FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Interpreting Hierarchichal Data Features - Towards Explainable AI

Luís Ramos Pinto de Figueiredo

MASTER'S DISSERTATION

U.PORTO

FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Daniel Castro Silva

Second Supervisor: Fábio Silva

July 25, 2018

Interpreting Hierarchichal Data Features - Towards Explainable AI

Luís Ramos Pinto de Figueiredo

Mestrado Integrado em Engenharia Informática e Computação

July 25, 2018

Abstract

As the field of artificial intelligence expands, increasingly complex models are constructed to solve increasingly complex tasks. One of the issues that rises from this development is the need for interpretability of the inner workings of the models. Correctly understanding these models leads the end user into better decision making and the developer into building better performing models, as key points of failure and success are identified.

In particular, neural networks (*NNs*) have shown great performance in fields such as speech and image recognition, surpassing the remaining models in key benchmarking challenges such as the *ImageNet Visual Recognition Challenge* [1] and the *COCO - Object Recognition Challenge* [2]. These models, however, suffer from being extremely opaque and are frequently referred to as *black-box* models, due to the inherent difficulty in their interpretation.

In this study, a step towards interpreting the process in which a convolutional neural network (*CNN*) learns from an image input is taken. By adding an intermediate layer trained in coarse classification and feeding its output into a fine classification layer, this new model can be compared to a baseline model. In constructing a model in this fashion, the study analyses the hierarchical properties of image data and evaluates the relevance of this information inside the model in comparison to the remaining extracted features. This allows more insight regarding the impact and usability of hierarchical image properties, providing more opportunities to understand the relevance of increasing model depth and feature transferability. The obtained results, even if only scratching the surface of the research potential of the field, demonstrate the potential use for hierarchical data in model interpretation, sparking discussion topics for future research.

Keywords: **Machine Learning, Explainable AI, Interpretable AI, Image Classification**

[1]Link to Imagenet - http://www.image-net.org/challenges/LSVRC/
[2]Link to the COCO dataset - http://cocodataset.org/#home

Resumo

À medida que o campo de inteligência artificial evolui, modelos cada vez mais complexos são construídos por forma a resolver tarefas igualmente mais complexas. Um dos desafios que surge deste desenvolvimento é a necessidade de interpretação dos mecanismos internos destes modelos. A compreensão destes mecanismos permite que programador identifique mais facilmente como construir um modelo mais poderoso assim como que o utilizador tome decisões mais relevantes, uma vez que os principais pontos de fracasso e sucesso são identificados.

Em particular, redes neuronais (NNs) têm mostrado um elevado desempenho nas áreas de reconhecimento de imagem e de fala, ultrapassando os restantes modelos em importantes desafios de comparação como o *ImageNet Visual Recognition Challenge* [3] e o *COCO - Object Recognition Challenge* [4]. Estes modelos, contudo, sofrem de serem extremamente opacos e são frequentemente designados de modelos *black-box* - modelos caixa negra - devido à inerente dificuldade de interpretação.

Neste estudo é dado um passo na direção de interpretar o processo de aprendizagem que uma rede convolucional (CNN) treinada para reconhecimento de imagens efetua. Ao adicionar uma camada intermédia treinada em classificação de *superclasses* e propagar o seu resultado para a camada de classificação final, este novo modelo pode ser comparado ao modelo de base. Ao construir um modelo desta forma, O estudo analisa as propriedades hierárquicas de dados de imagens e avalia a relevância desta informação dentro do modelo em comparação com a restante informação presente no modelo. Isto permite a recolha de mais informação relativamente ao impacto e utilidade das propriedades hierárquicas de imagens, criando mais oportunidades de perceber a relevância de aumentar a profundidade dos modelos atuais e da transferibilidade dos pesos do modelo. Os resultados obtidos, muito embora alcancem apenas um nível superficial do possível nesta área, demonstram a utilidade de dados hierárquicos na interpretação destes modelos e dão origem a mais tópicos de discussão para investigações futuras.

Palavras Chave: Aprendizagem Computacional, IA Explicável, IA Interpretável, Classificação de Imagem

[3]Link to Imagenet - http://www.image-net.org/challenges/LSVRC/
[4]Link to the COCO dataset - http://cocodataset.org/#home

Acknowledgements

To my supervisor, Professor Daniel Castro Silva, to my second supervisor and *AMT* manager, Fábio Silva. Thank you for all the support and education you provided me, all of which were critical in the elaboration of this project.

Luís Ramos Pinto de Figueiredo

"Live as if you were to die tomorrow. Learn as if you were to live forever"

Mahatma Gandhi

Contents

List of Figures

LIST OF FIGURES

List of Tables

Abbreviations

AI	Artificial Intelligence
aLIME	Anchor Local Interpretable Model-agnostic Explanations
AMT	A Matter of Trust (company name)
ANN	Artificial Neural Network
CNN	Convolutional Neural Network
DAE	Denoising Auto-Encoder
DARPA	US Defense Advanced Research Projects Agency
DDSM	Digital Databse for Screening Mammography
DL	Deep Learning
DNN	Deep Neural Network
ELU	Exponential Linear Unit
FEUP	Faculty of Engineering of the University of Porto
LIME	Local Interpretable Model-agnostic Explanations
LSTM	Long Short Term Memory network
ML	Machine Learning
MLP	Multi-Layered Perceptron
NLP	Natural Language Processing
NN	Neural Network
R-CNN	Regional-Convolutional Neural Network
RAM	Random Access Memory
ReLU	Rectified Linear Unit
RBM	Bernoulli Restricted Boltzmann Machine
ROI	Region of Interest
SGD	Stochastic Gradient Descent
UP	University of Porto
XAI	Explainable Artificial Intelligence

Chapter 1

Introduction

The work presented in this dissertation came to be due to the endeavours of *AMT - consulting* in expanding their market into the machine learning field. *AMT - consulting* is a Portuguese software consultancy company that is looking to build well-performing machine learning solutions, for which the better understanding of the developed systems could be the differentiating factor. One of the fields where machine learning is widely used is computer vision, in order to automate processes such as image classification and object detection, for which deep learning has shown exceptional potential (discussed in chapter 3). While these models are becoming increasingly more powerful, our ability to generate explanations of their decisions is still lacking, which makes target markets such as critical and medical systems unavailable. This work goes towards making the deployment of deep learning models in these fields a reality.

1.1 Goals and Motivation

Research in machine learning, deep learning and interpretable artificial intelligence (AI) is currently highly motivated behind the success it has shown in multiple AI challenges, the success in deployed solutions and in the ability to accurately extract information from data. In particular, deep learning has shown great results, trumping more traditional models and revolutionising the state of the art since the start of the current century. However, their applicability in fields like medicine, law and critical systems is hindered due to the lack of interpretability and explainability these models offer. This dissertation is an attempt at interpreting the inner workings of these models, in particular convolutional neural networks (*CNNs*), to increase our understanding of the information flow inside the model. Specifically, this project looks at the information flow and performance of a model on hierarchical image data.

From a business perspective, one of *AMT*'s goals is to combine deep learning approaches with natural language processing (*NLP*) techniques to create software that can automate tasks such as medical report generation from medical imagery data. It is currently possible to perform such tasks with traditional models like *Support Vector Machines* and *Decision Tree Classifiers*, because they offer significant explainability, but they fail to reach the potential of the new deep learning solutions available. Due to their black-box nature, these new solutions are often not allowed to be deployed in the medical field, as responsibility is a major concern. Being able to construct a solution for this problem that integrates deep learning would elevate AMT as experts in the field, having produced a unique solution, which could potentially help in saving the lives of patients by reducing the misdiagnosis rate for some disease types and quickening the diagnosis process.

1.2 Overview

We establish a testing environment to analyse and interpret the effect of feeding superclass predictions back into the classification model. This is achieved by constructing a baseline model, adapted from a known architecture, and proceeding to construct a new model that contains an additional prediction layer that is fed back into the model, forcing the model to have a degree of trained hierarchical features. The study analyses the effects of this approach on multiple levels such as training speed and testing performance, overfitting and layer relevance.

This document reports the process taken to create this environment, the impact of the analyses produced with it and suggests approaches for improvement and further research in the field.

1.3 Document Structure

Beyond this introduction, this dissertation contains an additional 6 chapters.

Initially, with the goal of familiarising the reader with necessary concepts of the machine learning field, some introductory background information is presented in a beginner friendly way in chapter 2. This chapter is meant for the non-expert reader and can be skipped if the reader is familiar with the basic concepts and techniques used in deep learning.

Following, the current state of the art and related works are presented and discussed in chapter 3, giving insight on the hurdles of research in the field. This chapter is divided into two parts: first, a traceback of the current deep learning architectures is built, showing the evolution of the architectures over the last decade and presenting the most relevant breakthroughs and their downsides for model interpretability. Following this segment, the most relevant attempts at explaining the behaviour of these models are discussed, as this is the focus of the project.

Afterwards, the project planning and the proposed solution are discussed in chapter 4. This chapter goes over the engineering problems that need to be solved during the project's development and explains the reasoning behind the changes that are introduced to the initial plan.

In chapter 5, a detailed description of the project is available. This chapter explains the environment in which the study is conducted, going over the model architectures and giving fundament for the design choices. In this chapter, the expected results are debated and the implications that such results would bring are presented.

Chapter 6 presents the obtained results. The reasons that result in the obtained results are debated and potential options to circumvent any negative results and lack of results are given. Relevant mistakes in the experiments that lead to any lack of results are also highlighted. For results that confirm the hypotheses that were discussed in chapter 5, potential future research opportunities and methodologies are suggested and motivated upon, identifying why these are a necessity in the field and the contributions that can be obtained from conducting them.

Finally, a review of the study and some further insight into future research can be read in chapter 7, where the most relevant hurdles and lessons learned throughout the development of the project are mentioned.

Chapter 2

Background

In this chapter, important background concepts for the understanding of this dissertation will be explained. Initially, broader machine learning terms will be introduced, followed by concepts and definitions specifically for the subset of machine learning challenges faced during this study, namely deep convolutional models.

2.1 Machine Learning Models

At its core, machine learning is a field of artificial intelligence (AI) wherein an algorithm is constructed to make some form of prediction [Nas07]. The difference between traditional AI and machine learning is that the algorithms are *trained* purely on data, instead of human instructions, as if *learning* by itself. The term *Machine Learning* was coined in 1959 by Arthur Lee Samuel [Sam59]. *Training* refers to the process of calibrating the model to fit a specific problem. The process of taking a set of unseen inputs and generating an output based on the data a model has been trained with is called *generalisation*, as the model is generalising the tendencies of the data it has learned onto new data. The output itself can either be a label for the given input, in which case the model is solving a *classification* problem, or a value for the given input, in which case the model is solving a *regression* problem. Technically, classification is a form of regression, in which the model is outputting an integer value, but these are usually distinguished from one another.

In either case, the model's goal is to approximate the real value for any given input as closely as possible. This value is called the *ground truth*.

In the case where the ground truth is known, the learning process is considered *supervised* [Nas07]. This additional information allows the model's performance to be calculated based on some metric to compare the difference between the ground truth and the model's prediction. This is the *error function* of the model. In the case where the ground truth is unknown, learning is considered *unsupervised*. This is often the case for problems where acquiring labelled data is prohibitively expensive. For these problems, one of the uses that machine learning models serve is to perform *clustering* of data. That is, to identify a decision boundary between data points such that highly-related data points fall under the same region. They're also used to perform dimensionality reduction on complex problems. More on this in section 2.2.

2.2 Feature Engineering

Feature engineering refers to the process of extracting a set of representative features from a dataset, to improve the generalization performance of a model. The majority of the data used in machine learning problems is, in its raw format, not representative of the patterns that are interesting to analyse. To better illustrate this, a fabricated example follows.

In the case of trying to predict the number of years of enrolment of a student in a faculty of engineering, the dataset at hand is that of the high school grades from every student who enrolled in that institution in the last years. However, these inputs will perform poorly on the generalization of the model, as students have a very high variability on their high-school grades depending on the subjects they are interested in. There is also a high bias towards favouring students who excel at a specific field that is particularly relevant for the course they're enrolling in. An example of this is that mathematical fields are more relevant than non-mathematical fields for the course of Software Engineering.

As such, selecting the most representative inputs (for example, the grades in the mathematics class) and crafting features such as the mean, median and standard deviation of relevant features from the dataset are considered fundamental steps in the process of a machine learner [Nas07].

This process is called feature extraction. After performing feature extraction, a step called feature selection is typically executed, as a means of dimensionality reduction. While the previous example is very simplistic, the result of a feature extraction process can result in thousands of extracted features or even millions, which often contain high correlation and irrelevant features, which in turn reduce the performance of the model and increase its execution time and memory requirements. As such, feature selection is the process of selecting the most representative subset of the extracted features that performs within the constraints of the problem at hand.

The combination of these two steps is called feature engineering and can be an iterative process. It is to note that the two steps are sometimes confused in the literature available; feature extraction is sometimes incorrectly called feature selection and vice-versa.

2.3 Deep Learning

Whereas in its counterpart, *shallow*-learning, input and output are usually directly related, in *deep*-learning, the relationship between input and output of the algorithm is determined by processing the input through multiple, intermediate processing layers [GSC16]. The result of this is that the features of the input set are iteratively learned by the model itself, leading to automated feature extraction and selection.

The use of deep learning techniques has risen to popularity due to its ability to model complex problems, where hand-crafted features have shown to be too simplistic [KSGE12].

In deep learning, the so called *hidden* layers are the intermediate layers that separate the input layer and the output layer. The more and larger the hidden layers of a network are, the more complex the model is, increasing the complexity of the features it can capture. This comes at the cost of requiring more data and computing power to learn them. This trade-off is the other reason why these models have only risen to popularity as of relatively recently, as the available datasets have been growing immensely with the rise of fields like Big Data.

2.3.1 Activation Function

In the context of deep learning, an activation function is a function that takes as input the output of a layer of a neural network and applies a transformation to it. The purpose of an activation function is to map the output, which can take any value between negative infinity and positive infinity, to a more usable value. In the hidden layers, this function is used to introduce non-linearity through activations like *ReLU* or *ELU*. An example of this non-linear activation can be found in Fig. 2.1.

In classification problems, the output layer is typically activated by functions like *Softmax*, that squash the output to be between 0 and 1, representing the probability distribution between the classes.

