MSc Data Science - Project Report

Artificial Intelligence in Cybersecurity: A Novel Deep Learning Architecture for Image-Based Malware Classification

Dhinta Jesslyn Foster
Student Number: [redacted]
Supervisor: Dr Paul Yoo

Department of Computer and Information Systems

Birkbeck, University of London

September 2020

Academic Declaration

This report is substantially the result of my own work, expressed in my own words, except where explicitly indicated in the text. I give my permission for it to be submitted to the JISC Plagiarism Detection Service. The report may be freely copied and distributed provided the source is explicitly acknowledged.

Signed: Dhinta J. Foster

Abstract

Malware remains as the number one cyber-threat globally (ENISA, 2019) and is stated as the cause for over 25% of all data breaches. The motivation of this project stems from a need to analyse and identify malware with greater efficiency and without relying on malware experts. Increasing variants of malware files are being developed daily with greater evasion and obfuscation ability. The behavior and impact (and subsequent malware defense strategies) of malware is dependent on the type of malware resulting in the task of malware classification to be a crucial step in the cybersecurity process. In the past decade, the technique of malware visualisation (by converting the binary structure of malware into images) has been utilised in the task of classifying malware families, given its success over traditional static and dynamic analysis of malicious files as there is a reduced need for domain expertise. Malware classification models powered by deep learning have achieved state-of-the-art accuracies and require little-to-no heavy feature engineering compared to other machine learning models.

Nevertheless, traditional deep learning architectures such as convolutional neural network (CNNs) variants do not consider the spatial hierarchies between features, and also lose some information on the precise position of a feature within the feature region, which is crucial for a malware file which has specific sections. Sabour et al (2017) introduced Capsule Neural Network (CapsNet) architectures, producing state-of-the-art results on image classification tasks and have seldom been used in the malware classification field. This project implements a novel CapsNet with hyperparameter optimized convolutional layers through a Random Search method, which overcomes CNN limitations by removing the need for a pooling layer and introduces capsule layers. The presented method is tested on a large, unbalanced malware dataset (MalIMG) and experimental results produce a testing accuracy of 91%, improving on a number of traditional deep learning models posited in recent malware classification literature.

Index Keywords: Malware, Cybersecurity, Image Classification, Capsule Neural Networks, Convolutional Neural Networks, Deep Learning, Machine learning

Acknowledgements

I would firstly like to thank Dr Paul Yoo for his excellent supervision and motivation throughout this research project, especially given the unprecedented changes to college life due to the on-going COVID-19 pandemic. Secondly, I am grateful to the professors and teaching assistants in the Department of Computer Science and Information Systems for providing me the opportunity to challenge myself in this academic field. A sincere thank you also goes to my friends and colleagues for putting up with me over the last two years. Finally, without the continuous encouragement from my parents, this postgraduate degree would not have been possible.

Contents

List of Figures

List of Tables

1 Introduction

Malicious Software, or Malware, is a term for software that contains code developed with the intention to cause damage to computer systems and databases, or to gain unauthorised access to sensitive information and private servers. Increasing variants of malware are being developed daily (Kalash et al, 2018), especially with greater ability in evasion of anti-malware defense systems. Independent information security research has shown that there are more than 350,000 new malware created daily and over a billion malware files are in existence in 2020 alone (seen in Figure 1). The proliferation of malware is partly due to constant and ever-changing modification and obfuscation techniques used by malware developers (Vu, Nguyen et a 2019). The importance of Cybersecurity and the information security is highlighted by the fact that the industry is a booming market. Global annual revenues totalled over $36.3 billion USD in 2018 alone, with 25% of the market share being generated from anti-malware applications [1]. Nevertheless, policymakers have stated that malware is still the number one cyber-threat, responsible for over a quarter of all global data breaches (ENISA, 2019).

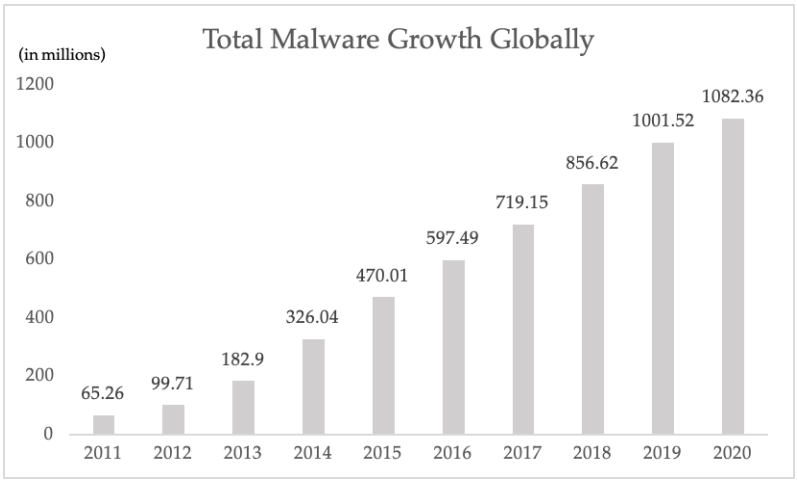

Figure 1: Annual Total Malware Statistics over the past 10 years has exponentially increased (AV-TEST, 2020)

.

The motivation of this project stems from the need for innovation of software with the ability to analyse and identify malware files with greater efficiency and without a need or reliance on malware experts, but also with the same efficacy as someone with domain expertise. Gibert et al (2020) also argue that given the

[1] Statista (2020). [online] Available on https://www.statista.com/topics/2208/security-software/

complexity and innovation in new malware, security analysts are in a constant battle with malware developers. Approaching this requirement with Deep Learning as a solution allows a purely data-driven approach for feature extraction and malware analysis. Malware analysis is split into two primary tasks: (1) malware detection and (2) malware classification. This project will focus on the latter. Once a file has been detected to be of malicious content, the next phase is to determine what family it belongs (i.e. files with either similar code or a similar origin). The task of malware classification is one of the most important parts of the cybersecurity analysis process. The impact of malware varies depending on which family the malware belongs, resulting in the response of commercial anti-malware strategies to differ. Within cybersecurity, one main aim of malware analysis is to capture additional properties for improved security measures (Ucci et al, 2018). Therefore, by capturing different properties of malware and recognising which family a sample within a dataset comes from, any defensive measures can become more effective.

1.1 Problem Description & Project Objectives

1.1.1 Problem Description

In the area of malware classification, there is currently no standard commercial taxonomy of malware classes. There are also new malware classes created daily and given the multitude of ways malware can be developed, creating and updating heuristics for all class types is a difficult and never-ending task. For the purpose of tackling the classification challenge given the enormous volume of new malware and the constant changes, it has become necessity to utilise techniques that requires little-to-no domain expertise in binary disassembly or malware behaviour analysis (Le et al, 2019), as well as not relying on complex feature engineering.

As software evolves, it is imperative that the cybersecurity industry adapts to increasing malware attacks and obfuscation techniques in a faster and more cost-efficient way without the reliance on domain experts. The newest malware families continue to coexist with older malware types resulting in classification of malware to be convoluted (Corrales, 2020). The techniques presented in this research project are an advantage to this field as it allows new classification criteria to be developed in an automated manner helping to reduce the amount of cyber-crime.

To solve this issue, I propose a novel artificial neural network - a Capsule Neural Network (CapsNet) solution - for classification that aims to learn features automatically without over-fitting. An emerging trend over the past decade in this

field is to employ object classification techniques. By transforming raw malware binary files into images, one can then frame malware classification as analogous to a supervised learning object classification problem, without the need of any deep expertise of the malware files themselves. The following subsection will describe the project aims and objectives in more detail, as well as summarising the methodology and contribution of this paper.

1.1.2 Aims & Objectives

Aims

The aims of this graduate data science project are to implement, train and evaluate a novel deep learning model in Python for malware classification and to present this project report as the final deliverable. Furthermore, this project aims to improve malware classification accuracy in a popular malware dataset ("Malimg dataset") by utilising a modified Capsule Network Model, that has yet to be implemented in this domain in existing literature.

Objectives

The detailed methods are presented in detail in Section 3, but in summary, the objectives, or how this project will achieve the aim, are as follows:

1. A popular malware dataset, "MalIMG" containing malware binaries from 25 different families will be used and transformed into as images.
2. The images will be split into a training and validation set for training a deep learning model and testing
3. Two simple Convolutional Neural Network architectures will be implemented and a random search will be used to refine the hyperparameters of the convolutional layers before the implementation of the novel architecture
4. A baseline Capsule Neural Network as outlined in Sabour et al (2017) will be implemented
5. An extended Capsule Neural Network with multiple convolutional layers will be implemented that is tailored to the dataset
6. An evaluation of the testing results will be compared as well as an extensive discussion of the limitations of the model that will provide a base for future work

1.2 Contribution of this work

As of writing (July-September 2020), there has only been one paper in this academic field that makes use of CapsNet in malware classification as well as proposing an ensemble version of CapsNets in this domain (Cayir et al, 2020). This project aims to contribute to current literature with a number of novel changes.

1. This work develops a new specification in the domain of malware classification by implementing a novel modified Capsule Neural Network architecture
2. Implements a Capsule Neural Network model on a popular benchmark in the field of malware classification: the MalIMG dataset
3. Compares a hyper-parameter optimised convolutional neural network architecture to a Capsule Neural Network and demonstrates the effectiveness of these optimisations within the convolutional layers of the CapsNet model
4. Presents a detailed evaluation of prior work and comparison of image-based deep learning solutions to malware classification
5. Determines the feasibility of a CapsNet model for deployment by presenting an extensive evaluation of model results, as well as proposals for future development to overcome current limitations

In addition to data science techniques involved in data pre-processing, model implementation and evaluation, I will also utilise cloud computing services to handle datasets and the computational intensity.

1.3 Amendments to the first proposal

A number of amendments were changed from the initial proposal given unforeseen challenges faced whilst in implementing this project. Given the on-going COVID-19 pandemic, access to university hardware was unfeasible but some of these risks were considered. Fortunately, the computational power and limited access to free virtual machines including GPUs from Google Colaboratory allowed this project without significant monetary cost. Nevertheless, the original dataset (Microsoft Malware Challenge Dataset, 2015) was too large to physically process on a virtual machine *without cost* given unzipped, it was 0.5 TB. As a result, the move to use an alternative, but appropriate dataset was implemented during the process with a slight alteration to the objectives.

Report Outline

Section 2 will analyse and evaluate prior work in this field. Section 3 sets out the design and implementation methodology as well as a theoretical overview of CNNs and for the proposed CapsNet model. We also evaluate the limitations of more commonly used architectures and why a CapsNet architecutre should be used to tackle such limitations. The data pre-processing stages including the process to convert raw malware binary files into images will be described. Finally, section 3 will outline the architectures and experimental setup. Section 4 will outline the experimental results with the discussion and evaluation set out in Section 5. The project will finalise with future work and concluding remarks.

2 Literature Review

The key objective of this project is to classify malware to their respective families and prior work outlines that there are many techniques for classification. Traditionally, malware analysis relies on static or dynamic analysis, by which patterns are matched against signatures extracted from known or existing malware families by disassembling a file or running the code on virtual machines (Bhodia et al, 2019). Naturally, these conventions rely heavily on domain expertise, therefore models that can be automated are intuitively more useful and time-efficient in comparison (Damodaran et al, 2017; Singh et al. 2016). Furthermore, these approaches are not conducive to efficient real-time malware classification monitoring. There is a growing trend in utilising machine learning techniques in malware analysis and an extensive survey of such methods is presented in Ucci et al (2018). This section outlines the strengths and weaknesses of supervised learning methods used with the same dataset (MalIMG).

Malware Classification as an image-recognition task

Nataraj et al (2011) were the first authors to take the approach of malware classification as an image-based classification task and their transformation of malware into images is the basis of this project[2], as well as many other academic papers. The authors observe that images belonging to the same malware family appear very similar in layout and texture after interpreting raw bytecode as a greyscale image. In order to deploy this malware image as a classification task, all malware executable binaries must first be reshaped into an image. The authors then apply a k-nearest neighbours (kNN) algorithm, where the unknown test malware image is assigned a label that matches a majority vote returned from the training set. Comparatively to papers that employ a dynamic analysis classification method, their accuracy rate hits 88%. However, the training time taken for classification is still a significant improvement, which may be favorable for classification in remote or IoT devices. One of the main benefits of this method is that disassembly nor dynamic code execution of a malicious file is not required. Nevertheless, this is an early paper and other papers in the past decade have already improved upon their feature extraction methodology.

