall

accuracy is more or less stable except for two significant decreases in columns 3 and 8. These two decreases happened when the two lowest scoring classes were added to the network ("alyssum" and "astilbe"). These classes are somewhat similar as they are both plants, and although they were grouped together, it appears that the binary classifiers responsible for specializing in distinguishing between the two classes did not do a good enough job in this case.

We performed the same test but now with 25 classes (again, 300 training images and 100 test images for each class). We show the final accuracy results in Tab. 4, and we show the growth of the network over time/class addition in Tab. 5 (using the simplified representation). Table 4 shows that the framework did not do a great job of correctly classifying most of the classes by the time it reached its final stage. Out of the 25 classes, 16 (64%) of them (marked in red to indicate an accuracy under 0.7) present poor results, 4 (16%) present results above 0.7 (marked in yellow), and only 5 (20%) present accuracy results above 0.9 (marked in green).

Table 4: ImageNet using 25 classes, final step accuracy results.

airplane propeller	0.11	artillery	0.08	bechtel crab	0.43	bobby pin	0.29	cat	0.53
almond tree	0.54	backpacking tent	0.48	bell cote	0.30	boxer	0.92	elevator shaft	0.26
ammunition	0.58	banana bread	0.98	blacktop	0.86	brachyuran	0.34	shih-tzu	0.94
amphibian	0.74	barley	0.57	blade	0.24	brain coral	0.24	space shuttle	0.99
anchovy pizza	0.86	beach	0.89	blinker	0.32	Britisher	0.31	stag beetle	0.99
						Overall accuracy	**0.55**		

Despite an expectation of lower accuracy results with Continual Learning, when compared with the state-of-the-art non-Continual Learning methods, the proposed method achieved very poor results for 64% of the classes and achieved good or excellent results for 46% of the classes. This is somewhat better than we initially expected because the algorithm has not yet been fine-tunned. In other words, this being the very first version of the MDNN framework, the main goal has been to prove its concept. In future versions, it is expected that these results will improve significantly.

Table 5 shows how the network dynamically adapts and builds itself without any human supervision as each class is learned. It is possible to see the network structure changing over time with the 24 steps of the network's expansion, displayed from left to right, top

to bottom. Also, it is interesting to compare the final structural results of the test with 25 classes (Tab. 5 bottom right) against the final structural results of the test with 11 classes (Fig. 19 bottom): despite there being a different number of classes in each test, we can clearly see that the structure of the network is completely different, showing that the dynamic modules work as expected.

It is important to stress that one of the major contributions of the MDNN framework is its capacity for dynamic changes, which create a deeper network without any kind of supervision and/or predefined static structure. To demonstrate and test this we used the CIFAR10 dataset.

Table 5: Using 25 classes from ImageNet – network's structure over time. Left to right, top to bottom, the 24 steps of the network growth (1/2).

