CNN ANALYSIS ON SECURITY DATA: DERIVING NATURAL ORDER IN A NONNATURAL WORLD

by

RANDY KLEPETKO, M.S.

DISSERTATION
Presented to the Graduate Faculty of
The University of Texas at San Antonio
In Partial Fulfillment
Of the Requirements
For the Degree of

DOCTOR OF PHILOSOPHY IN ELECTRICAL ENGINEERING

COMMITTEE MEMBERS:
Ram Krishnan, Ph.D., Chair
Artyom Grigoryan, Ph.D.
Yanmin Gong, Ph.D.
Jeff Prevost, Ph.D.

THE UNIVERSITY OF TEXAS AT SAN ANTONIO
College of Engineering
Department of Electrical and Computer Engineering
December 2022

DEDICATION

This dissertation is dedicated to the Creator without whom there would be no order.

ACKNOWLEDGEMENTS

This Masters Thesis/Recital Document or Doctoral Dissertation was produced in accordance with guidelines which permit the inclusion as part of the Masters Thesis/Recital Document or Doctoral Dissertation the text of an original paper, or papers, submitted for publication. The Masters Thesis/Recital Document or Doctoral Dissertation must still conform to all other requirements explained in the Guide for the Preparation of a Masters Thesis/Recital Document or Doctoral Dissertation at The University of Texas at San Antonio. It must include a comprehensive abstract, a full introduction and literature review, and a final overall conclusion. Additional material (procedural and design data as well as descriptions of equipment) must be provided in sufficient detail to allow a clear and precise judgment to be made of the importance and originality of the research reported.

It is acceptable for this Masters Thesis/Recital Document or Doctoral Dissertation to include as chapters authentic copies of papers already published, provided these meet type size, margin, and legibility requirements. In such cases, connecting texts, which provide logical bridges between different manuscripts, are mandatory. Where the student is not the sole author of a manuscript, the student is required to make an explicit statement in the introductory material to that manuscript describing the students contribution to the work and acknowledging the contribution of the other author(s). The signatures of the Supervising Committee which precede all other material in the Masters Thesis/Recital Document or Doctoral Dissertation attest to the accuracy of this statement.

This work is partially supported by NSF grants HRD-1736209 and CNS-1553696.

December 2022

CNN ANALYSIS ON SECURITY DATA: DERIVING NATURAL ORDER IN A NONNATURAL WORLD

Randy Klepetko, Ph.D.
The University of Texas at San Antonio, 2022

Supervising Professor: Ram Krishnan, Ph.D.

Neural Networks with back propagation are a powerful deep learning tool and were enhanced with the Convolutional Neural Networks (CNN). CNN gained notoriety in the detection and classification of objects within images. Images are grids, or matrices, of data aligned in a specific pattern. These structures are naturally defined by the light photons that were the source of the images. CNN have since proven valuable in non-image tasks, detecting and classifying variances in different data. When CNN are used with other sources, often those data have a naturally defined order, but what of cases where no natural order exists? CNN analyze the underlying structure of images, extracting features out of the mathematical patterns from which objects are identified. When data is not naturally ordered, duplicating similar mathematical structures should improve CNN performance. One example is digital data comprising of security breaches. That data rarely has a naturally derived order and is usually defined by specification or arbitrary log entry. Security is also an area where high performance is preferred so research that effects improvement has merit.

This topic is examined by exploring the mathematical structure of images, followed with an analysis of how a CNN are identifying those patterns. It then includes an algorithm for mimicking a similar structure in nonnatural security data, followed by testing between the original specification structure, statistically derived image related schemes, and a set of random orders. Also included is an examination of current visualizations tools to gain an understanding of the parameter behavior. By testing this hypothesis with different data sets and multiple models of CNN it is shown that using mathematical relationships to define the matrix structure, attempting to match those found in images, has strong merit for higher performance, but understanding the strengths

and weaknesses of a particular CNN model variant is imperative to maximize the benefit.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

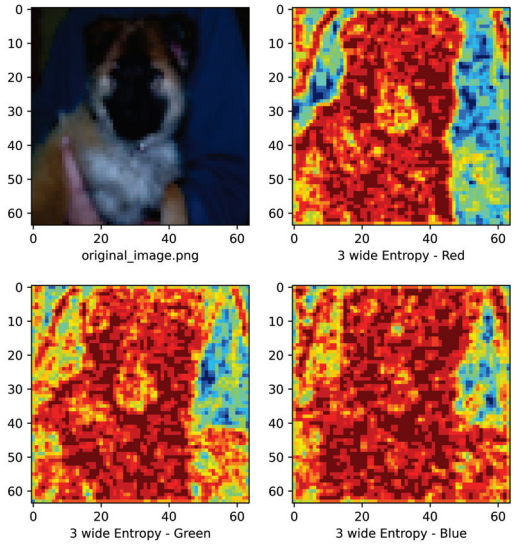
CHAPTER 1: INTRODUCTION AND MOTIVATION

Recent explosion in CNN architectures have pushed computer image recognition [1] to an art form. It has provided a variety of options depending on the application [2]. They are also used in non-image related fields, so understanding how they work with images should help us leverage their use in these other areas.

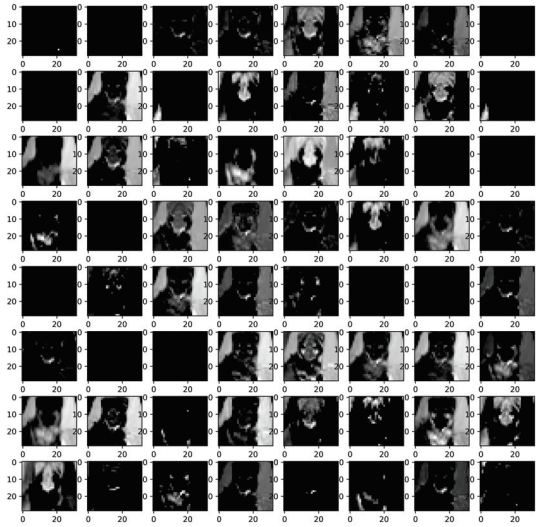
It has been shown that entropy can be used to both increase detail [3] and reduce noise [4]. By examining the entropy of an image, for example the dog in Figure 1.1a, and comparing the activation values found by analyzing it with a shallow CNN, Figure 1.1b, it is visible that the CNN is identifying patterns in entropy. This research hypothesizes that these new CNN models are finding novel ways of making identifiable information out of these patterns of entropy.

Exploration has been made in using CNN in fields other than image classification. Text [5], sound samples [6], and medical diagnostics of DNA [7] are examples on how this technology has other uses. Often times these sources of data have a naturally defined order such as the acoustical waves in a sound or DNA in a sequence. But many times these data sources do not have a naturally defined order, for example a series of sensors on a automated vehicle [8]. In most "*nonnatural*" cases the researcher defaults the matrix order to a structural relationship between features usually established by an arbitrary specification. The term "*nonnatural*" is used as a definition of ordering sources that were not defined in nature. This is opposed to "*unnatural*" which leads to the idea that they were ordered by something super-natural.

A particular subset of nonnatural data that has gained interest is detecting digital security issues. For example raw IP traffic [9, 10], computer process metrics [11], and industrial sensors [12] are examples where researches are evaluating the use of CNN in security related fields. The ability of a CNN to examine a large amounts of data looking for patterns from which important features are extracted is what make CNN successful. Compiling data sources in a properly organized



(a) Image with a 3 wide Entropy of Primary Colors



(b) CNN Level-2 Activations of Image

Figure 1.1: Image processed by CNN

structure for a deep learning algorithm to analyze should be of concern when using CNNs.

The activation plot in Figure 1.1b is an example of visualizing the convolutional layer. CNN models consist of many layers each performing a specific task. Some run convolutions via a series of filters, some pool data points together, while others perform mathematical operations over either one or a pair of grids. Comprehending what could be going on within these “black boxes" is improved with visualization techniques that let the user by eyesight understand the network internals.

1.1 Problem Statements

This research explores the use of training CNN models using nonnatural security data. Does order of that data matter when defining the matrix? Is there a method that can derive a preferred, or even an optimum order, one that produces the highest performance in the least amount of training time? Is it possible to leverage visualization tools when working in a nonnatural security domain?

1.2 Summary Of Contribution

The contributions of this work are:

- Show that ordering of rows and columns has a major impact on the performance of CNN when used to analyze nonnatural data but how much is model dependent.
- Define a methodology for ordering nonnatural data by statistical relationships.
- Show that using statistical relationships to define matrix order is a strong predictor of a good performing order, but the exact statistical relationship can depend on the model of CNN.
- Increase the state of malware detection technology by providing data preparation and visualization tools that assist in improving CNN performance when analyzing security data.

1.3 Related Publications

The initial hypothesis, that order matters, and defining order using statistical relationships based on portions of the work in Chapter 3.2.1 - 3.2.1, Chapter 4.1.2 & 4.3.1 and Chapter 5.2.1, was accepted for publication ICCCS-2019 conference.

- Randy Klepetko and Ram Krishnan. “Analyzing CNN Model Performance Sensitivity to the Ordering of Non-Natural Data", 4th International Conference on Computing, Communications and Security (ICCCS). IEEE Rome, Italy, 2019. Reproduced with permission from IEEE.

Additional development of the methodology and extending the types of data that can be organized using statistical relationships based on portions of the work in Chapter 3.1 & 3.2.1, Chapter 4.3.2- 4.3.6, and Chapter 5.2.2-5.2.6, was accepted for publication at SKM-2021.

- Randy Klepetko and Ram Krishnan. “Analyzing CNN Models’ Sensitivity to the Ordering of Non-Natural Data", International Conference on Secure Knowledge Management in the Artificial Intelligence Era. Springer, San Antonio, Texas, 2021. Reproduced with permission from Springer Nature.

Followed up with a Information Systems Frontiers journal publication where analyzing the use of visualization tools was added to enable some understanding for the individual models parameter response based on portions of the work in Chapter 3.3.1 and Chapters 5.2.7 & 5.2.8.

- Randy Klepetko and Ram Krishnan. “Visualizing CNN Models’ Sensitivity to Nonnatural Data Order", Information Systems Frontiers, 2022. Reproduced with permission from Springer Nature.

The latest publication explored using these techniques with internet protocol data sets and shared a new visualization tool to better understand the model parameters, the Model integrated Class Activation Map (MiCAM). It was accepted for publication to MLSC-2023 and is based on portions of this work in Chapter 3.3 & 3.2.1, Chapter 4.1.1 & 4.1 and Chapters 5.1, 5.3.1 & 5.3.

- Randy Klepetko and Ram Krishnan. “MiCAM: Visualizing Feature Extraction Of Nonnatural Data", 4th International Conference on Machine Learning and Soft Computing (MLSC 2023) Copenhagen, Denmark, 2023. Reproduced with permission from ACSTY.

Sections discussing ResNet modification, portions of Chapter 4.3.3 & 5.2.7, & all of Chapter 5.2.10, and IP maleficent classification analysis, portions of Chapter 5.3, have not been submitted for publication.

1.4 Organization Of Dissertation

The remainder of the manuscript is organized as follows: Chapter 2 discusses related work using CNN with nonnatural data. Chapter 3 outlines the methodology including a description of ordering the data. Chapter 4 describes the analysis procedure with data and model details. Chapter 5 describes the evaluation results. Chapter 6 summarizes and concludes.

CHAPTER 2: BACKGROUND AND LITERATURE REVIEW

This chapter explores the related background and research. First discussed is understanding images to identify the patterns that CNN discover and creates detectable features from. Then discussed are various image analysis CNN models used in these experiments, their history, capabilities. Next is included a description of some of visualization tools used with CNN. Following is an exploration of novel uses of CNN in nonnatural settings, and finally a background is provided of CNN used with nonnatural security data.

2.1 Image Analysis

Before neural networks, early image analysis used a number of mathematical procedures to help automate the decoding and analysis of images. These techniques were used in image enhancement, compression, feature extraction and rudimentary classification. Many of these methods are still used in our video and image compression schemes, and are continuously reviewed for novel insights and tools.

2.1.1 Convolutions

Convolution, the mathematical process of revolving a function over a set of numeric values has long developed relationship with image analysis. In 1960, Perrin with Kodak [13] first introduces the idea of using convolutions to identify edges and other sinusoidal patterns found within an image. There are many other examples of how convolutions are used in image analysis and they are the key process of how CNN operate.

2.1.2 Correlation

Correlation, the statistical analysis of two sets of data and the measuring the related differences between them has also long been used in image analysis. In 1961, Horwitz et. al. [14] proposed using correlation in a process he called auto-correlation for identifying numbers and letters on a page. This is done by comparing a known pattern and finding the highest correlation with multiple augmented versions of an unknown second source. In 1970 Arcese et. al. [15] also use correlation with a previously defined matrix to detect objects within a radar image. Correlation has repeatedly been used in both image and digital information processing since.

2.1.3 Pooling

Using pooling as a image feature extractor was first suggested in 1965 by Hubble et. al. [16]. They were studying the visual cortex receptors of a cat and discovered that data pooling was occurring between two of the layers. Since it became a normal process for minimizing images, but it wasn't until 2005, when Serre et. al. [17] introduced it's use as a feature extractor in an early CNN model.

2.1.4 Entropy

In the study of pattern recognition, entropy was introduced early on. In the year 1966, Bremermann [18] explores the use of entropy to study speech recognition. Although speech is thought of as non-image related, he clearly ties the sound pattern identification into a mathematical image on which a classification problem is solved. In it he includes entropy among other measurements to help identify features within the sound waves. He shows that these features are non-linear in general, and he is critical of the linear approaches used at the time including Fourier analysis. Entropy is another example of a mathematical concept that has repeatedly been used in image analysis.

2.2 CNN in Image Analysis

The concepts of neural networks was first published about by Roseblatt [19] in 1963, labeling each neuron a *perceptron*. There were a number of advances including layering the neurons by Fukushima et. al. In 1983 [20], where they introduce the *Neocognitron* or *new-recognizer*. Followed shortly by Rumelhart et. al. [21] with the introduction of back propagation for auto correcting training errors.

It was Werbos, in 1987 [22], who published a analytical description of the human visual system, and provided the insight on how to duplicate a similar structure using machines. In his article he explains that the visual system is broken into several parts. The front are the optical neurons (pixels) within the eye, followed by different layers of the brain, each of which are involved in assembling the image as features found within the pixels, organizing them into a story the brain understands.

Five years later, his work was translated into a practical system that could be coded in a machine by Hussain et. al. [23]. They define the basic building blocks and links between the layers calling it a *Feature Recognition Network*. They replicated a similar multi-level approach where middle layers extract features from the input pixels, followed by a decision network which compiles a story from those features. They made comparisons to the *Neocognitron* and showed it could perform as well when deciphering hand written numbers with one tenth the number of cells.

A diagram on how it works is shown with a cat image in Figure 2.1. The initial layers combine the pixels into features consisting of edges and textures. The following layers then combine and extract these features into new complex objects (eye's, arms, etc.) until in the final decision layer the picture is recognized. Initially this recognition result is random, with high error rates, but with training and back propagation the error rate is reduced.

That inspired Convolution Neural Networks from which many models were derived.

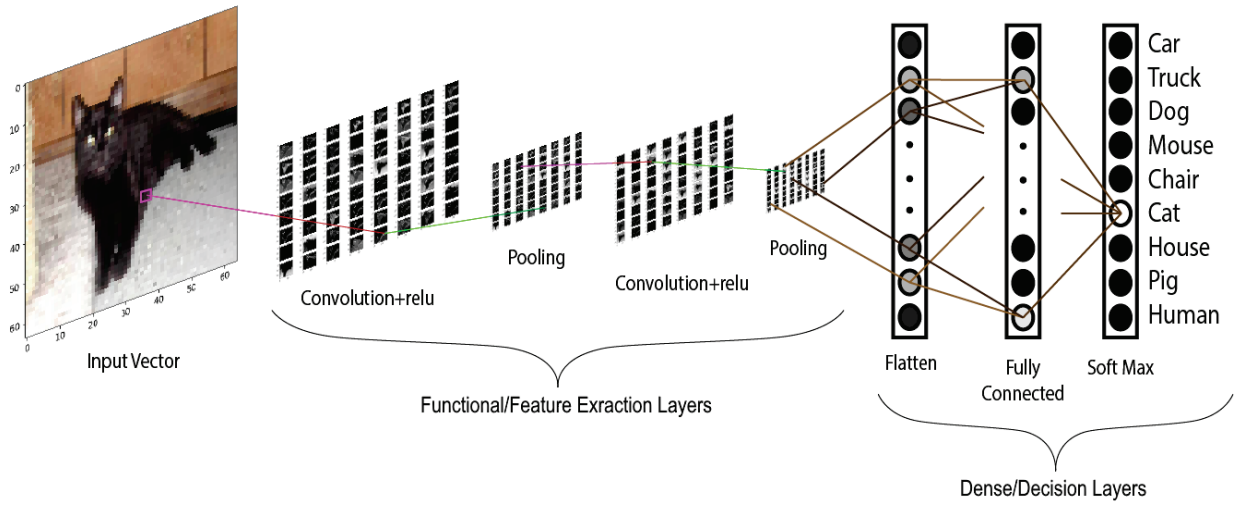


Figure 2.1: Visual diagram of a CNN activations identifying the image of a cat

Each new model uses novel techniques to accomplish higher precision in computer image object identification and classification. Following is a review of each model used in this research, they're features and capabilities. The technical details will be discussed in the later methodology Chapter 3 and the Appendices.