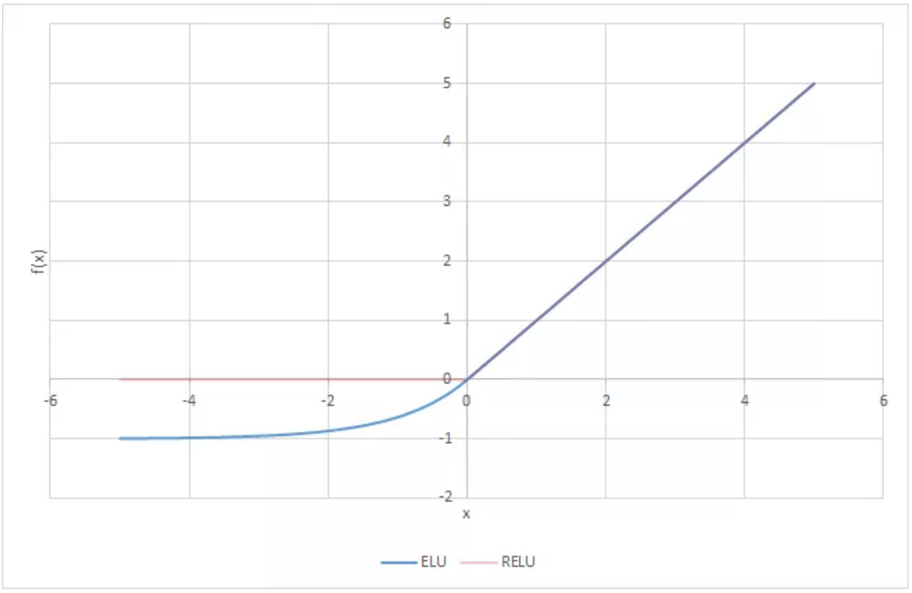


Figure 2.1: Elu and ReLU
Extracted from [Ser18]

2.3.2 Overfitting & Regularisation

Overfitting is a problem that emerges in machine learning when a model is overly trained, becoming too good at predicting the training data, thus failing at predicting unseen test data. This happens when the weight values of the weights matrix grow too large, forcing the model's decision boundary to bend too much in the direction of the training data.

Overfitting usually occurs when the model is fed the same information multiple times, since the weights are pushed in the same direction multiple times, thus overly fitting the curve. A clearer visualisation of a model's decision boundary in an *underfitting*, desired and overfitting scenario is shown in Fig. 2.2.

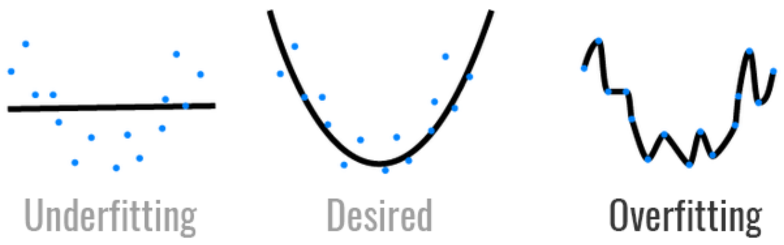


Figure 2.2: Overfitting Visualisation
Extracted from [Des]

Techniques to combat overfitting are called regularisation techniques. Regularisation techniques feed additional information to the model, which it can use to not approach a perfect solution too quickly.

One method that can be used to achieve regularisation is performing data augmentation. Data augmentation involves doing some kind of preprocessing on the dataset, such that new data samples are generated while keeping the associated label. In the case of image data, it can be used so that every time the model is fed the same image, that image has been slightly altered, thus changing the behaviour of the model on that image. Some of the operations that are typically used on image data include rotation by a fixed or randomized degree, shifting along an axis, cropping, flipping along an axis or even obfuscation of an area of an image. Not all operations can be used to all datasets though, so careful pre-analysis of what information of the image is being manipulated should be done.

Deep learning problems are particularly prone to overfitting, since the amount of data required to train a model is significantly higher. Because of this requirement, the dataset is often fed multiple times to the model, in so called training epochs. In order to mitigate the overfitting that occurs from this, additional regularisation techniques need to be employed.

Motivated on the process of sexual reproduction and its efficiency in optimisation ([LPPF10]), *Dropout* [SHK$^+$14] is a technique that forces a network's layer to randomly *drop* a neuron's output with a given probability during training, setting it to 0. The randomness of the dropout means that the neurons that get dropped change for every sample being fed into the model, thus making every sample training process unique. This forces the model to rely on multiple neuron's activations instead of overfitting on the prediction of a single neuron. The higher the dropout rate, the more robust to overfitting the model becomes, at the cost of slower learning, as the descriminative neuron activations will be inactive during training more often. As with all regularization techniques, dropout is only activated during the training phase of the model. To evaluate unseen data, the model utilizes all of its connections, which often leads to significantly better testing performances than training performances, which can be counter-intuitive to those who are unaware of this property.

For further reading, an expertly written explanation of the motivation and reasoning for this technique is available in section 2 of [SHK+14].

A more recent form of regularisation applicable to deep models is Batch Normalisation [IS15]. This technique avoids overfitting through normalising the activations of a layer's neurons over the training batch. This means that overly-weighted connections will be shifted towards the mean of the batch, forcefully preventing the model from relying on high weights to form its predictions. Additionally, it also serves as a form of regularisation because the batches generated in each epoch should, for most problems, be randomized. This means that the batches being fed to the model are typically not the same and thus have different means, which makes each batch unique, preventing the model from overly learning any particular weight.

Overfitting can also be prevented through *early stopping*. As the name suggests, early stopping involves stopping the training process before the model overfits. If the model has seen no improvement over a sufficiently large amount of training, then the training process is ended. In this fashion, the training is halted before overfitting can occur. One downside of this technique is that potential training may be lost if the stop condition triggers too soon, thus not allowing the model to reach peak performance. On the other hand, defining a stop condition that triggers too late won't prevent the model from overfitting. It is necessary to fine-tune the balance between the two scenarios. Figure 2.3 shows an example of a training- and validation error curve during a model's training process. The ideal point of early stopping is, intuitively, when the two curves start diverging.

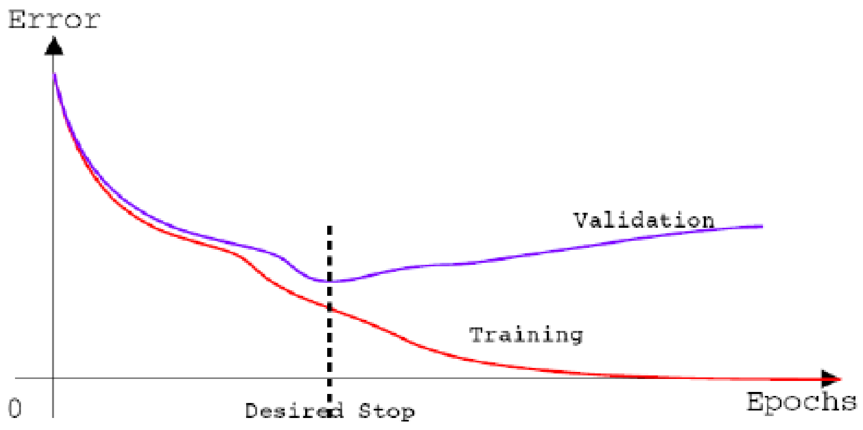


Figure 2.3: Early Stopping Visualisation
Extracted from [AGF07]

2.3.3 Convolutional Neural Networks

One of the problems with high-dimensional data, such as images, is that the processing and memory requirements for most models become too high. In the case of a neural network, the number of connections between neurons grows immensely, leading to memory and computational problems. Convolutional Neural Networks (*CNN*) are a subset of neural networks that are specifically built to handle image data and even found use for text and speech recognition. The core principle is that the input layers are processed in convolution layers that extract the features of the input through local *filters*. These filters, alternatively called kernels, take advantage of the spatial properties of image data: only pixels near each other are related. As such, instead of connecting each neuron to every neuron in the next layer, the information is passed in so called *convolutions*, which iteratively convolve around the image in a window of a fixed size. The application of these small-sized kernels, even as small as 3x3 pixel windows, results in an extracted feature map that can detect simple features like edges and shapes in the earlier hidden layers [KSGE12]. Typical CNN architectures such as the one shown in Fig. 2.4 have a pooling layer following a few convolutional layers. The purpose of these layers is two-fold: by pooling the image into a smaller one, the number of dimensions is significantly reduced, thus saving computational time. Secondly, this pooling operation allows the following convolutions to inspect information over a larger area of the image, thus allowing more complex features to be interpreted.

A beginner level tutorial of the CNN model can be found in [Pet17]. In his post, Veličković introduces the mathematics and motivation of convolutional and pooling layers through a simple example model, built with *Keras* and applied on the *MNIST* dataset. The effects of the convolutions are also shown on example images, to provide a better understanding.

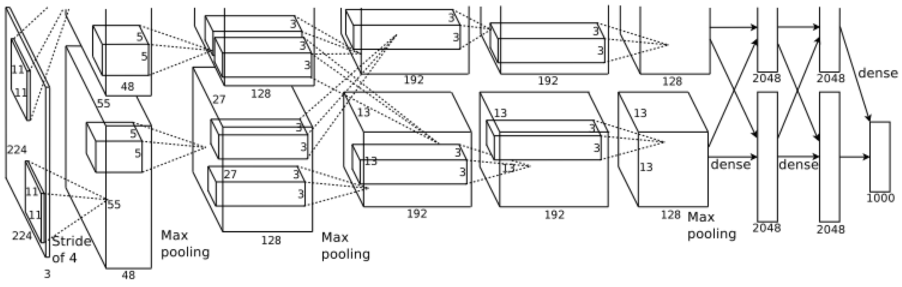


Figure 2.4: AlexNet Architecture
Extracted from [KSGE12]

2.4 Model Interpretability - Black Boxes and White Boxes

The models developed to solve machine learning problems can be loosely categorised into two types: black-box models and white-box models. White-box models are models that, given an input and respective output, can be inspected to find out how the output was produced. For lack of a better word, they can be considered "interpretable" by a human. This is a very desirable property, because it allows humans to get insight on why the model is producing its outputs, and helps us better understand why and when the model fails and why and when the model's prediction is correct. This is never a downside and is a necessary property in multiple fields, like Medicine, Law and most other high-risk fields, where responsibility is a concern.

In contrast, black-box models are models that cannot be directly understood through inspection. Deep learning models such as neural networks fall under this category [BCR97]. In fact, they are particularly difficult to inspect. The reason for this is that the output of the model is generated through a process that involves too many variables for human understanding (the network used in this study has 13 million parameters and is relatively small). However, deep learning models are significantly outperforming traditional models like decision tree classifiers due to the rapid growth of computing power and memory availability (see chapter 3), creating a larger gap between our understanding of the models being deployed. This study is one of many that try to shorten this gap.

Chapter 3

State of the Art

In this chapter, the current state of the art in convolutional neural network (CNN) solutions and their interpretations, as well as related works in the field are discussed. A look is taken at different attempts to uncover the inner workings of CNNs as well as alternative deep models. To conclude, a brief look into the problems that yet need to be solved is taken, as a way to contextualize the work presented in the study.

3.1 The Convolutional Neural Network

Regarding the choice of model for image classification, the breakthrough that followed the introduction of *AlexNet* [KSGE12] serves to establish the Convolutional Neural Network (CNN) model as the best performing base model in the field up to date. By massively outperforming every other existing model at the time it is introduced in 2012, achieving a top-5 error rate of 17%, it rapidly gains popularity and creates a new path for the image recognition research challenge, as the next best competitor model can only achieve 26.8%. The concept had already been introduced in 1998 by Yann Lecun in his paper [Lec98]. However, Lecun's application of the concept was in categorizing documents, not images. Additionally, the technology available at the time was not sufficient to adequately train a model as large as a CNN, so the concept is abandoned until *AlexNet* emerges. The architecture introduced in [KSGE12] is shown in 2.4.

Following AlexNet in 2012, the sector experiences a growth boom with major improvements every year, based on the same CNN concept. At the same competition in 2013, the error rate dropped to an astonishing 11.2% with the introduction of ZFNet, which was accompanied by the release of an informational paper by the authors in 2014, [ZF14], where a more intuitive and in-depth explanation of the CNN architecture is given.

The following year, 2014, another major improvement on the state of the art for the competition emerges with VGG NET [SZ14], reaching an error rate of only 7.2%, motivated on the construction of simple but very deep architectures. This publication serves to show that the optimisation of hyperparameters that affect the architecture of the network, such as kernel and padding size, number of convolutional layers per pooling layer, among others, can be overcome through increased depth. It cements the need for depth in order to be able to capture hierarchical features.

In the years to follow, tech giants *Google* and *Microsoft* also step in the competition, with the introduction of GoogLeNet and ResNet, respectively. GoogLeNet [SLJ$^+$14] entered the competition in 2014 as well, reaching 6.7% error rate, beating VGG by a small margin, which Google would come to improve to a mere 3.08% in 2016. This network, however, is infinitely more complex than VGG, consisting of 22 convolutional layers, some of which are running in parallel. Due to its sheer complexity, the model is impractical for researchers who don't have the resources available that a company such as Google has. Importantly though, GoogLeNet takes advantage of a recent methodology introduced at the time called Network In Network [LCY13], which uses 1x1 convolutions as a form of dimensionality reduction while preserving the complex feature maps extracted by the convolutional layers. This allows the model to be extremely efficient for its size. The importance of the parallelism introduced by GoogLeNet is its applicability to distributed computing. By showing that non-sequential stacking of layers can even outperform stacked architectures, the ability to rely on distributed computing is assured.

As mentioned, Microsoft's ResNet [WZL17] entered the competition in 2015, winning it with a performance of 3.6%, halving the previous record and outperforming human error for the first time. This network, which also beat the record of number of layers by consisting of 152 layers, continues the VGG legacy, showing that computing power can solve everything. The network, trained on an 8 GPU machine, still takes over 2 weeks to reach the performance showed at the competition. The reason why this network is relevant to the state of the art, however, is that it introduces a new system for preventing overfitting, as is clearly necessary for architectures as deep as this. This architecture adds a residue to the input, which is the result of the output of the convolutional + pooling layers. This forms a residual block, a new architecture that allows for information to be forwarded along the layers, rather than completely transformed at every layer. A visual representation of the process is shown in Fig. 3.1 for clarity.

Finally, the Regional-CNN [GDDM14], commonly abbreviated R-CNN, is an important contribution to the field, as it combines image segmentation technique with CNNs to significantly improve object detection in computer vision. Being the predecessor of multiple models that optimized its computational efficiency (Fast R-CNN in [Gir15] and even Faster R-CNN in [RHGS17]), R-CNN introduces the learning prowess of CNNs in object detection by initially applying image segmentation algorithms such as *Selective Search*, which segment the image in regions of interest, and then applies a CNN on those regions to determine the object that is present. The study published in 2017 by the *Facebook* research group [HGDG17] introduces a technique to predict a pixel-level mask for object detection over the image in parallel to the bounding-box algorithm of Faster R-CNN before feeding it to a CNN, outperforming every other single-model entry in the *COCO challenge suite*, which is the benchmarking challenge suite for object detection [1].