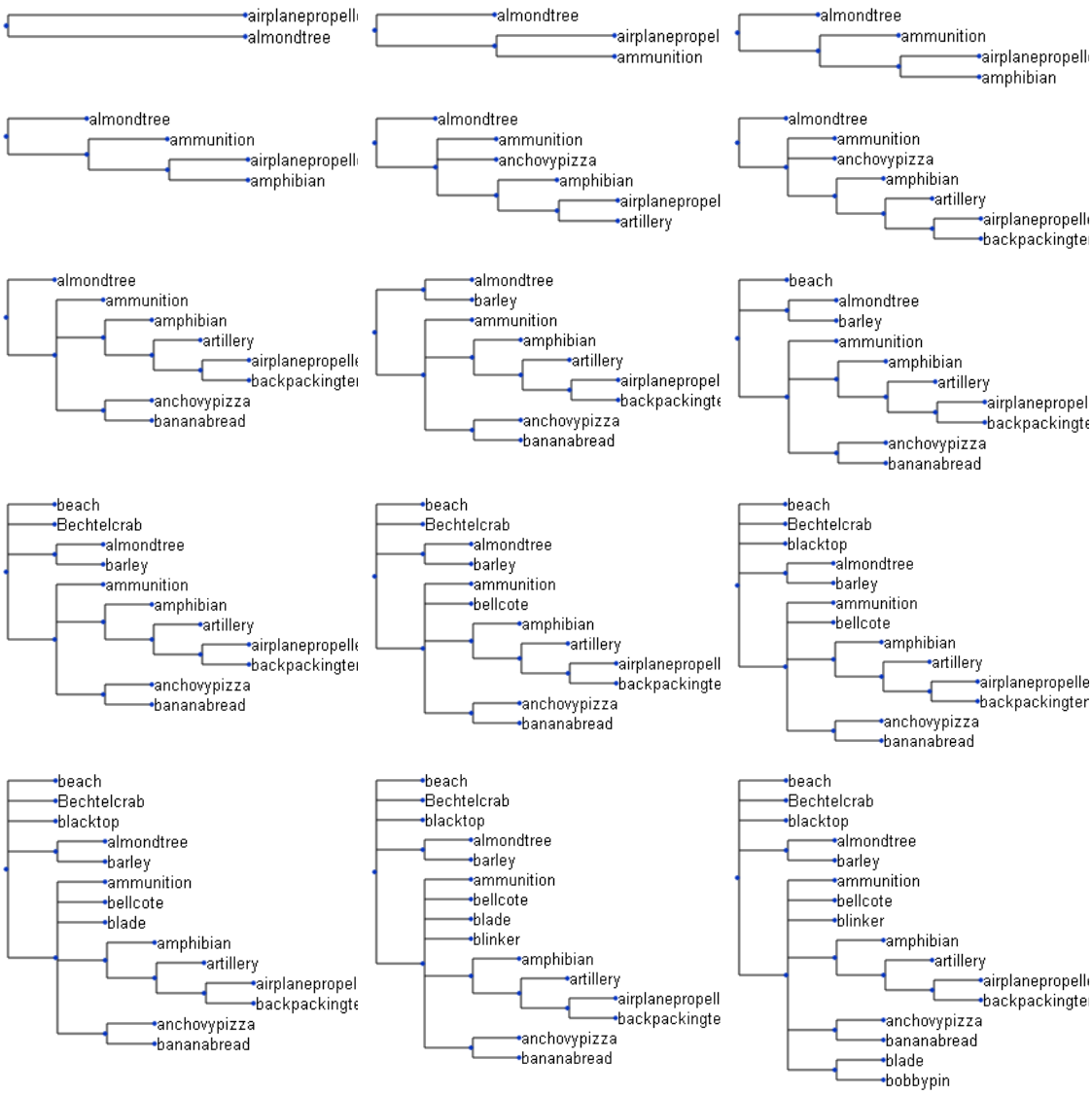
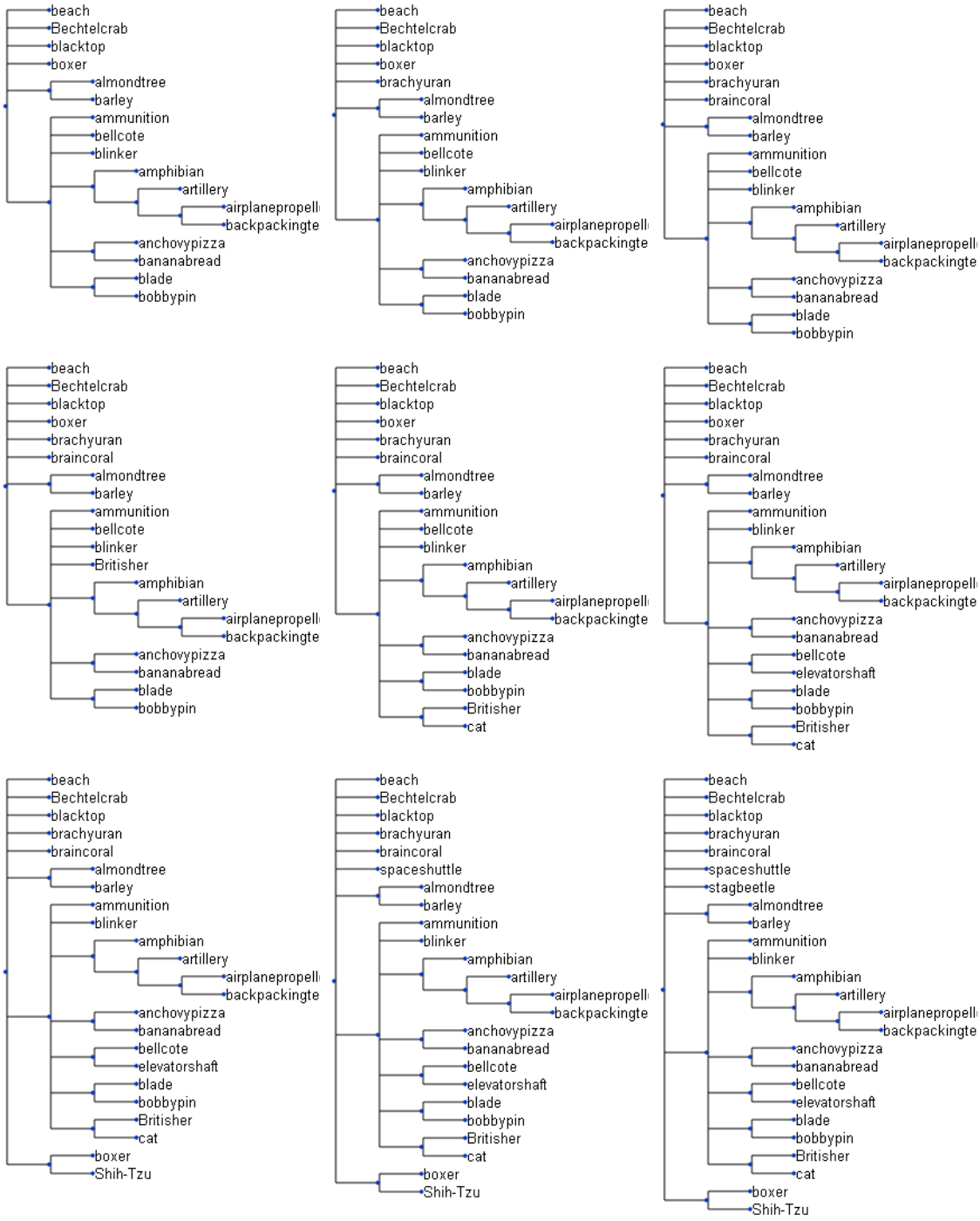


Table 5: ImageNet using 25 classes - network structure over time (2/2).



4.3 CIFAR10 TESTS

As mentioned earlier, the CIFAR10 dataset is made up of 10 classes each with 6,000 samples, making a total of 60,000 samples, where each class possesses 5,000 training samples and 1,000 test samples. The images are 32 × 32 pixels (see Fig. 18 bottom) with

3 colour channels (32 × 32 × 3). We use this dataset for two purposes: (i) to show how the network learns, i.e., how different "branches" of the network change dynamically and how that creates networks with different depths and structures. (ii) To show how results change for different learned configurations of the network (different final network structures).

By using CIFAR10 we assume from the start that we will have very poor results, due to: (a) the size of the images in the dataset is quite small compared to the ones in ImageNet (again, we call the attention to Fig. 18 where we compare the difference between image qualities). (b) We are working with Continual Learning, the dataset should/must be different from the standard dataset, CIFAR10 is a standard dataset. It is important to stress that for all CL methods (found in current literature), the accuracy results are always much lower than those of non-continual (state-of-the-art) DNN methods.