2.2.1 LeNet

In 1989, LeCun et. al. introduced the LeNet model in [24]. It was the first to use back propagation in a practical application. They identify and classify hand written numbers provided by the US postal system, simple black and white images. They're goal was to achieve a 1% error rate rate which they accomplished after 23 epochs of training. It is a shallow sequential model meaning it has few layers that are linearly aligned. The data sets used closely resembles one used in this research for the MiCAM visualization tool analysis, the MNIST [25] data set of handwritten numbers.

2.2.2 Inception Net

A number of CNN models came out between then and 2014. They explored different options by choosing various mathematical activation functions, convolution window sizes, and modifications to the depth and width of each layer. Inception Net by Szegedy et. al. with Google [26] took some novel approaches and published several iterations where each added new features to over come the limitations previous CNN models encountered.

In version one, they explored using multiple convolution window sizes, $(5x5, 3x3,$and$1x1)$. This added the capability of recognizing similar patterns at different scales. They then, in version two, factorized all of the (NxN) convolutions into a series of lower order $(1xN)$ and $(Nx1)$ convolutions. This reduced the number of mathematical parameters by almost an order of magnitude. The next version, V3, also included a factorized copies of $7x7$ window size with several other modifications including a new optimizer function, RMSProp, batch normalization in the classifiers and label smoothing. These later features were to reduce over fitting.

It consists of 13 separated convolution stages, each consisting of a group of factored convolutions and a pooling operation, with a total of 95 convolutional layers. Some stages are separated by parameter reduction pooling operations. It was the first non-sequential model that proved that the layers don't need to be stacked linearly by beating the competitors in the ImageNet 2014 challenge. There are later versions of Inception Net, but they were released after ResNet included features that align with the ResNet models which are discussed next.

2.2.3 ResNet

In late 2105 He et. al. submitted a paper [27] that added a new feature to the network topology that revolutionized CNN, the residual connection. This new link is for data to mathematically add the input of a convolution stage directly to the output, feeding the next stages input. It greatly

reduced the issue of a vanishing gradient which is a major issue when training deep networks. It accomplishes this by carrying though gradient changes over this connection, maintaining the majority of the gradient magnitude further through the network.

They were able to win first prize the 2015 ImageNet competition with a top five error rate of 3.57% taking the prize in all categories, classification, localization, and detection. They also won the categories of detection and segmentation in the 2015 COCO competition.

There are multiple published versions of ResNet, all based on how many convolution layers they are consist of. The convolutional layers are grouped by two or three in a stage with a residual link around them. This research focuses on the using ResNet-18, per it's name there are 18 convolutional layers with a pair of pooling feature extractors at the end. Our research experimented with other versions but found deeper models (ResNet-50, ResNet-101, etc.) took orders of magnitude longer to train with no better results.

2.2.4 Xception Net

Chollete published the Xception Network in 2017 [28]. Inspired by Inception Net, it reduced complexity and parameter count by adding depth separable convolutions. This is where the dimensions of the convolution filter are modified by factoring 3-dimension filters into 2-dimensions. By reducing the dimensions of the filter, the number of parameters required by the filter computations are reduced. It reduces the processing time, while maintaining the majority of the combined filters capability. This approach introduces using several $NxNx1$ and $1x1xN$ convolution filters in place of the individual $NxNxN$ filters. It also includes more *relu* activation functions supplying additional non-linearity.

Xception has 14 stages which consist of 40 convolutional layers. Most stages included a pooling operation and a residual link. The residual links are opposite sets of $3x3x1$ width wise convolutions and some links include $1x1xN$ depth wise convolution . Performance comparisons

versus previous models give Xception an advantage in fewer parameters hence processing time and a moderate improvement in accuracy.

2.2.5 MobileNet

A year later Howard et. al. published his work with MobileNets [29] with the goal to minimize the footprint of CNN to give them mobile access. Like Xception they also use depth separable convolution but use them to reduce the width of later layers instead of expanding them as done in Xception. This greatly reduces parameter count. Other models use pooling layers to accomplish this. To reduce the count even more, they add two more parameters a width and resolution multiplier. These positive values are less than or equal to one, and are used when training from scratch to decrease parameter count further while maintaining accuracy.

Several versions of MobileNet were tested with but this research found the first most responsive during training. It is very sequential, no links or multiple paths through the 13 stagee. Each stage consists of a series of depth wise and width wise convolutions. It is the only network that doesn't include any pooling operations as feature extractors, but uses convolutions with a strides larger than one to accomplish this.

2.2.6 DenseNet

Last revised in 2018, Huang et. al. [30] published DenseNet. Like residual links, they add connections around layers, but instead of using addition as the function for combining the input source with the output, they used concatenation, with each stage increasing the depth for the next stage, creating a *denser* input cluster. Hence they named it a densely connected network. This compiles all of the information previously gathered together as input from the earlier layers and forwards the details on for the later stages. It reduces the information lost by the addition process used in residual links, maintaining initial input integrity further mitigating the vanishing gradient.

They reduce parameter count by including bottleneck layers in between dense stages. These do not include the surrounding dense links, instead include a $1x1xN$ convolution to reduce the depth and a pooling layer for a reduction in width and height. It allows a deeper number of layers while limiting parameter count. This research worked with the smallest, Dense-121.

2.3 Visualizing Convolutional Neural Networks

A number of visualization tools have been created to assist in the engineering and development of CNN. Some image generating tools create graphs to provide a higher level understanding of the data flow within the model. Other visualization tools provide histograms of the parameters as they adjust over the training period. Visually revealing the hidden layers provides researchers comprehension behind neural network decisions. Theses tools are evolving as the field matures.

By providing transparency and an explanation [31] as to the network parameter intensities visualizations assist the researcher in all stages of the network development life cycle. Early in model construction they provide failure details letting the engineer to see how performance is affected by model changes. Visualizing the hidden layers enhance confidence that the model is identifying a proper set of features during network maturity. As the network exceeds human performance, the visualization tools provide a computer instructor, teaching the researcher novel ways of examining the data.

Flow and model diagrams were introduced since the very first deep learning models were published. They provide a visual representation of the mathematical objects that are coded within the software. They represent these as spheres or cubes, and as multiple mathematical objects are aligned in a layer, these graphical representations are placed next to each other in a row. A line between objects represent communication or parameter passing pathways. For convolutional layers, a plane of objects is used, and stacks of planes are a symbol which includes the third filter dimension. For brevity when the interpretation is understood, sometimes a higher dimension object

is represented by a lower level visual construct.

Another important class of visualization tools are classification response graphs which are designed to show the how responsive a pixel is to that particular classification made on a tested sample. These include Salience and CAM graphs. Most of these tools apply well with image data, but are not as well suited for data that is not visual in nature like cybersecurity. Often they are some form of flow and layer diagrams, class activation maps [32] (CAM), gradient visualization [33], sensitivity to perturbations [34], or a confluence of these.

CAMs were initially generated using a weighted sum and up-sampling the class activation maps from the penultimate layer to generate activation regions of the original image. CAMs have evolved using different parameters as the weight values for the ratio in summing the class activation maps. Detailed by Selvaraju et. al in 2016, GradCAM [31] uses gradients in a back propagation step with a *relu* function. LayerCAM [35] published by Jiang, et. al. collects the GradCAM maps from all of the individual layers and then sums them together in a normalized total that includes higher amount of detail from the shallower layers within the network.

GradCAM++ [36] by Chattopadhyay et. al. modified GradCAM by adjusting a normalizing factor used to determine the weights for the individual gradients from the feature activation maps. Devised by Wang et. al. in 2020 ScoreCAM [37], goes further by dropping the gradients altogether and include a contribution value to measure the importance of each activation map. EigenCAM submitted by Muhammad et. al. [38] replaces the gradients with an eigenvector that is derived from a combinations of the weights from all of the layers.

One thing all of these CAM systems have in common is they attempt to produce a two dimensional plot that shows how the features on the penultimate layer are related to the objects within the sample image. This works fine with shallower networks since the features within the penultimate layer are closely related to the pixels within the source image, but what about CNN models that are deep and the final feature set have no direct relation ship to the initial image. For

example a source image of 75x75 pixels (75 x 75 x 3) and the resulting DenseNet-121 penultimate layer (2 x 2 x 1028). A 2x2 grid does not distinctly map to points on a 75x75 grid. A better visualization tool is needed for these deeper layers. For this research, to identify the patterns that CNN layers are extracting features from nonnatural security data, a better visualization tool was developed, the Model integrated Class Activation Maps (MiCAM).

2.4 CNN in Nonnatural Data Analysis

This section presents other research that examines using nonnatural data as the source for CNN analysis. CNN were initially designed to mimic human vision and since have achieved super human performance in this task, but what about data sources that do not relate to vision?

2.4.1 Industrial Cases

In [39], Lihao and Yanni analyze the quality of rubber tire treads based on the parameters measured during the manufacturing process. There are four levels to the manufacturing process with eleven metrics sampled at each level. They vectorize these parameters, filter for noise, and feed those vectors to a CNN. They achieved a 94% accuracy with this process. Other than noise filtering they did not discuss data preparation or how it was organized as it was fed to the CNN.

2.4.2 As a Feature Extractor

Golinko et al. [40] examined using a one dimensional CNN as a feature extractor for other machine learning algorithms (kNN with k=1, SVN, and RF) with nonnatural “Generic" data and examined if ordering of the source data for the CNN has any impact on performance of the final classifying algorithm. They propose using statistical correlation as a method for identifying relationships of adjacent data and show that not pre-ordering the data for CNN feature extraction is detrimental to

performance. They show ordering by correlation offers significant improvement in most cases, especially for kNN and SVN, improving final average accuracy from 76% with no feature extraction to 82% if the features were ordered by correlation prior to CNN feature extraction.

2.5 CNN in Security

One field where CNN analyzes nonnatural data is the dynamic analysis of digital information for security purposes. This is when the data contains maleficent actors which is studied for general patterns that could be used in dynamically identifying any future infections. This is in contrast to static analysis which uses simple techniques of identifying fingerprints of known maleficent actors to counter specific infections. An analogy is training bank security guards to look for suspicious characters, guns and masks, versus identifying bank robber faces from a stack of photos.

2.5.1 Process Metric Analysis

Smith et al. in [41], performed dynamic malware analysis using process calls made by the executed malware code as the data source. Executed code submits a series of commands calls in sequence. These calls are command strings (ie. “ssh") which are pre-processed via a one hot encoding. When commands are issued during the same time segment they are included into a single on hot encoded vector. A series of these vectors represent the executed code as the calls are submitted to the kernel over time. These vectors were then analyzed as a group by several different machine learning techniques, comparing malware executed code and non-malware executed code. They show that a CNN can have a 94% accuracy with a 95% precision and 89% recall. They do not discuss the one hot vector ordering.

In [42], Tobiyama et al. compiled more process command call information from the machine logs. These include time of the process, name of the process, process ID, name of the command, the current working directory, the result from the command, and other information

included when the command was issued. They fed this information into a pre-trained RNN for feature extraction. The output of the RNN was fed to a CNN for analysis and achieved an AUC of 96%. They do not discuss the ordering of the data prior to feeding it to the network for analysis.

Abdelsalem et al. in [11], use metrics retrieved by hypervisors in a cloud environment that track the individual processes on a virtual machine. This set of 35 metrics are captured for each process running on the VM. They were then compiled into a process row, metric column matrix which is supplied to a CNN. They achieved an 89% accuracy, but did not attempt to optimize the data by maintaining static process rows or identify a preferred ordering during pre-processing. McDole et. al. [43] followed up with research analyzing different CNN architectures using the same data set and ordering scheme. Kimmell et. al. [44] includes the use of other deep learning models, recurrent neural networks (RNN), by testing the validity using long short term memories (LSTM) and Bi-Direction LSTMs. They also explore if the order has an effect on training and discover that it does affect performance for all models. Parts of this manuscript expand on these techniques with this data set.

2.5.2 IP Traffic Analysis

Arranging raw IP traffic packets from the publicly available security data set, CIC-IDS-2017, in a grid after the physical layer was stripped, Zhang et. al. [9] analyzed them using CNN, LSTM, and a hybrid of the two. They tested both binary classification (benign/maleficent) and multi-classification (benign + 10 maleficent types). They show all systems achieve quite remarkable, near-perfect results. For binary classification the best in precision was the hybrid which was slightly ahead of CNN, followed by LSTM. Regarding multi-classification, CNN had some minor advantage in precision over the hybrid and the LSTM was behind both. Parts of this manuscript expand of these procedures with this data set.

2.6 Research Goals

Extending this related work, our research goals are:

- Defining a methodology to derive a preferred or even an optimum order for any nonnatural data that is supplied to a CNN for analysis.
- Improving dynamic maleficent actor detection when using a CNN by properly pre-processing nonnatural security data with regards to row and column ordering.

CHAPTER 3: ORGANIZING NONNATURAL DATA FOR CNN ANALYSIS

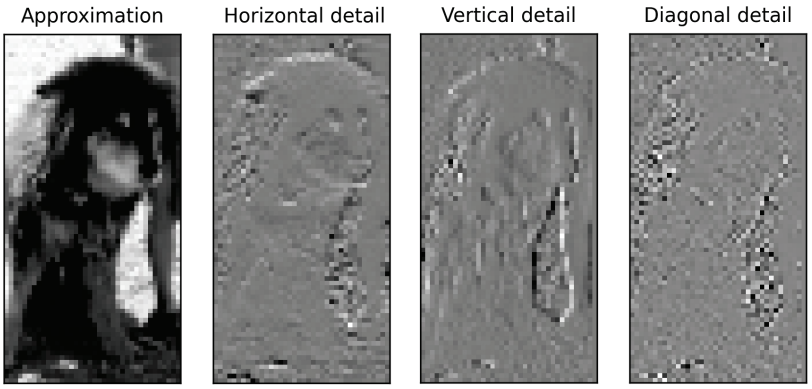
When comparing a CNN to a strictly densely connected neural network exactly what is it doing that helps it detect objects? It was shown in the initial developments of CNN by Hussain et. al. in [23] that the *Feature Recognition Network* accomplished the same object detection task as the fully connected *Neocognitron* designed by Fukushima et. al. [20] but with one tenth the number of cells. How did it do this and why objects in images?

When the architecture of a CNN are examined in detail they consist mostly of individual layers operating many convolutional filters over small windows. The are transcribing operations through the entire image, transforming it over into a set of new images. The simplest example of a filter is the *wavelet*, a unit filter of window size two, span of two, stride of two, and simply consists of two anti-symmetric digital pulses. When processed over a single channel in two dimensions, the wavelets filter out the high frequencies within the image or edge details of as shown in figure 3.1, and transformed the original image figure 3.1a into four new images, each displaying different characteristics of the original.

A CNN has many filters, as specified by the design engineer, where each parameter is algorithmically assigned values through the training process. Each of these filters are trying to find some pattern within the window of the convolution which means the patterns are *localized* within the filter window size. This research hypothesize that it is this localization of pattern identification and feature extraction that allow CNN to excel at image related tasks as compared to the processing required to perform the same task with a densely connected network.

Local data points are nearby, and are relatively grouped with each other. The hypothesis is that adjusting the pixel order will change the localizing patterns that the CNN are detecting. How can this hypothesis be leveraged when analyzing data that doesn't come in natural form?

(a) B&W Image of Dog



(b) Wavelet Filtering of Image

Figure 3.1: Image processed by Wavelet Filter

What makes up an object in an image? An object is a set of closely related pixels. These have similar pattern and maintain a relative relationship with their surrounding pixels as to shades of the associated channels (colors). For example the dog in figure 3.1a, the nose are a set of dark pixels, surrounded by gray fur. The image background is very bright white, and appears to have some stony texture. There is a distinct edge between the dog and background despite the diffusing hair.

All of these objects and sub-objects mentioned have mathematical relationships with the surrounding pixels. Statistical correlation informs us that pixels within the same object are highly correlated. Sub-objects of the same object could have a negative correlation, so the same object could have a high positive or negative values. Those pixels not related to the same object should have little relationship or close to zero correlation as compare to those within the same object. Accordingly those pixels should be spatially separate.

How to take advantage of this? Some form of correlation should be used to establish an order, and quite possibly build *artificial objects* out of the malware patterns to enhance detection.

This is discussed in further detail later in this chapter.

Nonnatural data organization is required prior to using CNN. The data sources must be placed in some sort of stable pattern for the CNN to perform properly. Often that data is not organized easily, and some care should be taken to ensure proper placement of the data. This chapter discusses this topic in detail by: first review how data is naturally organized when CNN are used in analyzing images and their related data expressions; second discuss how nonnatural data sources are collected and how that leads to an appropriate structure for CNN; and finally a technique for ordering nonnatural data within the structure that could improve CNN performance.