Within this field, the majority of prior work makes use of a variety of Convolutional Neural Networks (CNNs) architectures for malware as image-classification tasks. Firstly, Rezende et al (2017) apply the same techniques to the MalIMG dataset using Transfer Learning of a ResNet-50 Deep CNN, by first convert-

[2]Details in the Methodology Section 3.2

ing each byte plot to an RGB image. The model outperformed in accuracy compared to kNN and GIST classification models and achieved an impressive 98.62% accuracy and no feature engineering required. Knowledge from the ImageNet classification task can be transferred to other malware classification tasks. Lo et al (2019) apply an "Xception Convolutional Neural Network" with similar methodology achieving a 99.08% accuracy - a higher accuracy than other CNN variants (including VGG16) however one limitation posed is that other machine learning approaches had lower logloss. Despite both papers achieving impressive accuracy, both models could be improved further by adding additional layers and complexity to extract more features. That being said, the more computationally intensive these models become, the less applicability they could have for malware analysis in remote devices that may not have the hardware to cope with such level of computation.

Most recently, Vasan et al (2020) implement an Image-Based malware classification using ensemble of CNN architectures (IMCEC) by utilising transfer learning architectures pre-trained on the ImageNet dataset achieving a 99.50% accuracy rate. The IMCEC classified the majority of malware samples correctly even under obfuscation attacks. The authors note that the model performed better than existing methods employing similar benchmarks, but that being said, these authors have never tested this model in a real-world domain and could utilise other architectures such as the Inception-ResNet-V2.

Other machine learning methodologies have had equally high success rates. Ghouti & Imam (2019) present a Multiclass Support Vector Machine model to the same dataset and achieve a near perfect accuracy rate of 99.8%. Unlike existing classification models, their solution requires simple algebraic dot products to classify malware based on representative digital images, whist results in computational complexity is $O(n)$ compared to $O(n^2)$ for CNN, and larger complexities for other methods (GIST + SVM for example). Nevertheless, their method presents the standard challenges of selecting feature vector size through Principal Components Analysis (i.e. the curse of dimensionality) and that this model would require re-training each time new malware variants are presented.

Capsule Neural Networks in Malware Classification

Capsule Neural Networks (CapsNets) were first introduced by Sabour et al (2017) and are considered the latest breakthrough in computer vision[3]. There have been

[3]For more detailed overview of Capsule Neural Networks, please refer to Section 3.3.2 of this project

numerous applications in the past three years, especially within the medical field (e.g. Anupama et al, 2019 & Ashfar et al, 2019). On the other hand, within the malware classification domain and cybersecurity field, there are gaps. As of 2020, only one paper exists in literature, where Cayir et al (2020) propose a Random CapsNet For Imbalanced Malware based on bootstrap aggregating ("Bagging") methods. The authors design a CapsNet architecture on the Microsoft Malware Classification Dataset (BIG, 2015) which allows for two simultaneous image inputs, such that features from the two different file types (Byte files, ASM files) are concatenates and represented in the output vector. Their RNCF on this dataset achieves a 99.56% accuracy. Though an impressive accuracy result, there are arguably two limitations to this paper I tackle in my proposed methodology. Models based on the Random Forest algorithm make use of an "average ensembling" step, which essentially will base its prediction calculations on the dataset classes on an average of previously observed labels. Phaye et al (2018) argue that feature maps learned by the first convolution layer in the baseline CapsNet model only learns basic features, which may lack flexibility for deeper feature extraction. The following section will outline the theoretical underpinning of a convolutional neural network as well as a capsule neural network.

3 Methodology - Design & Implementation

3.1 Design

The project workflow was structured as per the project proposal. By dividing this project into several phases and key milestones, efficiency and productivity were optimised whilst carrying out the project:

- Phase I: Data Collection
- Phase II: Data Pre-processing
- Phase III: Model Development, Creation & Evaluation
- Phase IV: Report Write-up

The following section and its respective subsections will present the specification and design of the models proposed in order to meet the project requirements, the data pre-processing techniques, the theory behind the architecture as well as software and hardware implementations the malware classification techniques.

3.2 Dataset

Public access to malware datasets are limited, but the dataset used for training and evaluating the proposed model is the "MalIMG Dataset". First introduced by Nataraj et al (2011), this dataset consists of 9,339 malware binaries converted into greyscale labelled images with 25 different malware family classes. Details are available in Table 1. The authors describe the process by which a malware executable file can be "represented as a binary string of zeros and ones", and in turn "reshaped into a matrix and viewed as an image". Due to the common practice of recycling code to create variants of malware, there were visible similarities in the image texture of malware belonging to the same family (Ibid, 2011). To justify, this technique is advantageous to traditional static and dynamic approaches described in the literature review, given little requirement for file disassembly, file execution, or prior knowledge on features of different malware families.

Class Number	Class Type	Family Type	No. of Samples
1	Dialer	Adialer.C	125
2	Backdoor	Agent.FYI	116
3	Worm	Allaple.A	1591
4	Worm	Allaple.L	2949
5	Trojan	Alueron.gen!J	198
6	Worm:AutoIT	Autorun.K	106
7	Trojan	C2LOP.P	146
8	Trojan	C2LOP.gen!g	200
9	Dialer	Dialplatform.B	177
10	TDownloader	Dontovo.A	162
11	Rogue	Fakerean	381
12	Dialer	Instantaccess	431
13	PWS	Lolyda.AA1	213
14	PWS	Lolyda.AA2	184
15	PWS	Lolyda.AA3	123
16	PWS	Lolyda.A	159
17	Trojan	Malex.gen!J	136
18	TDownloader	Obfuscator.AD	142
19	Backdoor	Rbot!gen	158
20	Trojan	Skintrim.N	80
21	TDownloader	Swizzor.gen!E	132
22	TDownloader	Swizzor.gen!I	128
23	Worm	VB.AT	408
24	TDownloader	Wintrim.BX	97
25	Worm	Yuner.A	800

Table 1: MalIMG Dataset - Class Labels

Malware taxonomy splits a malware into various categories, or *class types*. Within the MalIMG dataset, there are the following class types:

1. Backdoor - A malware file that attemtpts to bypass an encrypted level of a computer product, or the authentication level[4]
2. Dialer - A malware that connects systems to phone numbers with the goal of fraud[5]
3. PWS - A type of Trojan[6]
4. Rogue - A malware that immitates a malware removal software, but instead installs malware[7]

[4]https://en.wikipedia.org/wiki/Backdoor_(computing)
[5]https://blog.malwarebytes.com/glossary/dialer/
[6]https://www.f-secure.com/v-descs/trojan-pws.shtml
[7]https://en.wikipedia.org/wiki/Rogue_security_software

5. TDownloader - Another type of Trojan[8]
6. Trojan - Usually concealed or hidden as safe or legitimate software, Trojan malware aim to get unauthorised access to a system to which they are downloaded[9]
7. Worm - A standalone malware with the intent to reproduce onto other computers and networks[10]

However, currently, there is no standard taxonomy of *family type* agreed amongst cybersecurity firms in the industry (Ucci et al, 2018), which could lead to a limitation once a machine learning, or deep learning model gets deployed to a live software[11]. Nevertheless, this project will classify the family types present in this dataset as new malware types tend to be closesly related to existing malware family types.

3.2.1 Data Pre-processing

Transforming binary code into images

To be able to utilise labelled malware data as a training input, the raw data contained in each malware file needs to be converted into an image. The authors of the MalIMG data process each malware binary file as a vector of 8-bit unsigned integers, which is then organized into a 2D array (Nataraj et al, 2011). This paper suggests recommended image widths for different file sizes and for files between 10kb - 30kb, a width of 64 is recommended. Though larger height and width combinations maybe allow for more features to be visibly represented, there is a trade-off for the computational power needed. Therefore, all 2D arrays are processed as an image with three channels and re-sized to 64 x 64 and visualised using the ImageDataGenerator module from the Tensorflow library[12].

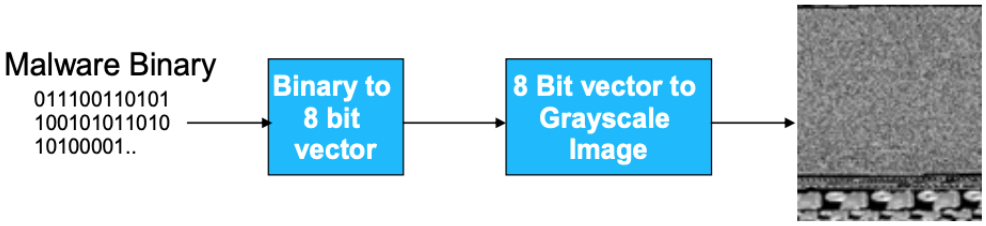

Figure 2: Process for malware images (Nataraj et al, 2011)

[8]https://blog.malwarebytes.com/detections/trojan-downloader/
[9]https://www.kaspersky.co.uk/resource-center/threats/trojans
[10]https://en.wikipedia.org/wiki/Computer_worm
[11]Discussed in Section 5.1 - Future Development
[12]https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator

Figure 3 provides a random selection of some of the processed images and interestingly, different sections of each image exhibit distinctive image features.

Figure 3: Sample images of the MalIMG Dataset

Gibert et al (2020) outline in a schematic diagram the traditional structure of a portable executable (PE) file (Figure 4) showing the characteristics of a PE file[13]. Conti et al (2010) demonstrate that visual analysis of binary data helps to distinguish structurally different regions of data, as would be typically found on a malware executable file, which allows an easier facilitation of malware file type classification. An example in figure 5 shows the text, data and resource sections of a Trojan file. As argued by Nataraj et al (2011), malware samples from different families exhibit different visual characteristics, which can be evidenced from the example in in Figure 5.

[13]Full details of each section can be found here: https://docs.microsoft.com/en-gb/en-_us/windows/win32/debug/pe-_format

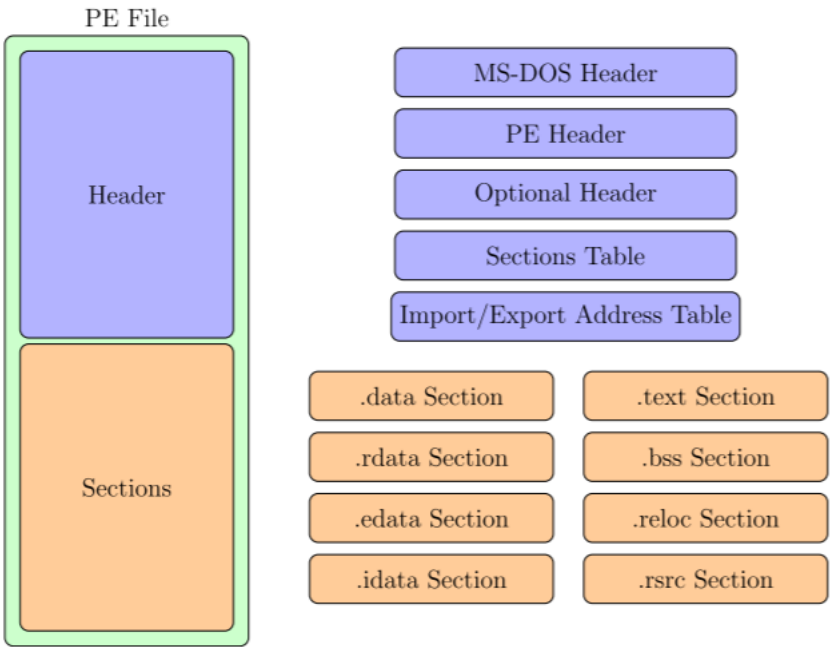

Figure 4: Portable Executable File Format (Gibert et al, 2020)

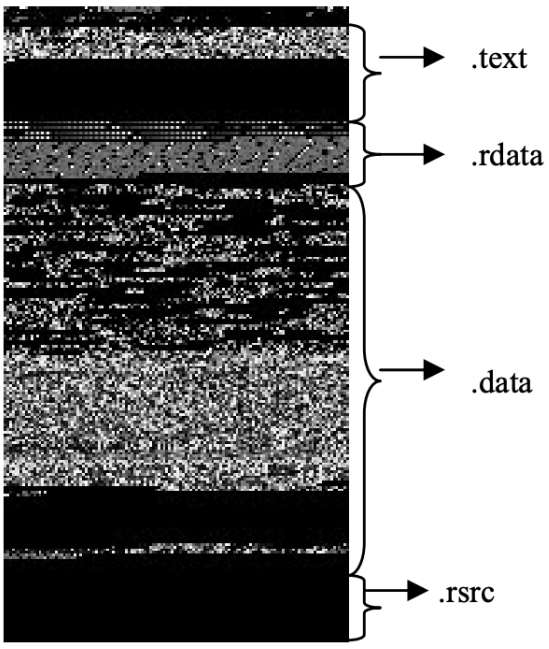

Figure 5: Sections of a Trojan file (Nataraj et al, 2011)

3.2.2 Unbalanced Classes in Dataset

From figure 6, we can see the dataset itself is highly unbalanced, where over 40% of the malware images are labelled with the Allaple.A and Allaple.L families

which will turn this task into one of an imbalanced classification. Generally, models generated by directly training on an imbalanced dataset result in a poor classification accuracy (Mitsuhashi et al, 2020). The dataset is pre-processed so that the configuration does not result in any biases due to class during the training phase. Such that the proposed models do not encounter the "Accuracy Paradox"[14] and that the class imbalance does not affect the accuracy scores, the "class_weight" parameter in the scikit-learn library is utilised to give higher weight to minority classes and lower weight to majority classes.