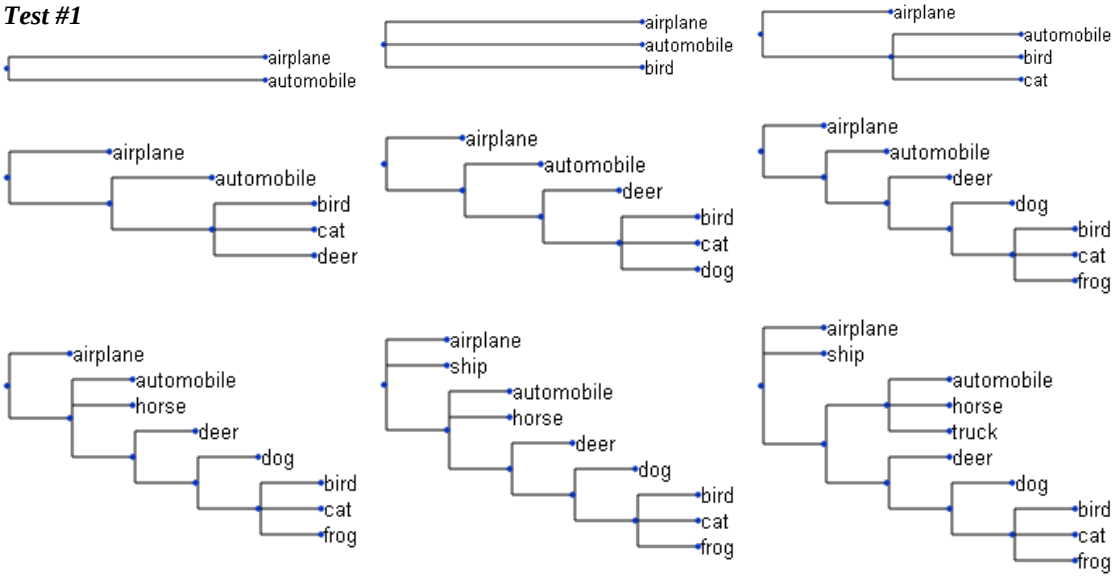
The big advantage of using CIFAR10 is that it allows us to train networks much faster because of its smaller sized images, which, in turn, allows us to perform multiple tests and validate the network's behaviour.

The 3 tests shown in Tab. 6 demonstrate how differently the networks learn the different classes, even though we always start with the same two classes and learn the others in the same order. The network's behaviour is somewhat "similar" to that of our brains, i.e., we associate certain objects with each other as we learn them, and then make new, different, associations as we learn more. For instance, two children that are familiar with "tomatoes" and "apples" are then presented with a new fruit, e.g., "peach" (similar shape and colour), one child may associate "peaches" with "apples" and the other may associate them with "tomatoes". And over time and with more sightings ("samples") of peaches the distinction between objects becomes clearer.

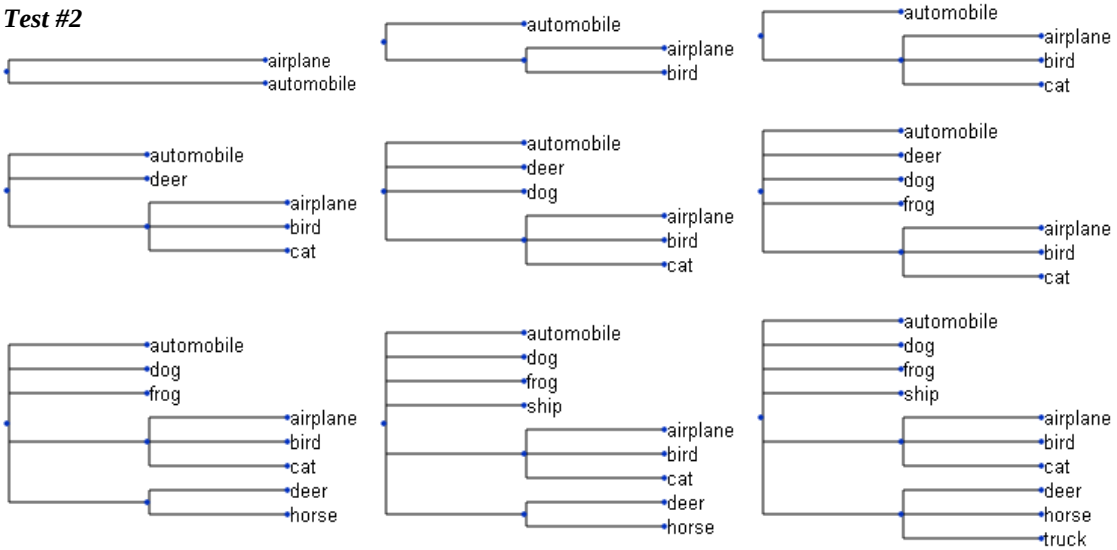
Looking at Tab. 6, we can see three different tests (#1, #2 and #3), in which straight away in step 2, where the third class is added to the initial two (2nd column, lines 1, 4 and 7 respectively for each test) the networks start learning and building themselves in different ways. In tests #1 and #3 the network has the same configuration but, test #2 shows a different configuration, and as we go along the different steps adding more and more, we start to see many different configurations. The last step in each test shows the final states of the 3 networks (3rd column, lines 3, 6 and 9), and we see that all the configurations are quite different.

Table 6: CIFAR10 network growth. Left to right, top to bottom, the 9 steps of the network structure growth for 3 tests.