3.1 Building Grids from Nonnatural Data Sources

Nonnatural data sources are rarely pre-organized into a grid like fashion. If a CNN is to analyze the data it must be organized as such. How that is accomplished will depend on the source of the data, how it is packaged as samples, and primarily what question the network is analyzing the data to answer.

Regarding the question getting asked, for example, take a stock market history, a series of companies each with prices changing through the day. What questions could be asked? For example, how one company is doing compared to it's past or other companies? How the market is doing as a whole? What should be sold, bought, and how much? Some questions use the same data organization others different.

After the questions are defined the researcher needs to decide which data and how to best organize that data to answer the question. Using the stock price example, one could ask, based on historic stock price, will the year have an up or down market. In this example, take the closing prices of 100 indices over a year, 200 business days, that's a 100x200 grid. Sample that over 20 years, and there is 4000 samples. Another example could be asking about a particular business. Take the price per hour and stack it over days, or take the beginning, median, and final price over

a week, by four weeks. One could also include outside data, open records like expense reports, profits / loses, and other sources of information. A decision needs to be made with regards to what data is available and germane to the question.

3.1.1 Mapping Feature Labels and Values

The CNN uses a grid structure, consisting of two axis. In algebra they are called the abscissa and ordinate but in this manuscript they are called rows and columns, both as vectors. The next step in creating the grid from nonnatural data is to identify and label associations from the data to the rows and columns. Start with a grid of V number columns and U rows. Before any values are assigned, the labels need to be assigned for each of the rows and columns. Mathematically using set theory, the column labels are $v_i, \forall i = 1 \dots V$, and rows $u_j, \forall j = 1 \dots U$. This provides two vectors of labels, $\mathbf{V} \ni \{v_1 \dots v_V\}$ and $\mathbf{U} \ni \{u_1 \dots u_U\}$.

Next the data associations are assigned to the labels. For the business example, the rows could be the businesses, labeled by stock index name, and the columns various economic measurements such as include opening, closing, and median stock price, number of reported employees, total revenues and expenses, net income, and taxes paid.

Now with the data association established by the row and column labels, the values of the sample are assigned all of $\mathbf{X} \ni x_{ij}, \forall i = 1 \dots V$ and $\forall j = 1 \dots U$, or in grid of size VxU in the form:

$$\mathbf{X} = \begin{bmatrix} & v_1 & v_2 & \dots & v_V \\ u_1 & x_{v_1u_1} & x_{v_2u_1} & \dots & x_{v_Vu_1} \\ u_2 & x_{v_1u_2} & x_{v_2u_2} & \dots & x_{v_Vu_2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ u_U & x_{v_1u_U} & x_{v_2u_U} & \dots & x_{v_Vu_U} \end{bmatrix}$$

Each point or pixel x_{ij} in this grid can be considered a unique *pixel*. This pixel is related to the

other pixels in a column and row, and are grouped together as column and row vector, u_i and v_j.

As the data is mapped onto the grid note that most data is ingested into a system serially. This means that data is brought in one element at a time, and it is up to the data organization and order that defines how it is delivered. This data is compiled into packets. Does the packet sample contain data about many labels, or just one? This question is asked along both the rows and columns. It is a detail that helps decides how to re-organize the data for analysis which is covered in the next couple of subsections.

3.1.2 Many Feature Labels Per Data Sample Packet

This is when the data is grouped together in large packets and values for many labels along one axis are available. In the business example, the the data packet of daily stock prices for all companies. The axis covered by many values is the businesses, and daily price is the other axis label. So this one data packet could be used to make a number of concrete analysis between businesses for that day. Statistical inferences like highest, lowest, mean and median price, and the spread of those prices can be extracted from this single sample. An important implication for the vector ordering mathematics later. These statistical inferences derived for these vectors labels are independent from other packets.

3.1.3 Single Data Sample Packet Per Labeled Feature

This is the opposite case, where data along the axis is divided among many samples. For this particular example of daily stock prices, this would be the businesses individually. The only statistical information possibly extracted about a business from the packet described in the previous subsection is the price of that business for a particular day. There is no other information about individual companies so any statistical questions regarding a single company requires many data packets. How these statistics are gathered involve a different mathematical perspective than the

previous case and are considered dependent on multiple packets. To use the same mathematics require remapping the data set. It is this remapping that adds a level of mathematical complication that is detailed later in the next section.

3.2 Vector Ordering

The focus of this study is what order is chosen for defining the axis of a grid? Is there one that is optimal? In this case, the focus in on two dimensional grids. So what order is important for the rows? What about columns? Are they dependent or independent? Does order matter at all? This next section details the ordering schemes that are compared later.

3.2.1 Dynamic Vector Order

This research found that past studies may or may not have had *"static"* ordering definitions. Often the organization of data was dependent on the data packets which may or may not consistently order the rows and columns of the data as it was compiled. The initial experiments compared this dynamic order scheme with static ordering schemes defined in the subsections below.

3.2.2 Static Order - Random: May or May not Increase Entropy

** The material presented in this subsection previously appeared in the proceedings of the 4th International Conference on Computing, Communications and Security (ICCCS 2019) in the article,* "Analyzing CNN Model Performance Sensitivity to the Ordering of Non-Natural Data", *co-authored with Ram Krishnan, Ph.D.*

One option for choosing an order is to pick it at random. This does not mean that the artificial pixels are randomly scattered onto the grid, but instead stochastically define the label order along the rows and columns. Although this adds a sense of variability to the order definition

it may or may not increase the entropy visualized when the grids are compiled. It is possible to choose and arbitrary order that reduces entropy.

Comparing changes in performance for several random ordering schemes determines whether any order has relevance. Depending on the data, this research either defined 10 random column and 10 row positions or 100 random options along a single axis. This provided 100 unique orderings that allowed this research test the basic hypothesis and the random results are used as a background to compare the others ordering schemes against.

3.2.3 Static Order - By Structure: Per Specification

** The material presented in this subsection previously appeared in the proceedings of the 4th International Conference on Computing, Communications and Security (ICCCS 2019) in the article,* "Analyzing CNN Model Performance Sensitivity to the Ordering of Non-Natural Data", *co-authored with Ram Krishnan, Ph.D.*

In most cases where previous researchers have used CNN with nonnatural data, the order is defaulted to match the structure of the packets as they are delivered to the digital interface. This order has been defined according to some human prioritized specification, one that does not consider the implication that a particular order may or may not have in regards to CNN classification performance.

The use of several structural relationships was explored as one method for establishing an order. Found were several relationships as determined by specification: log location, process number, parent/child and sibling status, related virtual machines, naming convention, or IEEE internet protocol. These orderings were studied in detail and are included in the processing and as part of the general backdrop along with the random 100.

Table 3.1: Statistical Correlation Functions

Statistical Correlation Function
$$\rho_{v_i v_j} = \frac{E(x_{v_i} x_{v_j}) - E(x_{v_i})E(x_{v_j})}{\sqrt{E(x_{v_i}^2) - E(x_{v_i})^2} \cdot \sqrt{E(x_{v_j}^2) - E(x_{v_j})^2}} \quad (3.1)$$

3.2.4 Static Order - By Statistics: Per Correlation

** The material presented in this subsection previously appeared in the proceedings of the 4th International Conference on Computing, Communications and Security (ICCCS 2019) and the proceedings of the International Conference on Secure Knowledge Management in the Artificial Intelligence Era (SKM 2021) in the articles,* "Analyzing CNN Model Performance Sensitivity to the Ordering of Non-Natural Data" *and* Analyzing CNN Models' Sensitivity to the Ordering of Non-Natural Data, *co-authored with Ram Krishnan, Ph.D.*

Perhaps images provide some insight on how to best order the matrices. CNN's are used to identify objects within images. What makes up an object in an image? Statistically, an object is a set of highly related pixels. All of the pixels will have a similar shade. Pixels outside the object boundaries usually have little statistical relationship to those inside an object. It is this fact that led to many advances in image compression techniques [45–47].

This research hypothesizes that *artificial objects* can be created by grouping the rows and columns to increase the average statistical relationship between neighboring features while decreasing the overall entropy of the image. By taking either rows or columns as a V number of vectors v, research found a relationship, statistical correlation $\rho_{v_i v_j}$ as shown in equation 3.1 to base this order from, maximizing this relationship value between any two vectors v_i and v_j for all vectors along an axis (row or column). This order was tested to see if it has a positive impact on CNN performance. Also tested, an opposite hypothesis by dispersing the *artificial objects*, minimizing the correlation between vectors to see if it had a negative impact on performance. These column orderings are included in the evaluation details. This consists of three relationship functions, correlation (equation 3.1), the absolute value of the correlation $\rho_{ABSv_i v_j} = |\rho_{v_i v_j}|$ to increase

object edge creation, and anti-correlation, $\rho_{ANTIv_iv_j} = 1 - |\rho_{v_iv_j}|$, to test a counter hypothesis dispersing the objects and increase the entropy.

The derivation of this correlation function is easy when there are many vector labels per data packet, when the statistics are independent of multiple packets. Each vector sample is immediately related to each other by calculations within the same packet. Or mathematically if the individual sample $\mathbf{X}$ which consists of P packets so $\mathbf{X} = [\mathbf{X}_p]_{p=1}^{P}$ where the packet is $\mathbf{X}_p = [x_{pq}]_{q=1}^{Q}$ with Q labels. This allows easy evaluation of the cross vector correlation by calculating the statistical relationships between samples labels across all of the packets. Since $\mathbf{X}_p \ni \{v_1 \dots v_N\}, \therefore \rho_{v_iv_j} = f(x_{v_i}, x_{v_j}) \forall \mathbf{X}_p$ where $f(x_{v_i}, x_{v_j})$ is defined in equation 3.1.

It was a struggle to derive a proper correlation value when the data packets are organized by a sample per feature label, or the statistics were dependent on multiple packets. Mathematically it is flipped around. Q labels so that $\mathbf{X} = [\mathbf{X}_q]_{q=1}^{Q}$ where a label has P packets $\mathbf{X}_q = [x_{pq}]_{p=1}^{P}$. Initially it was attempted to use the same correlation function, but because of a data size and inclusion of irrelevant samples the results suffered from a *vanishing correlation*. This occurs when a large set of samples not related to the label are included in the calculations. For this research the queries were pared down so only packets related to a single vector pair, u_i and u_j, were calculated at a time. The data set was reduced for this specific relation value to only include samples when these two vectors were included. The data was further reduced by restricting the opposite axis v to a single label. This basically modifies the correlation equation to:

$$\rho_{v_ku_iu_j} = \frac{E(x_{v_ku_i}x_{v_ku_j}) - E(x_{v_ku_i})E(x_{v_ku_j})}{\sqrt{E(x_{v_ku_i}^2) - E(x_{v_ku_i})^2} \cdot \sqrt{E(x_{v_ku_j}^2) - E(x_{v_ku_j})^2}} \tag{3.2}$$

It was repeated for the opposite vectors $v_k \forall k$. This reduced the query time from what was months for all vector pairs to processing around a single opposite vector v_k, $\rho_{v_ku_iu_j} \forall i, j$, in roughly 24 hours. Once these calculations were finished a full set of vector pair correlation values was available, $\rho_{v_ku_iu_j} \forall i, j, k$.

Summing the correlations for a single pair resulted in a statistical relationship value between the vectors $\rho_{SUMu_iu_j}$:

$$\rho_{SUMu_iu_j} = \sum_{k=1}^{V} (\rho_{v_k u_i u_j}) \tag{3.3}$$

To derive a relative importance order in the feature labels per sample correlations (equation 3.1 above) a sum is taken of all vector correlations for a single metric:

$$\rho_{TOTv_i} = \sum_{j=1}^{V} (\rho_{v_i v_j}) \tag{3.4}$$

This is *total feature labels per sample correlation* on which to order their importance, largest to smallest. It is repeated for the opposite axis, resulting in a second *total samples per feature label correlation*:

$$\rho_{TOTu_i} = \sum_{j=1}^{U} (\rho_{SUMu_iu_j}) \tag{3.5}$$

Along with the fully correlated rows ordered derived from equation 3.3, this research took the opportunity to tests some other options derived from this function. Included were similar relationships mentioned with the dependent packet association with both the absolute value of the correlations, $\rho_{ABSp_ip_j} = \sum_{j=1}^{M} \left|\rho_{m_k p_i p_j}\right|$ and anti-correlations $\rho_{ANTIp_ip_j} = \sum_{j=1}^{M} (1 - \left|\rho_{m_k p_i p_j}\right|)$.

This statistical relationship value is ranked by importance of each metric column and process row with each other. A methodology was built to construct the order. It is generic process with regards to feature label, f_i, row or column, and the function used to derive the statistical relationship value $\rho_{f_i f_j}$. The ordering methodology uses the steps in Algorithm 3.1.

Algorithm 3.1 Derive Statistical Relationship Order

For features along an axis, f_i, define a function, $\rho_{f_i f_j} \forall i, j$
From $\rho_{f_i f_j}$ define $\rho_{TOT f_i} \forall i$
Create a selection pool of features $P \ni f_i$
while $P \neq \varnothing$ **do**
  Create an empty bidirectional queue Q for features f_i Find $max(\rho_{TOT f_i}) \forall f_i \in P$ Place the corresponding feature $f_{max(\rho)}$ onto Q Remove $f_{max(\rho)}$ from P Create two pointers left, L, and right, R; $L, R \in Q$ Point L and R towards $f_{max(\rho)}$ in Q **while** $P \neq \varnothing$ *and not(**STOP**)* **do**
    if $\exists \rho_{f_L f_i} \forall f_i \in P$ *or* $\exists \rho_{f_R f_i} \forall f_i \in P$ **then**
      Find $max(\rho_{f_L f_i}, \rho_{f_R f_i}) \forall f_i \in P$ Place the feature $f_{max(\rho)}$ next to f_L or f_R on Q Remove $f_{max(\rho)}$ from P Move the pointer, L or R, to the new feature $f_{max(\rho)}$ in Q
    else
      Stack current queue Q into a final ordered axis V **STOP**
    end
  end
end
Result: A vector V of features f_i that are ordered by the relationship function, $\rho_{f_i f_j}$
Derive Statistical Relationship Order

Occasionally there are ties. This was especially true for the anti-correlated functions. Many pair of feature vectors have no correlation between them. The ties were settled by examining the next set of neighbors to see which set increased the relative total relationship value of the entire grid.

3.2.5 Static Order - Correlation of Data Subsets

** The material presented in this subsection will appeared in the proceedings on the 4th International Conference on Machine Learning and Soft Computing (MLSC 2023) in the article,* "MiCAM: Visualizing Feature Extraction Of Nonnatural Data", *co-authored with Ram Krishnan, Ph.D.*

Since this study uses correlation on the entire set of data, what if this statistical analysis is only performed on a subset of the data being analyzed. For example, what if the statistics were derived only from a data subset that was classified as maleficent. Would deriving the order from these statistics generate specifically maleficent *artificial objects*? In cases where the data sets have

sub-classes, this hypothesis is tested.

3.3 Visualizations

For further analysis, this research studied several visualization tools used on specific data samples from the testing set to explore what the CNN models are doing within the hidden layers. To see the differences between a poor-performing order and a good one, the extreme options were chosen from the results. These two extreme ordering options from each experiment were then analyzed using the visualization tools.

3.3.1 Publicly Available Visualization Tools

** The material presented in this subsection previously appeared in the journal, Information Systems Frontiers (2022) ,* "Visualizing CNN Models' Sensitivity to Nonnatural Data Order", *co-authored with Ram Krishnan, Ph.D.*

Initial attempts to use visualization tools consisted of evaluating those that have been published and readily available within the software environment. Used were *Saliency*, *GradCAM*, *GradCAM++*, and *ScoreCAM* tools. An example of a set of visualization results is found in Figure 3.2 which shows the original image followed by a Saliency [48], ScoreCAM [37], GradCAM [31], and GradCAM++ [36] plots. In this case a LeNet-5 CNN was trained to discriminate between cat and dog images. These plots visibly locate different features identified within the image. Will these visualization tools help identify malware features within grids of cybersecurity data? Along with analyzing grid order with deeper CNN models, we also explore the use of image analysis visualization tools to assess if they can provide additional insight into these deeper models analyzing nonnatural data. The resulting plots and analysis of their feasibility can be found in the following chapters.

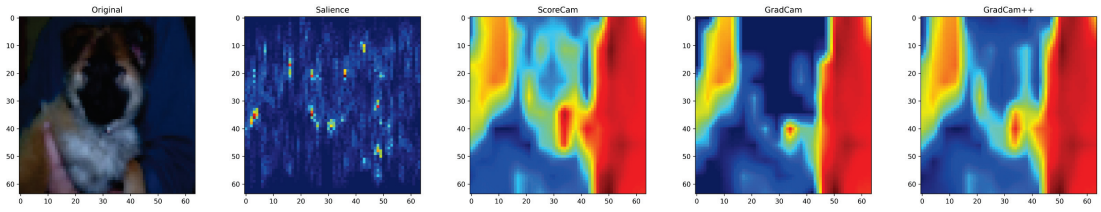


Figure 3.2: CNN Visualization Tool Results of Dog Image

3.3.2 New Visualization Tool: The Model integrated Class Activation Maps (MiCAM)

** The material presented in this subsection will appeared in the proceedings on the 4th International Conference on Machine Learning and Soft Computing (MLSC 2023) in the article,* "MiCAM: Visualizing Feature Extraction Of Nonnatural Data", *co-authored with Ram Krishnan, Ph.D.*

To fully visualize feature extraction a tool was built that is a combination of a model diagram with class activation maps. A model diagram is a flow plot that has the network layers displayed with the data pathways identified so the engineer can visually see the related connections between layers. This flow diagram is rather trivial when working with sequential models, but can be quite complex when dealing with network like Inception Net, that have multiple interconnections between layers. A class activation map (CAM) is a combination via a weighted sum of all of the activation maps for the filters a single layer. The weights for this sum define the type of CAM.