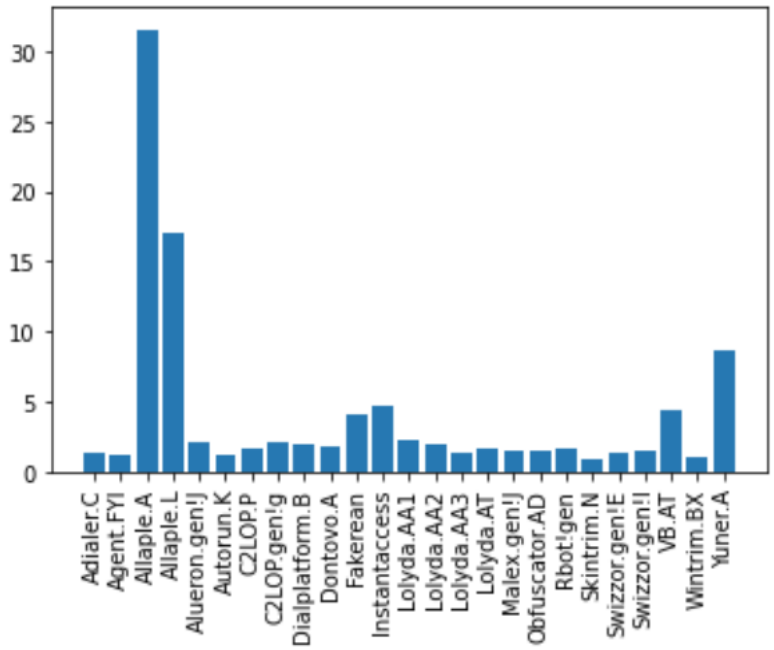

Figure 6: MalIMG Dataset - Unbalanced Classes (% split of dataset)

3.2.3 Training & Testing Datasets

In order to train, test and evaluate the deep learning model, the dataset is split into a testing and training subset. Firstly, 80% of the dataset was used to fit our model and 20% was left for testing the trained model. As discussed in 3.2.2, a data-level approach is used to approach the nature of an imbalanced dataset prior to splitting the dataset. Moreover, given the nature of Capsule Neural Networks and their output of a vector, rather than a scalar, the epoch size must be a common multiple of both the training and test subsets. Given the number of samples in the MalIMG dataset does not allow for an epoch size that is a least

[14]https://en.wikipedia.org/wiki/Accuracy_paradox

common multiple of both the training and test set, the dataset is under-sampled to allow for this, rather than stick with the 80/20 split. Similar to the rationale taken by Mitsuhashi et al (2020), the dataset for this project is randomly under-sampled with 7,004 samples used for the training data and 1,133 samples used for the test data in all final experiments. Furthermore, in this project, no data augmentation is adopted.

3.3 Theoretical Frameworks

In order to detail the architecture implementation and experimental setup, this subsection will outline the theoretical underpinning of the algorithms used in prior work and the justifications why the proposed models are used to overcome their limitations. We assume that the reader of this paper has a baseline understanding of machine learning concepts such as artificial neural networks, perceptrons, fully-connected layers and model training through backpropagation.

3.3.1 Convolutional Neural Networks

There have been major successes in the field of computer vision across multiple domains. Image classification tasks with the use of deep learning architectures such as Convolutional Neural Networks (CNNs) have had remarkably low test errors across a variety of domains (Mukhometzianov et al, 2018). The latest development in this field was the introduction of Capsule Neural Networks (CapsNets) by Sabour et al (2017) as an alternative model for image recognition tasks to overcome limitations from CNNs.

A CNN is a deep learning algorithm which is inspired from the biology of the human brain. The algorithm aims to replicate the connectivity of brain neurons and the complex cells in the primary visual cortex (Lindsay, G. 2020). A CNN is trained by taking an input pattern and assigns weights and biases to specific features present in this pattern (or image). A traditional CNN is built on an input layer, a number of hidden layers and an output layer. These hidden layers are composed of (1) Convolutional layers, (2) activation functions (e.g. a Rectified Linear Unit, or ReLU), (3) pooling layers and (4) fully-connected layers (Ayaz, 2018). The convolution layers read input patterns to extract features via an activation function (which is applied to the convolutional layer output in order to introduce non-linearity into the model as well as reducing model complexity (ibid, 2018).

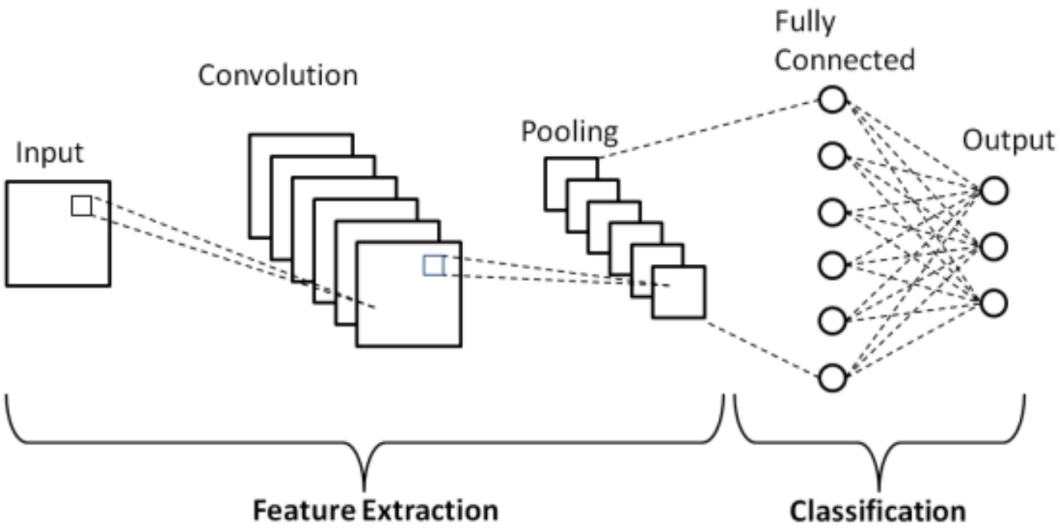

Figure 7: Convolutional Neural Network

.

Convolutional Layers & Activation Functions

Convolutional layers are fundamental layers in a CNN that are usually used for image classification with the objective to extract the high-level features of the input image. These layers are also utilised in Capsule Network architectures (See section 3.3.2). The main parameters defining a baseline convolutional layer are:

1. Number of filters
2. Filter size
3. Step Size (or stride) of the filter

Neurons in CNNs are connected to a small region subset of the input image and exploit spatially local correlations. A learnable filter is extended over the input, starting at the top left and then extended over the depth of input image. The initial convolution layer will extract low-level features of an image and the more layers that are added, i.e. how deep the network is, the more features that will be extracted. A filter consists of the layer of connection weights with the input being a two-dimensional part of the input pattern and the output being a single scalar (this will differ for a Capsule Network). The step size (or stride) determines whether a filter gets convolved throughout all permutations of the image. The stride number determines how many convolutions will be avoided, for example a stride of 2 will result in half of the convolutions being avoided.

An activation function is an important part of the CNN algorithm and is a

function that occurs after most layers of a neural network. It defines the output of a node given the inputs. The main aim of an activation function is to determine whether a node should or should not be activated through a fixed mathematical operation on the scalar output. In this project, Rectified Linear Unit, or ReLU, will be utilised due to its ability to accelerate convergence in artificial neural networks and its simplicity of being a threshold[15].

$$f(x) = max(0, x)$$

Pooling

A pooling layer is an operation within a CNN that is added after the convolutional layer (and successive layers) in order to reduce the potential of overfitting. Pooling works by reducing the spatial size of the representation in order to decrease the number of parameters[16]. The three main types of pooling are Min, Max and Average, where either the minimum, maximum or average pixel values of the batch are selected respectively. The batch size is equivalent to the filter size. MaxPooling is the most common out of the three, extracting important, or brighter pixel features (e.g edges) in each region of the feature map, whereas average pooling takes into account all parts of the convolutional output prior to transferring to the next layer of the network.

One fundamental feature of a CNN is that they acquire a new set of features from each layer of the architecture given an input pattern. Moreover, any spatial or positional information with how the features are positioned tend to be discarded in these pooling layers. This project proposes to implement a novel capsule neural network, where unlike utilising a pooling layer, capsules have the property of equivariance. Formally, capsules allow a commutative property. Spatial information is not lost as a capsule network accepts inputs and outputs vectors instead of a scalar value in the CNN model and is a key justification to overcome limitations when using deep learning for malware classification.

3.3.2 Capsule Neural Networks

Capsule Neural Networks (CapsNets) are the latest development in the field of deep learning architectures, introduced by Sabour et al (2017) to overcome limitations from the traditional CNN approach. A capsule is a group of neurons that store different features or information about the input pattern or image

[15]Available online at Stanford University CS231 Course Notes - https://github.com/cs231n
[16]Available at: https://cs231n.github.io/convolutional-networks/pool

it is aiming to classify (Intel, 2018)[17]. Capsules exist at a higher-dimensional vector space, where each dimension would represent certain information about an image's position, rotation or other notable features. The Capsule Networks aim to replicate the process of how our brain processes vision, known as *inverse graphics*. Humans deconstruct an image into a number of hierarchical sub-parts (ibid, 2018) and a relationship between these sub-parts is constructed in order to recognise familiar objects. This is the main approach that capsules try to emulate. Capsules are placed in every part of the image, and the capsule output denotes whether or not that sub-part of the image is located in that position.

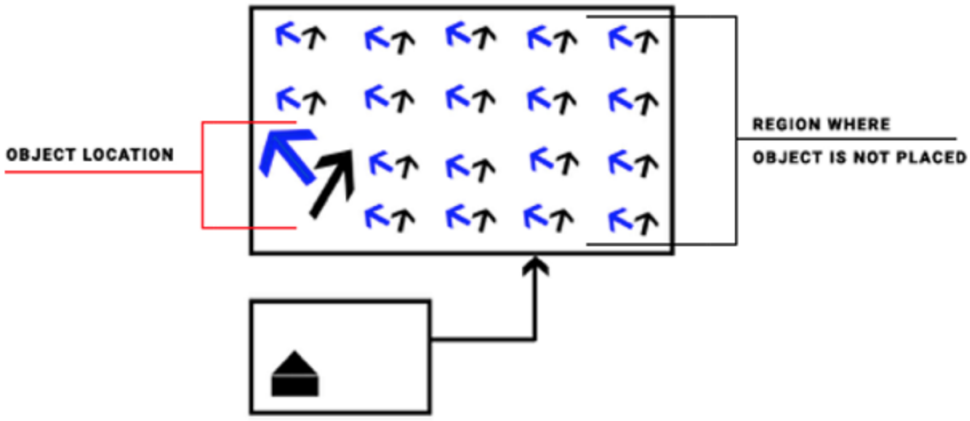

Figure 8: The location where the rectangle and triangle are placed, results in a larger vector output (Intel, 2018)

Capsule Networks aim to overcome the two key disadvantages of CNNs. Firstly, they fail to consider spatial hierarchies between features, i.e. the spatial orientation of features extracted from the input patterns are disregarded. Figure 9 shows an illustration of the "Picasso Problem": The image on the right is not a face, despite having eyes, nose and a mouth. CNNs only look for features and disregard the pose and the positioning of such features - this type of model would not know the difference between the two faces in this figure.