[1]Link to the COCO object detection challenge 2018 - http://cocodataset.org/#detection-2018

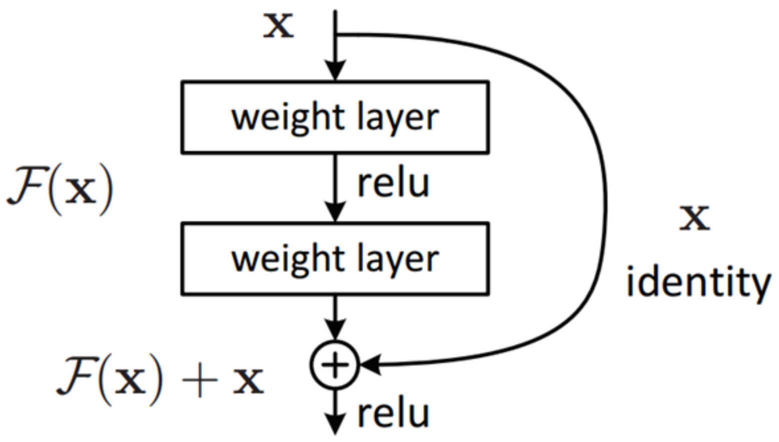


Figure 3.1: Residual Block Example
Extracted from [Des16]

3.2 Interpretability

Model interpretability, also refered to as explainability, is a property that is essential to a variety of fields where AI can and is currently applied, as explained in chapter 2. When a data scientist is capable of understanding how the model operates under different conditions, improving the model becomes significantly easier. However, our understanding of the models we're constructing has not been able to evolve as quickly as the models themselves, most notably so with the widespread use of neural networks and deep learning in the current decade.

Research in the field dates back to as early as 2003 [GDL03], but as neural networks only began wide-spread use later, most research started by the end of the decade. Visualisation approaches fall under two categories: *decompositions*, where a model is decomposed into a more interpretable form (e.g. a *Taylor*-series), and sensitivity analyses, where the sensitivity of the model to changes to the input is tested.

An attempt at visualisation is presented in [EBCV09]. In their work, Erhan et. al. look at the activation maps of the network for each of the 10 classes in the MNIST dataset, which is one of the modern datasets that was already available at the time. With this procedure, one can generate images that represent the various shapes that activate the network at each layer. However, while this visualisation technique is useful for the classical artificial neural network, the multi-layered perceptron (MLP), it can't be applied to more complex models such as the ones debated previously due to the filtering and pooling operations, which convert the original image into uninterpretable shapes.

While the concept of *decompositions* to visualise the behaviour of deep architectures is introduced back in 2006 in [PES$^+$06], decomposing a convolutional network is significantly more complex due to the numerous pooling operations.

Significant research into methods to uncover the inner workings of deep convolutional models begins in 2013, shortly after the publication of AlexNet. The approach suggested to interpret the behaviour of a deep model is trying to visualise the feature and activation maps of the model. The paper published in 2013 by Zeiler and Fergus [ZF14] implements the deconvolution (the inverse of the convolution) on the backwards pass over a trained network's output, thus reconstructing the original image at each layer, in the so called *DeConvNet*. This creates heatmaps, as shown in Fig. 3.2, which help in understanding which pixels are resulting in high activations by the network.

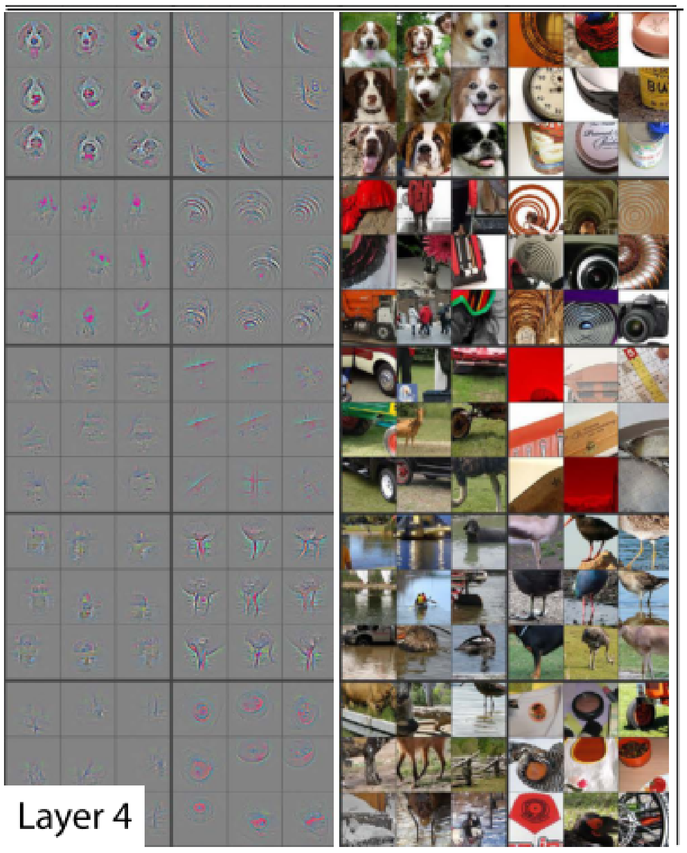


Figure 3.2: Deconvolution Result
Extracted from [ZF14]

Shortly after the DeConvNet publication, a new publication ([SVZ13]) reveals a method to generate an image representing the class interest of the network across all training samples. The results generated by this procedure show evidence that the features learned by a convolutional network are typically not the same as those a human would identify as representative, as can be seen in Fig. 3.3. Another source that explains how to produce these results and shares the code to do so can be found in Chollet's blog post about keras, [Cho17]. This insight is a reminder that the ability to accurately classify images does not imply that a model has a deep understanding on the information it is handling and that more insight is necessary to apply these models in the fields mentioned in chapter 1

2016 sees a surge in published research works in the field, a part of which are due to the funding provided by the *US Defense Advanced Research Projects Agency (DARPA)*. Under the name of

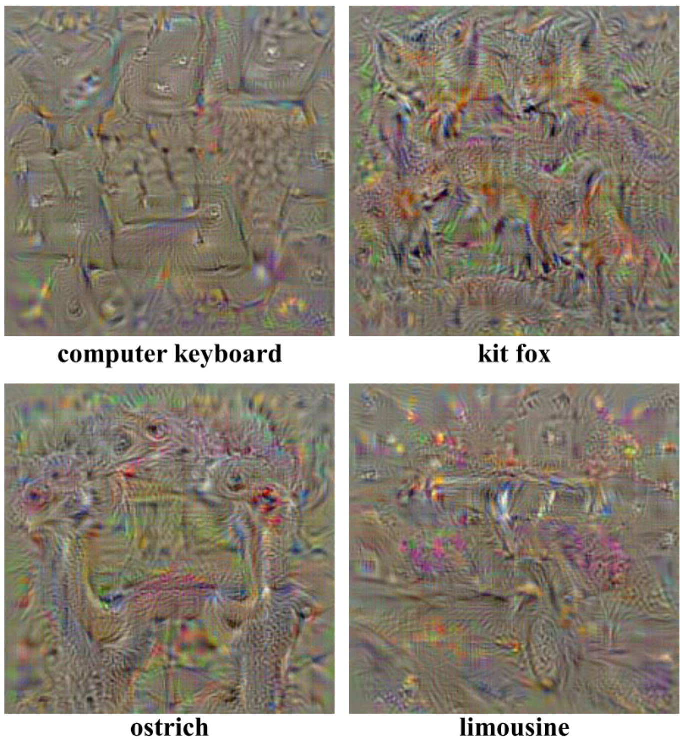


Figure 3.3: Class Interest
Extracted from [SVZ13]

Explainable AI (XAI)[2], this funding programme, started in 2015, aims to create more interpretable models in hopes of applying them to previously restricted fields - in particular, military operations.

The first result from this project is the publication of LIME (Local Interpretable Model-agnostic Explanations) [RSG16b]. This innovative work uses image segmentation to segment the input image of any classifier into regions of interest and then intelligently obfuscates a selection of those regions before feeding them into the classifier. In this method, the system can identify which regions cause the biggest loss in performance when obfuscated, which implies they are of high relevance to the activation of the classifier. Because there are too many possible combinations for these regions, the system utilises an interpretable white-box model such as a decision tree to learn which combinations provide the largest performance drop. An example figure of this process can be seen in Fig. 3.4

The core philosophy of manipulating the input as a means of obtaining information on the model is called *sensitivity analysis*, as it tests the sensitivity of the model to a local change. This kind of analysis already pre-dates back to 2003 in works such as [GDL03] , but the implementation of machine learning models as a means of learning the behaviour of the final model is initially introduced in LIME. This type of approach is intuitive as it is applied to the original image and is run on a white-box model. In fact, it can be applied to any model, making it a flexible way of providing insight to the user. However, with this flexibility comes the inability of providing information from inside the hidden layers of the model - it merely shows us an input-output relation.

[2]Link to DARPA XAI - https://www.darpa.mil/program/explainable-artificial-intelligence

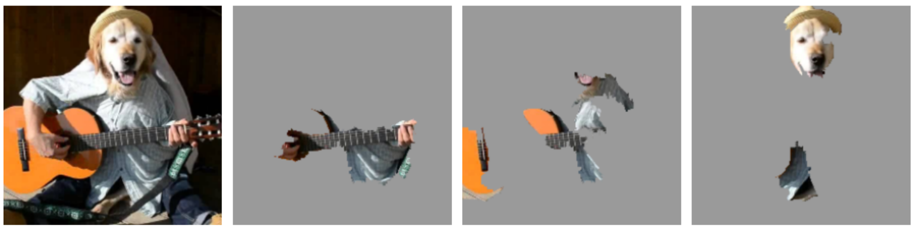

(a) Original Image
(b) Explaining *Electric guitar*
(c) Explaining *Acoustic guitar*
(d) Explaining *Labrador*

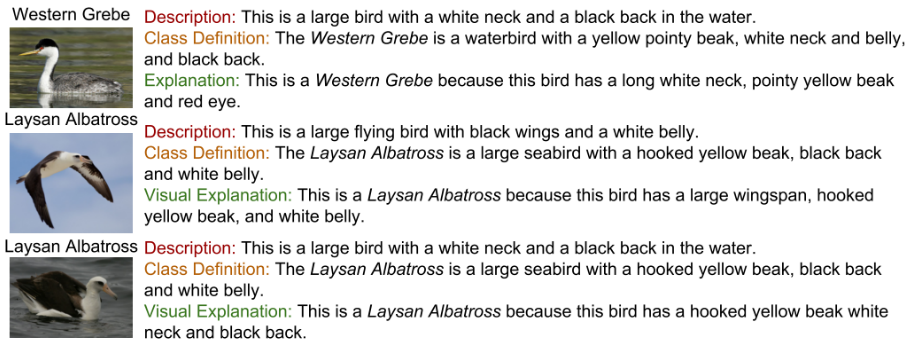
Figure 3.4: LIME Explanation
Extracted from [RSG16b]

Nonetheless, the second publication resulting from DARPA followed shortly afterwards, an improvement on LIME called aLIME for Anchor LIME [RSG16a]. This approach uses so called *anchors* to create *if-then* rules as an additional explanation to be fed to the model. The model then tries to learn the shortest yet most representative ruleset to describe a set of images. The goal of this process is to maximise the ruleset's coverage over the dataset whilst sacrificing the least ammount of precision, as it is possible to make a 100% precise ruleset if it is sufficiently strict to describe only a single image. This allows for a more precise and human-interpretable explanation on the process. Naturally, this comes at the downside of requiring a manual construction of the anchors specific to each problem. While the algorithm itself is model agnostic, this incurs an overhead that is specific to each problem at hand.

A set of five research investigations by Binder and his team are made public in 2015 and 2016 [BBM+15] [BML+16] [MBB+16]. In this work, Binder et. al. investigate the information flow of images as they propagate along a convolutional network. These studies look into possible ways of decomposing a convolutional network. The proposed method is through a Taylor-series decomposition and is better explained in [MBB+16]. Through this decomposition it's possible to generate a heatmap that describes the input activation on a pixel-level. The mathematical fundaments for why this decomposition is valid are explained in [MLB+17], another published work by the same team.

The extension of this method is proposed in [BML+16]. Here, a relevance score is computed for each neuron at each layer, such that, after propagation through the network, a relevance score can be granted to each pixel of the input image. In this fashion, it is possible to apply local renormalization layers to the network, which was previously not possible as fundamented in [BBM+15]. In order to qualify that the obtained results are valid, the team develops a metric for the relevance of a pixel based on sensitivity to changes of the input image. The premisse is that, if the pixels predicted by the algorithm to be the most relevant are correctly predicted, then randomly assigning RGB values to them will cause a larger disturbance in the model's performance than if the same transformation is applied to other pixels. Similarly, if the least relevant pixels are changed, then the expected result is a lesser loss of performance. The presented results are consistent with this premise, indicating that the algorithm corresponds to an effective decomposition of the network.

Still in 2016, a research work in generating visual explanations for images is introduced by Hendricks et. al. [HAR+16]. In an effort to construct an algorithm to generate relevant textual descriptions of an image, a dataset of bird images is used to produce descriptions that give insight on what the network recognised to be a part of the image by identifying sub-components. An example output of this process is shown in Fig. 3.5.


Western Grebe
Description: This is a large bird with a white neck and a black back in the water.
Class Definition: The *Western Grebe* is a waterbird with a yellow pointy beak, white neck and belly, and black back.
Explanation: This is a *Western Grebe* because this bird has a long white neck, pointy yellow beak and red eye.

Laysan Albatross
Description: This is a large flying bird with black wings and a white belly.
Class Definition: The *Laysan Albatross* is a large seabird with a hooked yellow beak, black back and white belly.
Visual Explanation: This is a *Laysan Albatross* because this bird has a large wingspan, hooked yellow beak, and white belly.

Laysan Albatross
Description: This is a large bird with a white neck and a black back in the water.
Class Definition: The *Laysan Albatross* is a large seabird with a hooked yellow beak, black back and white belly.
Visual Explanation: This is a *Laysan Albatross* because this bird has a hooked yellow beak white neck and black back.

Figure 3.5: Birds Explanation
Extracted from [HAR+16]

In order to produce these descriptions, a *Long Short Term Memory (LSTM)* generator model is trained on their dataset, with a reinvented loss function and multiple new metrics to evaluate image descriptions, class relevance and explanation performance. The proposed metrics aim to measure the relevance of a certain image description to the description of the class as well as its ability to explain the information in the image. This is difficult to achieve since the same image description (example: black bird with long beak) is relevant for multiple classes of birds. As such, the metrics provide an initial solution to quantify this precision.

3.2.1 Feature Transferring

The work published in 2014 in [YCBL14] represents the first attempt at creating a method of quantifying the transferability of layers in problems of the same domain. The purpose of this study is not to outperform existing models, but to analyse the behaviour of a well-known model in different test conditions, to better understand its learning patterns. The goal is to try and determine the degree to which features can be learned when the model's weights are transferred from a different model, a technique called *transfer learning*.