This tool takes the model diagram and instead of displaying an object (i.e. layer) as a graphical construct (sphere or rectangle) it displays the CAM for that layer. After the MiCAM diagram is complete the result is a map clearly showing the various features that each layer defines as important in identifying the class of a tested sample. A diagram of the process steps used to generate MiCAM plots is found in Figure 3.3.

The multiple steps to the process are identified in alphabetical order. In the beginning the researcher has the chosen model and the data seen in (A). The model is trained in step (B) while at the same time, the model layout is extracted from the model definition. From the result, the trained model in (C) and the layer layout are pulled out and the activation model is defined

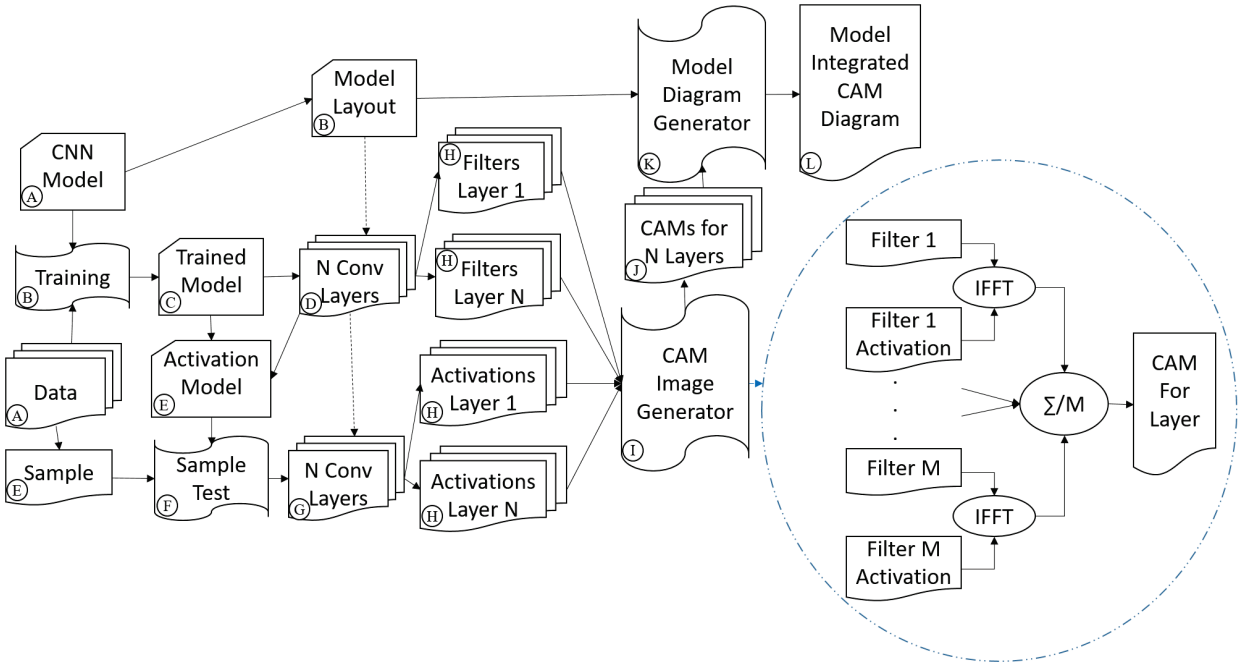


Figure 3.3: MiCAM Generation Process

(D). This model has the pre-trained layers from the trained model laid out with the filters' outputs exposed for sampling later.

With the activation model, a sample is taken (E) and tested to determine how it is classified in (F). Using the activation model post-test and the model layout, the outputs or activations are extracted (G) for all of the filters and the associated filters' weights in (H). In step (I), using an inverse Fourier transform, the inverted convolution is taken between a filters' activation and its kernels' weights. The results of are summed together, weighted by the particular filter final bias into a single CAM plot for each layer. This CAM plot is then up-sampled to match the original input grids dimensions.

To enhance the details within the CAM plots, the full RGBA pallete is used, by associating different variations of the CAM data within the plotted pixels. Every plot has a maximum and minimum range that is scaled to 256 discrete intensities. These pixel values can be positive or negative, so MiCAM uses a set of *relu* functions to display these variations in intensities by matching

one of the 4 degrees to a specific range of values. For blue it uses the full range of minimum to maximum for this plot, scaled to the 256 color levels. For red it displays the positive peaks using the *relu* of the values, scaling from zero to the maximum of this plot. For green it displays the negative peaks using relu of the negative value or zero if the values are positive, scaling from zero to the minimum. For alpha and size, it uses the full range for the plot, but scale the results to the minimum and maximum values for all of the CAM plots within the model. The results are very dynamic images that display a full range of the extracted features.

After generating the images, there is a stack of CAM plots for all of the layers within the model (J). For layers that are not convolutional, it uses an evenly weighted sum of the outputs across the filter dimension, and then up-sample them to provide a graphic for each layer. For layers that are one-dimensional (flatten and dense) MiCAM fits the linear data within the input grid, scaling elements up if there are fewer data points within the layer than the width and height of the source data. The CAM plots are then integrated with the Model Layout in the Model Diagram Generator (K) which produces the final MiCAM diagram.

The code uses the “pydot/graphviz" graphical diagram module which has an interface for integrating images in place of objects. The code required some slight modification for passing two list of parameters. One is the list of layers that has CAM plot images, and the second is the image files list of CAM plots. Both lists must be the same length, and for proper diagram generation the layer names in the first list should align with the filenames in the second list. The code is under open source license and found at `https://github.com/rklepetko/MiCAM.git` for easy access.

This concludes this chapter on the reasoning behind the steps and methodology used to test this researches hypothesis. Several implementations of this process are explored in the next chapter.

CHAPTER 4: EVALUATION PROCESS

The previous chapter discussed the hypothesis, reasoning behind it and the methodology proposed to test it. This chapter discusses the steps performed in evaluating this hypothesis and methodology. Discussed are the data sources and the method used to define the data structure organization. This is followed by a description of the test bed including hardware and software applications. The chapter closes with a technical description of each CNN model tested.

4.1 Data Sources

This section describes the data sources and how they are organized to match the methodology defined earlier in Chapter 3.

4.1.1 Dataset-1: MNIST Handwritten Numbers

** The material presented in this subsection will appeared in the proceedings on the 4th International Conference on Machine Learning and Soft Computing (MLSC 2023) in the article,* "MiCAM: Visualizing Feature Extraction Of Nonnatural Data", *co-authored with Ram Krishnan, Ph.D.*

So that a baseline understanding of how the new visualization tool MiCAM behaves, one data set that is analyzed within this research is the MNIST data set, compiled and released by Deng [27], an image library consisting of hand written numerical text. The 10 image classes are from "0" to "9" and comprise of 60,000 samples from 250 census takers and 250 high school

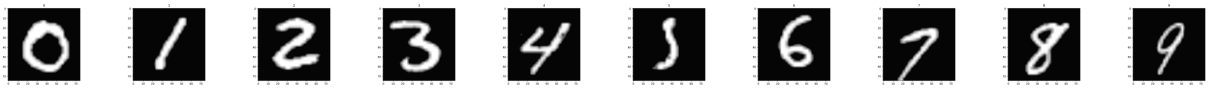


Figure 4.1: MNIST Data Samples

Table 4.1: Virtual Machine Process Metrics

Metric Category	Description
Status	Process status, Current working directory
CPU information	CPU usage, CPU user space, CPU system/kernel space, CPU children user space, CPU children system space.
Context switches	Voluntary context switches, Involuntary context switches
IO counters	Read requests, Write requests, Read bytes, Write bytes, Read chars, Write chars
Memory information	Swap memory, Proportional set size (PSS), Resident set size (RSS), Unique set size (USS), Virtual memory size (VMS), Dirty pages, Physical memory, Text resident set (TRS), Library memory, Shared memory
Threads	Used threads
File descriptors	Opened file descriptors
Network information	Received bytes, Sent bytes
Group Information	Group ID real, Group ID saved, Group ID effective

students. Another set of testing data was compiled from a separate group of 250 census and high school students, but comprised of only 10,000 samples. This research joined the two, shuffle them and use 20% of the data for testing, 20% in validation, or 14,000 of the samples per set, with the remaining used for training. Each sample was fitted in to a 20x20 grid, normalized for shading, and centered on a 28x28 image. For analysis on deeper models, this research up-sampled the images to 75x75 pixels in size. Visual examples of the MNIST data are seen in Figure-4.1. Several MNIST samples with the MiCAM diagrams are shared in the results chapter **??** as a base line on evaluating feature extraction.

4.1.2 Process Metrics of Malware Infections

** The material presented in this subsection previously appeared in the proceedings of the 4th International Conference on Computing, Communications and Security (ICCCS 2019) and the proceedings of the International Conference on Secure Knowledge Management in the Artificial Intelligence Era (SKM 2021) in the articles,* "Analyzing CNN Model Performance Sensitivity to the Ordering of Non-Natural Data" *and* Analyzing CNN Models' Sensitivity to the Ordering of Non-Natural Data, *co-authored with Ram Krishnan, Ph.D.*

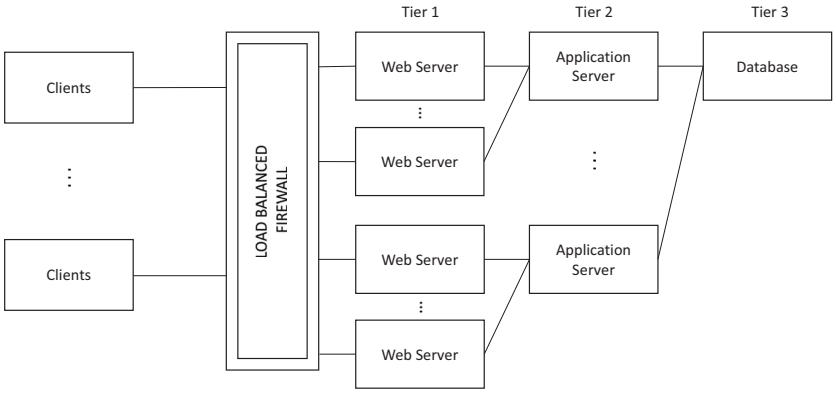


Figure 4.2: 3-Tier Web Service

This source of the data are samples taken from virtual machines in a cloud IaaS environment. These virtual machines are arrayed as a LAMP stack hosted web-site as shown in Figure-4.2. The application server is injected with malware half way during the experiment. Each sample is for a specific process running on the VM kernel, and contains a series of M number of metrics per process (Table 4.1) during a segment in time. Stacking all of the P number of processes that are captured in a sample during a single slice of time results in a matrix:

$$\mathbf{X}_t = \begin{bmatrix} & m_1 & m_2 & \dots & m_{M=75} \\ p_1 & x_{m_1p_1} & x_{m_2p_1} & \dots & x_{m_Mp_1} \\ p_2 & x_{m_1p_2} & x_{m_2p_2} & \dots & x_{m_Mp_2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ p_{P=150} & x_{m_1p_P} & x_{m_2p_P} & \dots & x_{m_Mp_P} \end{bmatrix}$$

For these experiments the 35 metrics expanded out with one hot encoding to $M = 75$ metric columns and room in the matrix for as many as $P <= 150$ process rows. The 29+ million process samples were organized around 114 experiments (infections), and consisted of 31,064 sample matrices, about half of which are considered infected. The experiments were split between 60% training, 20% validation, and 20% testing. The entire grid set for each experiment

Table 4.2: Metric and Process Correlation Functions

Metric Statistical Correlation Function

$$\rho_{m_i m_j} = \frac{E(x_{m_i} x_{m_j}) - E(x_{m_i})E(x_{m_j})}{\sqrt{E(x_{m_i}^2) - E(x_{m_i})^2} \cdot \sqrt{E(x_{m_j}^2) - E(x_{m_j})^2}} \tag{4.1}$$

Process Statistical Correlation Function

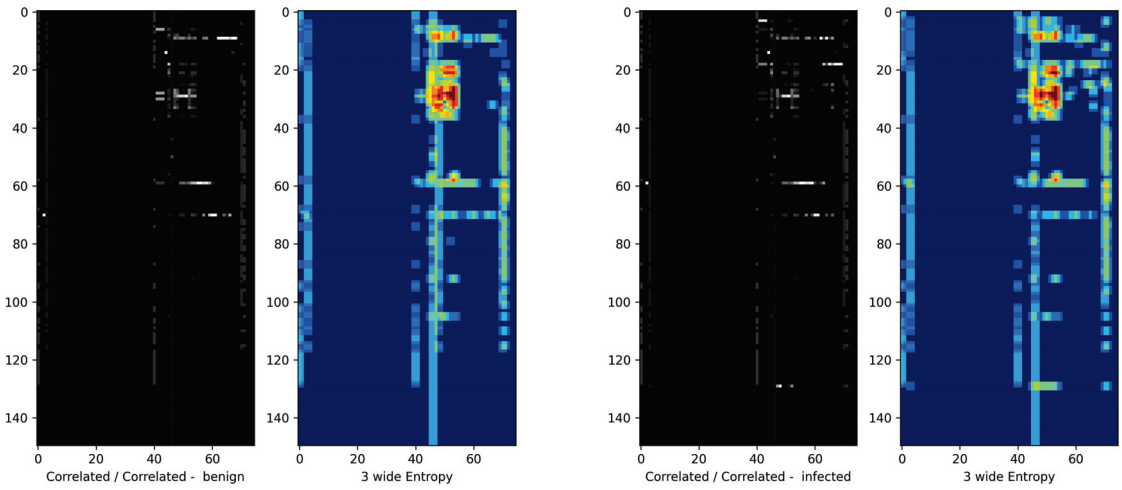
$$\rho_{m_k p_i p_j} = \frac{E(x_{m_k p_i} x_{m_k p_j}) - E(x_{m_k p_i})E(x_{m_k p_j})}{\sqrt{E(x_{m_k p_i}^2) - E(x_{m_k p_i})^2} \cdot \sqrt{E(x_{m_k p_j}^2) - E(x_{m_k p_j})^2}} \tag{4.2}$$

was included in the group assigned it, so no experiment was split between training, validation, and testing.

The data source for the rows and columns were organized in packets differently. For the columns, each feature was a specific metric, and all samples included all of metrics, or feature labels. Since this is a many feature labels per data packet situation, so the statistic functions are independent of many packets. Substituting the grid values into the appropriate statistical correlation function results with the Equation-4.1.

The data set for the rows was by process, with an individual distinct process contained within the individual data packet. This is the data packets per feature scenario, or the statistic functions are dependent on multiple packets. After substitution, results in the correlation function found in 4.2. Following the methodology spelled out in Algorithm 3.1, Subsection 3.2.1 a correlation value and order was derived for every pair of metrics and processes. Note the difference between these two functions. The metrics function is per every metric pair for all processes, while the process function is per every pair of processes per metric.

As shared previously in Chapter 3 this research defined several order options for both rows and columns. First 10 random orders for each which provided 100 distinct grids. Included are several structural relationships as one method for establishing an order. Found were several

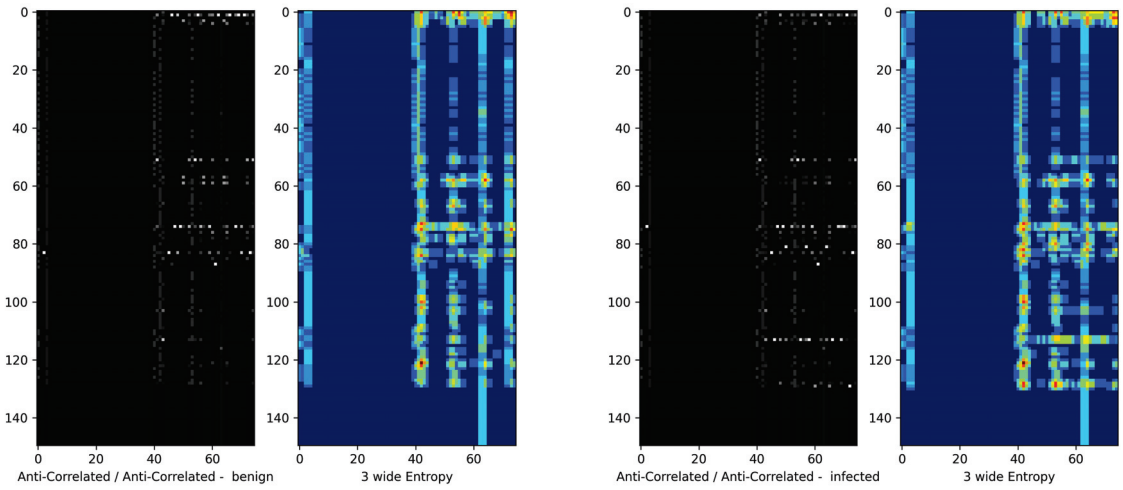


(a) Benign (b) Infected

Figure 4.3: Visual Plot of Correlated Samples with 3 Wide Entropy

relationships as determined by specification, log location, process number, parent/child and sibling status, related virtual machines, and naming convention. They performed no better, if not worse, than the average of random options. They are included in processing as part of the general backdrop along with the random 100. They were examined in detail when using the LeNet-5 CNN model, but unless noted are ignored for analysis in the remaining evaluation.