[17]Available at: https://software.intel.com/content/www/us/en/develop/articles/understanding-capsule-network-architecture.html

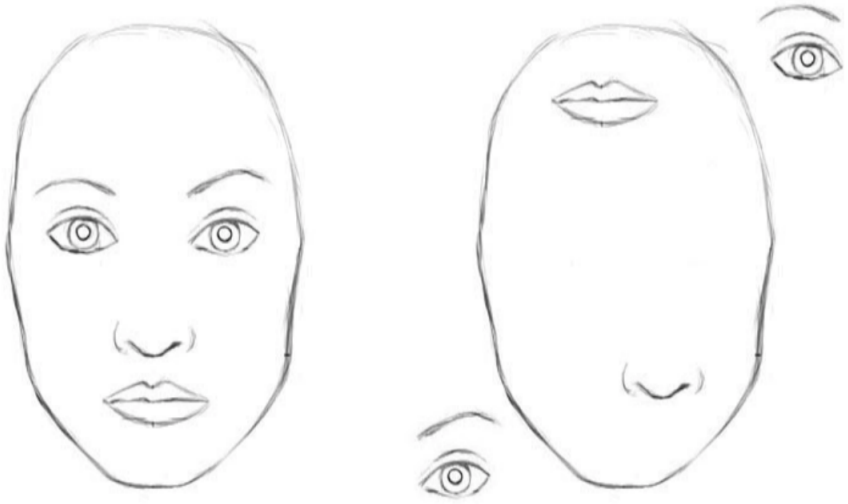

Figure 9: The "Picasso Problem": A Convolutional Neural Network would see these two faces as identical (Tarrasse (2018))

.

CNNs also lack rotational invariance. One of the primary justifications to why a CapsNet *theoretically* should perform better than a CNN specifically for the domain of malware classification, is that malware binaries represented as images have clear sections (as shown in Fig. 4). In other words, the proposed model should allow for better hierarchical relationships, which is important in a dataset that is compromised of images with set sections. To justify a CapsNet for this project, the use of dynamic routing and reconstruction regularisation, over the use of MaxPooling layers will result in a rotationally invariant model which takes into consideration spatial awareness within a test dataset. The hypothesis then would be that the model's ability to reduce the number of false positives and false negatives whilst classifying.

CapsNets architectures typically have convolutional layers similar to CNNs but introduce an alternative to the pooling operation through the use of capsules, where layer-based "squashing" and dynamic routing are used. The capsule layer typically is the last layer in the network. After a capsule layer, their entire output is squashed as an entire vector rather than focusing on a scalar-output like CNNs. By introducing these two features of a capsule, the limitations regarding losing spatial information is solved.

Architectural Design

There are three key sections of a Capsule Neural Network (Sabour et al, 2017):

1. Primary Capsules
2. Higher Level Capsules
3. Loss Calculation

Primary Capsules

In the first primary capsules process, input images are processes in three steps: (1) convolutional layers, (2) reshape function and (3) squash function. Similarly to CNNs, the convolutional layers output a feature map. This is followed by a reshaping function which takes a feature map and reshapes into a number of vectors of a specified dimension. In the original baseline model (Sabour et al, 2017), a capsule learns the right transformations through the following formal summary of steps below[18]:

1. Convolutional layers are implemented to produce a feature map.
2. "Prediction" vectors $\hat{\mathbf{u}}_{j|i}$ - representing the features and probabilities of recognition - are inputted into to all, but the first layer.
3. Apply a transformation matrix (or a matrix multiplication, $\mathbf{W}_{ij}$, to the previous layer's vector which represents important spatial information between low-level and high-level features (Misra, 2019): $\hat{\mathbf{u}}_{j|i} = \mathbf{W}_{ij}\mathbf{u}_i$. For example, if we had a $(n * m)$ matrix, the transformation would be from u_i to $\hat{u}_{j|i}$ from the m-dimension to the n-dimension (Hui, 2017)[19] where $(n \times m) \times (m \times 1) = (n \times 1)$.
4. Finally, calculate a weighted sum of the input vectors via a dynamic routing algorithm as described in Hinton et al (2017) in order to determine output location: $\mathbf{s}_j = \sum c_{ij}\hat{\mathbf{u}}_{j|i_i}$, where c_{ij} are coupling coefficients that are calculated by dynamic routing.
5. After the above four steps, the length of a CapsNet output vector, $\mathbf{v}_j$ represents the probability of a feature being present in the current input. A new activation function is proposed: a non-linear "squashing" function is used to transform "short vectors to almost zero length and long vectors to a length below 1" (Sabour et al, 2017) prior to sending to the next layer.

$$\mathbf{v}_j = \frac{\left\|\mathbf{s}_j\right\|^2}{1+\left\|\mathbf{s}_j\right\|}\frac{\mathbf{s}_j}{\left\|\mathbf{s}_j\right\|}$$

[18] As outlined in my project proposal
[19] https://jhui.github.io/2017/11/03/Dynamic-Routing-Between-Capsules/

In other words, the squashing function allows the length of each vector to represent the *probability* that a certain feature or object is present in the respective location. The squashing function, $\mathbf{v}_j$, ensures positional information that is present in higher dimensions is preserved (Intel, 2018).

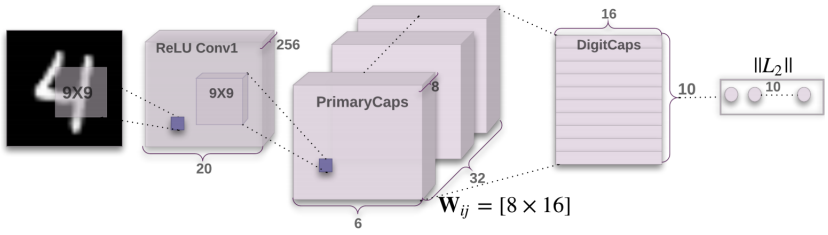

Figure 10: A baseline 3-layer CapsNet Architecture used for the MNIST dataset (Sabour et al, 2017)

Higher Level Capsules

The final stage of the CapsNet is the highest level of capsule layer[20]. Capsules from the primary layer would route to this layer using *dynamic routing* (see algorithm outlined below in Figure 11) which is an algorithm that groups capsules into this parent capsule. In the architecture described by Sabour et al (2017), 16-dimensional vectors are outputted which contain the instantiation parameters for the decoder, in order to reconstruct the object (ibid, 2017).

Procedure 1 Routing algorithm.

1: **procedure** ROUTING($\hat{\mathbf{u}}_{j|i}$, r, l)
2: for all capsule i in layer l and capsule j in layer $(l+1)$: $b_{ij} \leftarrow 0$.
3: **for** r iterations **do**
4: for all capsule i in layer l: $\mathbf{c}_i \leftarrow \texttt{softmax}(\mathbf{b}_i)$
5: for all capsule j in layer $(l+1)$: $\mathbf{s}_j \leftarrow \sum_i c_{ij}\hat{\mathbf{u}}_{j|i}$
6: for all capsule j in layer $(l+1)$: $\mathbf{v}_j \leftarrow \texttt{squash}(\mathbf{s}_j)$
7: for all capsule i in layer l and capsule j in layer $(l+1)$: $b_{ij} \leftarrow b_{ij} + \hat{\mathbf{u}}_{j|i}.\mathbf{v}_j$
return $\mathbf{v}_j$

Figure 11: Routing by agreement (Sabour et al, 2017)

The intuition behind this algorithm is that by routing a capsule to another capsule in a higher level based on its relevancy, the likeliness of every feature property in a given image is taken into account. Therefore, if the similarity is low, the neuron is not activated. This is known as *routing-by-agreement*. The vector $(\hat{u})_{j|i}$ is a prediction vector of capsule i based on the output of capsule j above (Hui, 2017). If an activity vectors (s_j) has close similarity to a prediction vector, both capsules are arguably highly related, and should create the parent vector.

[20]Known as the DigitCaps in the original paper.

Similarity is measured using the scalar product of the prediction and the activity vector. (Ibid, 2017).

$$b_{ij} \leftarrow \hat{u}_{j|i}\dot{v}_j$$

For CapsNets, an iterative dynamic routing is completed by computing the capsules output using a coupling coefficient as an intermediate value c_{ij}, which mathematically is equivalent to the softmax of the similarity score, i.e.:

$$c_{ij} = \frac{\exp b_{ij}}{\Sigma_k \exp b_{ik}}$$

Evaluation: CapsNet Loss Function

For a Capsule Neural Network, a specific loss function named Margin Loss is used. This is used over traditional neural network loss functions such as mean squared error (MSE). A separate margin loss, L_c, is calculated for each class present in an image:

$$L_c = T_c \max(0, m^+ - ||v_c||)^2 + \lambda(1 - T_c)\max(0, ||v_c|| - m^-)^2$$

The original paper sets the ideal parameters based on experiments as $m^+ = 0.9$ and $m^- = 0.1$ and are kept as constant for this project. The lambda coefficient λ prevents initial learning from reducing the size of activity vectors for classes and is set at 0.5. The total loss sums the total loss of all classes. If an object of a specific class, c, is present then $T_c = 1$ (Hui, 2017)[21].

3.4 Classification Approach & Experiment Implementation

The following configurations will outline the implementation and experimentation strategy for the proposed dataset.

3.4.1 CNN & CapsNet Models - Hyperparameter Configuration

The objective of the models is to correctly classify the 25 classes of family types in the MalIMG dataset. In order to compare the accuracy and effectiveness of different deep learning models for malware classification, the first experiment is to implement a simple Convolutional Neural Network with the second experiment to optimise a number of hyperparmeters in this CNN. Alongside the CNN

[21] All implementations of the capsule layers are available in this project's GitHub repository.

are two CapsNets architectures. Firstly, the same architecture proposed in the original paper (Sabour et al, 2017) is used to test the dataset and secondly, a different architecture is proposed with multiple convolutional layers in order to combine the experiment with the second CNN. For all of our experiments, a seed is set for reproducibility and the train/test split is described in section 3.2.3. An objective of this project is to also show whether a capsule network would provide a better accuracy rate than a convolutional neural network. Despite some of the impressive accuracy rates (as discussed in Section 2) achieved in previous work with this dataset, in order for a classification model to be useful and applicable to resource constrained devices, this project aims to replicate these results with less computationally-intensive architectures.

Experiment 1

One of the simplest CNN architectures is the LeNet-5 (LeCunn, 1998) which consists of two convolutional layers and three fully-connected layers. This project follows an initial hyperparameter setup as suggested by Mallet (2020) due to its experimental performance for malware datasets, despite a relatively simple architecture which is based of the LeNet-5. The first convolutional layer has 30 filters, with a 3*3 kernel size and the second has a reduced 15 filters both followed by a MaxPooling layer of size 2*2 respectively. This is followed by a fully-connected layer of 128 neurons, 50 neurons and the final fully-connected layer of 25 neurons with a softmax activation function. For regularisation tests, a dropout layer is added in a second CNN implementation prior to the fully-connected layers being added of 25% and another of 50% after the first fully-connected layer in order to avoid overfitting (Brownlee, 2013). Both use ReLU activation.