In the experiment, the team compares the performances of two models side by side on ImageNet. One model has had its layers transferred from the learning process on a different dataset, the other is pre-trained on the same dataset. With this setup, the study looks at how "transferable", in other words, how "generic" the features extracted at each layer are.

Additionally the study compares the performance of the two models when the base training is performed on similar datasets (sharing the same super-classes, such as *man-made objects*) against dissimilar datasets (not sharing the same super-classes, such as *man-made objects* versus *natural objects*).

This work creates a system to quantify the transferability of the layers between models, which allows for interpretation of the kind of features being extracted at each step. The conclusion is that the deeper layers of the model extract more specific features, thus being less transferable, which goes in line with the assumptions from the basic CNN model as explained in 2. It also provides insight on the relevance of hierarchical features, showing that the performance of the models vary depending on the super-class being used as the base for the initial layers.

3.2.2 The need for Explainable AI - Adversarial Attacks

In 2013, Szegedy et. al. undeniably proved that, even though the convolutional operation, intuitively, should allow CNNs to *see* in a similar fashion to humans, this is not the case. In their work [SZS+13], the team creates an algorithm to apply a learned disturbance on the input image, such that the human classification of the image is the same (in fact this disturbance, for the most part, cannot be seen by humans), yet the model fails at the classification task. An example of this process is shown in Fig. 3.6, where the image in the left is the original, correctly predicted image, the image in the center is the disturbance map and the image to the right is the incorrectly predicted image after applying the disturbance map on the original image.

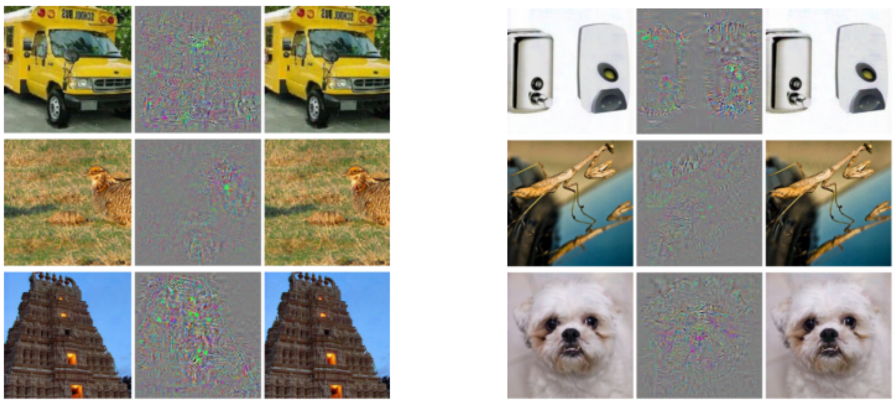


Figure 3.6: Adversarial Disturbance
Extracted from [SZS+13]

This inability to perform on only slightly altered inputs is revisited in other publications, such as [PMG+16]. Different algorithms are tested to produce the disturbances on the original input image, with the goal of reducing the amount of model-specific knowledge that is necessary to produce the disturbance maps, and [PMG+16] succeeds in creating an algorithm that only requires information on the input and output matchings. Essentially, the study confirms that the only information necessary to cause the model to fail is the output of the model to any given input. This cements the inapplicability of CNNs to critical systems and fields, at least in their current implementation.

3.2.3 Defining Explainable AI

Understanding the need for more explainable models, the pressing question that follows is how to define them. The effort of defining explainable models focuses in describing the desired properties of the produced models and evaluation criteria for these properties, to which end a major source stands out - [Lip16]. In his work, Lipton describes 4 primary evaluation criteria for explainable models - *trust, causality, transferability* and *informativeness*. Trust is a subjective criteria and also depends on the model's task, but it refers to the degree to which the end-user is able to rely and depend on a model's output for the given circumstance. Causality refers to the ability to infer causal relationships by analysing the model, transferability refers to the degree to which a model's generalization power can be used on unseen data and informativeness describes the ability of the model to provide meaningful, new information to the end-user. An example for this is having the model point to other data inputs that also generated the same classification - this gives more information to the user without describing the model itself. Another remark in Lipton's work regards the ethical component of deploying a machine learning model. With the increased usage of machine learning and AI in social fields like crediting and filtering job applications, eliminating any bias from the model that could incur an ethical dilemma is a concern.

There are, however, different definitions for explainable AI and model explainability [Joh16] [Fel17] and, to our knowledge, no accepted global standard and metrics for this problem exist so far.

3.3 Conclusions

In this chapter, the most relevant and revolutionary publications in the field of image classification and explainable AI are introduced. The history of innovation in image classification is provided before delving into the research of how to interpret these. A short description of the work presented in each study is provided, as well as the context in which the research occurs, to better compare the presented work to other studies at the time. The major contributions to the field are discussed and motivated upon for each work, as a means to provide motivation for the choices made for this dissertation, which are further discussed in the next chapter.

This chapter makes clear that much work needs to be done before deep convolutional networks can be applied in safety-critical fields like medicine and public transportation. There is insufficient insight on the exact decision process of these models, yet they prove to be substantially more powerful than any other solution in the market. Attempts at decomposing these models show great promise to provide some degree of post-training insight, but our insight on the hierarchical layering inside the networks is still mediocre.

Chapter 4

Project Planning

In this chapter, the initial planning for the development process of the project is introduced and motivated upon. This is followed by a comparison with the work that was actually produced, in which the setbacks that drove to any changes are presented. Additionally, relevant requirements are analysed and the solutions to the problems introduced by the requirements are discussed. This includes relevant technology choices, available datasets and models and even field of study.

4.1 Initial Problem - Medical Imaging

The initial planned purpose of this study is to create a model for breast cancer image classification. The goal is to be able to classify medical images produced through radiology and identify lesion areas for breast cancer, if present. For this, the image is fed to the model, which can then produce the mentioned classification. Given that a sufficiently accurate model is built and trained, the second step is to feed the output of the model to a second model, which can create a medical report, as similar to the report a specialist would write. This second model is to be trained with some dataset of medical reports and is to incorporate *natural language processing (NLP)* techniques. If successful, this would be a step towards automating medical report generation and potentially provide insight that a human would usually not identify.

This approach, however, proves to be far too ambitious for a master's thesis, for multiple reasons: the first and most relevant is the time constraint. The construction of either of the two models is a task sufficiently time consuming for a dissertation project by itself. Secondly, the datasets to work with were not available since the start of the project. As opposed to image and text datasets, for which standardised, benchmarking versions exist, medical datasets are small, scarce and typically require special permission to access due to patient privacy issues. Nonetheless, two datasets are identified as potential candidates for the study, namely *INbreast* [1] and the *Digital Database*

[1] INbreast homepage - http://medicalresearch.inescporto.pt/breastresearch/index.php/

for Screening Mammography [2] (*DDSM*). However, after further analysis, these show too much complexity for the available time - an example of this is that DDSM is a 20-year-old dataset. Additional problems regarding the quality of these datasets are also concerning, such as the quality of the data gathering process, data formatting and class balance. While the last two problems can be mostly addressed through pre-processing techniques to some degree, this again proves to be time consuming enough to last the duration of the project. Additionally, as mentioned in previous chapters, the medical field requires special attention to the decision process of AI models. There is no guarantee that a prediction generated by a black-box model such as a CNN, even if extremely accurate, is accepted by a specialist in the field.

After assessing these concerns, the decision to drop the NLP portion of the project is taken and full focus on the classification of medical images is given. The research focus changes from generating reports to creating methods of interpreting the model's behaviour, as this is a necessity in the medical field. However, due to the complexity involved in working with breast cancer imagery, AMT suggests working with a different dataset. The suggestion is to construct a superset consisting of 7 medical image datasets found in http://www.andrewjanowczyk.com/deep-learning/. This approach is initially accepted, as tutorials on how to individually work with these datasets are available, and the creation of a new, workable dataset in the field is an enticing opportunity to help future research.

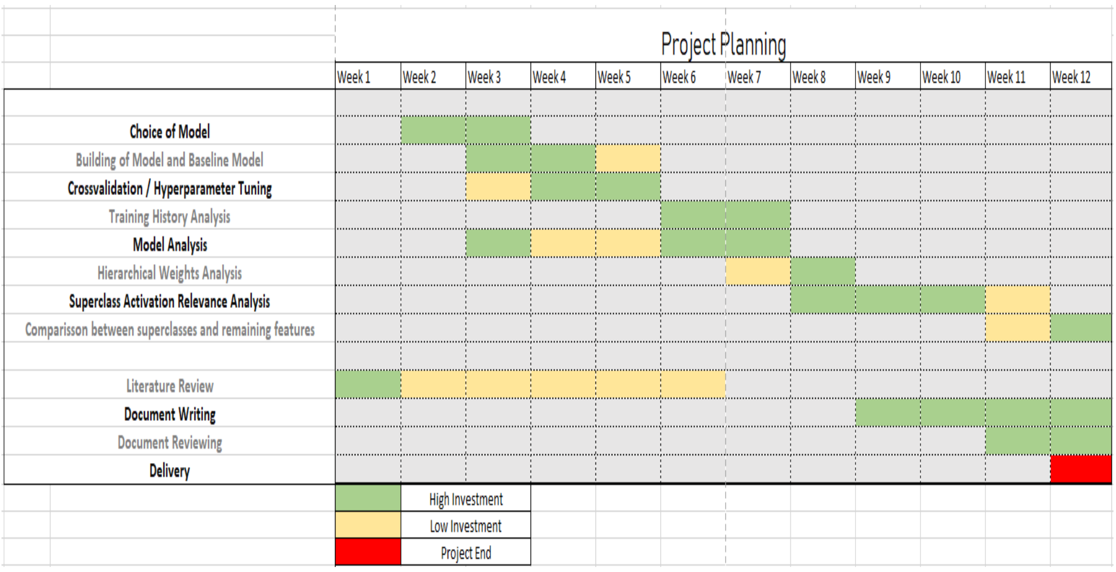
However, after preprocessing attempts at unifying these datasets, the conclusion is that they differ too much between each other. Consisting of images generated by different imaging techniques and machines, having different zoom, resolution, differing number of target classes, containing significant noise, overly different sizes - ranging from 6MB to 1.6GB - and, above all, each dataset being relevant to a different disease type, that is identified by a different set of features and has different properties, this suggestion proves unworkable.

Since ultimately the goal of the study is now to explore explainable AI and to try to contribute towards making more interpretable models, the decision to move out of the medical field into the standard image classification benchmarking datasets is taken. This has the advantage of making any results easier to compare with past and future research, as well as a guarantee of the quality of the dataset.

4.2 Time Plan

Given the remaining 12 weeks of work, the change in trajectory of the project and the need to recreate the background and state-of-the-art chapters of the dissertation, a new time plan needs to be devised. Because the technology choices from the previous work are still relevant, no time needs to be invested into choosing the best development environment. A diagram of the planned development process is shown in 4.1.

[2]DDSM homepage - `http://marathon.csee.usf.edu/Mammography/Database.html`



Project Planning

	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Week 11	Week 12
Choice of Model		🟩	🟩									
Building of Model and Baseline Model			🟩	🟩	🟨							
Crossvalidation / Hyperparameter Tuning			🟨	🟩	🟩							
Training History Analysis						🟩	🟩					
Model Analysis			🟩	🟨	🟨	🟩	🟩					
Hierarchical Weights Analysis							🟨	🟩				
Superclass Activation Relevance Analysis								🟩	🟩	🟩	🟨	
Comparisson between superclasses and remaining features											🟨	🟩
Literature Review	🟩	🟨	🟨	🟨	🟨	🟨						
Document Writing									🟩	🟩	🟩	🟩
Document Reviewing											🟩	🟩
Delivery												🟥

🟩	High Investment
🟨	Low Investment
🟥	Project End

Figure 4.1: Initial Project Planning

The process of choosing a model requires an analysis on the available models in the literature and their respective performances. In particular, a relevant characteristic here is the amount of processing power and memory required to use the chosen model, as time is a significant constraint. In this phase, the architecture for the new model is also chosen, that is, a part of the work is directed towards deciding how, why and where the model will be altered to create a new model. The next stage consists of building the chosen baseline model and test model. This requires familiarising the development team with *Keras* and *Tensorflow*, performing initial training runs, evaluating training and validation times and implementing utility functions to process data, generate graphics, logging and storing of the model. In parallel to this process, cross-validation and hyperparameter tuning is left to run over night due to it being an extremely long process, in an attempt to fit the model perfectly to the task at hand. With a trained and cross-validated model, the analysis of the training histories produced by the training process of both models begins. This is the step in which we determine whether or not there is any performance boost with the new model and start obtaining results. Once this is done, the weights and relevance analysis can begin. This involves constructing an algorithm that can, with the available *API* from *Keras*, access the layers' neurons and reconstruct the process of feeding an image through the model at any given point, granting access to any layer's output on demand. After the successful implementation of the algorithm, the weights and activation relevances can be analysed and discussed. The final step is to apply the same logic towards comparing the relevance of multiple layers, in this case, being able to adapt the algorithm to can compare the output of the coarse classification layer to the remaining densely-connected layer.

During this time, a literature review in explainable AI and image classification benchmarking is done, while the elaboration of this document is left towards the last few weeks of the project, as having the results available makes the writing significantly faster.

The planning manages to squeeze every step necessary to build a working model into the project, iterate on a new testing model to generate results with, while still reserving some time to analyse these results in time to publish them. The time reserved for analysis is limited and the number of analyses possible with this planning is equally limited. This proved to be a concern during the development of the project.

4.2.1 Planning Decisions - Dataset Choice

In order to compare the performance of the models that machine learners build, companies and communities work actively in order to create databases that can be used as benchmarking tests. In this section, the most relevant image classification datasets are introduced.

Arguably the most famous dataset in the field is the MNIST dataset [3]. It consists of 60 thousand training and 10 thousand test images of hand-written digits, with a resolution of 28 x 28 in gray scale. Due to its simplicity and relatively small image resolution, it rose to popularity due to the low computational requirements necessary to train a working model. This made it especially popular as an introductory dataset for beginners in the field.

Cifar-10 [4] is a dataset of 50 thousand training images and 10 thousand test images, with a resolution of 32 x 32, split into 10 categories, which were taken from the 80 million tiny images dataset [5]. As with MNIST, its small image resolution allows for fast training and good performance on relatively simple models.

Cifar-100 [6] is another dataset with the exact same properties as Cifar-10, except it is divided into 100 classes, called fine classes. The reason for the name is that the dataset also contains 20 coarse labels, which correspond to class families of the finer classes. This enables the study of hierarchical feature extraction of models trained on the dataset. In this work, the names *super classes* and *coarse classes* are used interchangeably, both referring to the class families.

ImageNet [7] is a dataset consisting of over 14 million images and over 20 thousand classes. Due to its sheer size, most research only uses a portion of the dataset. The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) is a yearly challenge hosted by ImageNet to incentivise the development of better performing models, where a smaller ImageNet dataset of roughly 1.3 million training images, 100 thousand test images and 1000 classes is used. This is the dataset that is typically used by research groups for benchmarking models. Having a varying resolution and also having significantly larger images than the other benchmark datasets discussed in this section, ImageNet is significantly more challenging, emulating real-world data more accurately.