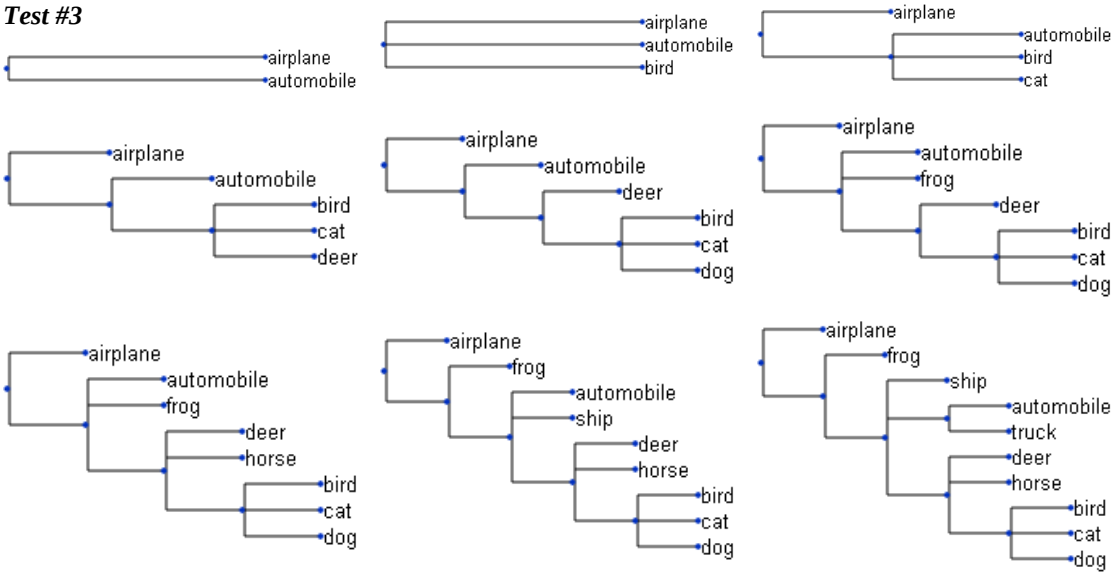
Test #1



Test #2



Test #3



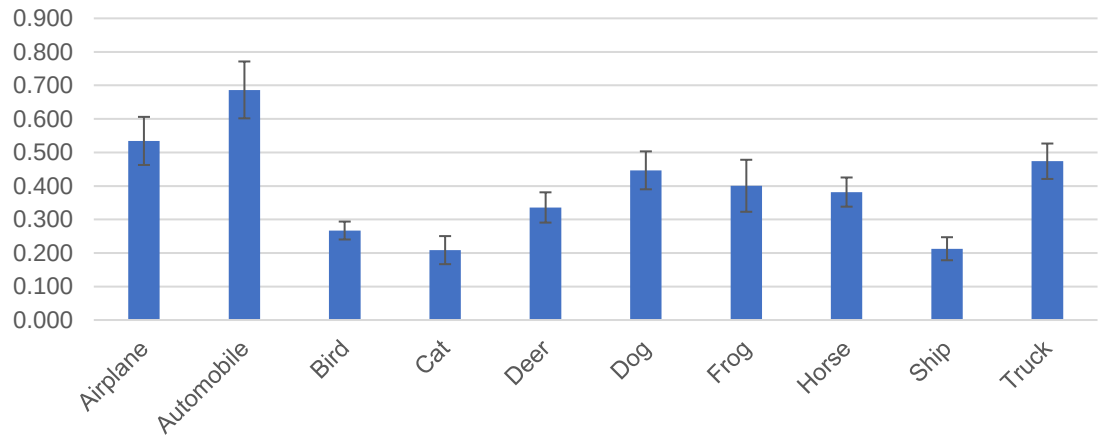
The next question that arises is: if these configurations are so different, are the results also very different? Table 7 presents the results from those three tests and it shows the accuracy for each class over time (as the classes are inserted) for the entire network, from its initialization all the way up until the end of learning all 10 classes, and then a mean overall accuracy is shown. Again, as with previous examples, red means very poor results and yellow means results over 0.7.

Table 7: CIFAR10 accuracy over time, results from test #1, #2 and #3.

Classes \ *Step*	***Test***	2 (init)	3	4	5	6	7	8	9	10
Airplane	#1	0.76	0.51	0.51	0.59	0.58	0.52	0.55	0.45	0.58
	#2	0.77	0.55	0.31	0.26	0.25	0.22	0.18	0.15	0.17
	#3	0.80	0.47	0.43	0.49	0.52	0.68	0.47	0.56	0.61
Automobile	#1	0.97	0.94	0.79	0.88	0.90	0.92	0.84	0.81	0.71
	#2	0.96	0.92	0.79	0.66	0.45	0.38	0.23	0.15	0.09
	#3	0.95	0.87	0.86	0.90	0.90	0.83	0.88	0.82	0.62
Bird	#1		0.73	0.63	0.47	0.45	0.40	0.14	0.12	0.27
	#2		0.75	0.48	0.30	0.25	0.19	0.16	0.16	0.18
	#3		0.84	0.65	0.49	0.39	0.29	0.35	0.30	0.24
Cat	#1			0.72	0.53	0.26	0.12	0.05	0.05	0.09
	#2			0.47	0.35	0.11	0.08	0.06	0.06	0.05
	#3			0.64	0.50	0.41	0.28	0.39	0.34	0.27
Deer	#1				0.37	0.52	0.39	0.20	0.20	0.41
	#2				0.72	0.69	0.64	0.48	0.47	0.51
	#3				0.46	0.54	0.51	0.31	0.28	0.23
Dog	#1					0.55	0.63	0.19	0.19	0.55
	#2					0.72	0.69	0.65	0.65	0.64
	#3					0.38	0.33	0.39	0.36	0.32
Frog	#1						0.26	0.18	0.18	0.27
	#2						0.59	0.66	0.66	0.67
	#3						0.45	0.54	0.62	0.74
Horse	#1							0.75	0.77	0.37
	#2							0.50	0.50	0.41
	#3							0.43	0.42	0.35
Ship	#1								0.27	0.33
	#2								0.28	0.29
	#3								0.14	0.08
Truck	#1									0.31
	#2									0.67
	#3									0.58
Overall accuracy	#1	0.86	0.73	0.66	0.57	0.54	0.46	0.36	0.34	**0.39**
	#2	0.87	0.74	0.51	0.46	0.41	0.40	0.37	0.34	**0.37**
	#3	0.88	0.73	0.65	0.57	0.52	0.48	0.47	0.42	**0.41**

In conclusion, all results are quite poor, nevertheless, we can see that despite the accuracy changes, there are no major differences between final accuracy results, meaning that the network can grow in different ways and maintain a consistent performance. Of course, as is the case with any ANN the MDNN is in need of some tuning, and after that, the overall results are expected to improve.

Figure 20 presents the final accuracies for 15 tests, showing the average accuracy per class and the standard deviation graphically (top) and numerically (bottom). With the average final accuracy also presented. These results validate the consistency of the network's performance.



Class	Airplane	Automobile	Bird	Cat	Deer	Dog	Frog	Horse	Ship	Truck	**Overall**
Average	0,53	0,69	0,27	0,21	0,34	0,45	0,40	0,38	0,21	0,47	**0,40**
Stand. Dev.	0,14	0,17	0,05	0,08	0,09	0,11	0,16	0,09	0,07	0,11	0,02

Figure 20: Final averages and standard deviations per class from 15 tests on CIFAR10 dataset.

Finally, Tab. 8 shows a comparison between our proposed method and the top-ranking methods in the state-of-the-art for classification accuracy on the CIFAR10 dataset. It is immediately noticeable that our method presents a huge difference from the state-of-the-art methods, nevertheless, again we stress that the current focus is on its ability to learn classes continually. While our method does not surpass the top-ranking methods, this result does validate that it works. Over time, we expect to drastically improve this accuracy.