Experiment 1 - Baseline Architecture
30 Conv, 3 x 3 2x2 Max Pooling
15 Conv, 3x3 2x2 Max Pooling
25% Dropout Flatten
Dense, 128 nodes 50% Dropout
Dense, 50 nodes Dense, 25 class neurons with Softmax

Table 2: Experiment 1 - Model Configuration

Experiment 2 - Random Search

There are a number of algorithms to iterate through various trials of different hyper-parameter configurations. Berstra & Bengio (2012) show through their empirical experiments that trials searched and tested randomly were more efficient for optimizing hyper-parameters over Grid Search or a Manual Search. To improve the CNN architecture in experiment 1, a Random Search is used to run through the number of neurons in the first convolutional layer as well as the second convolutional layer, the optimal epoch number and the optimal batch size. The random trial experiments were run over the following configurations over 200 epochs with the parameter configurations shown in Table 3.[22]

Random Search Experiments - Parameter Configurations	
Neurons - Conv Layer 1	[1, 5, 10, 15, 20, 25, 30, 100]
Neurons - Conv Layer 2	[1, 5, 10, 15, 20, 25, 30, 100]
Batch Size	[100, 200, 1000, 5000]
Epochs	[10, 50, 200]

Table 3: Experiment 2 - Random Search Parameters

The optimised hyperparameters that produced the best accuracy of 92.4% in the random search experimental trials were 25 filters with a 3*3 kernel size for convolutional layer 1, 5 filters with a 3*3 filter size for convolutional layer 2, followed by a fully-connected layer of 128 neurons and then 50 neurons. Identical dropout layers for regularisation are also used as implemented in experiment 1. The ideal batch size is also 200. These will also form the ideal hyperparameters as per the experiments for the convolutional layers in Experiment 4. The schematic of this CNN are as follows in the below table:

Experiment 2 - Optimized Hyperparameters
25 Conv, 3 x 3 2x2 Max Pooling
5 Conv, 3x3 2x2 Max Pooling
25% Dropout Flatten
Dense, 128 nodes 50% Dropout
Dense, 50 nodes Dense, 25 class neurons with Softmax

Table 4: Experiment 2 - Model Configuration

[22]The top 11 results of the Random Search Experiment are available in Appendix I.

Experiment 3 - Baseline Capsule Network

Two CapsNet architectures are also implemented. Firstly, the baseline CapsNet model as presented by Sabour et al (2017) is implemented (as detailed in Fig. 4). The architecture also has two convolutional layers, but only one fully-connected layer. The first convolutional layer has 256 filters with 9*9 kernel size, with a stride of 1 and uses a ReLU activation, which outputs the local features as inputs into the capsule layers. The primary capsules (PrimaryCaps) correspond to "inverting the rendering process" and the secondary capsules are a convolutional capsule layer with "32 channels of convolutional 8D capsules with a 9*9 kernel size*. Ibid (2017) state the capsule output sees the outputs of all 256*81 Convolutional layer, whose "fields overlap with the location of the center of the capsule".

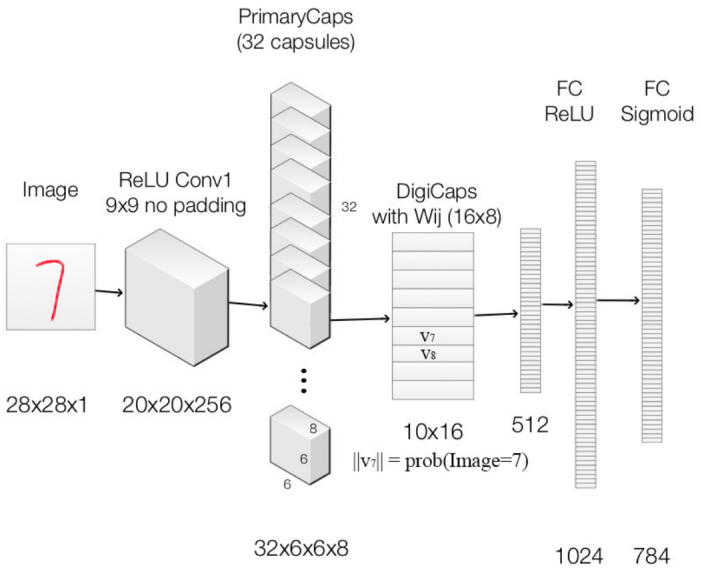

Figure 12: The original capsnet architecture (Hui, 2017)

3.4.2 Modifying the CapsNet Model

Experiment 4 - Malware Capsule Network

After implementing the baseline CapsNet model as proposed by Sabour et al (2017), another candidate model is implemented with the aim to improve classification accuracy, whilst reducing the margin loss as described in the theoretical framework section. The issue with the baseline CapsNet is that the architecture was built specifically for the benchmark MNIST and CIFAR-10 datasets, and the capsules now must be tailored specifically for the Malimg dataset.

This proposed CapsNet architecture increases the number of convolutional layers in an attempt to learn more features. Building upon experiment two, the proposed model chooses to double the amount convolutional layers prior to the capsule layer, as suggested in literature that the more layers the model has, the more features the model can extract[23]. This approach follows a similar methodology as Cayir et al (2020) as discussed in the prior work. Mathematically, a capsule neural network is far more complex of a structure due to its ability to input *and* output vectors over scalars[24]. The advantage of not adding further convolutional layers is that the training time should be minimised.

The first and second convolutional layers have 3*3 kernel size and 25 filters and the third and fourth have 5 filters and a 3*3 kernel size, which are then reshaped to 128 feature vectors (corresponding to a similar structure in experiment 2 in order to be able to compare results). These vectors are inputted into a single capsule layer containing 25 capsules (for 25 family classes in the MalIMG dataset). The routing iteration is 3 in the capsule layer and the dimension of each capsule is 8, as suggested to be optimal in Sabour et al (2017). Other differences to the original paper is that the implementation uses a decay factor = 0.9 (for the learning rate decay) and step = 1 for the epoch, as shown to be optimal rates in experiments run by Guo, X (2017)[25]. Furthermore, due to GPU limitations on Google Colaboratory, accuracy and error testing is done after 50 epochs, rather than the 200 for the CNN experiments. That being said, visually the accuracy convergence came into place well before 50 epoch iterations (discussed in section 4). Visually, the architecture semantics can be seen in the diagram and described in the table below:

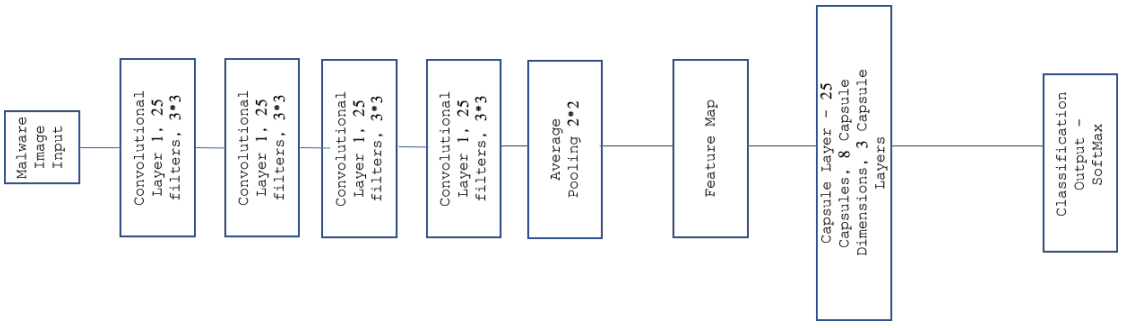

Figure 13: Experiment 4 - Malware CapsNet Schematic

[23]https://cs231n.github.io/convolutional-networks/
[24]The proposed model for experiment 4 alone has 23,572,992 total trainable parameters.
[25]Available online at: https://github.com/XifengGuo/CapsNet-Keras

Experiment 4 - Malware CapsNet Layers & Parameters
25 Conv, 3 x 3 ReLU
25 Conv, 3x3 ReLU
5 Conv, 3 x 3 ReLU
5 Conv, 3x3 ReLU Average Pooling 2x2
Feature Map
Capsule Layer - 25 Capsules 8 Capsule Dimensions 3 Capsule Layers
SoftMax

Table 5: Experiment 4 - Malware CapsNet

3.5 Extract, Transform, Load (ETL) Pipeline

With any data science project, a crucial part of the methodology is the "Extract, Transform, Load" (ETL) process, or the process by which data is acquired, processed and loaded in order to utilise. In order to implement the architecture described in the above section, the dataset is processed through Google Colaboratory with a zip file of the malware images being downloaded through the Google Drive OAuth API given the speed of access[26]. A similar cloud solution is recommended if this model is to be deployed onto a production anti-malware software.

Software and Hardware Specifications

Given the computational power required for training the proposed models is beyond the scope of available hardware, this project will carry out all experiments using a cloud-based alternative. Google Colaboratory is a Jupyter Notebook-style environment, which makes available Google's GPU TPU instances for free, allowing for faster and more time efficient training. Traditional machine learning libraries (Tensorflow, Keras) and data analytics and visualisation tools are also used (Pandas, Numpy, Scikit-learn)[27].

[26]Please see Appendix and GitHub Repository for more details on code implementation.
[27]Please see Appendix for library versions

4 Experimental Results

Unlike a conventional software engineering project, testing a machine learning or deep learning project refers to the evaluation of accuracy and prediction of the proposed models[28]. From a quality assurance perspective, model performance is evaluated through a testing set of the dataset. As discussed in Section 3, the dataset is split into a training subset and all models are tested on a testing subset of 1,133 images. Experiments 1 and 2 described in section 3.4 are evaluated in a number of ways for classification models. For the two Convolutional Neural Network implementations, the commonly used method of a confusion matrix are produced and also related measures of efficiency and effectiveness are calculated as relayed below:

- Precision = $\frac{TP}{(TP+FP)}$
- Recall = $\frac{TP}{(TP+FN)}$
- Accuracy = $\frac{TP}{(TP+FP+TN+FN)}$
- F1 = $\frac{2*TP}{2*TP+FN+FP}$
- MCC = $\frac{(TP*TN)-(FP*FN)}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$

Precision is a measurement to determine of those predicted as a certain class, how many are actually relevant (i.e. true positives / actual results) and Recall refers to the proportion of relevant results classified by the model. Chicco et al (2020) and Boughrbel (2017) also have shown that the Matthews Correlation Coefficient (MCC) is an optimized performance metric that is designed to deal well with evaluation of data imbalance and overcome issues that accuracy may lead to poor generalization results given it may predict the largest size class. For experiments 3 and 4 on the CapsNet model, the main evaluation metrics discussed are the minimisation of the loss function (margin loss) and the accuracy rate. As mentioned in section 3, the data-level approaches of correcting for class imbalance help to over-sample the minority classes and under-sample the majority classes, similarly to Mitsuhasi et al (2020). Training time will also be analysed for the experiments.

4.1 Experiment 1 Results

The first convolutional neural network as describe was initially used to classify the malware images which resulted in the model's accuracy of 89.6%. The fol-

[28]Blackbox testing for neural network models: https://dzone.com/articles/qa-blackbox-testing-for-machine-learning-models

lowing table shows summary evaluation statistics for the test dataset on the first experiment - the accuracy, precision, recall, F1 and MCC for each malware class.

Class	Recall	Precision	F1 Score	Accuracy	MCC
1	1.000	1.000	1.000	1.000	1.000
2	1.000	1.000	1.000	1.000	1.000
3	0.997	0.998	0.997	0.998	1.000
4	1.000	1.000	1.000	1.000	1.000
5	1.000	1.000	1.000	1.000	0.979
6	1.000	0.913	0.248	0.914	0.001
7	0.643	0.996	0.666	0.992	0.813
8	0.926	0.995	0.877	0.994	0.861
9	1.000	1.000	1.000	1.000	1.000
10	1.000	1.000	1.000	1.000	1.000
11	1.000	1.000	1.000	1.000	1.000
12	1.000	1.000	1.000	1.000	1.000
13	1.000	0.999	0.971	0.999	0.971
14	0.941	1.000	0.970	0.999	0.970
15	1.000	1.000	1.000	1.000	0.973
16	1.000	1.000	1.000	1.000	1.000
17	1.000	0.999	0.889	0.999	1.000
18	1.000	1.000	1.000	1.000	1.000
19	1.000	1.000	1.000	1.000	1.000
20	1.000	1.000	1.000	1.000	0.638
21	0.571	0.996	0.615	0.991	0.452
22	0.545	0.996	0.545	0.991	0.990
23	1.000	1.000	1.000	1.000	1.000
24	0.923	1.000	0.960	0.999	1.000
25	0.001	1.000	0.001	0.914	0.919
Averages	0.901	0.996	0.830	0.952	0.892

Table 6: Experiment 1 Results.

The summary statistics presented in Table 1 show initially some promising results despite the simplicity of the model. 14 out of the 25 classes showed near perfect accuracy rates. The per-class recall shows that 17 out of 25 classes shows the proportion of positive results were identified correctly, with classes 7, 21 and 22 showing just over half were calculated correctly. Per-class precision shows nearly all classes show that the correctly classified inputs were correct, with class 6 showing a 91.3% precision. The per-class MCC scores consider the proportion of each class in the confusion matrix and only scores highly if the classifier performs well on positive elements, as well as negative ones[29]. The generalised MCC score shows a result of 89.2%.