Regarding the choice of dataset, importance is given to it being challenging enough to emulate a potential application, while being sufficiently small to be trained with the available resources. This immediately rules out the MNIST dataset, as it is too simple. Cifar-10 is also too simple and it doesn't support hierarchical classification *as-is*. This leaves Cifar-100 and ImageNet. While

[3]Link to MNIST homepage - http://yann.lecun.com/exdb/mnist/
[4]Link to Cifar-10 homepage - https://www.cs.toronto.edu/ kriz/cifar.html
[5]Link to 80 million tiny images dataset - http://groups.csail.mit.edu/vision/TinyImages/
[6]Link to Cifar-100 homepage - https://www.cs.toronto.edu/ kriz/cifar.html
[7]Link to ImageNet homepage - http://image-net.org

ImageNet is surely the more complex of the two and, as such, has more potential to represent real-world problems, it comes at the disadvantage of requiring more computational resources to work with, as the images are 64 times larger after resizing (from a resolution of 32x32 to 256x256). Combined with the problem of having less than 12 weeks of work left, the decision of working with Cifar-100 and its 20 super-classes is taken, with a focus on identifying the relevance of these 20 superclasses in the evaluation process of the model.

4.2.2 Planning Decisions - Model Choice

One of the possible solutions to save time and even have a more powerful model is to use a pre-trained model. By adopting a well studied model, less time needs to be dedicated into building and optimising the architecture of the model and the time sink of training the model can also be reduced, as only the final layers would need to be re-trained. In spite of this possibility and the limited time available, the decision to use transfer learning techniques is not embraced. One of the key aspects that this study focuses on is the analysis of the training history of a model that is initially fit to predict super classes and then re-trained to predict fine classes. The impact that this re-training of the convolutional layers has on the performance of the model cannot be examined on a pre-trained model as its training history is not available.

From the available models in the literature, GoogLeNet and VGG16 stand out, the former due to it being the best performing model, the latter due to its extremely good performance on a significantly lower computational resource requirement. Due to the time constraint of building, optimising and training a set of working models in 3 weeks, the decision to implement an even simpler version of VGG16 is taken. The exact architecture proposed is described in chapter 5, but the motivation behind the choice of a simpler model is that the trade-off between performance and development time of dropping a few layers of depth from the model is worth it in this scenario. The primary reasons for this are threefold: considering that the goal of this study is to analyse the model's inner workings and not to create a highly performing model, the loss of accuracy is, by itself, not a large concern. This premise is revisited in chapter 6 and 7. The second reason is that the architecture itself does not change as the depth increases, which means that the fundamental behaviour of the model is maintained between convolutional layers. The final reason is that, with the available hardware, the VGG16 and VGG19 models require implementation of code to handle the increased memory requirements, or, in turn, a decrease of the batch size used in training. The first option leads to a loss of time in investigating more interesting aspects of the model, while the second leads to less than optimal training results.

4.3 Languages and Frameworks

The choice of programming language and associated domain specific frameworks significantly affects the speed of development and execution of any software project.

Python[8] is the staple language for data science and machine learning solutions for its ability to manipulate matrices with the *numpy* [9] module. There are multiple modules available for visualising data and generating graphs, such as *matplotlib* [10]. Python also has multiple machine learning frameworks available which are still in active development. The industry trend, as seen by the graph displayed in Fig. 4.2 [11], indicates that Python is leading technology for the field of machine learning by number of job postings and is also experiencing the largest growth since 2015.

Keras[12] is a framework built for ease of implementation of deep learning solutions and comes with an API for building, training and evaluating convolutional models and even contains some tools for model visualisation and analysis. It has support for multiple backends such as *Theano*[13] and *Tensorflow*[14]. An alternative to Keras that is also available for Python is *scikit-learn*[15], a framework built to introduce core machine learning concepts to new developers that contains APIs for implementing, training, evaluating, visualising and manipulating models. Scikit-learn however does not yet fully support deep learning implementations, serving as a tool for more traditional models such as *Random Forests*. Since Tensorflow is backed by Google as well as an active open-source community, the choice of combining Keras and Tensorflow on top of Python is taken. Nonetheless, Scikit-learn provides some methods that Keras does not, such as the ability to run the *gridsearch*[16] algorithm for cross-validation and constructing scores like the *area under receiver operating characteristic curve*[17]. As such, when necessary, these methods are retrieved from SciKit-learn and combined with the Keras model's implementation.

[8]Python homepage - https://www.python.org/

[9]Numpy - http://www.numpy.org/

[10]Matplotlib - https://matplotlib.org/

[11]Fossbytes link - https://fossbytes.com/wp-content/uploads/2016/12/machine-learning-data-science-programming-language.png

[12]Keras - https://keras.io/

[13]Theano homepage - http://deeplearning.net/software/theano/

[14]TensorFlow - https://www.tensorflow.org/

[15]http://scikit-learn.org/stable/index.html

[16]GridSearch Cross Validation on SciKit - http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

[17]SciKit documentation on AuC - http://scikit-learn.org/stable/modules/generated/sklearn.metrics.auc.html

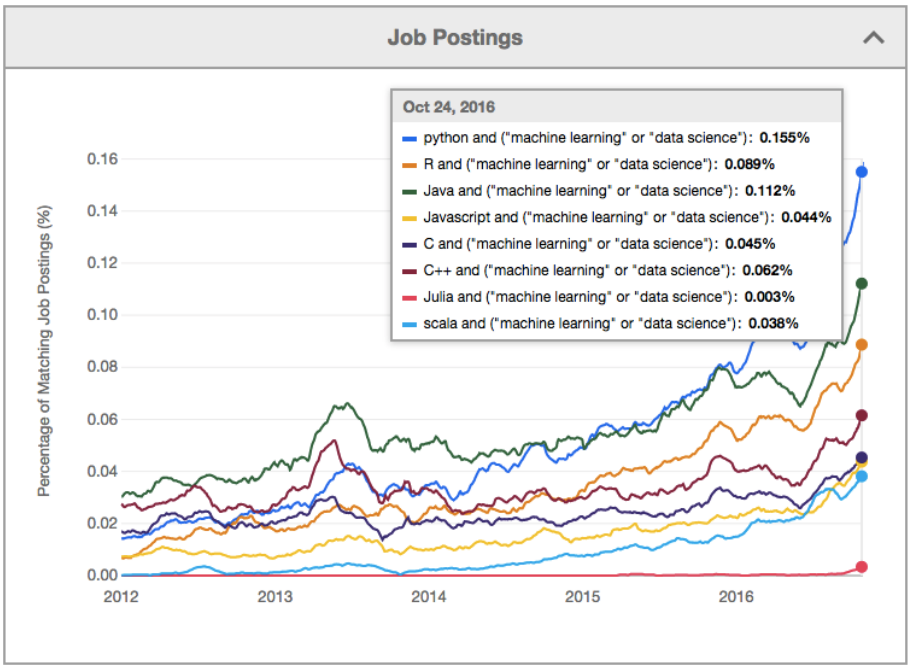


Figure 4.2: Machine Learning Job Query
Extracted from fossbytes

Chapter 5

Model And Experimental Setup

This chapter focuses on the architecture of the developed models and the analyses that are run on it. The motives behind each analysis are presented as a fundament for the decisions taken and expected results are described before being compared to the obtained results in chapter 6. Design decisions of the experiments tested in the project, as well as the results that originate these choices are discussed here, as chapter 6 is reserved for the analysis of the conducted experiments' results. The exception to this is the base model's performance results, which are presented in this chapter instead of chapter 6, since this model is used to fundament most decisions regarding the base architecture.

To classify and analyse the Cifar-100 dataset, two models are constructed. The first is the baseline model, which is used to compare against the obtained results from the remaining models. The second model is an adaptation of the base model's final layers, re-designed to incorporate information of both coarse and fine classes.

5.1 Base Model

The base layers consist of three stacks of convolutional layers, each stack being a sequence of two convolutional layers followed by a 2x2 max pooling layer and a dropout layer. As by the VGG design, the kernel size of 3x3 is equal across all convolution layers. Similarly, all convolutional and dense layers are activated by the *elu* activation, safe the network output layers, which are naturally activated by the softmax activation function.

The first stack takes a (32, 32, 3) dimensional input, which corresponds to the dimensions of the Cifar-100 images. These are then convolved into an output of (32, 32, 128) twice before being fed into the pooling layer which reduces the output to (16, 16, 128). The chosen dropout rate for this layer is 0.1.

The second stack takes a (16, 16, 128) input and convolves it twice into a (16, 16, 256) output which is fed into a maxpooling layer of size 2x2, thus outputting a (8, 8, 256) volume into the dropout layer, which is tuned to have a dropout rate of 0.25.

The third stack takes the (8, 8, 256) output as its input and convolves it twice into a (8, 8, 512) output. This is then pooled into a (4, 4, 512) volume, which is fed into the dropout layer with a rate of 0.5. This is then fed into a densely connected layer of 1024 neurons after being flattened. This is again fed into a dropout layer of rate 0.5.

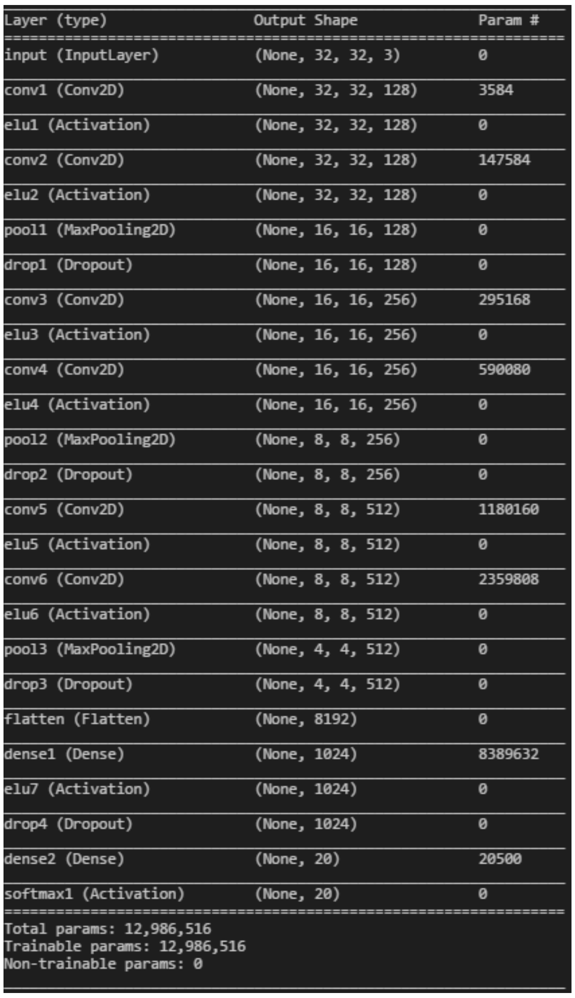
The optimiser of choice is Adam [KB14], a stochastic gradient-based algorithm, as with *stochastic gradient descent (SGD)*, that adapts the learning rate and momentum of the gradient according to local trends, in order to avoid overshooting and undershooting, while being computationally efficient.

This is the base architecture used across all models. For the base model that is used as a comparison tool, this output is then fed into one of two densely connected output layers, activated by softmax, which create the class probability distribution. If the desired prediction is that of the coarse labels, then the used model connects to a 20 neuron layer, otherwise, if the desired prediction is of the fine labels, the layer has 100 neurons. The summary of the architecture can be seen in Fig. 5.1.

The choice of this architecture comes from comparing the network of [Kru17] with the original VGG design. Due to the need of reducing the depth of the model, the number of stacks and the depth in some of the stacks is reduced, thus allowing the model to fit in memory. The convolutional output filter dimensions are kept at 512, but the input dimensions of 64 from the VGG design are kept. This leads to significant performance loss and unsatisfactory results.

In the original VGG design, the initial layers have a filter dimension of 64. Some suggested implementations for Cifar-10 even use a filter dimension of 32, as the task is less complex. However, given the relative shallowness of the presented architecture, the number of layers is not sufficient to achieve good performance with these parameters, as the information extracted in this step is not sufficiently complex. An initial attempt at fixing the performance loss incurred by this issue is to reduce the output filter dimensions. This leads to even worse performance. As such, gridsearch and k-fold cross validation is run to determine a new filter dimension for the first layers, in increments of powers of two for computational efficiency. The chosen k value for cross-validation is $k=3$. This value is chosen so low due to the computational requirements associated with having larger values of k. During cross-validation, another set of parameters is tested, namely the activation function. *ReLU* is tested against *Elu*, as a way to validate the sometimes inconclusive literature on the matter. The results show that a filter size of 128 and *elu* activation functions produce the best performing model for the given architecture. This results in an equal solution to [Kru17].

The learning rate is set at 1e-4 after cross validation testing for powers of 1/10 [ZB17]. No attempt is made at producing adaptive or stepwise learning rates, since the marginal gain of performance is not a significant concern for this study.



```
Layer (type)                 Output Shape              Param #
=================================================================
input (InputLayer)           (None, 32, 32, 3)         0
_________________________________________________________________
conv1 (Conv2D)               (None, 32, 32, 128)       3584
_________________________________________________________________
elu1 (Activation)            (None, 32, 32, 128)       0
_________________________________________________________________
conv2 (Conv2D)               (None, 32, 32, 128)       147584
_________________________________________________________________
elu2 (Activation)            (None, 32, 32, 128)       0
_________________________________________________________________
pool1 (MaxPooling2D)         (None, 16, 16, 128)       0
_________________________________________________________________
drop1 (Dropout)              (None, 16, 16, 128)       0
_________________________________________________________________
conv3 (Conv2D)               (None, 16, 16, 256)       295168
_________________________________________________________________
elu3 (Activation)            (None, 16, 16, 256)       0
_________________________________________________________________
conv4 (Conv2D)               (None, 16, 16, 256)       590080
_________________________________________________________________
elu4 (Activation)            (None, 16, 16, 256)       0
_________________________________________________________________
pool2 (MaxPooling2D)         (None, 8, 8, 256)         0
_________________________________________________________________
drop2 (Dropout)              (None, 8, 8, 256)         0
_________________________________________________________________
conv5 (Conv2D)               (None, 8, 8, 512)         1180160
_________________________________________________________________
elu5 (Activation)            (None, 8, 8, 512)         0
_________________________________________________________________
conv6 (Conv2D)               (None, 8, 8, 512)         2359808
_________________________________________________________________
elu6 (Activation)            (None, 8, 8, 512)         0
_________________________________________________________________
pool3 (MaxPooling2D)         (None, 4, 4, 512)         0
_________________________________________________________________
drop3 (Dropout)              (None, 4, 4, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 8192)              0
_________________________________________________________________
dense1 (Dense)               (None, 1024)              8389632
_________________________________________________________________
elu7 (Activation)            (None, 1024)              0
_________________________________________________________________
drop4 (Dropout)              (None, 1024)              0
_________________________________________________________________
dense2 (Dense)               (None, 20)                20500
_________________________________________________________________
softmax1 (Activation)        (None, 20)                0
=================================================================
Total params: 12,986,516
Trainable params: 12,986,516
Non-trainable params: 0
_________________________________________________________________
```

Figure 5.1: Base Model Architecture

As for the decay rate, cross validation in powers of 1/10 sets it at 1e-4, although the performance doesn't significantly differ from the default of 0.0.