Table 8: A comparison of the classification accuracy between our method and the top-ranking methods in the state-of-the-art. Table extracted from Papers with Code[1] (in 2020/20/09).

Rank	Model	Percentage Correct	Extra training data
1	BiT-L (Kolesnikov *et.al.*,2019)	99.37	Yes
2	GRIPE + transfer learning (Huang *et.al.*, 2019)	99.00	No
3	TResNet-XL (Ridnik *et.al.*,2020)	99.00	No
n/a	MDNN (ours)	39.50	No

4.4 Testing the Framework with Real World Images and Camera Streams

The last goal of the dissertation was to create a framework (MDNN) capable of learning from real world images from a video stream. To show that this is possible with our network, we used the previously established network from the test with 11 ImageNet classes (see Sec. 4.2) and added two more classes to it. These two new classes consist of image samples acquired using a regular HD webcam in a home/laboratory environment. Those classes were “teacup” and “glass bottle”.

The video data was acquired by filming different objects of those classes with different view angles. Images to train our network with were then extracted as frames from the final videos in intervals of 0.5 seconds and then fed to the network to perform the learning process. New objects were filmed in order to generate the test data. These new objects were from the two classes that were learned but they were not present in the training videos. This was to make sure the resulting network was not generalized to the specific objects present in the training videos. Three hundred and ninety samples/frames of each of new class were used to train the network and one hundred and thirty were used to test its performance.

In Fig. 21, the top two rows show examples of images of the two new classes, “teacup” and “glass bottle” that were added to the existing network. The bottom row shows how the original network structure (from the ImageNet test with 11 classes, left, also shown

[1] https://paperswithcode.com/sota/image-classification-on-cifar-10

in Fig. 19) grew to learn the two new classes which naturally resulted in a slightly larger network (right).

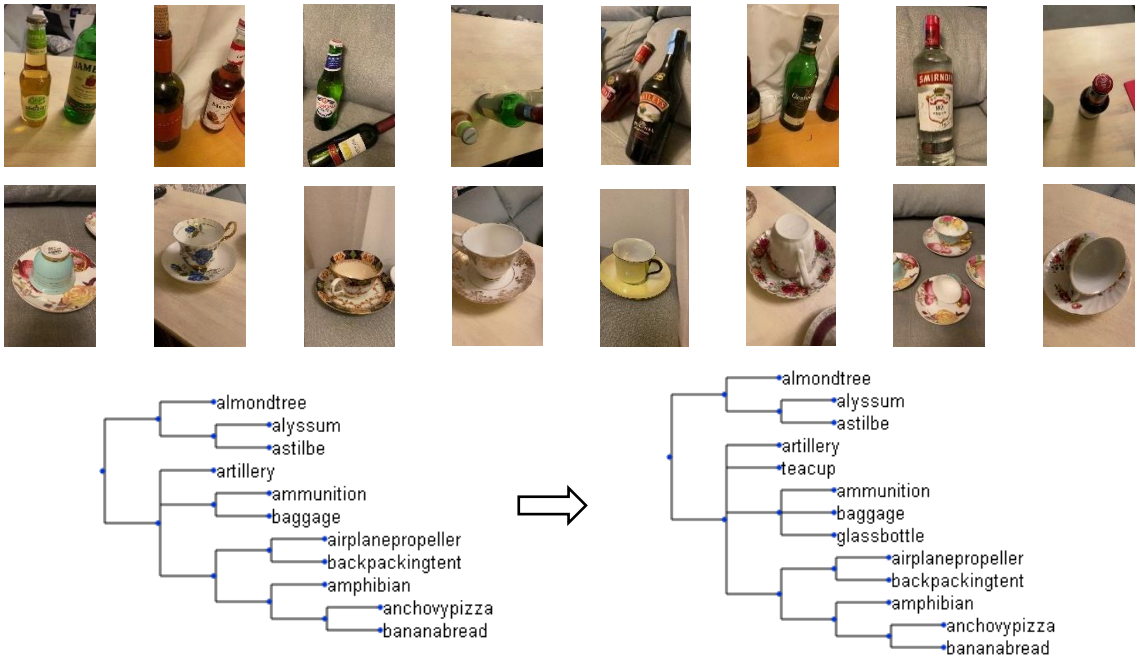


Figure 21: The top two rows are images of the two new classes that were added the network, those classes being “glass bottle” and “teacup” respectively. The Bottom row shows the transition between the network from the ImageNet test with 11 classes (left, from Fig. 19) and the same network after the addition of two new classes (right).

Table 9 shows how the two new classes affected the accuracy results for each of the 11 previously learned ImageNet classes. It also presents the new, overall, accuracy of the network (0.82). The final accuracies for “glass bottle” and “teacup” were 0.89 and 0.99 respectively. The final accuracies for all the classes learned by the network are presented in the last column (“13”). Out of the 13 classes, 2 (~15%) (marked in red) presented very poor results, 6 (~46%) presented results above 0.70 (marked in yellow), and 5 (~38%) presented accuracy results above 0.90 (marked in green).

4.5 Discussion

The obtained results from our tests were in line with what was expected based on what we had seen from intermediate tests during the development of the architecture.

Table 9: Accuracy over time/class - addition of 2 new classes obtained from video footage to the network previously created using 11 ImageNet classes shown in Tab. 3.

Classes \ *Step*	2 init	3	4	5	6	7	8	9	10	11	12	13
airplane propeller	1.00	1.000	1.00	1.00	1.00	1.00	1.00	0.98	0.98	0.94	0.93	0.93
almond tree	0.97	0.92	0.92	0.95	0.96	0.96	0.96	0.94	0.96	0.94	0.97	0.96
alyssum		0.79	0.79	0.79	0.80	0.80	0.32	0.32	0.32	0.32	0.32	0.32
ammunition			0.92	0.95	0.92	0.88	0.89	0.88	0.75	0.72	0.76	0.77
amphibian				0.96	0.94	0.94	0.95	0.96	0.98	0.97	0.96	0.96
anchovy pizza					0.98	0.98	0.98	0.99	0.99	0.89	0.89	0.89
artillery						0.85	0.85	0.88	0.91	0.91	0.87	0.87
astilbe							0.68	0.67	0.68	0.68	0.67	0.68
backpacking tent								0.91	0.91	0.83	0.83	0.84
baggage									0.56	0.80	0.87	0.87
banana bread										0.90	0.95	0.95
glass bottle											0.89	0.89
teacup												0.99
Overall accuracy	0.99	0.90	0.91	0.93	0.93	0.92	0.83	0.84	0.80	0.81	0.83	**0.82**

The first thing to note is that while the accuracies may not all be exceptional, the framework works. Classes are learned and, as they are added, the network is still able to identify old ones (although there is still a fair bit of degradation to the accuracy as more and more classes are added).