After compiling the statistically related orders with the previously defined order sets, there are a total of 252 distinct grid orders to compare. A visual example of the grids in different ordering sets is shown in figure 4.3 and figure 4.4. Shown are two data samples, one benign and another infected, using different row and column ordering schemes. Included is a 3-square pixel entropy filter plot to highlight possible patterns the CNN may be detecting. One order set, figure 4.3, has both rows and columns correlated while the other, figure 4.4, has them anti-correlated. How the algorithm constructs objects using the correlated order is visible and so is dispersing them into tiny objects using the anti-correlated order.



(a) Benign (b) Infected

Figure 4.4: Visual Plot of Anti-Correlated Samples with 3 Wide Entropy

4.1.3 IP-Traffic of Security Attacks: CIC-IDS-2017

** The material presented in this subsection will appeared in the proceedings on the 4th International Conference on Machine Learning and Soft Computing (MLSC 2023) in the article,* "Mi-CAM: Visualizing Feature Extraction Of Nonnatural Data", *co-authored with Ram Krishnan, Ph.D.*

The CIC-IDS-2017 data set has captured live, raw IP traffic that is intentionally subjected to various forms of attack vectors. There were 12 attack classes, ten of which were of a sizable sample. The sample count and break down by class is included with the results in Table 5.7 found in the next chapter. This traffic is compiled by session, with the sessions labeled benign or by attack class. Each packet in the session has the physical layer of the IP packet stripped, the first fourteen bytes, and only the following 160 bytes kept. If the original packet wasn't 174 bytes long, the remaining portion of the 160 bytes are supplied with zeros. The first ten packets of the session are then compiled in order of transmission, and if there aren't ten packets, the remaining are filled

with zeros. The result is a 10x160 byte grid.

$$\mathbf{X}_f = \begin{bmatrix} & b_1 & b_2 & \dots & b_{B=160} \\ p_1 & x_{b_1p_1} & x_{b_2p_1} & \dots & x_{b_Bp_1} \\ p_2 & x_{b_1p_2} & x_{b_2p_2} & \dots & x_{b_Bp_2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ p_{P=10} & x_{b_1p_P} & x_{b_2p_P} & \dots & x_{b_Bp_P} \end{bmatrix}$$

This is the basic single sample from the data set before it is reorganized into a 40x40 square. The current order of this gird is IP specification for the columns and transmission time for rows. Transmission time is a natural order, an instance in a sequence, but IP specification, human defined, is a nonnatural order. Is IP specification the best order? Will statistical correlation on the data be a high performing order? These are the questions this study is trying to resolve. The hypothesis is that this structural order isn't a preferred order for CNN analysis, and if the grid is re-ordered using a statistical relationships, those that are found in images, it should be able to improve the CNN performance. Note that statistical analysis between bytes, a nonnatural axis, is not dependent on multiple samples, so the use of the simpler independent correlation as the ordering function is appropriate.

To test these hypothesis, first 100 random column ordering schemes were generated to process and compare. Since the calculations between bytes are independent per sample the function Equation-4.1 was used and the ordering algorithm shared in Algorithm 3.1. To diversify the number of ordering options available to analyze the correlation relationships were used within different data subsets. The first data set was total of all samples. Next, the study separated between the benign and maleficent and used the correlation of each of these data subsets. Then subsets of each attack types were extracted and generate correlated orderings from each. The idea is to see if it is possible to focus on a specific artificial objects by re-arraigning the order to match the correlation generated from that subset sample type. Also generated were absolute values of the

correlations ($\rho_{ABSp_ip_j} = \sum_{j=1}^{M} \left|\rho_{m_kp_ip_j}\right|$) and anti-correlation ($\rho_{ANTIp_ip_j} = \sum_{j=1}^{M}(1 - \left|\rho_{m_kp_ip_j}\right|)$) orderings for each of the data sets.

This resulted in 146 ordering schemes to analyze. After reordering, the samples were then translated into a 40x40 grid by splitting the 160 bytes into four sections and stacking them on top of each other in order. They were randomly reordered the samples and split them into 60% training, 20% validation, and 20% testing sets.

The reordering process of the IP packet data was written in python and processed sequentially on one of the test beds. The process for determining the correlations took approximately an hour for the entire data set, while because of the smaller population sizes, the time to generate all twelve maleficent classes plus benign subset correlations took approximately 18 minutes. The time it took to generate the 146 ordered lists from the correlation values and the random orderings took approximately 12 minutes. In contrast, the time it takes to prepare the data for the CNN, generating the tensor-flow records from the data set for all 146 ordering schemes took 33 hours, and the processing time for the CNN to train for ten epochs on the ordering schemes took close to 300 hours. So in relative terms, the ordering process took less than two hours, while CNN prep and training took over 150 times that.

This data set has also been labeled by type of attack, so in the results a classification task is included. It is strongly suspected that performance for classification tasks will reflect the outcomes seen in the identification tasks.

4.2 Test Beds

The test beds used for the research contained in this document were three server class desktops with additional GPU capability. The machines were running Ubuntu OS and Tensorflow™with Tensorboard™under Python as the underlying engine to perform the CNN analysis. Comparing

between these machines, the Tesla had the ability to handle larger CNN models with two cores and more GPU memory, while the GeForce machines would process faster with the later CUDA capable features. The analysis was run using a desktops with the following specifications:

4.2.1 GeForce GTX Machine

- Central Processor Unit: Intel©Core™i7-8700 CPU @ 3.2 GHz x 12
- Memory: 16.0 GB
- Graphical Processor Unit: GeForce™GTX 1070i/PCIe/SSE2
- OS: 64-bit Ubuntu©22.04.2 LTS (Gnome 42.4)
- CUDA™: 11.6
- Python: 3.10.6

This desktop was purchased and assembled locally, within the lab.

4.2.2 GeForce RTX Machine

- Central Processor Unit: Intel©Core™i9-12900K x 24
- Memory: 64.0 GB
- Graphical Processor Unit: Nvidia GA102 [GeForce™RTX 3080Ti]
- OS: 64-bit Ubuntu©22.04.2 LTS (Gnome 42.4)
- CUDA™: 11.6
- Python: 3.10.6

This desktop was purchased and assembled locally, within the lab.

4.2.3 Tesla Machine

We run our remaining CNN analysis using a desktop with the following specifications:

- Central Processor Unit: Intel©Core™i7-8700 CPU @ 3.2 GHz x 12
- Memory: 15.5 GB
- Graphical Processor Unit: GeForce™GTX 1070i/PCIe/SSE2
- OS: 64-bit Ubuntu©20.04.2 LTS (Gnome 3.36.8)
- CUDA™: 11.1
- Python: 3.8.10

This desktop was purchased and assembled locally, within the lab. Extra care was needed to re-engineer the case airflow. The Tesla is passively cooled but a heat generator. Initial attempts to use it resulted in immediately overheating the GPU card and halting the machine. To get the processor to perform properly airflow was increased by forcing it through the GPU card frame by increasing the air pressure within the case using high performance fans.

4.3 CNN Architectures

** The material presented in this section previously appeared in the proceedings of the 4th International Conference on Computing, Communications and Security (ICCCS 2019) and the proceedings of the International Conference on Secure Knowledge Management in the Artificial Intelligence Era (SKM 2021) in the articles,* "Analyzing CNN Model Performance Sensitivity to the Ordering of Non-Natural Data" *and* Analyzing CNN Models' Sensitivity to the Ordering of Non-Natural Data, *co-authored with Ram Krishnan, Ph.D.*

Next is a discussion of the various CNN architectures the hypothesis is tested with. They were chosen based on two criteria, were they research significant on their release, and are they

feasible to apply in this application. The first criteria was covered in the related chapter 2 earlier in this document.

Initially the research examined the use of a shallow CNN model, LeNet with $relu$ as an activation function. This model was used since it is one of the first CNN models to make major milestones of object identification in image analysis, but is relatively simple to construct and easy to understand. It is described in the first subsection below.

I was then tested to determine the statistical relationship hypothesis would hold true with other forms of CNN. Initially experiments with ResNet-50 found that the training times took longer per epoch and more epochs than a LeNet-5. LeNet-5 would usually saturate training in 20 epochs, but Resnet-50 would take as long as 50. A shift to Auto-Keras ©was made and within by 20 epochs it would settle on a shallow CNN with a couple of dense layers, much like LeNet, but fail to produce any meaningful results.

To determine which CNN models would behave, a modularly broad but targeted approach as provided by re-coding the test ground to use the recently released Keras ©application set of deep learning models. Using a limited set of ordered experiments every model was tested for training saturation. Because of the methodology, using the same data set for the different models was simply changing the model name within the script. Post calculation analysis found five models that would saturate training much quicker than the others, within three epochs, so those were chosen to compare. They are described describe them below in order of their release date.

4.3.1 LeNet

This research examines two versions of LeNet. The first is LeNet-5 was used in analyzing the process metric malware data. The second is another LeNet-5 used to analyze the maleficent IP traffic data. There were slight variations between these models. They were chosen specifically for their matching specifications in research related to these data sets.

The LeNet-5 model for detecting malware in process metrics comprises of:

- Two Convolutional layers where the first consisting of 32 nodes and the second of 64 nodes. Each convolutional layer uses a 3x3 filter and relu as the activation function.
- After each convolutional layer is a max-pooling layer with a downsize of a factor of two to one.
- Two dense layers, the first consisting of 1024 nodes and the second 512 nodes. Relu was also used as the activation function for the dense layers.
- Predictive layer used binary cross entropy loss function.
- Training was processed for 20 epochs with a batch size of 64.

The LeNet-5 model for maleficent detection in IP packet flows comprises of:

- Two Convolutional layers where the first consisting of 32 nodes and the second of 64 nodes. Each convolutional layer uses a 3x3 filter and relu as the activation function.
- After each convolutional layer is a max-pooling layer with a downsize of a factor of two to one.
- Two dense layers followed by dropout. The first consisting of 4096 nodes and the second 1600 nodes. Relu was also used as the activation function for the dense layers.
- Predictive layer which used a softmax function for binary or multi classification.
- Training was processed for 10 epochs with a batch size of 8.

Complete model details as programmed and used in these experiments can be found in Figure A.1 which shows the layout of the Lenet CNN model and the layers parameters are found in Table A.1 and Table A.2 within Appendix A.

4.3.2 Inception

The Inception model of CNN took some novel approaches to manage the challenges previous CNN were encountering. This first had to do with image scale. Objects trained at one scale had difficulty in identifying those same objects at a different scale. To overcome that challenge, they replaced single convolutional layer with a parallel set of convolution layers, and then aggregated the results for the next layer.

The second thing they do is introduce depth and width *(aka point)* separable convolutions. This takes the understanding that a NxN matrix can be constructed by multiplying a $1xN$ vector with a $Nx1$ vector. Normal convolutions are performed with filters that are as deep in channels as the layers input source. This mans that if the number of input data points is $P = WxHxD$ then an NxN filter would have $NxNxD$ parameters.

By transforming the filter into two, using depth and width wise separation, the result is two filters with fewer parameters. One has filter has $NxNx1$ parameters and the other has $1x1xN$. That means unless $N = 1$ the number of parameters in the separated filter solution is fewer than the non-separated filters. This effect is greater deeper within the CNN layers as the third channel, depth, increases. By reducing the parameter count of the filter, but maintaining mathematical cohesion, you reduce computational redundancy and increase performance.

Version three also included some new concepts with regards to training and optimization. The first is the *RMSProp* optimizer algorithm. This helps during training, adjusting the learning rate depending on the gradient change. By comparing the current gradient with a moving average the learning rate can be adjusted accordingly. RMSprop uses the squared average, hence the name *Root Mean Square*. They also included *batch normalization* between classifiers and include *label smoothing*. Both of these regularizing features helped reduce over fitting when the CNN become over-confident in a single result.

The layers of Inception are grouped in modular stages. The first two stages are a linear series of convolution and max pool layers, one feeding the next. These stages are designed to construct lower level features. This is followed by a set of Inception stages. These branch out and perform a group of convolutions and average pooling layers in parallel, and then aggregate the results into a single concatenated feature set for the next stage. Since there is some modularity to these Inception stages they are often grouped together into five categories.

Once the Inception stages are finished, the final stage is the decision, which in our scenario is a a basic 6144 node dense network with a flattening and 50% dropout as a preamble. The layer flow is shown in figures B.1-B.8 with layer details found in table B.1 within Appendix B.

4.3.3 ResNet

One of the major refinements in CNN architecture development is the concept of the skip or residual connections which are introduced with Residual CNN or ResNet for short. These skip connections re-introduce the input of a CNN stage of layers by linking the source directly to the output, maintaining the initial information integrity for the subsequent layers and stages.

It does this through two processes, an *Identity Shortcut* and a *Projection Shortcut*. The identity shortcut is simply adding the input values to the output values, and requires that the vectors match in dimensions. The projection shortcut performs a convolution on the input to transform the dimension to match the output dimensions. It accomplishes this using either strides or depth wise convolutions.

It was determined by adding these skip connections, and re-introducing the source data to the following layers, that one of the major impedance in CNN training was alleviated, the *Vanishing Gradient*. This occurs during the back propagation step in training, when the difference int gradient weighs within later layers are repeatedly multiplied by small values nullifying gradient within the earlier stages so no additional learning is achieved. By re-introducing the input vectors to the later

layers, it appears the later layers are able to carry though the changes in gradient to the earlier layers through these skip connections.

ResNet architectures come in different sizes depending on the number of layers they use. This study initially attempted to use ResNet model of several sizes, but found the larger models' with erratic results and taking substantially longer to saturate training. Selected was the smallest version published, ResNet-18 and it produced similar erratic results, but trained saturated punctually. This model as included in our analysis.

As the name suggests, there are 18 convolution layer in it. These are grouped in 10 stages. The first stage is feature pre-fetching, followed by 8 stages consisting of two convolutions paired with a skip connection. The third, fifth, and seventh stages have projection connections while the remaining have identity shortcuts. All convolutions are 3x3 full vector (no separation). All stages use a 1 stride convolution initially, but those stages with a projection shortcut use a stride of two for the second convolution.

Once the residual stages are finished, ResNet includes a concatenation of an average and max pooling of the results. The final stage is the decision, which in our scenario is a a basic 80 node dense network with flattening and 50% dropout as a preamble. To test why certain behaviors are noticed when analyzing data with ResNet-18, two other models were design to compare the differences between the pooling layers. One only uses the max pooling operation, and the other just uses the average pooling operation. The remaining layers the model is unchanged. This provides three ResNet models to analyze, ResNet-18, ResNet-18max, and ResNet-18avg. The layer flow of the normal ResNet-18 is shown in figures C.1-C.5 with layer details found in table C.1 within Appendix C.

4.3.4 Xception

The Xception net is a combination of the concepts found in the Inception and ResNet models. It uses similar shortcuts as ResNet, and separate out the convolutions like Inception, but they only include the depth separable convolutions on the normal non-skip path and the width separable convolution on a projection versions of the bypass paths. The resulting paths are summed like they were in ResNet for the next stage input. By including these features they were able to not only out achieve the accuracy of these models, but did so at higher efficiency with fewer parameters.

It has 15 convolution stages. First an initial feature prefetch stage followed by 12 shortcut stages. The first, second, third and twelfth stage use projection shortcuts, the rest use identity. Once the shortcut stages are finished, there are a pair of depth separable convolutions. The final stage is the decision, which in this scenario is a a basic 162 node dense network with flattening and 50% dropout as a preamble. The layer flow is shown in figures D.1-D.10 with layer details found in table D.1 within Appendix D.

4.3.5 MobileNet

MobileNet is as it's name implies, was designed for a small footprint for IOT applications. They capitalized on the separation concept and constructed several versions with minimal parameter counts. Unlike the other recent models, it stays true to the sequential form of the early CNN models. It also uses two other multipliers that reduce the computation cost. These parameters are called the width and resolution multiplier, and are between 0 to 1.

There are several versions of MobileNet, but this research only found the first to saturate in a reasonable amount of time. It consists of 15 stages. The initial prefetch stage, followed by 13 convolution layers, and finally a decision layer. All of the convolution layers consist of a depth wise $(3x3)$ filter followed by a point $(1x1)$ filter. The decision layer has a dense network consisting

of 8192 nodes with flattening and 50% dropout as a preamble. This is the one network that includes no pooling operations. The layer flow is shown in figures E.1-E.7 with layer details found in table E.1 within Appendix E.