The remaining hyperparameters produce too many combinations to gridsearch for with k-fold cross-validation. Nonetheless the attempt is made, but the computational requirements prove too steep. As such, the dropout rates of [Kru17] are kept, after manual testing. These values are also in line with the common knowledge of increasing dropout rates in deeper layers found in related literature; an example of this is [SHK$^+$14]. The results obtained from the training process reveal that the dropout layers' configuration is enough to fully prevent overfitting while retaining the ability to train the model without underfitting in less than 100 epochs when combined with minor data augmentation, as seen in the training history presented in Fig. 5.3. The same is true for the hierarchical model.

5.1.1 Training Methodology

For all training procedures, the random number generator seed for the numpy module is fixed for reproducibility effects.

The loss metric used for evaluating the model is the log of the categorical cross-entropy. This metric is chosen over accuracy as it measures the degree of certainty of the model in its prediction, as motivated in chap. 2.

The training procedure for the baseline model consists of randomly augmenting training batches of the Cifar-100 training set before feeding them to the model for training for a sufficiently large number of epochs, such that the early stopping condition is triggered.

As for the data augmentation technique used, horizontal and vertical shifting of 10% is used, as well as random horizontal flipping. This is extracted from [Ker]. The non-application of data augmentation leads to severe overfitting after as few as 5 epochs, as shown in Fig. 5.2.

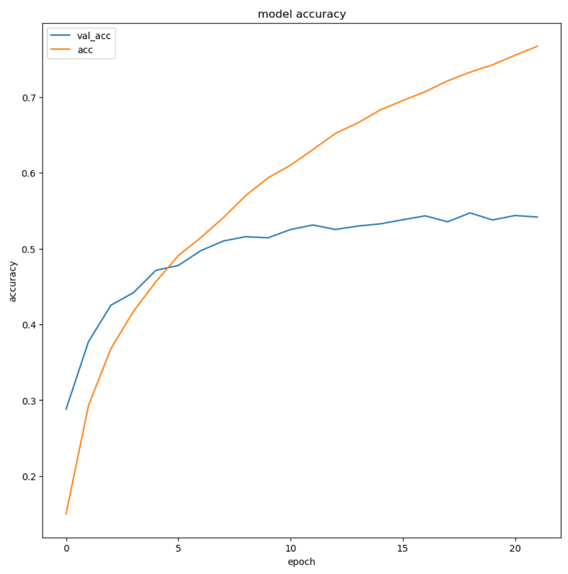
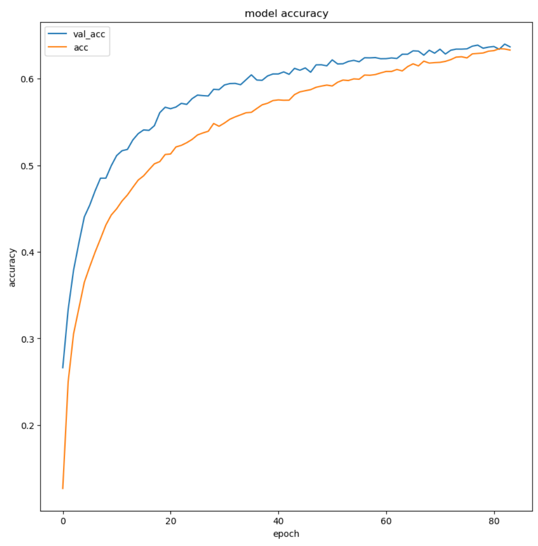


Figure 5.2: No Augmentation Accuracy

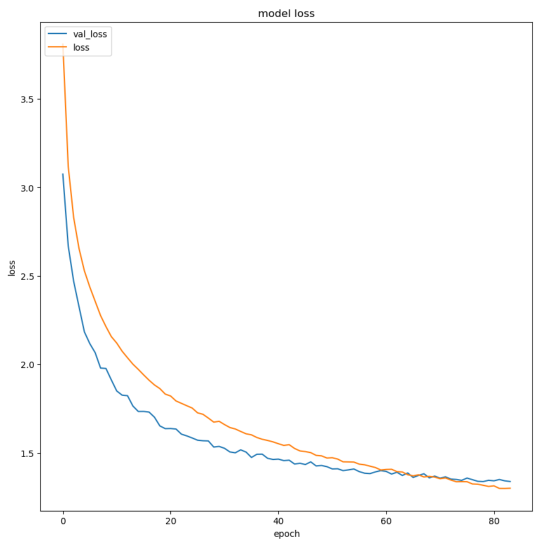
The early stopping criteria of choice is the validation categorical cross-entropy loss over multiple training epochs. *Delta* is chosen to be 0.001 over a *patience* period of 5 training epochs, as this allows the model to stop training before 100 epochs, by which point the training and testing errors have converged without overfitting (see Fig. 5.3). Increasing the patience yields wasted training time, as no significant improvement is achieved, while increasing the delta leads to a less fit model. An argument towards increasing the delta to save training time can be made, as the last 20 to 30 training epochs do not grant a very significant boost to the model's performance.

5.1.2 Baseline Performance

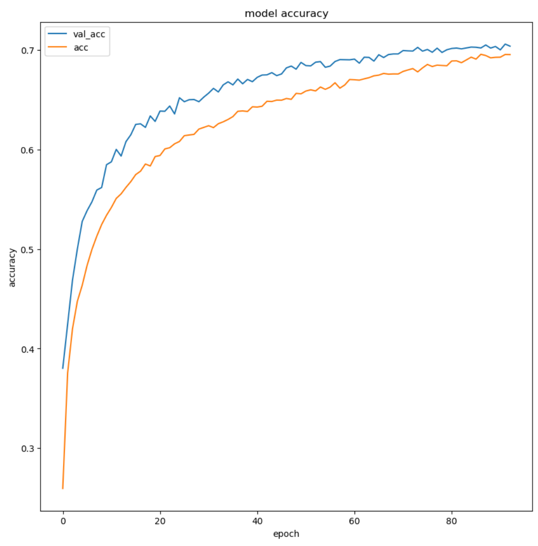
The baseline model that is built achieves a performance of 70.39% accuracy on the coarse targets, with a validation loss of 0.936. On the fine targets, the performance drops to 63.67% accuracy and a loss of 1.337. The training history can be visualised in Fig. 5.3 and the final results are presented in Table 5.1.



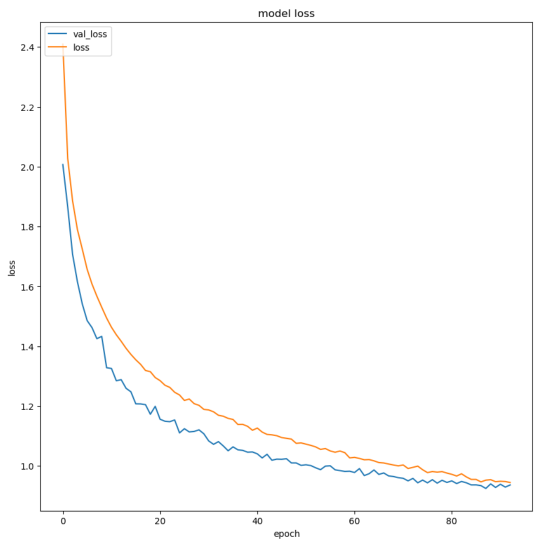
(a) Baseline Fine Accuracy



(b) Baseline Fine Loss



(c) Baseline Coarse Accuracy



(d) Baseline Coarse Loss

Figure 5.3: Base Model Training History

	Accuracy	**Loss**
Coarse Classes	70.39	0.936
Fine Classes	63.67	1.337

Table 5.1: Baseline Model Test Performance

5.2 Hierarchical Model

The hierarchical model differs only slightly from the baseline model in architecture. Its architecture is shown in Fig. 5.4. This model introduces an additional densely-connected layer of 20 neurons between the two densely-connected layers in the base model, which are capable of producing a coarse classification. This classification is an output of the model, allowing the model to be trained on this criteria. For this reason, this layer will be called *coarse layer* for the remainder of the document. This layer's output is activated by an *elu* activation layer and then fed into the fine layer as an additional input, creating a total input shape of 1044 (20 + 1024). The coarse layer is also activated by a softmax function to generate the coarse predictions.

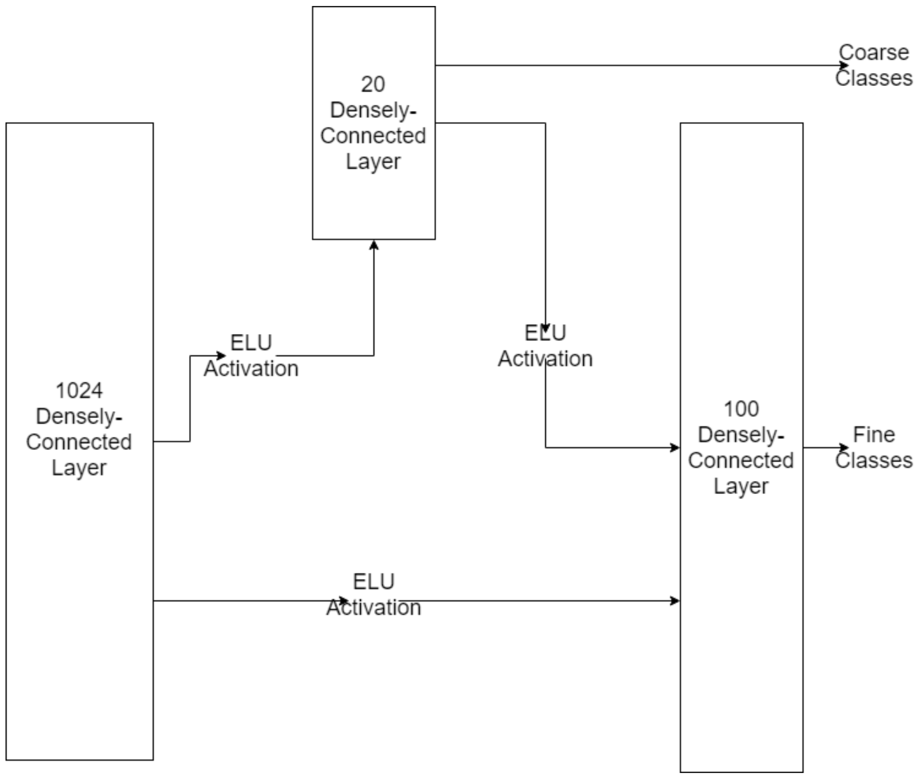

20 Densely-Connected Layer
Coarse Classes
1024 Densely-Connected Layer
ELU Activation
ELU Activation
100 Densely-Connected Layer
Fine Classes
ELU Activation

Figure 5.4: Hierarchical Model Architecture

What significantly changes is the training methodology. The model is initially trained on the coarse targets. This means that the loss function for the model consists of weighing the validation error on the coarse labels at 100% and 0% on the fine labels. This creates a model that is identical to the baseline model trained on coarse targets.

After this training procedure, the experiment introduced in this study begins. This involves a comparison between the training behaviour and performance of the hierarchical model and the baseline model. The hierarchical model, pre-trained on coarse labels, is re-trained to predict the fine labels, and the results of the two compared.

For the re-training process, the coarse layer is frozen, locking its weights. This ensures that the densely-connected layer trained to optimise the coarse targets is fixed and won't be updated. After freezing, the loss function for the model is updated to weigh the coarse loss at 0% and the fine loss at 100% and the fine targets are fed to the model. This means that every layer in the model is being re-trained, safe the coarse layer.

One concern with this approach is that the convolutional layers are being retrained. This is a necessity to produce some of the experiments in the study, but it comes at the price of a drop in performance on the coarse targets, since, even if similar, the extracted features from the coarse labels aren't exactly the same as those for the fine labels. This also raises a concern regarding the effect on the model's performance on the fine targets themselves, since one of the inputs to the fine layer is the coarse output. If the coarse output isn't sufficiently accurate, then it could result in "misinformation" being fed into the fine layer, that is, the model would perform better without that layer. Whether this scenario is happening or not can already be evaluated on the performance of the model after the first training epoch. Should this scenario happen, the expected loss and accuracy of the model are worse than those from the baseline model. This would imply that the features extracted can't describe both super classes and fine classes.

5.2.1 Experiment - Learning Curves Analysis

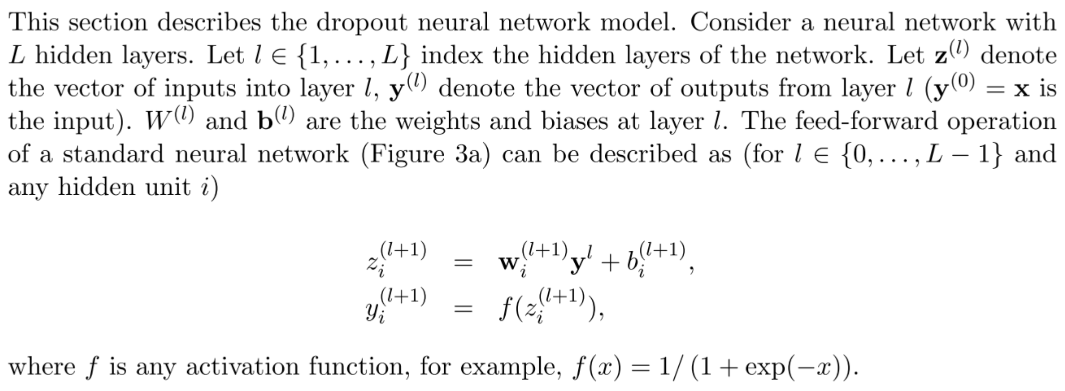
Our hypothesis is that, since the features that are representative of a coarse class are *similar* to those that represent a fine class due to their hierarchical properties, the training process should be significantly faster for this second phase. We hypothesise that the initial accuracy of the model should be significantly higher than the baseline fine model in the first epochs of training and that the training process should converge significantly sooner than the baseline model. Since the baseline model is more accurate in the prediction of coarse labels than fine labels, we hypothesise that the hierarchical model should slightly outperform the baseline model in the fine target prediction, as it has additional information to work with.