The two datasets we used each help us learn different things. The CIFAR10 tests were a good way to test the stability/reliability of the network, and we can see that, with the 15 separate tests, the final accuracy results are all very similar, even though the network structured itself quite differently each time. This is a sign that the network's performance is relatively stable. The ImageNet tests were a good way to see how successful the network can be with more realistic images, as they are of a far superior quality when compared with CIFAR10. The difference in image quality between the two datasets we presume is also why the accuracy results were far superior for the ImageNet tests.

The test with 25 ImageNet classes also revealed some very interesting results regarding the grouping of modules: there is a clear pattern to be noticed where objects that could be considered similar by humans were also considered to be similar by the network. For example, in the final state of the network, in Tab. 5, some interesting groupings were boxer & Shih-Tzu (dogs), blade & bobby pin (metallic objects), bell cote & elevator shaft (places), anchovy pizza & banana bread (food), almond tree & barley (plants) and also some classes like ammunition, blinker, artillery and airplane propeller are all mechanical-type objects and they aren't too far apart and look like, they might also group together if

more similar objects were to be added to the mix. Even in the CIFAR10 tests which only had 10 classes, it is possible to see that there was a tendency to group animals and vehicles separately. This natural grouping of similar objects is exactly the type of behaviour we were hoping for, because an increased amount of similarity-based grouping means a better chance that the sub classifiers will become specialists at distinguishing between similar objects.

It is also worth noting that some of the classes that had worse accuracies, namely artillery and airplane propeller, were quite close together in the network. We think this may be because some other similar classes (like blade, elevator shaft, ammunition, and space shuttle) have not yet been grouped with them and could be causing the network some confusion. We plan to do more tests of this kind, because we think it will help us decide on how to continue to improve the network.

Finally, the tests performed with objects filmed by us, showed that the network can learn from image data acquired in the “real world”, which was one of the objectives for the framework. The addition of the two new classes did not appear to have much of an effect on any of the previously learned ones in terms of accuracy, some of them decreased a small amount and some of them improved.

While none of these test results are as high as the current benchmarks for these datasets that keep getting ever closer to 100%, that was not our objective with this initial architecture. We once more emphasize that, our objective was to create a network that can learn classes one at a time and still be able to classify old ones, and we are very pleased with the test results so far, especially considering that there are so many points we can still improve upon.

5. Conclusions and Future Work

ABSTRACT

Continual Learning is starting to attract a lot of attention, and although it still has much room for improvement, when its main obstacles have been overcome, will be a huge player in the Artificial Intelligence field.

5.1 CONCLUSIONS

In this dissertation, we presented a proof of concept of an image classification framework, capable of learning new classes while maintaining knowledge of previous ones. The structure of our framework is based on a modular network of smaller artificial neural networks that get added to the network when new classes are learned. This way, other parts of the network are left untouched which allows them to retain their knowledge.

The idea of a constant addition of new smaller neural networks to the main network creates an initial concern for scalability, but the network is structured in such a way that different parts of the network are used for different tasks, which greatly reduces this problem. For example, in Sec. 3.4.5 we demonstrated how the network's structure reduces the scalability problem using an example (Fig. 12 and Tab. 1). In this example we demonstrated how a different percentage of the network would be used depending on the predicted class, where the results for that example ranged between 3 and 8 binary classifiers, out of a total of 14, equating to 21% and 57% respectively. This example could predict a total of 9 classes and, if we had decided to simply have one binary classifier per class, even our worst-case scenario (8 classifiers used) would still be 11% faster, and our best-case scenario (3 classifiers used) would be 67% faster. Naturally, with our network being so dynamic, there will be some cases where a few classes use more binary classifiers than if there was simply one for each class, but in the vast majority of cases they will use much less. And a network where all the binary classifiers are in parallel would always require all the binary classifiers to be retrained when new classes are added as opposed to only the essential ones, as is the case with our network.

The two most similar studies we found to ours were NBDT: Neural-Backed Decision Tree (Wan *et.al.*, 2020) and LR: Latent Replay for Real-Time Continual Learning (Pellegrini *et.al.*, 2020). Where NBDT is not a continual learning method but is partially similar because our architecture also resembles a tree-like structure and is based on ANN, but our nodes have two or more outputs whereas theirs always have two. Our nodes are dynamic and independent from each other, allowing us to add new classes to the network, where NBDT are trained once to learn a set of classes (as is the case with most ANN frameworks).

The other network, LR, is similar because it is, in fact, a continual learning method, they also make a division between low-level features and high-level features, and store extracted features. The main differences are that (i) their feature extraction method is not static (in some cases it is trained slowly), where ours is permanently fixed so that our binary classifiers always maintain validity. (ii) When they train new classes, they re-apply their stored samples from known classes. Therefore, their high-level feature extraction for old classes is not affected too badly by the slow training of their low-level feature extraction. We also re-use samples of known classes during training but only from specific classes for specific cases, depending on the location of the module in question (see Sec. 3.6). (iii) The classification part of their network is made up of one ANN of a fixed size, where ours is made up of multiple ANN with more being added as needed.