[29]https://en.wikipedia.org/wiki/Matthews_correlation_coefficient

One notable advantage of a simple, modified LeNet-5 architecture is the time taken to train the model with this dataset takes 3 minutes, 21 seconds over 200 epochs, which if applied to devices which may have lower memory availability may present a benefit over any gains in accuracy from a more complex model. Nevertheless, the following experiments aim to optimize this accuracy rate by experimenting with hyperparameter optimization as well as adding in Capsule Layers.

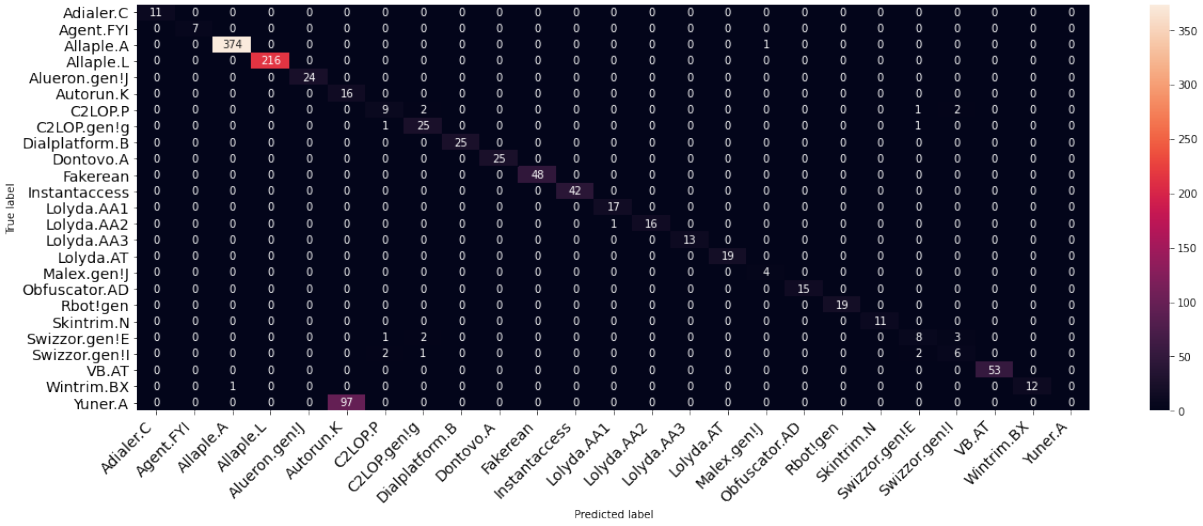

Figure 14: Experiment 1 - Confusion Matrix

4.2 Experiment 2 Results - Hyperparameter Optimization

To optimize the hyperparameters in the convolutional layers prior to implementing with a capsule neural network, a Random Search is performed on the initial CNN design as done in the first experiment. Each training round went through 200 epochs and randomly searched through the following subsets of hyperparameters as described in Section 3.

The best accuracy was 91.1% in training with the following subset of hyperparameters: Conv1 Layer = 25 neurons, Conv2 Layer = 5 neurons, Epochs = 200, batch_size = 200. The summary statistics are presented in table 7 and the training loss and accuracy charts are below:

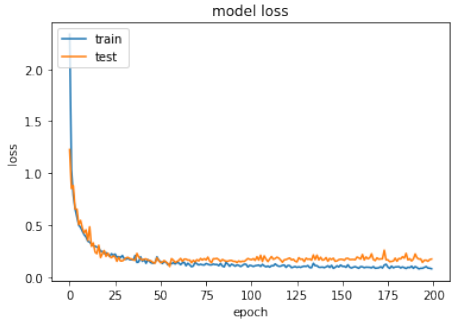

Figure 15: Experiment 2 - Model Accuracy

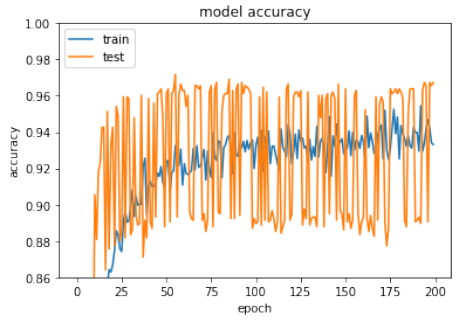

Figure 16: Experiment 2 - Model Loss

Class	Recall	Precision	F1 Score	Accuracy
1	0.000	0.999	0.000	0.989
2	0.000	0.931	0.000	0.923
3	0.000	1.000	0.000	0.669
4	0.028	0.928	0.054	0.756
5	0.083	0.762	0.150	0.748
6	0.000	1.000	0.000	0.985
7	0.429	0.807	0.000	0.803
8	0.000	1.000	0.000	0.976
9	0.000	1.000	0.363	0.978
10	0.000	0.980	0.000	0.958
11	0.229	0.878	0.000	0.851
12	0.000	1.000	0.000	0.963
13	0.000	1.000	0.000	0.985
14	0.000	0.987	0.000	0.972
15	0.000	1.000	0.000	0.989
16	0.000	1.000	0.000	0.983
17	0.000	1.000	0.000	0.996
18	0.000	0.979	0.000	0.966
19	0.000	1.000	0.000	0.983
20	0.000	1.000	0.000	0.990
21	0.000	1.000	0.000	0.988
22	0.000	1.000	0.000	0.990
23	0.113	0.735	0.196	0.710
24	0.000	1.000	0.000	0.989
25	0.000	1.000	0.000	0.914
Average	0.0352	0.920	0.0305	0.883

Table 7: Experiment 2 Results.

The summary statistics on table 2 show some excellent and more realistic results as an improvement on some per-class accuracy results. The top-1 accuracy rate was on class 17 with a 99.6% accuracy, with a 100% per-class precision. During

the iterations with epochs, the model loss decreased over time during training but was slightly higher in the testing phase. Similarly with accuracy, average model accuracy fluctuated between 92 to 96%, whereas the testing phase saw the average accuracy fluctuate between 88% and 96% during convergence. Interestingly, there has been a large Recall-Precision trade off in this experiment. The per-class precision shows that the number of positive results correctly classified were a large portion of the per-class positive results, however poor recall rates suggest that this is a much lower portion of all relevant results.

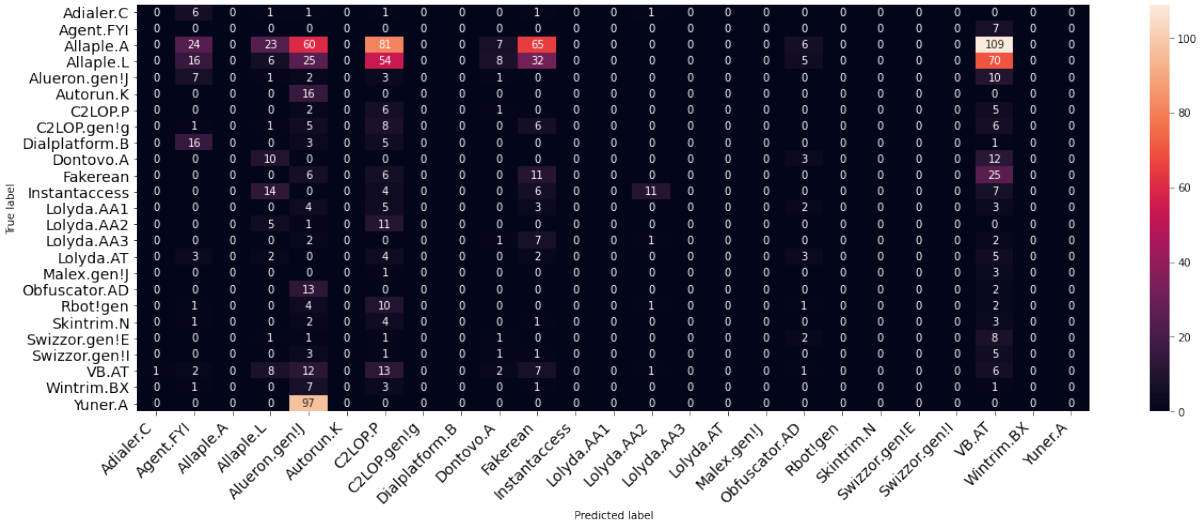

Figure 17: Experiment 2 - Confusion Matrix

4.3 Experiment 3 & 4 Results - Capsule Neural Networks

The first of the two CapsNet implementations followed the Sabour et al (2017) implementation with no modifications. Naturally, the implementation as described by the authors (Ibid, 2017) was aimed to fit the MNIST and CIFAR-10 datasets, not the MalIMG dataset which has a different image input size. With this model, the CapsNet accuracy did not improve from a 4.2% accuracy in training, with a 3.5% accuracy on the test dataset, which is worse than perhaps a random classification guess. The second and more robust CapsNet implementation carried out has four convolutional layers based on the hyperparameter optimized CNN model (Experiment 2) and a final capsule network to replace the pooling layers. Due to GPU limitations, this model was iterated over 50 epochs rather than 200 and resulted in an average CapsNet training accuracy of 98.9% and a final testing accuracy of 90.9%.

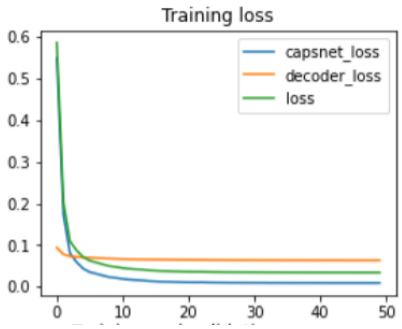

Figure 18: Experiment 4 - Model Loss

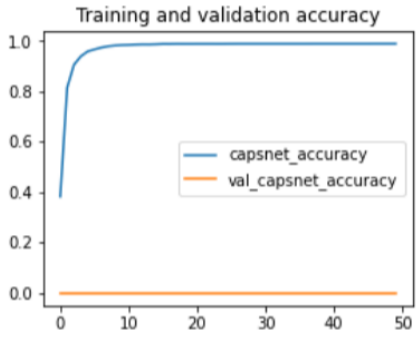

Figure 19: Experiment 4 - Model Accuracy

In Figure 18 and 19, we can see that loss decreases over epoch iterations, whilst training accuracy increases, as would be expected. This final implementation provides a 7% improvement over the hyperparameter optimized CNN model in the previous experiment. The next section will discuss these results as well as make comparisons to other models in literature, as well as evaluating the limitations.

5 Discussion & Evaluation

5.1 Comparisons between CNN & CapsNet Models

At first glance, the CapsNet in experiment 4 achieves a 98.9% training accuracy, with generalised validation accuracy on the held-out testing set achieving 90.9% which is an improvement from the generalised CNN accuracy of 88.3% in experiment 2. The comparisons between our traditional image classification method of convolutional neural networks and our novel CapsNet architecture on the Mal-IMG dataset is that the CapsNet has provided a significant gain in accuracy over a simple CNN model from the point of view of accuracy. Both the training and testing phases of the CapsNet model result in the margin loss being negligible. However, these results are not without limitations and the model would require alterations to achieve industry level efficacy. Despite a relatively commendable generalised accuracy rate, the per class accuracy rates vary and the confusion matrices presented in the previous section show that the models failed to classify some of the family classes that were a minority class.

Another consideration to make is that if these models were to be used in deployment, whether in hardware or anti-malware software, there is an argument to choose a model that may perform slightly worse in terms of accuracy if it is faster to train. The optimised CNN model took around 3 minutes 30 seconds compared to a capsule neural network of 151 minutes 40 seconds. For example, if part of the anti-malware analysis needed to be deployed into a low-memory Internet of Things (IoT) device, a less complex solution may be preferred. Table 8 presents four different architectures that have been used in prior work on the same dataset:

Architecture	**Accuracy**
GIST Feature Selection + kNN (Nataraj et al, 2011)	97.18%
M-CNN (Kalash et al, 2018)	98.52%
ResNet50 (Rezende et al, 2017)	98.62%
XCeption (Lo et al, 2019)	99.03%
Hyper-parameter Optimized CNN (This Project)	95.20%
Novel CapsNet (This project)	90.90%

Table 8: Malware Classification - Accuracy comparison table

There are a number of alternative deep learning architectures that exist in the field of malware classification. Compared to other alternative convolutional neural network models and machine learning models, our capsule neural network improves upon the majority of static and dynamic analysis approaches (Ucci et

al, 2018). However, despite our model achieving a good performance, a number of tweaks would be needed to improve its efficacy in comparison to deep convolutional neural network architectures in order to achieve state-of-the-art performance levels. The following subsection evaluates the limitations of the presented model and proposes a number of ways to improve for future development.