4.3.6 DenseNet-121

DenseNets took the principals of ResNets a step farther, and instead of carrying the input from only the immediately previous block of convolutions to the next it concatenates together inputs from all of the previous convolution stages to the following convolutional stages input. The blocks are filter separated and organized into one $1x1$ depth filter followed by a second $3x3$ width filter with a padding of one to maintain matrix size.

The networks start with a lower level feature extractor stage, and then follows through a series of dense interconnected stages, interconnected with pooling layers to reduce dimensionality, and finally a dense decision layer after flattening the matrix and adding a %50 dropout. DenseNet-121 is used in this research comprises of 121 layers, one convolution in the prefetch stage, 119 within the dense stages, and the final decision layer. The dense stages are split in into 4 groups, each one incrementing in size. The layer flow is shown in figures F.1-F.25 with layer details found in table F.1 within Appendix F.

CHAPTER 5: EXPERIMENTAL RESULTS

This chapter discusses the experimental results. It starts with the new visualization tool and how it behaves with images. It then explores the malware process metrics, how order effects performance, including results from current and MiCAM visualizations tools. It is followed with an analysis of the maleficent internet protocol, an examination of ordering options and a display of the MiCAM plots of some samples.

5.1 MiCAM Images Results

** The material presented in this section will appeared in the proceedings on the 4th International Conference on Machine Learning and Soft Computing (MLSC 2023) in the article,* "MiCAM: Visualizing Feature Extraction Of Nonnatural Data", *co-authored with Ram Krishnan, Ph.D.*

This section reviews the resulting plots from the visualization tool as it is is processed on a CNN analyzing the MNIST handwritten image data set. The resulting MiCAM plots are large when compared to other CAM plots. They are usually vertically aligned following the model layout. Since not only the convolutional layers, but the pooling, adding and concatenation layers, along with the final flatten and dense layers at the end of are all plotted, the combined plot contains a visual representation of the parameter intensities for each layer. For example, DenseNet-121, with 121 convolutional layers has a total of 429 individual layers within the model. For brevity the diagrams are not all included but can be found on GitHub at: `https://github.com/rklepetko/MiCAM.git`. Snapshots of elements that illuminate the value of this visualization tool are shared below.

Examined first is the LeNet-5 MiCAM plot (Left side of Figure 5.1) which clearly shows how the convolution layers build the identifying features. Examining the dense layers closely it can be seen the variation in the color pixel intensities relate to specific features the network has

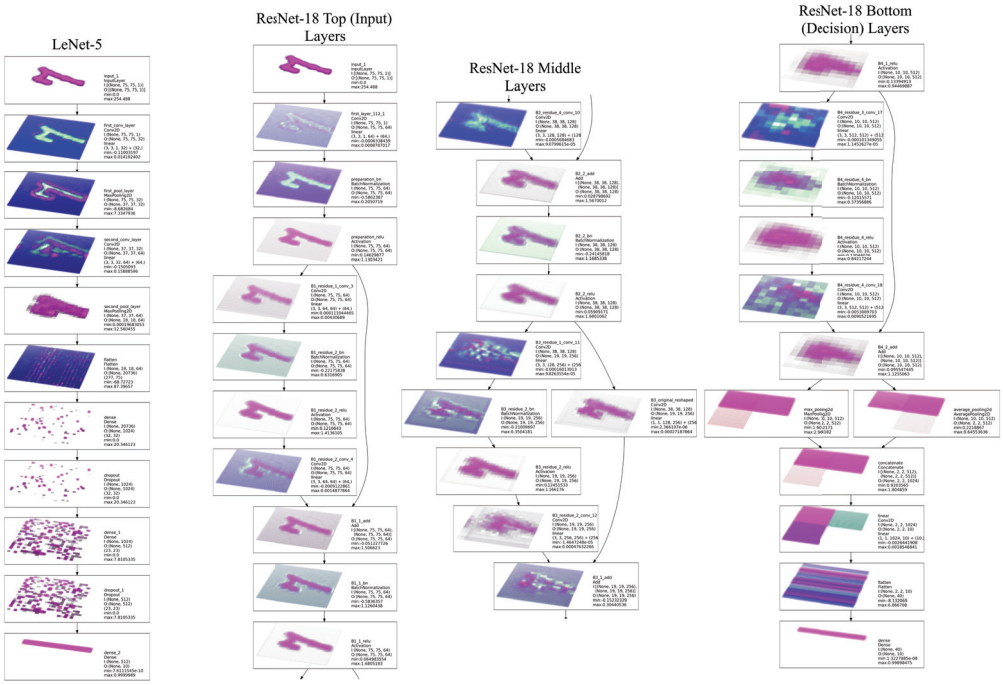


Figure 5.1: MiCAM Plots of LeNet-5 (left one) and MiCAM Plot Clips of ResNet-18 (right three) analyzing an MNIST sample“7"

identified.

It is even clearer when examining ResNet-18 MiCAM plot (the right three plots of Figure 5.1) as shown in the top, or input stages, the middle of the model, and the final bottom or decision layers. It’s seen in these graphs how the residual links re-introduce features extracted from earlier layers. It is also visible within the final layers how the ResNet-18 network collapses the number of extracted features to relatively few, 40, as compared to LeNet-5 which was 20736.

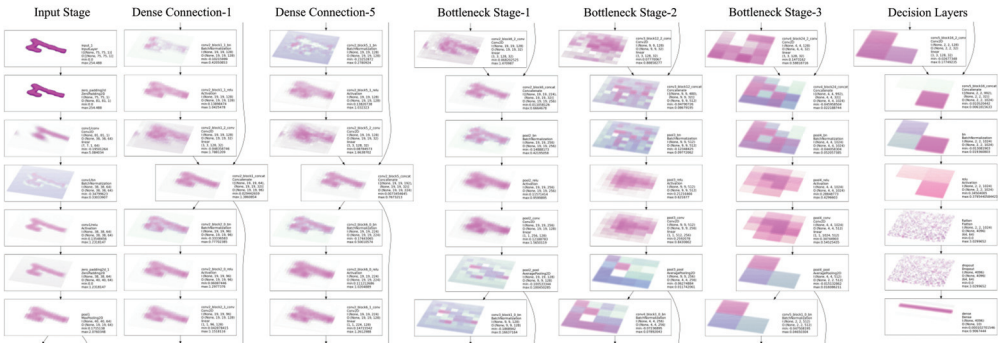


Figure 5.2: Clips of MiCAM Plot from a DenseNet-121 analyzing an MNIST sample “7"

Examining the DenseNet-121 MiCAM plot of the same sample (Figure 5.2), included are 49 of the 429 layers. From left to right are shown the details of the input layers, the first and last dense connection before the first bottleneck, the three bottle neck stages, and the final decision layers. In the dense connection plots, the reintroduction of the input stages initial features (outline of a "7") is visible as the data cascades through all the way to the first bottleneck stage, maintaining a higher level of details for feature extraction precision. Also visible is that the bottle neck layers compile and extract the features for discrimination later.

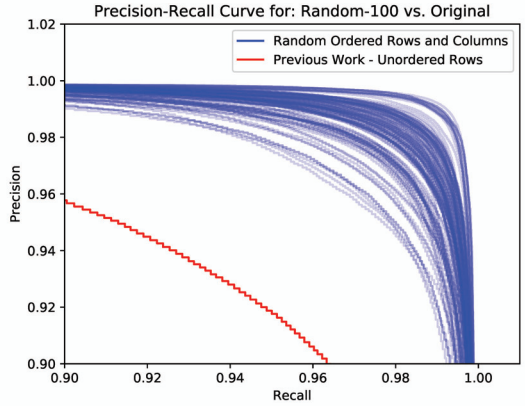
5.2 Process Metrics of Malware Infections

** The material presented in this section previously appeared in the proceedings of the 4th International Conference on Computing, Communications and Security (ICCCS 2019) and the proceedings of the International Conference on Secure Knowledge Management in the Artificial Intelligence Era (SKM 2021) in the articles,* "Analyzing CNN Model Performance Sensitivity to the Ordering of Non-Natural Data" *and* Analyzing CNN Models' Sensitivity to the Ordering of Non-Natural Data, *co-authored with Ram Krishnan, Ph.D.*

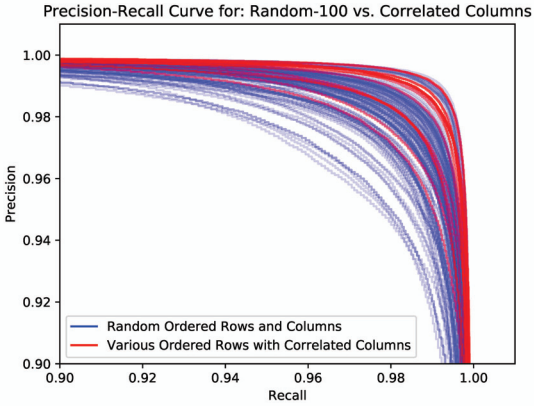
This section shares the order analysis experiments results with the process metric malware data set. It begins with detail results from using a LeNet-5 model in the initial experiments, then share results from the remaining modesl. It then shares visualization tool plots, including MiCAM, of data samples in different ordering schemes.

5.2.1 LENET-5

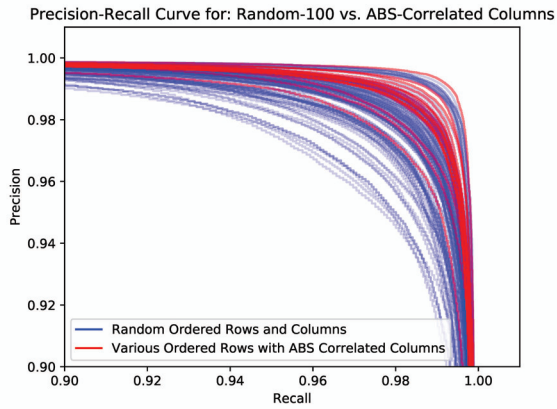
This subsection shares the initial results found when a LeNet-5 was used to analyze the process metric malware data set. Since malware infections are rare compared to the normal machine activity it is appropriate to compare the precision/recall curves versus accuracy which would weighted by the excessive benign activity. P/R curves also inform on the relative precision required to minimize false positives which in security scenarios have harmful results. In figure 5.3 and 5.4 there is



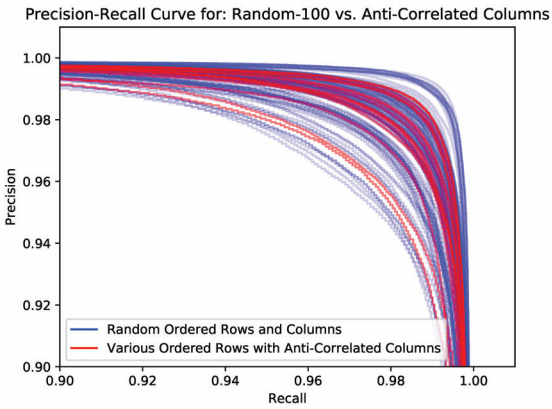
(a) Static Random-100 vs. Original Structural / Dynamic Order



(b) Static Random-100 vs. Static Correlated Columns Order



(c) Static Random-100 vs. Static ABS-Correlated Columns Order

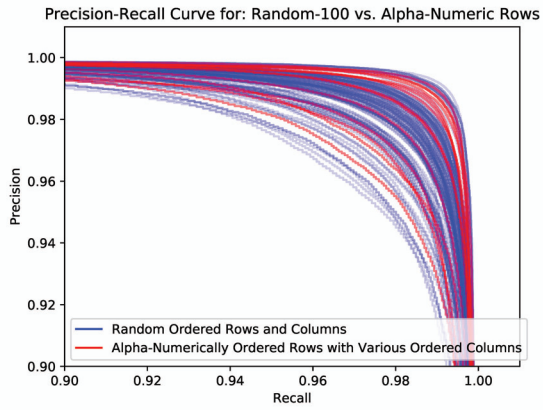


(d) Static Random-100 vs. Static Anti-Correlated Columns Order

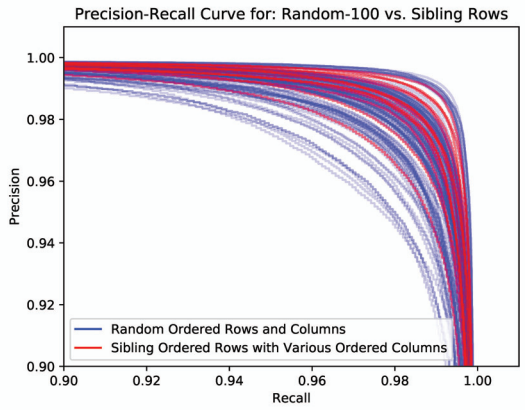
Figure 5.3: 100 Randomly Ordered Rows and Columns PR curves vs. Different Columns Ordering Sets

a sample spread in performance for 100 (10x10) random variations of the row and column ordering of the source data matrix. They are blue as a backdrop while the other row and column ordering that are methodically defined and displayed are in pink.

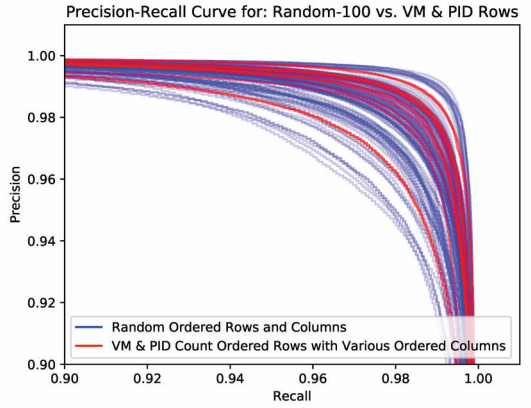
In figure 5.3a these static but random order results are compared to the original results in the paper [11] by Abdelsalem et al. where row ordering between experiments was not taken into consideration, so the was some dynamic variability in the row order. It is obvious that maintaining a static ordering of the rows between experiments is imperative to reach high performing results.



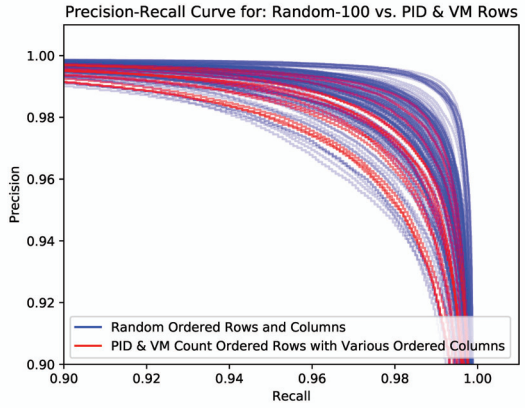
(a) Static Random-100 vs. Alphanumeric Row Order



(b) Static Random-100 vs. Static Sibling Related Row Order

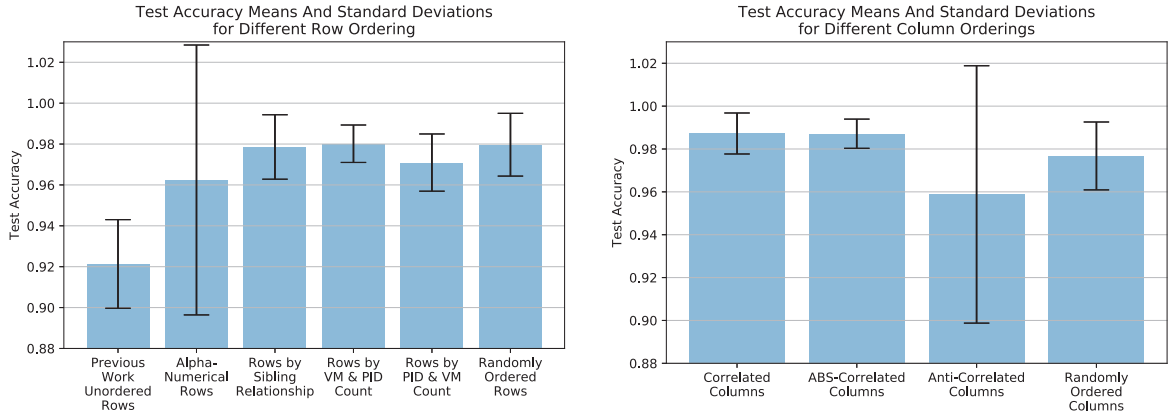


(c) Static Random-100 vs. Static Related VM ID and Parent ID Count Order



(d) Static Random-100 vs. Static Related Parent ID and VM ID Count Order

Figure 5.4: 100 Randomly Ordered Rows and Columns PR curves vs. Different Row Ordering Sets



(a) Accuracy of different row ordering

(b) Accuracy of different column ordering

Figure 5.5: Test Accuracy Mean and Standard Deviation for Different Row and Columns ordering

The graphs following compares the 100 random curves with different metric column orderings. For each methodical metric column ordering all of the different process row orderings were included so the column order can be analyzed with some independence from the rows. In figure 5.3b correlated metric columns are compared with other row orderings. Correlated columns appear to reside in the upper spectrum of the randomly ordered curves.