The modular approach we employ in our network is responsible for many of the qualities we have talked about (it allows us to add new components to the network, it allows for automatic training of only the necessary sections of the network, it allows us to make classifications with only a percentage of the network etc.) but there are also other advantages that we have not yet mentioned. The modular network also grants us the following possibilities:

- Removal of classes – for whatever reason, if we desired to remove a class this would not stop the network from working;
- Selective manual retraining of sections – the algorithm used for automatic retraining of binary classifiers including the balanced selection of data can also be forcibly applied whenever we want;
- Temporary removal/freezing of classes – with this architecture one could easily "freeze" sections of the network in cases where there was previous knowledge that some classes that did not need to be considered. E.g., if the network knows 10 classes but there is a situation where we are only interested in 3, then we could "freeze" the other 7 classes to boost classification speed and possibly precision. This concept requires further testing in order to provide accurate metrics;
- Identify an input as "unknown" – because our binary classifier's final layers use sigmoid activation functions which give us a percentage of certainty that the class is "true", it is therefore possible to apply a threshold value to all the classifiers so that we can establish if a presented input is an unknown class;

- Manual network structuring/re-structuring – the modular network also allows us to manually define the structure of the nodes and locations of the class endpoint modules and let the network do the rest. Although we believe our algorithms optimize this structure, if a particular use case required certain classes to be grouped in specific ways or if any other kind of manual alteration was required, this could be easily done, and the self-training binary classifiers would do the rest based on their positions in the network. For instance, one could quite easily substitute an endpoint module "dog" with a node containing "[Bulldog, Labrador, Chihuahua, Jack Russell]" or even manually force a distinction between large and small dogs using sub-nodes such as "[[Bulldog, Labrador], [Chihuahua, Jack Russell]]".
- Categorization of input before final result – if one were to apply informational labels to the nodes in the network (e.g., "animals", "vehicles", etc.) one would have access to this information before a final classification was made as the network first progresses through the nodes before reaching the endpoints. If sub-nodes were also labelled, it would be possible to receive a progression of information during the classification process, e.g., "animal" > "mammal" > "quadruped" > "dog".

To conclude, the following contributions were made in this dissertation:

- Proposal of a new model/architecture for Continual Learning;
- Development of a dynamic tree-like modular network with formulas that allow tasks to be achieved using only the necessary sub-sections;
- Application of multiple ANN-based binary classifiers as inter-comparable independent modules;
- Development of a Continual Learning model/architecture that allows for a variety of manual changes to be made such as class removal, specific section retraining and manual re-structuring.

Initial testing showed promising results, enough that we definitely consider it to be a viable approach to Continual Learning, but we are anxious to continue doing more tests to better evaluate some of the parameters that we plan to work on next.

The architecture was put together using many different base methods and technologies and our next step is to start researching alternatives for our existing components so that we can start improving all the parts of the architecture one by one.

5.2 FUTURE WORK

The way our architecture is structured with separate blocks opens up many possibilities for future work, because there are various pieces to the puzzle that can all be improved.

After analysing the test results and seeing that the concept is viable, we are excited to start improving upon the base structure we have laid down. Some of the things look forward to starting work on are the following:

- Trying other feature extraction methods other than ResNet50, while also applying different binary classifier structures to work with the different extracted feature dimensions – evaluating speed/quality compromises and also seeing which methods are best for different applications;
- Improve existing knowledge – this was one of our initial objectives and still is, we believe that with our easily configurable network that it shouldn't be too hard to use new data of existing classes to improve the network's accuracy for them;
- Making use of the false values of the binary classifiers – currently, we only use the true value during classification and training, but we believe we can develop some more "intelligent" algorithms that make better use of the data we have available;
- Improving the new node placement algorithm – currently, we use an average value of the results of all the classifiers when placing new nodes. We think it would be interesting to experiment with some other formulas and to try to optimize this process;
- Explore the training threshold value – this value is one of the main configurable parameters in our network. We want to do tests and study the effects of changing this value and possibly implement a dynamic threshold value that is calculated based on the state of the network;
- Clustering of unknown classes to be learned at a later point – we plan to make use of our network's ability to identify inputs as "unknown" and use this to set aside groups of images to later be clustered into classes and trained (and await labelling). This will be a big step in the applications of this architecture in autonomous agents;

- Knowledge sharing – not quite as direct as the other points, but we are very interested in exploring the possibility of multiple networks with this structure communicating with each other and improving each other's performance.

5.3 PUBLICATIONS

The work returned two publication, one already submitted and accepted in an international conference, the other one is in preparation to submit in a journal, for the last one more tests and validation are being prepared.

Turner, D., Cardoso P.J.S., Rodrigues J.M.F. (2021) ***Continual Learning for Object Classification***. Accepted In: Antona M., Stephanidis C. (eds) Universal Access in Human-Computer Interaction. Applications and Practice. HCII 2021.

Turner, D., Cardoso P.J.S., Rodrigues J.M.F. (2021) ***A Modular Dynamic Network for Continual Learning***. In preparation for Applied Sciences, Special Issue in: Artificial Intelligence Applications and Innovation.

References

Abiodun, O. I., Jantan, A., Omolara, A. E., Dada, K. V., Mohamed, N. A., & Arshad, H. (2018). State-of-the-art in artificial neural network applications: A survey. Heliyon, 4(11), e00938.

Aljundi, R. (2019). Continual Learning in Neural Networks. arXiv preprint arXiv:1910.02718.

Aljundi, R., Babiloni, F., Elhoseiny, M., Rohrbach, M., & Tuytelaars, T. (2018). Memory aware synapses: Learning what (not) to forget. In Proceedings of the European Conference on Computer Vision (ECCV) (pp. 139-154).

Aljundi, R., Rohrbach, M., & Tuytelaars, T. (2018). Selfless sequential learning. arXiv preprint arXiv:1806.05421.

Alpaydin, E. (2020). Introduction to machine learning. MIT press.

Chen, C. P., & Liu, Z. (2017). Broad learning system: An effective and efficient incremental learning system without the need for deep architecture. IEEE transactions on neural networks and learning systems, 29(1), 10-24.

Dangeti, P. (2017). Statistics for machine learning. Packt Publishing Ltd.

De Lange, M., Aljundi, R., Masana, M., Parisot, S., Jia, X., Leonardis, A., ... & Tuytelaars, T. (2019). Continual Learning: A comparative study on how to defy forgetting in classification tasks. arXiv preprint arXiv:1909.08383.

Deng, J., Dong, W., Socher, R., Li, L. J., Li, K., & Fei-Fei, L. (2009, June). Imagenet: A large-scale hierarchical image database. In *200*9 IEEE Conference on Computer Vision and Pattern Recognition (pp. 248-255). IEEE.

Ebrahimi, S., Meier, F., Calandra, R., Darrell, T., & Rohrbach, M. (2020). Adversarial Continual Learning. arXiv preprint arXiv:2003.09553.