5.2 Future Development

Capsule Neural Networks are the latest development in the field of deep learning but a lot more research and experimentation must be developed before they can be implemented into industry-level deployment to protect against malware, or other cyber-attacks. This project successfully implemented a CapsNet and has shown some promise, but a final generalised accuracy of 91% may not be sufficient to defend against zero-day attacks if a particular malware family class goes unrecognised.

In future, there are a number of approaches to improve the proposed methodology in this paper in order to improve model effectiveness, efficiency and time taken. From an algorithm-level, a similar approach to making CNNs more efficient is to increase the number of hidden layers. Phaye et al (2018) propose a "Dense Capsule Neural Network" (DCNets) and a "Diverse Capsule Network" (DCNet++) as an adaptation of the baseline CapsNet in order to improve classification performance on complex datasets. The authors show in experimental results that the DCNet achieves a state-of-the-art performance on the benchmark MNIST dataset of 99.75% and could be considered as a candidate model for future development. Nevertheless, the current trade-offs lie in the fact that in order to train such a deep capsule network, much more computational power is required. This may not be the most ideal currently given that new malware is being created daily and that model training may not currently be fast enough to keep up with malware development, or may be too costly to deploy continuously. Nevertheless, as development of computational power improves, and the access to GPUs and other powerful virtual machines become more available, then these models may prove to beat traditional CNN architectures.

Another proposal for future work is to utilise transfer learning, or to pre-train a model prior to deep learning. Erhan et al (2010) show empirical evidence to suggest that unsupervised pre-training supports better generalization from the training set. As part of preliminary testing, this project tested autoencoder-generated features of the Malimg dataset prior to training (as discussed in the project proposal). However, the findings and results of a basic autoencoder with

regularisation on this dataset proved to do a poor job at image compression. As a result, this step was decided not to be used for the final training and is left for future development.

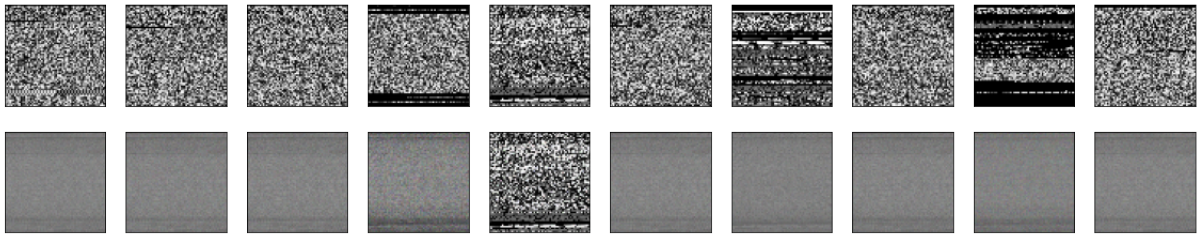

Figure 20: Preliminary Experiments - Autoencoder Generated Features example

From a data-level perspective, currently there is no agreed taxonomy of malware family classes which makes training of malware classification models less efficient (Ucci et al, 2018), especially if deployed in real-time as models have to be trained to new datasets. If cyber-security frameworks such as ISO 27001 were updated to include a new taxonomy, then a much more robust testing and validation set could be used to improve the proposed models[30]. However, for the malware family classes in the Malimg dataset, another pre-processing technique that may be to process the images in colour, rather than greyscale. Vu et al (2019) achieve a 99.14% classification accuracy on the malimg dataset by proposing a "gradient-based" colour scheme, which allows feature details to appear that do not become visually available in purely a greyscale image of that represents the bytes file. Transforming an image whereby a different byte value is a different colour may improve the feature extraction of the hidden layers in a capsule neural network during the training.

For the future, if the appropriate computational power is made available, further work on deeper capsule neural networks whilst making use of pre-processing methods such as autoencoder-generated features, or colourised images will be implemented. Furthermore, investigation into ETL processes to allow real-time data to be trained using this method will also be carried out.

[30]https://www.itgovernance.co.uk/iso27001

6 Concluding remarks

This project report presents the implementation of a novel deep learning approach to the task of malware classification as well as a comparison to traditional deep learning approaches. This was achieved through processing a large number of malware bytes files as a vector and visualising as images. The images in turn were used as input into a Capsule Neural Network architecture, making the task of malware classification analogous to image classification rather than traditional static or dynamic approaches. The Capsule Neural Network model proposed is of one of the early implementations of a CapsNet in the field of malware classification being the second paper to use this method. One of the key benefits of the CapsNet model over traditional deep learning models (e.g. CNNs) is that it takes into account the spatial information which is usually lost in a pooling layer of a CNN.

The ability to utilise deep learning models meant that heavy feature extraction or domain expertise in malware classes was not required. As a result, the proposed model could prove useful in a world where thousands of new malware files are developed daily. The experimental results show a 91% testing accuracy which outperformed a number of other papers in literature utilising traditional machine learning and deep learning approaches. Moreover, the dataset used in this project is highly imbalanced and the class weights module in the scikit-learn library is used to deal with the imbalance, and as suggested in literature, CapsNet models are the best in deep learning to overcome the issues with an imbalanced class distribution (Cayir et al, 2020).

Nevertheless, performance can still be improved in two ways. 91% accuracy may not be sufficient performance or efficacy given the impact a malicious file may have on systems, data, network and so forth. Pre-trained models and deeper capsule network architectures are discussed as potential solutions to increase generalisation performance in future as well as different data pre-processing techniques. In future, testing on live datasets is also proposed.

7 Bibliography

[1] P. Afshar, K. N. Plataniotis and A. Mohammadi, "Capsule Networks for Brain Tumor Classification Based on MRI Images and Coarse Tumor Boundaries," ICASSP 2019 - 2019 IEEE (ICASSP) United Kingdom, 2019, pp. 1368-1372.

[2] Agarap, A.F. (2020). Implementing an Autoencoder in TensorFlow 2.0. [online] Medium. Available at: https://towardsdatascience.com/implementing-an-autoencoder-in-tensorflow-2-0-5e86126e9f7 [Accessed 20 Apr. 2020].

[3] Ahmadi, M., Ulyanov, D., Semenov, S., Trofimov, M. and Giacinto, G. (n.d.). Novel Feature Extraction, Selection and Fusion for Effective Malware Family Classification. [online] Available at: https://arxiv.org/pdf/1511.04317v2.pdf [Accessed 3 Apr. 2020].

[4] M. A. Anupama, V. Sowmya and K. P. Soman, "Breast Cancer Classification using Capsule Network with Preprocessed Histology Images," 2019 International Conference on Communication and Signal Processing (ICCSP), Chennai, India, 2019, pp. 0143-0147.

[5] AV-TEST. 2020. Malware Statistics Trends Report. [ONLINE] Available at: https://www.av-test.org/en/statistics/malware/. [Accessed 12 August 2020].

[6] Bergstra, J., Ca, J. and Ca, Y. (2012). Random Search for Hyper-Parameter Optimization Yoshua Bengio. Journal of Machine Learning Research, [online] 13, pp.281–305. Available at: https://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf [Accessed 1 Sep. 2020].

[7] Bhodia, N., Prajapati, P., Di Troia, F. and Stamp, M. (2019). Transfer Learning for Image-Based Malware Classification. arXiv:1903.11551 [cs, stat]. [online] Available at: https://arxiv.org/abs/1903.11551 [Accessed 17 Sep. 2020].

[8] Brownlee, J. (2018). A Gentle Introduction to Dropout for Regularizing Deep Neural Networks. [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/.

[9] Boughorbel, S., Jarray, F. and El-Anbari, M. (2017). Optimal classifier for imbalanced data using Matthews Correlation Coefficient metric. PLOS ONE, 12(6), p.e0177678.

[10] Boyle, T. (2019). Methods for Dealing with Imbalanced Data. [online] Medium. Available at: https://towardsdatascience.com/methods-for-dealing-with-imbalanced-data-5b761be45a18.

[11] Cayır, A., Ugurünal, H. and Dag (2020). Random CapsNet Forest Model for Imbalanced Malware Type Classification Task. [online]
Available at: https://arxiv.org/pdf/1912.10836.pdf [Accessed 17 Apr. 2020].

[12] Chicco, D. and Jurman, G. (2020). The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. BMC Genomics, 21(1).

[13] Corrales, J. 2020. Deep Learning in Cybersecurity. [ONLINE] Available at: https://www.buguroo.com/en/blog/deep-learning-in-cybersecurity-the-definitive-tool. [Accessed 20 August 2020].

[14] Conti, G., Bratus, S., Shubina, A., Lichtenberg, A., Ragsdale, R., Perez-Alemany, R., Sangster, B. and Supan, M. (2010). A Visual Study of Primitive Binary Fragment Types. [online] Available at: http://www.rumint.org/gregconti/publications/taxonomy-bh.pdf [Accessed 15 Sep. 2020].

[15] Damodaran, Anusha Di Troia, Fabio Visaggio, Corrado Aaron Austin, Thomas Stamp, Mark. (2015). A comparison of static, dynamic, and hybrid analysis for malware detection. Journal of Computer Virology and Hacking Techniques. 10.1007/s11416-015-0261-z.

[16] Das, A. (2019). Malware Classification using Machine Learning. [online] Medium. Available at: https://towardsdatascience.com/malware-classification-using-machine-learning-7c648fb1da79 [Accessed 13 Feb. 2020].

[17] Siddharth Das (2017). CNN Architectures: LeNet, AlexNet, VGG, GoogLeNet, ResNet and more. [online] Medium. Available at: https://medium.com/analytics-vidhya/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5.

[18] ENISA Europa. (2018). ENISA Threat Landscape Report 2018. [online] Available at: https://www.enisa.europa.eu/publications/enisa-threat-landscape-report-2018.

[19] Eldor, T. (2018). Capsule Neural Networks – Part 2: What is a Capsule? [online] Medium. Available at: https://towardsdatascience.com/capsule-neural-networks-part-2-what-is-a-capsule-846d5418929f [Accessed 19 Apr. 2020].

[20] Erhan, D., Ca, Y., Ca, A., Ca, P.-A., Ca, P. and Com, B. (2010). Why Does Unsupervised Pre-training Help Deep Learning? Yoshua Bengio Aaron Courville Pierre-Antoine Manzagol Pascal Vincent Samy Bengio. Journal of Machine Learning Research, [online] 11, pp.625–660. Available at: https://www.jmlr.org/papers/volume11/erhan10a/erhan10a.pdf [Accessed 27 Sep. 2020].

[21] Foster, D (2020). MSc Data Science Project Proposal.

[22] Ghouti, Lahouari Imam, Muhammad. (2020). Malware Classification Using Compact Image Features and Multiclass Support Vector Machines. IET Information Security. 10.1049/iet-ifs.2019.0189.

[23] Gibert, D. and Bejar, J. (2016). Convolutional Neural Networks for Malware Classification. [online] Available at: https://www.covert.io/research-papers/deep-learning-security/Convolutional%20Neural%20Networks%20for%20Malware%20Classification.pdf [Accessed 1 Feb 2020].

[24] Gibert, Daniel Mateu, Carles Planes, Jordi. (2020). The rise of machine learning for detection and classification of malware: Research developments, trends and challenges. Journal of Network and Computer Applications. 10.1016/j.jnca.2019.102526.

[25] Guo, Xifeng. 2020. CapsNet Implementation in Tensorflow 2.2. [ONLINE] Available at: https://github.com/XifengGuo/CapsNet-Keras. [Accessed 12 August 2020].

[26] Hassen, M. and Chan, P.K. (2017). Scalable Function Call Graph-based Malware Classification. Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy.

[27] Hubens, N. (2018). Deep inside: Autoencoders. [online] Medium. Available at: https://towardsdatascience.com/deep-inside-autoencoders-7e41f319999f [Accessed 27 Apr. 2020].