The absolute value of the statistical correlation between metrics graphs are examined in 5.3c. In this set of curves not only do they reside the upper spectrum of the random curves, but possibly an optimum result appears, one better than any random ordering.

Next examined is the counter example, Anti-Correlated columns ordering. As expected, in 5.3d the curves do not align to the upper end of the randomly ordered curve spectrum, and most curves show rather poor results in comparison, residing on the lower end of the spectrum.

Next examined are the results for defined row orderings. Again in each methodical row order case all of the various column orderings are used for analysis independence between the rows and columns.

The results for the orderings based off of initial alphanumeric relationships are found in

graph 5.4a. It shows that alphanumeric row ordering results are spread relatively evenly among the randomly ordered curves.

Examining the results for the orderings that are based off of sibling relationships in graph 5.4b the results are again spread among the random curves. In general they reside on the higher than average end of the randomly ordered curve spectrum, but nothing stands out as exemplary.

Studying the results for the orderings that are based off of the number of machines that make the process call followed by the number of execution or Process ID counts in graph 5.4c it shows the results are spread among the random curves. Again they are higher than average, but nothing stands out as exemplary.

Studying the results for the orderings that are based off of Process ID counts followed by number of machines that call the process in graph 5.4d it shows the results are again spread among the random curves. These curves though reside on the lower set of random curves, informing that this is not a desired ordering.

As a comparison for other studies that only take the accuracy in consideration for evaluation, included are the following two graphs that display the means and standard deviation spread for the various row and column ordering performance accuracy. In 5.5b is shown the accuracy spread for the various row ordering and 5.5a shares the accuracy found different column ordering including the non-ordering from the previous research by Abdelsalam et al. [11].

5.2.2 Inception Net

This section shares the same experiment results using the process metric malware data set, but instead using the Inception Net CNN model. In figure 5.6 all of the PR curves are shown as the light pink background while the dark lines represents the subset of PR curves that are generated running the model using a specific order set. Note that these plots of the Inception Net results

are scaled in to 50%-100%. It is easy to identify that Inception prefers correlated columns and ABS-correlated rows, while and correlated rows offer another well performing alternative. Anti-correlated rows should be avoided.

5.2.3 ResNet-18

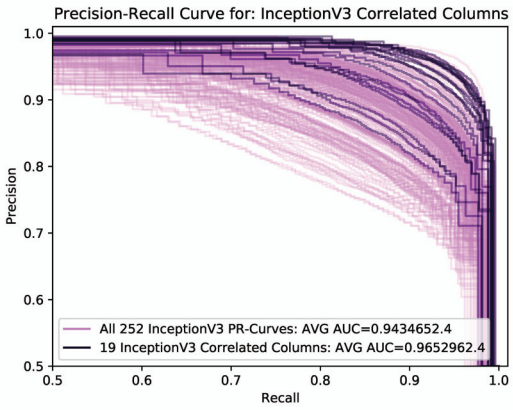
Following are the results from ResNet-18 in figure 5.7 running the same experiments. Note that these plots for ResNet-18 are at scaled at 0-100%. It's obvious by the wide varieties in PR curves that this model is very susceptible to minor changes in order. For this model anticorrelated rows and columns perform better than average, while the other orderings have only minor variation around the poor average.

5.2.4 Xception Net

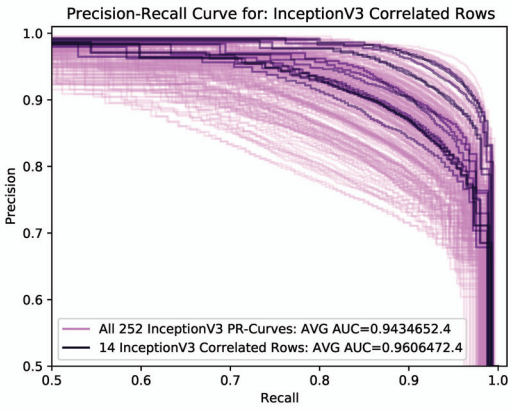
The next model, Xception Net, results are found in figure 5.8. Note that this model seems order ambivalent with near perfect results every time, but it is visible that the statistically related order performs well if not better than average. Only the ABS-correlated columns fell below average, but this was by only 0.0007 AUC. It appears the best performance is found using correlated rows and anti-correlated columns.

5.2.5 MobileNet

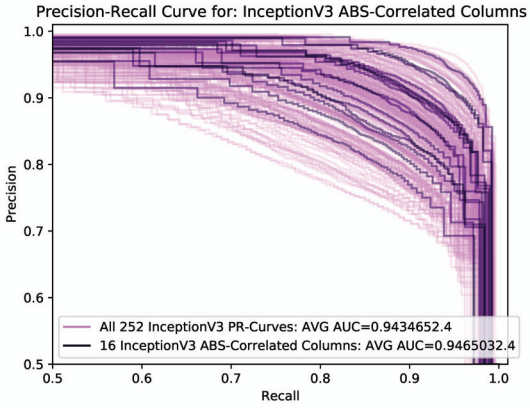
MobileNet is included as a small format option with it's intention to be used in mobile devices. The results are found in figure 5.9. Like ResNet-18, MobileNet seems to be very reactive when there are changes in the order so these plots are at full zoom, 0-100%, to observe all of the curves. Unlike ResNet-18 (0.898 AUC) it appears to respond better on average (0.958 AUC). It also appears that it loves any statistical relationship in column order, but choosing a random order is better than any



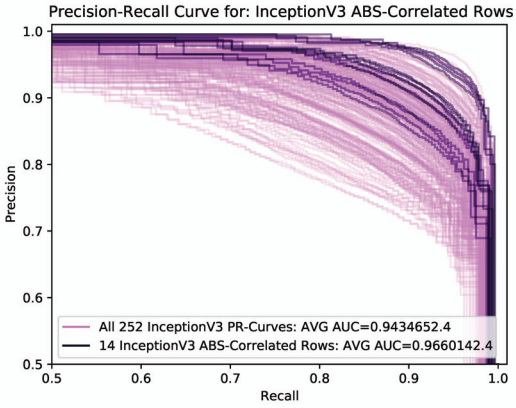
(a) Correlated Columns



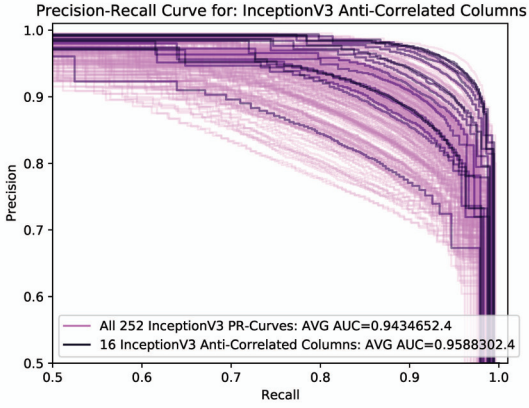
(b) Correlated Rows



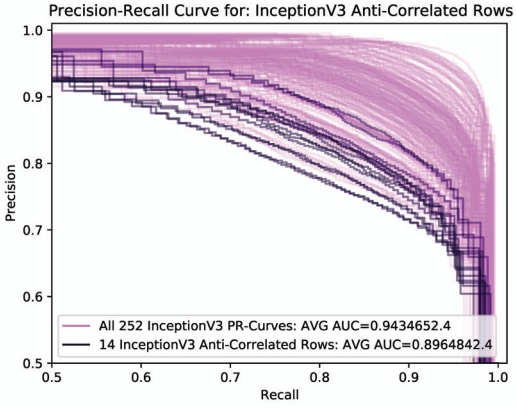
(c) ABS-Correlated Columns



(d) ABS-Correlated Rows

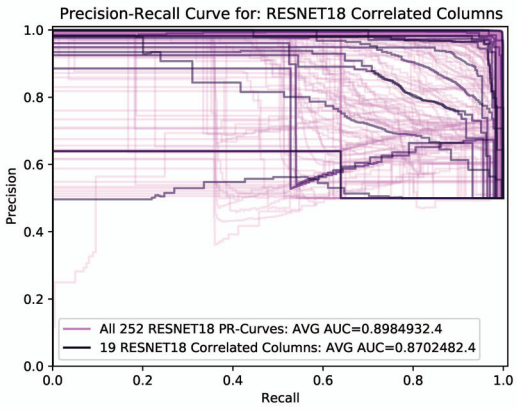


(e) Anti-Correlated Columns

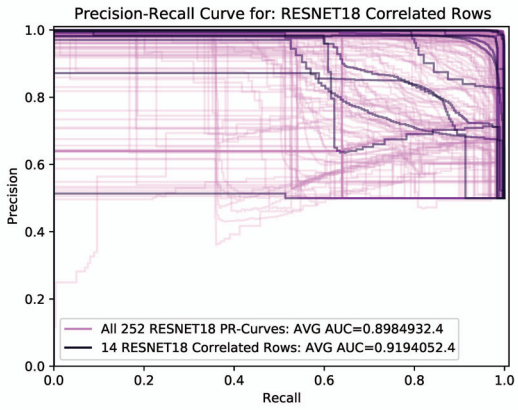


(f) Anti-Correlated Rows

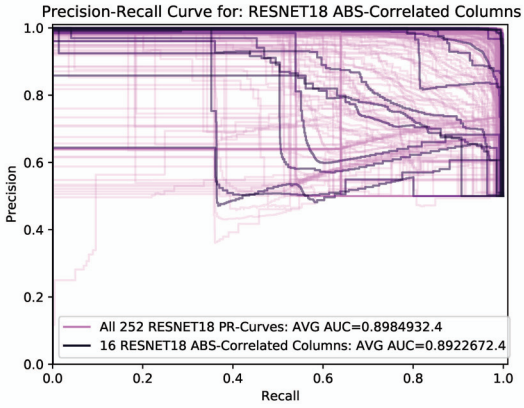
Figure 5.6: Inception V3 CNN model PR Curves



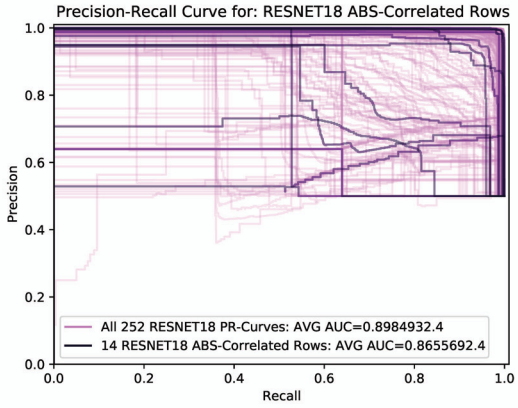
(a) Correlated Columns



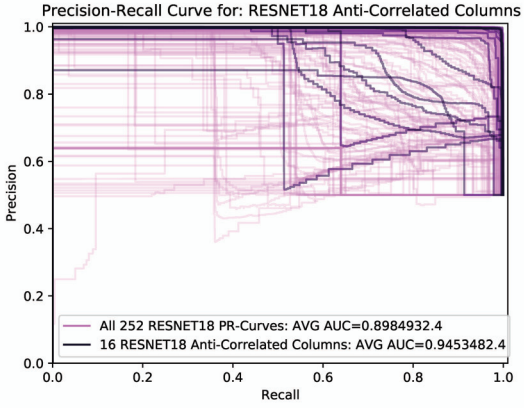
(b) Correlated Rows



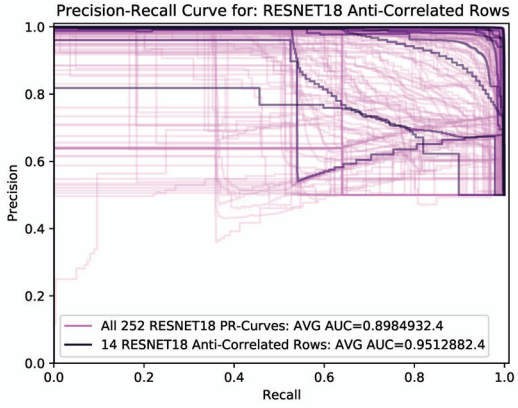
(c) ABS-Correlated Columns



(d) ABS-Correlated Rows

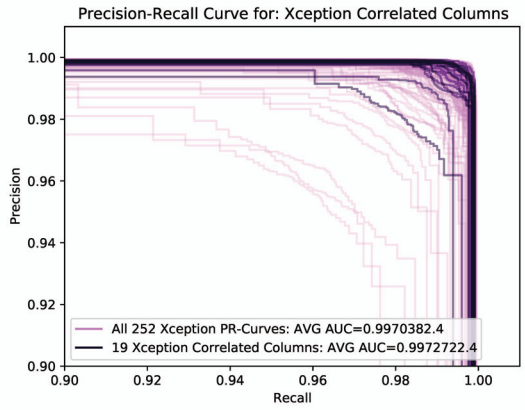


(e) Anti-Correlated Columns

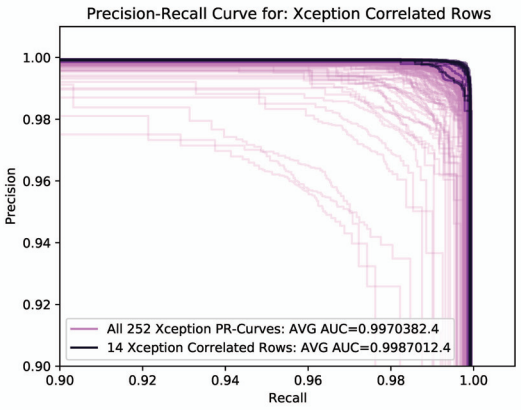


(f) Anti-Correlated Rows

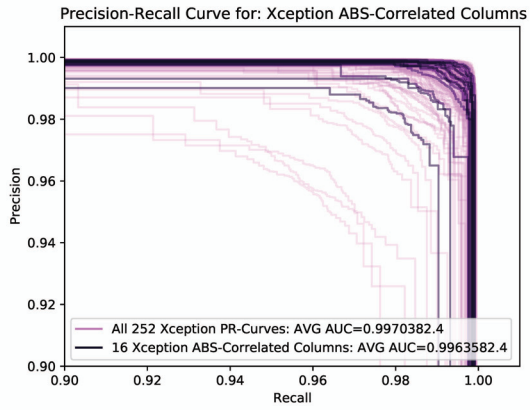
Figure 5.7: ResNet-18 CNN model PR Curves



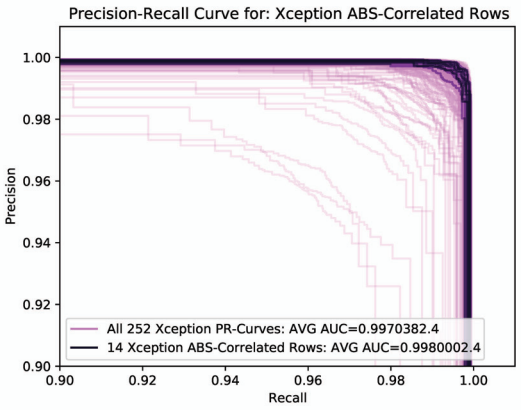
(a) Correlated Columns



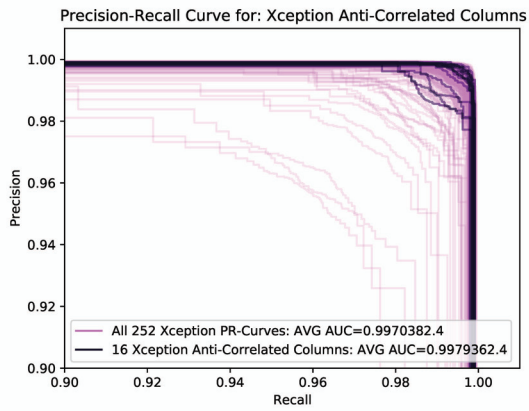
(b) Correlated Rows



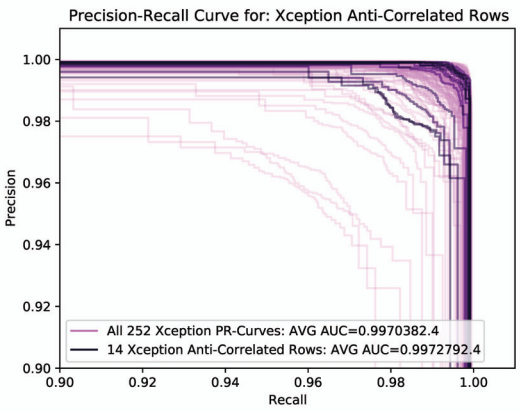
(c) ABS-Correlated Columns



(d) ABS-Correlated Rows



(e) Anti-Correlated Columns



(f) Anti-Correlated Rows

Figure 5.8: Xception CNN model PR Curves

Table 5.1: Process/Metric Malware Analysis Mean AUC for Precision Recall Curves

CNN Architecture	All Options	Corr Rows%	ABS Corr Rows%	Anti Corr Rows%	Corr Cols%	ABS Corr Cols%	Anti Corr Cols%
LeNet-5 *20 ep*	99.55	99.68	99.58	99.09	99.59	99.60	99.44
Inception-V3	94.35	96.06	96.60	89.65	96.53	94.65	95.88
ResNet-18	89.85	87.02	86.56	94.53	91.24	89.23	95.13
Xception	99.70	99.73	99.80	99.73	99.87	99.64	99.79
MobileNet	95.87	93.76	92.29	91.86	96.55	97.01	97.55
DenseNet-121	99.53	99.70	99.43	99.20	99.60	99.52	99.56

thing analyzed for row order.