The second hypothesis related to the learning process is related to the behaviour of the model's performance on the coarse labels as it is being trained on the fine labels. As described earlier, we expect a significant drop of performance on the coarse targets during the start of the training procedure. However, if the assumption that learning the representative features for the fine targets is parallel to learning those representative of the coarse targets holds, then the model should be able to recover from this drop, even if this is not being considered in the loss function of the model. If the results confirm the hypothesis, then we can be sure that the convolutional layers are extracting information that is relevant to solving both problems, providing further evidence that the hierarchical properties of the data can be used to boost a model's performance. Nonetheless, we hypothesise that the final model's performance on the coarse targets will be lower than the baseline model's, as it is ultimately not entirely trained on that problem.

With regard to overfitting, we expect the re-training to tend faster towards overfitting, which needs to be covered by the early stopping condition. This is due to the fact that the model is learning faster, as it has already been trained before - even if fitting a slightly different problem.

5.2.2 Experiment - Relevance Analysis

Aside from validating the ability of CNNs to learn hierarchical features, this project also aims to measure if the model is considering the correct coarse class as the most relevant output from the coarse layer. This is because the coarse layer contains a high correlation to the output of the fine layer. Knowing that the fine class distribution is uniform and that each coarse class has 5 associated fine classes, if the model correctly predicts the coarse class for the sample, then a random guess for the fine class would have a 20% chance of being correct. This, combined with the remaining extracted features from the 1024 neuron layer, makes the output of the coarse class a valuable source of information for predicting the fine class if it is accurate. The problem is that the model could be determining the wrong coarse layer as relevant, which would indicate that either multiple coarse classes could be super classes for the fine class or that the model is extracting information in some unknown fashion. As such, we need to determine if the model is actually assigning the correct super class to the fine class. With this in mind, we create an experiment to try and measure the relevance of the correct coarse class output on the fine layer. In order to compute the relevance score, a look at the operation executed by the dense coarse layer is taken. This operation, as described in the work of [SHK⁺14], is shown in Fig. 5.5.


This section describes the dropout neural network model. Consider a neural network with L hidden layers. Let $l \in \{1, \ldots, L\}$ index the hidden layers of the network. Let $\mathbf{z}^{(l)}$ denote the vector of inputs into layer l, $\mathbf{y}^{(l)}$ denote the vector of outputs from layer l ($\mathbf{y}^{(0)} = \mathbf{x}$ is the input). $W^{(l)}$ and $\mathbf{b}^{(l)}$ are the weights and biases at layer l. The feed-forward operation of a standard neural network (Figure 3a) can be described as (for $l \in \{0, \ldots, L-1\}$ and any hidden unit i)

$$\begin{aligned} z_i^{(l+1)} &= \mathbf{w}_i^{(l+1)}\mathbf{y}^l + b_i^{(l+1)}, \\ y_i^{(l+1)} &= f(z_i^{(l+1)}), \end{aligned}$$

where f is any activation function, for example, $f(x) = 1/(1+\exp(-x))$.

Figure 5.5: Dense Layer Operation, extracted from [SHK⁺14]

As can be seen, the impact of each neuron on the next layer is dependant not only on its output, but also the weight of the connection that leads to the neurons in the next layer. Since the bias term is independent of the inputs, it serves no use in determining the relevance of each input. The second component that can influence the relevance is then the activation function $f(x)$. Since the layer's activation function is *elu* and for values of $x > 0 \to f(x) = x$, this function also doesn't impact the relevance of the neuron.

By ruling out those factors, we can determine that the relevance of each neuron towards the next layer's output is simply its output value multiplied by the weight that the next layer gives to that neuron, for each neuron. Using the same notation as in Fig. 5.5, let $Relevance(y_{i\to j}^{l\to l+1})$ represent the relevance of the output y_i^l towards the output y_j^{l+1} and let $W_{j,i}^{(l+1)}$ represent the weight of layer $l+1$ at index i for neuron j. The relevance score can then be described as:

$$Relevance(y_{i\to j}^{l\to l+1}) = W_{j,i}^{(l+1)} * y_i^l \tag{5.1}$$

As described earlier, we expect the most relevant coarse class input to fine layer to be the associated coarse layer of the image. To test this hypothesis, the model is fed with images from the testing set, and the input relevance for the fine layer's highest N activations from the coarse layer is computed. We then compare the frequency with which the predicted coarse class falls within these top N inputs.

Besides analysing the relative relevance of the coarse class, another analysis on the relevance is its relation to the accuracy of the predicted coarse and fine class. We hypothesise that, due to the increased accuracy of the coarse class prediction relative to the fine class, the fine class prediction is, on average, more accurate, when the network has learned to identify the coarse class output as relevant. That is to say, we hypothesise that the percentage of correctly predicted fine classes will be higher when the corresponding coarse prediction is considered relevant. Equally, we expect the accuracy on the fine labels to be lower when the coarse class isn't being considered relevant, as this means the model has recognised a different set of features to be relevant than expected.

To test this, we compare the accuracy of the model on the samples that had the coarse prediction be considered the most relevant to the accuracy of the model across all samples. We can identify 4 distinct scenarios:

Scenario 1 - Both coarse and fine label are correctly predicted.

Scenario 2 - Only the coarse label is correctly predicted.

Scenario 3 - Only the fine label is correctly predicted.

Scenario 4 - Neither label is correctly predicted.

We compute the percentage of samples that fall under each scenario when the model considers the coarse class as the most relevant input from the coarse layer and when it doesn't, as well as the percentages over all samples.

This type of analysis serves to understand how the model performs based on the kind of information it is retrieving. However, if, as we hypothesise, predicting the coarse label is solving a problem parallel to predicting the fine class, then the accuracy on the fine class and on the coarse class are two dependant variables. This is because the model will be inclined to train a particular coarse neuron to be very relevant when it is often correct in its prediction. Equally, it will consider it irrelevant when the coarse prediction is wrong. But, since we know that predicting the coarse class wrong means the fine class is more likely to be wrongly predicted as well, we have a logical relationship between considering a coarse neuron irrelevant and wrongly predicting the fine label - the same for predicting the coarse class correctly and considering the neuron relevant.

This consideration means that this type of analysis serves to understand the influence of the coarse layer on the fine layer only to some degree, as the expected poor performance on the samples whose coarse output isn't considered relevant by the fine layer is partly attributed to them being more difficult to predict. It does, however, serve to understand to some degree the ability of the model to rely on the alternative features extracted by the 1024 neuron layer as well as its reliance on the coarse output.

Chapter 6

Results Analysis and Discussion

In this chapter, the results obtained from the experiences described in chapter 5 are presented and discussed. The impact and significance of these results is debated and the questions that remain unanswered are discussed. Some potential solutions to obtain more relevant results are also present.

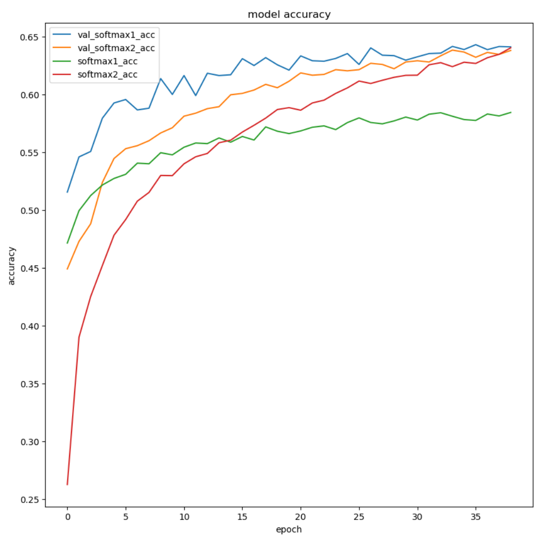
6.1 Learning Curves Analysis

The results from the training procedure are shown in Fig. 6.1, while the testing performance can be seen in table 6.1. In the showcased images, the *val* prefix (as in *val_softmax1_acc*) signifies the score on the validation set, while the lack of this prefix means the score refers to the training set. Softmax1 is the name for the coarse class output and Softmax2 stands for the fine class output. The *loss* and *val_loss* scores stand for the global loss and validation loss of the model, since this is a multi-output model. As explained in chapter 5, during the coarse training process, the loss function for the model weighs the coarse loss at 100% and the fine loss at 0%, which is why the *loss* and *val_loss* curves are identical to the *softmax1_loss* and *val_softmax1_loss* curves, respectively. Equally, during the second training process, the global losses are equal to the loss on the fine targets.

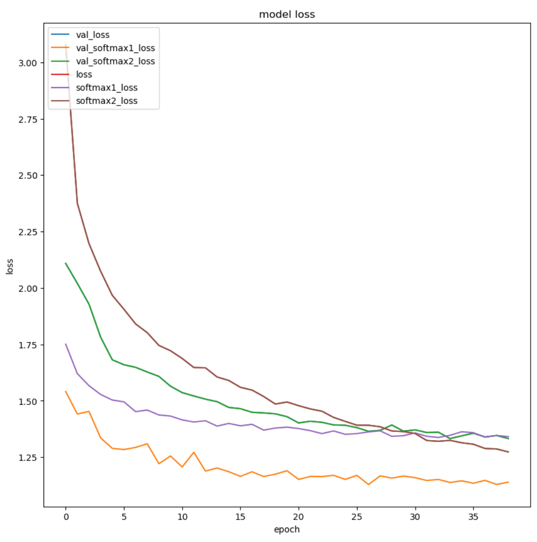
	Accuracy	**Loss**
Coarse Classes	64.10	1.139
Fine Classes	63.79	1.332

Table 6.1: Hierarchical Model Test Performance

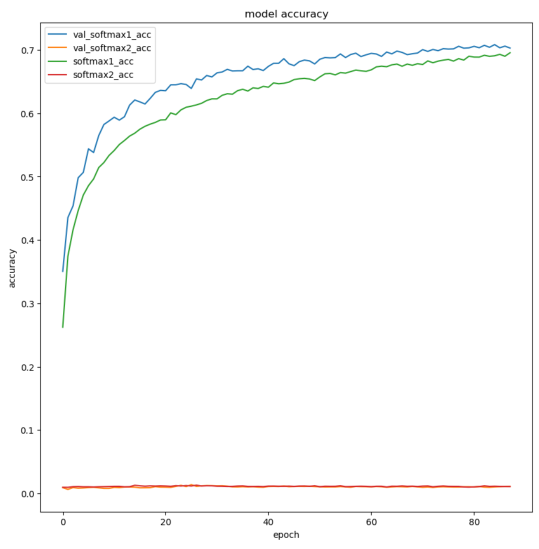
When comparing these results to Fig. 5.3, the first noticeable difference is the drop in performance on the coarse classes. As described in chapter 5, we expected a significant drop in the coarse prediction at the start of the re-training procedure, because the convolutional layers are being re-trained. What's surprising is that the initial drop is as big as it is - the testing performance



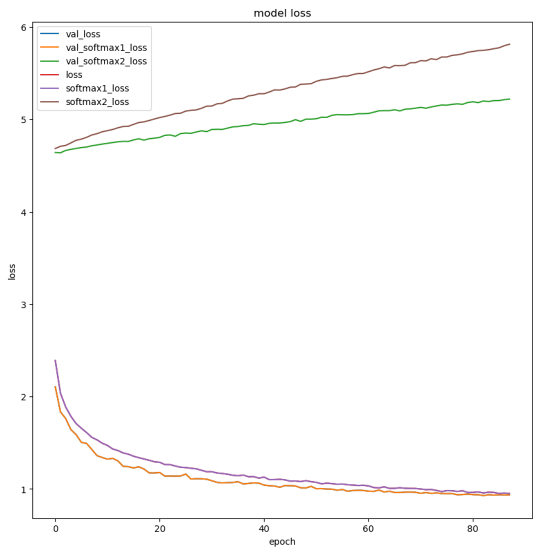
(a) Hierarchical Fine Accuracy



(b) Hierarchical Fine Loss



(c) Hierarchical Coarse Accuracy



(d) Hierarchical Coarse Loss

Figure 6.1: Hierarchical Model Training History

drops from 70% down to 52%. This means that the features that were learnt are being extremely refitted to solve the new problem. One possible explanation for these results is that the model may be learning unwanted correlations from the data alongside the representative features, which in turn do not entirely translate over to the fine labels. But at least some of the drop is explained by the fact that predicting the fine targets is *not* the same problem as classifying the coarse labels. Nonetheless, a significant amount of learning is kept at the convolutional level. This can be seen by the fact that the performance of the model after the first training epoch is 52% on the coarse labels and 45% on the fine labels, compared to the 38% and 27% from the baseline model, respectively. The same can be seen in the duration of the training process, as the early-stop condition is triggered before the 40th epoch, compared to the 85 epochs the baseline model needs.

These results show evidence that even if a model is to be trained in its entirety on a problem, loading weights from a model pre-trained on a parallel problem can still save a relevant amount of training time, as a significant portion of the fine-tuning is still kept.

Another interesting observation is that the fitting of the convolutional kernels on the fine targets classification problem is still solving the coarse label classification problem, as predicted, which can be seen by the rising accuracy and decreasing loss on the coarse targets in Figs. 6.1a and 6.1b. The fact that the densely-connected coarse layer's weights are frozen but are still applicable to the features extracted in the convolutional layer, regardless of whether they are being extracted to solve the fine or the coarse classification problem, is indicative of the relevance of this layer in the decision taking process of the network. In fact this is obvious when we take a look at Fig. 6.1c, where the accuracy of the model on the fine labels is stagnant because the fine layer isn't being trained, even though the convolutional layers are.

This implies that to better understand the decision process of the model, there is a need to understand the relation between the convolutional layers and the densely-connected layers that follow them, as the convolutional layers are clearly extracting relevant information in both training processes. Considering how the convolution operation is carried out in a human-like way [SZS+13], analysing regions of an image at a time, this begs the question of whether the issues and lack of understanding of the effect that adversarial attacks ([SZS+13]) have on CNNs could be due to the operation being performed by the dense layer. Unlike convolutional layers, the operation performed by a dense layer is based on the human understanding of the brain and the neural connections inside it. However, the number of connections that a neural network has at its disposal is multiple orders of magnitude lower and shallower than available in a human brain, making it a tall task to expect it to be able to extract information similar to a human.

The obvious suggestion to try and solve this problem is to approximate a neural network closer to a human brain and test its performance, but the reason why this is impossible is that we do not yet have the necessary computing power to accurately emulate a human brain nor the amount of data that would be necessary to do so. As such, research into this problem could focus into analysing the trends of the performance of models with increasing (and decreasing) depth and complexity. In other words, a potential research problem is to analyse the effect of adding more dense layers before the final classification and testing the performance of these adversarial algorithms in forcing the model to fail. We hypothesise that more complex models would tend to be less prone to such attacks, given that they are sufficiently trained and not overfitting. The same test needs to be conducted on increasingly large datasets, that is, we expect that a model that has seen more and different samples of a class to become less prone to adversarial attacks.