[28] Hui, J. (2017). "Understanding Dynamic Routing between Capsules (Capsule Networks)." [online] Available at: https://jhui.github.io/2017/11/03/Dynamic-Routing-Between-Capsules/ [Accessed 27 Sep. 2020].

[29] Intel. (n.d.). Understanding Capsule Network Architecture. [online] Available at: https://software.intel.com/content/www/us/en/develop/articles/understanding-capsule-network-architecture.html [Accessed 27 Sep. 2020].

[30] Jia, B. and Huang, Q. (2020). DE-CapsNet: A Diverse Enhanced Capsule Network with Disperse Dynamic Routing. Applied Sciences, 10(3), p.884.

[31] Jiang Y., Li S., Wu Y., Zou F. (2019) A Novel Image-Based Malware Classification Model Using Deep Learning. In: Gedeon T., Wong K., Lee M. (eds) Neural Information Processing. ICONIP 2019. Lecture Notes in Computer Science, vol 11954. Springer, Cham

[32] M. Kalash, M. Rochan, N. Mohammed, N. D. B. Bruce, Y. Wang and F. Iqbal, "Malware Classification with Deep Convolutional Neural Networks," 2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS), Paris, 2018, pp. 1-5, doi: 10.1109/NTMS.2018.8328749.

[33] Keras.io. (2016). Building Autoencoders in Keras. [online] Available at: https://blog.keras.io/building-autoencoders-in-keras.html.

[34] Kohlbrenner, M., Berlin, T., Hofmann, R., Ahmmed, S. and Kashef, Y. (2017). Pre-Training CNNs Using Convolutional Autoencoders. [online] Available at: https://www.ni.tu-berlin.de/fileadmin/fg215/teaching/
nnproject/cnn_pre_trainin_paper.pdf [Accessed 17 Apr. 2020].

[35] D. Kornish, J. Geary, V. Sansing, S. Ezekiel, L. Pearlstein and L. Njilla, "Malware Classification using Deep Convolutional Neural Networks," 2018 IEEE Applied Imagery Pattern Recognition Workshop (AIPR), Washington, DC, USA, 2018, pp. 1-6.

[36] Kwabena Patrick, M., Felix Adekoya, A., Abra Mighty, A. and Edward, B.Y. (2019). Capsule Networks – A survey. Journal of King Saud University - Computer and Information Sciences.

[37] Le, Q., Boydell, O., Mac Namee, B. and Scanlon, M. (2018). Deep learning at the shallow end: Malware classification for non-domain experts. Digital Investigation, [online] 26, pp.S118–S126.
Available at: https://www.sciencedirect.com/science/article/pii/S1742287618302032 [Accessed 30 Jan. 2020].

[38] Lecun, Y., Bengio, Y. (1997). Convolutional Networks for Images, Speech, and Time-Series. [online] Available at: https://pdfs.semanticscholar.org
/563e/821bb5ea825efb56b77484f5287f08cf3753.pdf

[39] Lecun, Y., L Eon Bottou, Bengio, Y. and Patrick Haaner Abstract (2006). Gradient-Based Learning Applied to Document Recognition. PROC. OF THE IEEE. [online] Available at: http://vision.stanford.edu/cs598$_spring07/papers/Lecun98.pdf$.

[40] Lindsay, G. (2020). Convolutional Neural Networks as a Model of the Visual System: Past, Present, and Future. Journal of Cognitive Neuroscience, pp.1–15.

[41] Lo, W.W., Yang, X. and Wang, Y. (2019). An Xception Convolutional Neural Network for Malware Classification with Transfer Learning. [online] IEEE Xplore.
Available at: https://ieeexplore.ieee.org/document/8763852/ [Accessed 27 Apr. 2020].

[42] A. Makandar and A. Patrot, "Malware analysis and classification using Artificial Neural Network," 2015 International Conference on Trends in Automation, Communications and Computing Technology (I-TACT-15), Bangalore, 2015, pp. 1-6.

[43] Mallet, H. (2020). Malware Classification using Convolutional Neural Networks — Step by Step Tutorial. [online] Medium. Available at: https://towardsdatascience.com/malware-classification-using-convolutional-neural-networks-step-by-step-tutorial-a3e8d97122f [Accessed 20 Feb. 2020].

[44] Mallet, H. (2020). hugom1997/Malware_Classification. [online] GitHub. Available at: https://github.com/hugom1997/Malware_Classification [Accessed 27 Sep. 2020].

[45] Misra, A. (2019). Capsule Networks: The New Deep Learning Network. [online] Medium. Available at: https://towardsdatascience.com/capsule-networks-the-new-deep-learning-network-bd917e6818e8.

[46] Mitsuhashi, R. and Shinagawa, T. (2020). High-Accuracy Malware Classification with a Malware-Optimized Deep Learning Model. [online] Available at: https://arxiv.org/pdf/2004.05258.pdf [Accessed 17 Sep. 2020].

[47] Mukhometzianov, R. and Carrillo, J. (2018). CapsNet comparative performance evaluation for image classification. [online]
Available at: https://arxiv.org/pdf/1805.11195.pdf [Accessed 27 Apr. 2020].

[48] Nataraj, L., Karthikeyan, S., Jacob, G. and Manjunath, B.S. (2011). Malware images. Proceedings of the 8th International Symposium on Visualization for Cyber Security - VizSec '11.

[49] Parker, L.R., Yoo, P.D., Asyhari, T.A., Chermak, L., Jhi, Y. and Taha, K. (2019). DEMISe. Proceedings of the 14th International Conference on Availability, Reliability and Security - ARES '19.

[50] Pechyonkin, M. (2018). Understanding Hinton's Capsule Networks. Part I: Intuition. [online] Medium. Available at: https://medium.com/ai%C2%B3-theory-practice-business/understanding-hintons-capsule-networks-part-i-intuition-b4b559d1159b.

[51] Phaye, S.S.R., Sikka, A., Dhall, A. and Bathula, D. (2018). Dense and Diverse Capsule Networks: Making the Capsules Learn Better. arXiv:1805.04001 [cs]. [online] Available at: https://arxiv.org/abs/1805.04001 [Accessed 27 Sep. 2020].

[52] Prabhu (2018). Understanding of Convolutional Neural Network (CNN) — Deep Learning. [online] Medium. Available at: https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148.

[53] Rezende, E., Ruppert, G., Carvalho, T., Ramos, F., de Grus, P. (2017). Malicious Software Classification using Transfer Learning of ResNet-50 Deep Neural Network. [online] Available at: https://www.lasca.ic.unicamp.br/paulo/papers/
2017-ICMLA-edmar.rezende-malicious.classif.transfer.learning.ResNet-50.pdf

[54] Riter (2018). Capsule Networks As a New Approach to Image Recognition. [online] Medium. Available at: https://medium.com/@RiterApp/capsule-networks-as-a-new-approach-to-image-recognition-345d4db0831 [Accessed 19 Apr. 2020].

[55] Ronen, R., Radu, M., Feuerstein, C., Yom-Tov, E. and Ahmadi, M. (2018). Microsoft Malware Classification Challenge. arXiv:1802.10135 [cs]. [online] Available at: http://arxiv.org/abs/1802.10135 [Accessed 21 Jan 2020].

[56] Sabour, S., Frosst, N. and Hinton, G. (2017). Dynamic Routing Between Capsules. [online] Available at: http://papers.nips.cc/paper/6975-dynamic-routing-between-capsules.pdf [Accessed 27 Apr. 2020].

[57] Samarth, S., Phaye, R., Sikka, A., Dhall, A. and Bathula, D. (2018). Dense and Diverse Capsule Networks: Making the Capsules Learn Better. [online] Available at: https://arxiv.org/pdf/1805.04001.pdf [Accessed 27 Apr. 2020].

[58] Sarvam Blog. Malimg Dataset (2014). https:// sarvamblog.blogspot.com/2014/08/.

[59] Singh, T., Di Troia, F., Corrado, V.A. et al. Support vector machines and malware detection. J Comput Virol Hack Tech 12, 203–212 (2016). https://doi.org/10.1007/s11416-015-0252-0

[60] Piotr Skalski (2019). Gentle Dive into Math Behind Convolutional Neural Networks. [online] Medium. Available at: https://towardsdatascience.com/gentle-dive-into-math-behind-convolutional-neural-networks-79a07dd44cf9.

[61] Stanford University CS231 Class ntoes (cs231n.github.io.) (n.d.). CS231n Convolutional Neural Networks for Visual Recognition. [online] Available at: https://cs231n.github.io/.

[62] Statista. 2020. Security software - statistics. [ONLINE] Available at: https://www.statista.com/topics/2208/security-software/. [Accessed 12 August 2020].

[63] Stewart, M. (2020). Comprehensive Introduction to Autoencoders. [online] Medium. Available at: https://towardsdatascience.com/generating-images-with-autoencoders-77fd3a8dd368 [Accessed 20 Apr. 2020].

[64] Tarrasse, M. (2018). What is wrong with Convolutional neural networks? [online] Medium. Available at: https://towardsdatascience.com/what-is-wrong-with-convolutional-neural-networks-75c2ba8fbd6f [Accessed 27 Sep. 2020].

[65] Ucci, D., Aniello, L. and Baldoni, R. (2019). Survey of machine learning techniques for malware analysis. Computers Security, 81, pp.123–147.

[66] Vasan, D., Alazab, M., Wassan, S., Safaei, B. and Zheng, Q. (2020). Image-Based malware classification using ensemble of CNN architectures (IMCEC). Computers & Security, 92, p.101748.

[67] Vu, D.-L., Nguyen, T.-K., Nguyen, T., Nguyen, T., Massacci, F. and Phung, P. (2019). A Convolutional Transformation Network for Malware Classification. [online] Available at: https://arxiv.org/pdf/1909.07227.pdf [Accessed 1 Apr. 2020].

[68] C. Xiang, L. Zhang, Y. Tang, W. Zou and C. Xu, "MS-CapsNet: A Novel Multi-Scale Capsule Network," in IEEE Signal Processing Letters, vol. 25, no. 12, pp. 1850-1854, Dec. 2018.

[69] M. Yousefi-Azar, V. Varadharajan, L. Hamey and U. Tupakula, "Autoencoder-based feature learning for cyber security applications," 2017 International Joint Conference on Neural Networks (IJCNN), Anchorage, AK, 2017, pp. 3854-3861.

8 Appendix I - Implementation

All the implementations of the data pre-processing and code are available via the private Birkbeck GitHub repository linked to the MSc Project Report and are available online at: https://github.com/Birkbeck/msc-data-science-project-2019-20-files-dhifoster. The README file and markdown within each Python notebook will outline how to access the dataset and how to reproduce results. Six files exist in the repository (Birkbeck/msc-data-science-project-2019-20-files-dhifoster) and relate to each experiment as outlined in Section 3.

1. Experiment 1 - Data Exploration & Initial CNN
2. Experiment 2 - Hyperparameter Optimized CNN
3. Experiment 3 - Baseline CapsNet
4. Experiment 4 - Malware CapsNet
5. Preliminary Experiments - Autoencoders
6. README

9 Appendix II - Experimental Results

The best ten results (resented with their respective achieved accuracy and loss rates) of Experiment 2's Random Search results are as follows after 200 epochs[31]:

Iteration	# Neurons Conv1	# Neurons Conv2	Epochs	Batch Size	Accuracy	Loss
1	15	100	200	200	0.921508	0.028651
2	25	15	200	1000	0.915329	0.030599
3	**25**	**5**	**200**	**200**	**0.924139**	**0.025660**
4	25	25	10	5000	0.359230	0.015561
5	15	15	50	100	0.910015	0.031763
6	10	25	200	200	0.901185	0.023285
7	5	100	200	1000	0.885322	0.051289
8	20	25	50	1000	0.662068	0.090263
9	25	15	10	1000	0.626670	0.020522
10	5	30	50	1000	0.669112	0.065426

[31]Conv1 and Conv2 refer to the number of neurons in the first convoultional layer and the second convolutional layer respectively.

10 Appendix III - Software versions

The software and libraries used are detailed below, as well as the version number:

- Python 3.6.9 - Google Colaboratory Environment with GPU-processed runtime
- Tensorflow 2.3.0
- Matplotlib 3.2.2
- Numpy 1.18.5
- Pandas 1.05
- sklearn 0.22.2