Fang, W., Wang, C., Chen, X., Wan, W., Li, H., Zhu, S., ... & Hong, Y. (2019). Recognizing global reservoirs from Landsat 8 images: A deep learning approach. IEEE Journal of selected topics in applied earth observations and remote sensing, 12(9), 3168-3177.

French, R. M. (1999). Catastrophic forgetting in connectionist networks. Trends in Cognitive Sciences, 3(4), 128-135.

Giles, C. L., Kuhn, G. M., & Williams, R. J. (1994). Dynamic recurrent neural networks: Theory and applications. IEEE Transactions on Neural Networks, 5(2), 153-156.

Hall, J. S. (2007). Self-improving AI: An analysis. Minds and Machines, 17(3), 249-259.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

Hensman, P., & Masko, D. (2015). The impact of imbalanced training data for convolutional neural networks. Degree Project in Computer Science, KTH Royal Institute of Technology.S

Huang, Y., Cheng, Y., Bapna, A., Firat, O., Chen, D., Chen, M., ... & Wu, Y. (2019). Gpipe: Efficient training of giant neural networks using pipeline parallelism. In Advances in Neural Information Processing Systems (pp. 103-112).

Kolesnikov, A., Beyer, L., Zhai, X., Puigcerver, J., Yung, J., Gelly, S., & Houlsby, N. (2019). Big transfer (BiT): General visual representation learning. arXiv preprint arXiv:1912.11370.

Krizhevsky, A., & Hinton, G. (2009). Learning multiple layers of features from tiny images, Master Thesis U. Toronto.

Kwong, K. K., Belliveau, J. W., Chesler, D. A., Goldberg, I. E., Weisskoff, R. M., Poncelet, B. P., ... & Turner, R. (1992). Dynamic magnetic resonance imaging of human brain activity during primary sensory stimulation. Proceedings of the National Academy of Sciences, 89(12), 5675-5679.

Liu, W., Wen, Y., Yu, Z., & Yang, M. (2016, June). Large-margin softmax loss for convolutional neural networks. In *ICML* (Vol. 2, No. 3, p. 7).

Mallya, A., & Lazebnik, S. (2018). Packnet: Adding multiple tasks to a single network by iterative pruning. In Procs of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 7765-7773).

McCarthy, J. (1969). Programs with common sense. In Marvin, M. (Ed.), Semantic information processing (pp. 403–418). Cambridge: MIT.

McCarthy, J., Minsky, M. L., Rochester, N., & Shannon, C. E. (2006). A proposal for the Dartmouth summer research project on artificial intelligence, August 31, 1955. AI Magazine, 27(4), 12-12.

McClelland, J. L., McNaughton, B. L., & O'Reilly, R. C. (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. Psychological Review, 102(3), 419.

Nielsen, M. (2015). Neural Networks and Deep Learning. Determination Press. (online on: http://neuralnetworksanddeeplearning.com/, last accessed 2020/09/20))

Nilsson, N. J. (2014). Principles of artificial intelligence. Morgan Kaufmann.

O'Shea, K., & Nash, R. (2015). An introduction to convolutional neural networks. arXiv preprint arXiv:1511.08458.

Pan, S. J., & Yang, Q. (2009). A survey on transfer learning. IEEE Transactions on Knowledge and Data Engineering, 22(10), 1345-1359.

Parisi, G. I., Kemker, R., Part, J. L., Kanan, C., & Wermter, S. (2019). Continual lifelong learning with neural networks: A review. Neural Networks.

Parker, J. A., Kenyon, R. V., & Troxel, D. E. (1983). Comparison of interpolating methods for image resampling. IEEE Transactions on medical imaging, *2*(1), 31-39.

Pellegrini, L., Graffieti, G., Lomonaco, V., & Maltoni, D. (2020). Latent replay for real-time continual learning. arXiv preprint arXiv:1912.01100.

Ratcliff, R. (1990). Connectionist models of recognition memory: constraints imposed by learning and forgetting functions. Psychological Review, 97(2), 285.

Rebuffi, S. A., Kolesnikov, A., Sperl, G., & Lampert, C. H. (2017). ICARL: Incremental classifier and representation learning. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 2001-2010).

Requeima, J., Gordon, J., Bronskill, J., Nowozin, S., & Turner, R. E. (2019). Fast and flexible multi-task classification using conditional neural adaptive processes. In Advances in Neural Information Processing Systems (pp. 7957-7968).

Ridnik, T., Lawen, H., Noy, A., & Friedman, I. (2020). TResNet: High Performance GPU-Dedicated Architecture. arXiv preprint arXiv:2003.13630.

Ring, M. B. (1998). CHILD: A first step towards continual learning. In Learning to learn (pp. 261-292). Springer, Boston, MA.

Sharma, N., Jain, V., & Mishra, A. (2018). An analysis of convolutional neural networks for image classification. Procedia computer science, 132, 377-384.

She, Q., Feng, F., Hao, X., Yang, Q., Lan, C., Lomonaco, V., ... & Qiao, F. (2019). OpenLORIS-Object: A Dataset and Benchmark towards Lifelong Object Recognition. arXiv preprint arXiv:1911.06487.

Shlens, J., Field, G. D., Gauthier, J. L., Grivich, M. I., Petrusca, D., Sher, A., ... & Chichilnisky, E. J. (2006). The structure of multi-neuron firing patterns in primate retina. Journal of Neuroscience, 26(32), 8254-8266.

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 2818-2826).

Tan, M., & Le, Q. V. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks. arXiv preprint arXiv:1905.11946.

Turing, A. M. (1950). Computing machinery and intelligence. Mind, 59, 433–460.

Wan, A., Dunlap, L., Ho, D., Yin, J., Lee, S., Jin, H., ... & Gonzalez, J. E. (2020). NBDT: Neural-Backed Decision Trees. arXiv preprint arXiv:2004.00221.

Wang, Z. J., Turko, R., Shaikh, O., Park, H., Das, N., Hohman, F., ... & Chau, D. H. (2020). CNN Explainer: Learning Convolutional Neural Networks with Interactive Visualization. arXiv preprint arXiv:2004.15004.

Wu, J. (2017). Introduction to convolutional neural networks. National Key Lab for Novel Software Technology. Nanjing University. China, 5, 23.