Another observation is regarding the final coarse and fine accuracy that the model achieves. While we expected the model to be able to generalise better towards the fine targets when compared to the baseline model due to the added layer, the results do not match our expectations. Achieving a peak performance of only 0.12% higher accuracy and 0.005 lower loss, these results do not draw any decisive conclusions. As discussed in chapter 5, we believe this is the case due to the re-training of the convolutional layers, which force the coarse classification error higher, making this coarse classification less impactful than anticipated and even demonstrated in other studies such as [ZB17].

6.2 Relevance Analysis

Through analysing the relevance of the neurons in the coarse layer towards the prediction of the fine target through the methods described in chapter 5, an attempt at measuring the applicability of this layer can be made. Table 6.2 shows the percentage of coarse classifications that are in the top N most relevant coarse activations for the fine class being predicted. Table 6.3 shows the accuracy on fine and coarse labels distributed over the 4 scenarios described in chapter 5. The *global* row shows the ratios over all samples, the *relevant* row over the samples whose coarse layer output was the most relevant input to the fine layer and the *not relevant* row contains the remaining samples.

	Percentage
N = 1	0.64
N = 2	0.73
N = 3	0.75
N = 4	0.82
N = 5	0.83

Table 6.2: Frequency of coarse class prediction in top N most relevant connections

	Coarse & Fine	**Coarse**	**Fine**	**None**
Global	0.498	0.156	0.142	0.204
Relevant	0.641	0.147	0.066	0.147
Not Relevant	0.244	0.172	0.278	0.306

Table 6.3: Relevance Scenarios Percentages

The results in table 6.2 show a discouraging percentage of correctly attributed relevance. Only 64% of the samples were maximally activated in the fine layer by the corresponding super class. Even when considering the 5 most relevant inputs to the fine layer from the coarse layer, a surprising 17% of the time the coarse class prediction is not present. This means that the model deems that information to not be descriptive of the fine target a significant number of times. This observation can be partially attributed to the fact that each coarse class has 5 fine classes associated to it and to the fact that the model itself only achieved a 64% coarse class accuracy. This means that even after the model manages to distinguish between those 5 classes, there will still be a 36% chance that it was neither of those 5. In sum, the information being provided by the coarse layer isn't perfectly descriptive. Nonetheless, the model does find the correlation to be significant 64% of the time, which is an interesting value, as it is also the accuracy of the model on the coarse targets, which is tied to the training process. We suspect this is no coincidence, as when the model is being trained and outputs a wrong coarse class, we expect this to not be a descriptive feature of the fine class, and as such we expect the model to not consider it relevant. With this consideration in mind, the results available from this project are not sufficient to specify this as the reason, and further research on this problem should be done.

The lack of increase in performance when compared to the baseline model initially seems contradictory: both loss and accuracy of the model are virtually equal to the baseline model, yet the model has every bit as much information as the baseline model available and has correctly interpreted the coarse output to be relevant more than half the time. The reason for this lack of improvement is most likely due to the accuracy on the coarse labels being essentially the same as on the fine labels, making any improvements to be gained from this approach marginal. However, this result serves to justify a potential choice of using an external or branched model to produce coarse predictions and then feed them back into the model, as we have shown evidence towards the coarse class's relevance. Hierarchical models built on this idea already exist, but this study manages to measure the impact of this architectural choice based on its relevance, rather than its effect on performance.

More insight on this behaviour is drawn from table 6.3. Here we obtain yet another confirmation that the fine and coarse predictions are related. This can be seen in scenario 1, that is, samples that had both coarse and fine targets correctly classified, contains 49.8% of the samples. If these were independent variables, the expected percentage would be the multiplication of the fine accuracy and the coarse accuracy, for a total of $0.6410 * 0.6379 = 0.4089$, or 40.89% of the samples. The percentage of correctly predicted samples rises significantly, up to 64.1%, when we analyse those that had the coarse prediction be the most relevant of the coarse outputs. Equally, it drops significantly, down to 24.4%, when the coarse classification is not relevant. These results prove that when the model identifies the coarse layer's prediction to be relevant, then it also tends to be accurate. The same can be seen when comparing the samples that had no correct prediction: those with the coarse prediction considered irrelevant are twice as likely to have no association to a correct target than otherwise, namely 30.6% versus 14.7%.

In order to draw significant conclusions from these results, a more thorough analysis needs to be conducted, as there is a significant amount of correlation between a sample's coarse output being relevant and this output being accurate. In other words, when the model attributes a high weight to a neuron connection, it's because this neuron is helpful in determining the correct target. This means that for the cases where the coarse class is being incorrectly predicted, there is a high likelihood that the neuron for that class won't represent the highest relevance and that the fine prediction will also be wrong. This becomes even more apparent when we consider the observation that the convolutional layers are producing information that is relevant for both fine and coarse classification.

In particular, it is interesting to analyse the effect that increasing the performance of the coarse classification model has on the relevance of the coarse prediction. For the aforementioned reasons we suspect that with a better-performing model, the relevance of this layer's output would be more often given to the correct class. Another interesting analysis to conduct is the magnitude of this relevance, as we suspect that whenever the correct coarse class is deemed relevant, the magnitude of its relevance is significantly higher than the magnitude of the most relevant class otherwise. Should future research find this to be the case, then it would be possible to determine with relative certainty that the samples whose coarse prediction isn't being considered relevant are those for which the model is unsure of the correct classification, more likely to be mistaken on and as such needs to rely on the features extracted from the alternative layers.

The analysis in this study focuses in determining the prevalence of the coarse prediction in the coarse output relevance, but understanding the overall relevance of the coarse output to the fine layer is an aspect which could not be covered in this project. Because the model identifies the correct super class to be the most relevant coarse output the majority of the time, we hypothesise that the output of this layer also be significantly more relevant to the fine layer's output than the other layer, at least on a neuron-by-neuron level. The next step in this project would be to compute the relevance score for each of the 1044 neurons that produce the fine layer's input and determine the impact that the coarse layer has on the final output. In particular, quantifying the relevance of the layer for varying models, with increasing performances, would allow us to obtain a better grasp on the potential of these implementations without needing to compare the actual performance gain. Understanding the effect of adding more convolutional stacks to a model, thus increasing its depth, can also be done through relevance analysis. Importantly, if it's possible to understand the impact of hierarchical features through relevance analysis, then more insight can be gained on the effect of increasing a model's depth through means other than comparing performance metrics.

As such, we propose potential further research into model interpretability to focus on computing the input relevance of the connections to a layer, while analysing this effect with varying model architectures, sizes and complexity.

Chapter 7

Conclusion

This study raises awareness for the current issues with black-box models that, while extremely performing, due to their inexplicable nature, are inapplicable to some fields to which they can be major contributions. Tackling this issue in its entirety is a huge task, to which this study contributes by identifying potential research opportunities and showcasing how answering these questions could aid in solving the issue. The work presented in this study also serves as further empirical analysis towards the potential of transfer learning and hierarchical models through the analysis of the obtained results. To achieve these results, convolutional neural network models are trained to perform image classification with the aim of analysing the training and evaluation process. Insight regarding the usefulness of coarse predictions in generating fine predictions is obtained through model re-training and relevance analysis, creating a set of observations that, while not confirmative of the hypotheses presented, serve to highlight potential future research opportunities on a qualitative level. Beyond highlighting the need for this future research to be developed, the presented results are also useful as further evidence for previously tested hypotheses regarding the use of hierarchical information in artificial and convolutional networks.

The primary lessons learned throughout the development of this project fall back to software engineering and project management, less so for the actual field of study, interpretable AI. The initial planning of the project to be developed is extremely lacking, which can be partially attributed to the remote communication that took place during this period, which highlights the need for continuous and working communication, a property that is constantly mentioned in current development processes such as *agile*. Not having a formally defined research goal and motivation by the start of the project, coupled with the lack of preparation of the dataset to be used, leads to an initial project that is bound to fail. Nonetheless, the importance of adaptation to the project is highlighted by the relatively quick change of trajectory in the project's goal, which allowed the project to still create relevant observations and extract useful insight even with only 12 weeks of development time, which had to be conjugated with the need of investigating the state of the art of interpretable AI solutions in CNNs.

As for the hurdles in the field of research, the importance of working with a well-known dataset and the need for computational power stand out. Ultimately, the quality and size of the dataset being used are major contributors towards the success of any study and deployment, as that is all the information that the model has at its disposal - any research project on this subject should prioritise the choice of dataset as early as possible, keeping quality attributes such as usability, integrity and size in mind. The decision to adopt a benchmarking dataset that has seen widespread testing and iterating by the community is the main contributing factor towards every result obtained in this project.

Looking into the future, this study raises some relevant questions that need to be answered. Does pre-training a layer to extract hierarchical features such as a coarse classification result in more relevant extracted features when compared to training a model from scratch? What is the exact impact of re-training initial layers of a model while freezing intermediate layers? What other forms of insight beyond those introduced in this study can be gained through this process? While the next step for this project in particular is to answer the first question by comparing the relevance of the coarse classification layer to the other densely-connected layer, a step that was planned but couldn't be completed, it is our hope that the remaining questions are not left unanswered.

References

[AGF07] Matteo Arvetti, Giuseppina Gini, and Michele Folgheraiter. Classification of EMG signals through wavelet analysis and neural networks for controlling an active hand prosthesis BT - 2007 IEEE 10th International Conference on Rehabilitation Robotics, ICORR'07, June 12, 2007 - June 15, 2007. 00(c):531–1424413206, 2007.

[BBM$^+$15] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation. *Plos One*, 10(7):e0130140, 2015.

[BCR97] J. M. Benítez, J. L. Castro, and I. Requena. Are artificial neural networks black boxes? *IEEE Transactions on Neural Networks*, 8(5):1156–1164, 1997.

[BML$^+$16] Alexander Binder, Grï¿½goire Montavon, Sebastian Lapuschkin, Klaus Robert Mï¿½ller, and Wojciech Samek. Layer-wise relevance propagation for neural networks with local renormalization layers. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9887 LNCS:63–71, 2016.

[Cho17] Francois Chollet. How convolutional neural networks see the world, 2017.

[Des] Julien Despois. Memorizing is not learning! — 6 tricks to prevent overfitting in machine learning.

[Des16] Adit Deshpande. The-9-Deep-Learning-Papers-You-Need-To-Know-About, 2016.

[EBCV09] Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. Visualizing higher-layer features of a deep network. *Bernoulli*, (1341):1–13, 2009.

[Fel17] Ed Felten. What does it mean to ask for an "explainable" algorithm?, 2017.

[GDDM14] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 580–587, 2014.

[GDL03] Muriel Gevrey, Ioannis Dimopoulos, and Sovan Lek. Review and comparison of methods to study the contribution of variables in artificial neural network models. *Ecological Modelling*, 160(3):249–264, 2003.

[Gir15] Ross Girshick. Fast R-CNN. *Proceedings of the IEEE International Conference on Computer Vision*, 2015 Inter:1440–1448, 2015.

[GSC16] Ian Goodfellow, Juergen Schmidhuber, and Aaron Courville. *Deep Learning*, volume 10. 2016.

[HAR+16] Lisa Anne Hendricks, Zeynep Akata, Marcus Rohrbach, Jeff Donahue, Bernt Schiele, and Trevor Darrell. Generating visual explanations. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9908 LNCS:3–19, 2016.

[HGDG17] Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross Girshick. Mask R-CNN. *Proceedings of the IEEE International Conference on Computer Vision*, 2017-Octob:2980–2988, 2017.

[IS15] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. 2015.

[Joh16] Jesse Johnson. Goals of Interpretability, 2016.

[KB14] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. pages 1–15, 2014.

[Ker] Keras Example.

[Kru17] Andrew Kruger. Keras Convolutional Neural Network for CIFAR-100 · Andrew Kruger, PhD, 2017.

[KSGE12] Alex Krizhevsky, Ilya Sutskever, and Hinton Geoffrey E. ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems 25 (NIPS2012)*, page 1–9, 2012.

[LCY13] Min Lin, Qiang Chen, and Shuicheng Yan. Network In Network. pages 1–10, 2013.

[Lec98] Yann Lecun. Gradient Based Learning Applied to Document Processing. 1998.

[Lip16] Zachary C. Lipton. The Mythos of Model Interpretability. (Whi), 2016.

[LPPF10] A. Livnat, C. Papadimitriou, N. Pippenger, and M. W. Feldman. Sex, mixability, and modularity. *Proceedings of the National Academy of Sciences*, 107(4):1452–1457, 2010.

[MBB+16] Grégoire Montavon, Sebastian Bach, Alexander Binder, Wojciech Samek, and Klaus-Robert Muller. Deep Taylor Decomposition of Neural Networks. *Proceedings of the Workshop on Visualization for Deep Learning at International Conference on Machine Learning (ICML)*, pages 1–3, 2016.

[MLB+17] Grégoire Montavon, Sebastian Lapuschkin, Alexander Binder, Wojciech Samek, and Klaus Robert Müller. Explaining nonlinear classification decisions with deep Taylor decomposition. *Pattern Recognition*, 65(May 2016):211–222, 2017.

[Nas07] Nasser M Nasrabadi. Pattern recognition and machine learning. *Journal of electronic imaging*, 16(4):49901, 2007.

[PES+06] Brett Poulin, Roman Eisner, Duane Szafron, Paul Lu, Russ Greiner, D S Wishart, Alona Fyshe, Brandon Pearcy, Cam MacDonell, and John Anvik. Visual Explanation of Evidence in Additive Classifiers. *Proc. Conference on Innovative Applications of Artificial Intelligence (IAAI06)*, pages 1822–1829, 2006.

[Pet17] Petar Veličković. Deep learning for complete beginners: convolutional neural networks with keras, 2017.

[PMG$^+$16] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical Black-Box Attacks against Machine Learning. 2016.

[RHGS17] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149, 2017.

[RSG16a] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Nothing Else Matters: Model-Agnostic Explanations By Identifying Prediction Invariance. (Nips), 2016.

[RSG16b] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. 2016.

[Sam59] Artur L Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229, 1959.

[Ser18] Sefik Ilkin Serengil. ELU as a Neural Networks Activation Function - Sefik Ilkin Serengil, 2018.

[SHK$^+$14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[SLJ$^+$14] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich, Chapel Hill, and Ann Arbor. Going Deeper with Convolutions. pages 1–9, 2014.

[SVZ13] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps. pages 1–8, 2013.

[SZ14] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. pages 1–14, 2014.

[SZS$^+$13] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. pages 1–10, 2013.

[WZL17] Songtao Wu, Shenghua Zhong, and Yan Liu. Deep residual learning for image steganalysis. *Multimedia Tools and Applications*, pages 1–17, 2017.

[YCBL14] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? 27, 2014.

[ZB17] Xinqi Zhu and Michael Bain. B-CNN: Branch Convolutional Neural Network for Hierarchical Classification. 2017.

[ZF14] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8689 LNCS(PART 1):818–833, 2014.