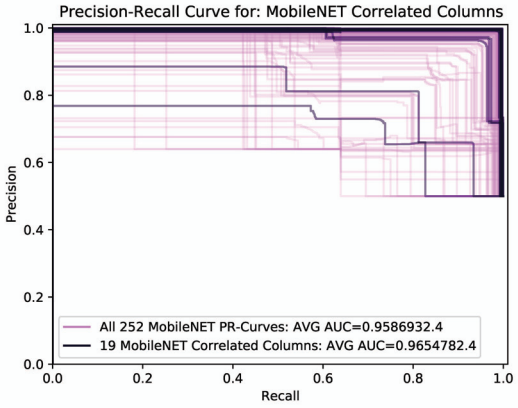
5.2.6 DenseNet

Our final model to examine, DenseNet-121, was analyzed and the results are found in figure 5.10. This like Xception had a vary high AUC regardless of row or column ordering, with almost near perfect results each time. Only a couple of curves drop below 97%, and the figures are zoomed in at 80%-100% for that reason. It displays that correlated rows and columns are the best option, but all of the statistical relationships provide an average if not better result.

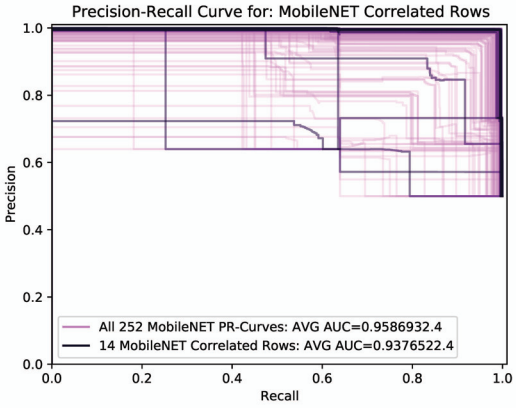
5.2.7 Model Comparison

For comparison, a breakdown of the different model performances while detecting malware from process metric samples are included in the following tables. First table 5.1, is the average (mean) area under the precision recall curves for the different ordering schemes along the both axis. The second table 5.2, is transformation of the same numbers but in relation to the percentage of improvement (or degradation) against the average performance for all of the ordering schemes.

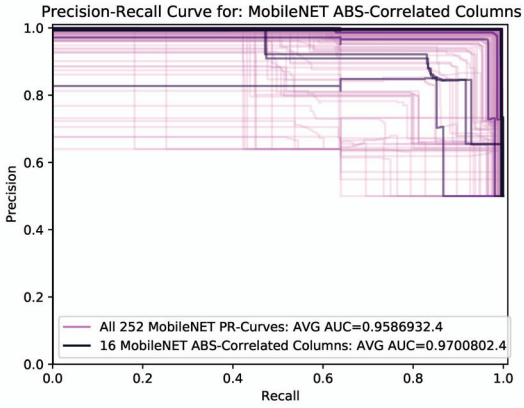
These tables clearly show that using correlation to order the shorter metric column axis always improved performance over the average for all models, and all but two of the models when



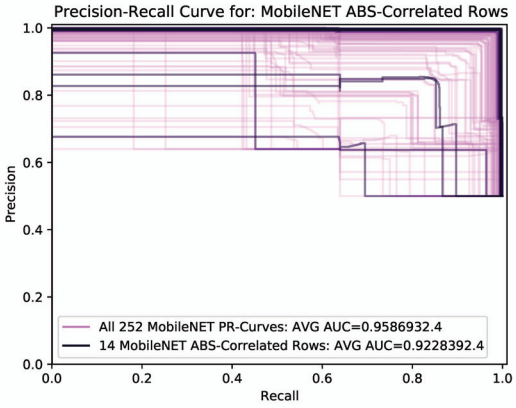
(a) Correlated Columns



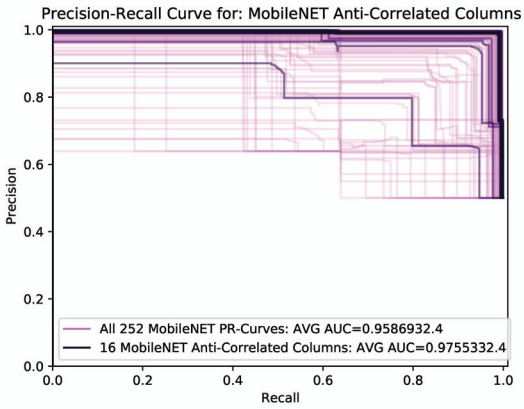
(b) Correlated Rows



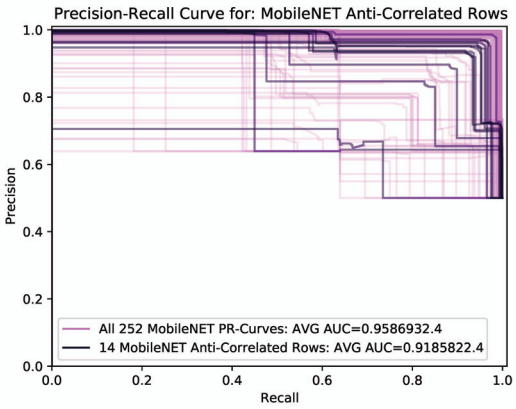
(c) ABS-Correlated Columns



(d) ABS-Correlated Rows

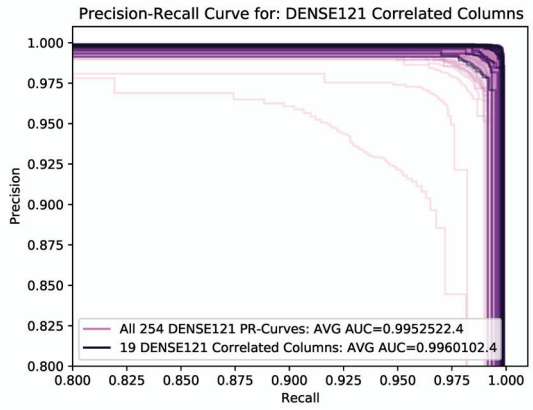


(e) Anti-Correlated Columns

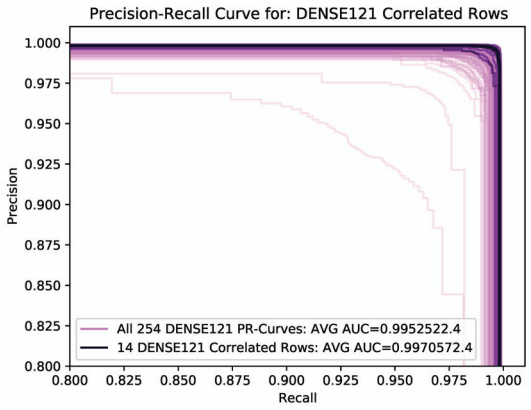


(f) Anti-Correlated Rows

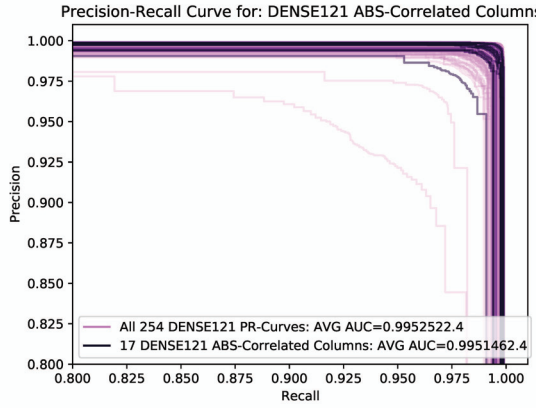
Figure 5.9: MobileNet CNN model PR Curves



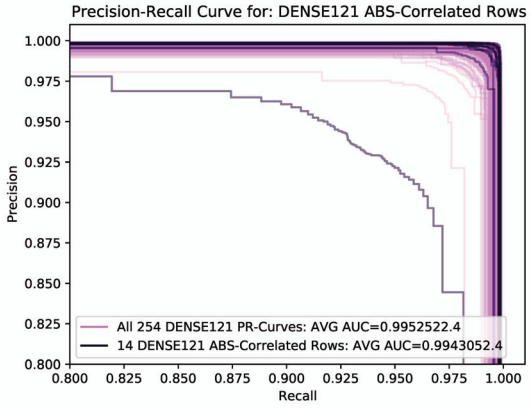
(a) Correlated Columns



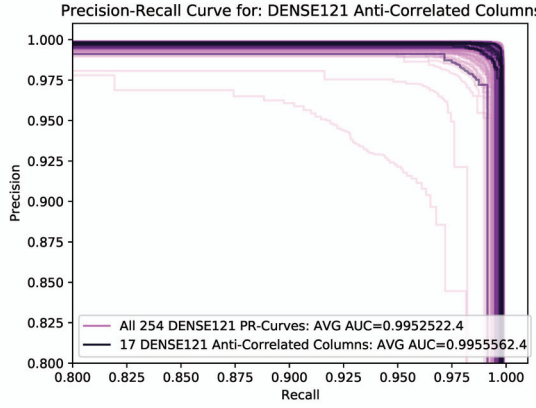
(b) Correlated Rows



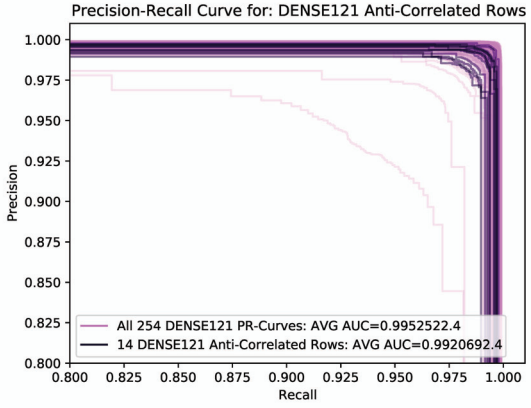
(c) ABS-Correlated Columns



(d) ABS-Correlated Rows



(e) Anti-Correlated Columns



(f) Anti-Correlated Rows

Figure 5.10: DENSE121 CNN model PR Curves

Table 5.2: Process/Metric Malware Analysis Percentage Improvement Over Average (Mean) Performance

CNN Architecture	100%-All Mean	Corr Rows%	ABS Corr Rows%	Anti Corr Rows%	Corr Cols%	ABS Corr Cols%	Anti Corr Cols%
Lenet-5 *20 ep*	0.45	28.89	6.67	-102.22	8.89	11.11	-24.44
Inception-V3	5.65	30.27	39.82	-83.19	38.58	5.31	27.08
ResNet-18	10.25	-27.88	-32.41	46.11	13.69	-6.11	52.02
Xception	0.30	10.00	33.33	10.00	56.67	-20.00	30.00
MobileNet	4.13	-51.09	-86.68	-97.09	16.46	27.60	40.68
DenseNet-121	0.47	36.27	-21.28	-70.21	14.89	-2.13	6.38

using correlation for the longer process rows. In particular the ResNet-18 and MobileNet showed significant degradation. To understand why this was the model structures were analyzed and of particular note was that the feature reduction/extraction process for these two models was different compared to the others. Most CNN models use several pooling layers through out the network for feature extraction and parameter reduction, but ResNet-18 uses a single pooling operation, sum of the max pool and average pool at the end of the network, while MobileNet uses depth wise and width wise convolutions with large strides.

To test if these structural difference have as affect on the ordering performance, two other experiments were run, but with a modified ResNet-18 architecture. The first removed the average pooling layer so feature extraction was performed by a max pooling layer only. The second does the opposite, and removes the max pool and leaves the average pool. These models are labeled ResNet-18max and ResNet-18min, and those results are shared later in section 5.2.10.

5.2.8 Analyzing Malware Results Using Existing Visualization Tools

** The material presented in this subsection previously appeared in the journal, Information Systems Frontiers (2022) ,* "Visualizing CNN Models' Sensitivity to Nonnatural Data Order", *co-authored with Ram Krishnan, Ph.D.*

For further analysis, several visualization tools are used on specific data samples from

Table 5.3: Process/Metric Malware Analysis Worst Four and Best Four Performing Order Schemes per CNN Architecture

CNN Architecture	Row Order (Processes)	Column Order (Metrics)	mAP	Model Rank
Inception-V3	Anticorrelation	Random5	%87.66	Worst
Inception-V3	Anticorrelation	Random4	%87.98	
Inception-V3	VMPID	Random2	%88.03	
Inception-V3	Anticorrelation	Random1	%88.04	
Inception-V3	ABS-Correlated	Anticorrelated	%98.33	
Inception-V3	Alphanumeric	Anticorrelated	%98.47	
Inception-V3	ABS-Correlated	Random5	%98.54	
Inception-V3	ABS-Correlated	Correlated	%98.68	Best
ResNet-18	Random1	Original	%50.31	Worst
ResNet-18	Correlated	Random9	%50.7	
ResNet-18	VMPID	Random1	%51.11	
ResNet-18	ABS-Correlated	Random1	%51.56	
ResNet-18	Random10	Random3	%99.96	
ResNet-18	Random10	Original	%99.97	
ResNet-18	PIDVM	Random6	%99.99	
ResNet-18	Random1	Random9	%99.99	Best
Xception	Random7	Random4	%96.32	Worst
Xception	Sibling	Random6	%97.11	
Xception	Random7	Random6	%98.04	
Xception	Random7	Random1	%98.41	
Xception	Correlated	Random6	%99.92	
Xception	Random3	Random8	%99.92	
Xception	Random4	Random1	%99.92	
Xception	Random3	Random5	%99.92	Best
MobileNet	Alphanumeric	Original	%58.92	Worst
MobileNet	ABS-Correlated	Random8	%62.09	
MobileNet	Alphanumeric	Random1	%62.11	
MobileNet	Anticorrelated	Random5	%64.65	
MobileNet	Sibling	ABS-Correlated	%99.8	
MobileNet	ABS-Correlated	Anticorrelated	%99.81	
MobileNet	ABS-Correlated	Correlated	%99.81	
MobileNet	Correlated	Random5	%99.82	Best
DenseNet-121	ABS-Correlated	Random5	%96.36	Worst
DenseNet-121	Alphanumeric	Random9	%97.13	
DenseNet-121	Anticorrelated	Random10	%98.43	
DenseNet-121	Random7	Random2	%98.52	
DenseNet-121	Alphanumeric	Random1	%99.85	
DenseNet-121	Random3	Random3	%99.87	
DenseNet-121	Alphanumeric	Correlated	%99.87	
DenseNet-121	VMPID	Random1	%99.87	Best

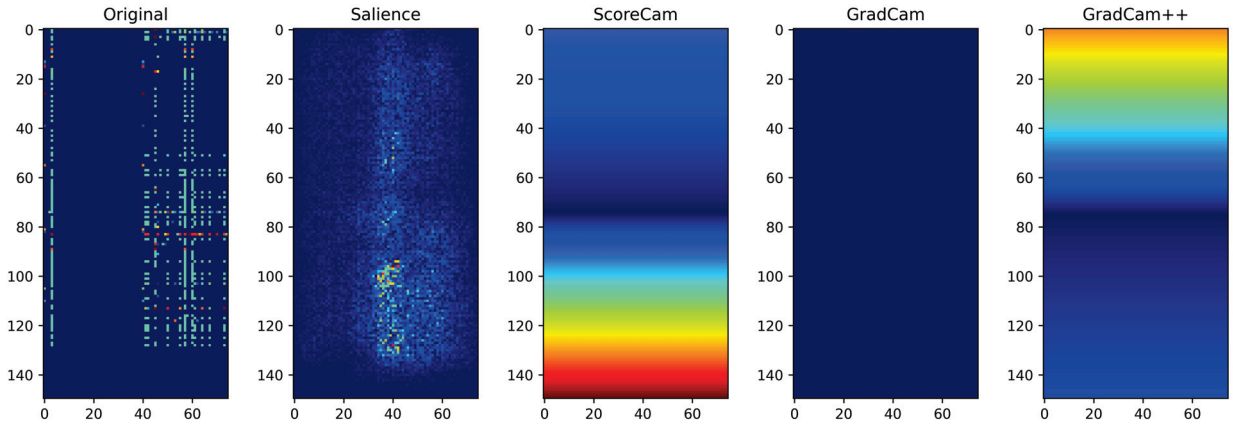


Figure 5.11: Inception-V3's Worst Order (Anticorrelated/Random-5) Benign Sample #159 Visualizations (Pred: 3.6e-9)

the testing set to explore what the CNN models are doing within the hidden layers. To see the differences between a poor-performing order and a good one, extreme options were chosen the from each model's results. In the Table 5.3 displays the worst and best ordering options for each model including associated mAP score achieved during testing. Then the two extreme options of each model, the highlighted rows, were then analyzed using *Saliency*, *GradCAM*, *GradCAM++*, and *ScoreCAM*.

With every model for the worst and best orders a set of five visualization plots was produced for a benign and infected sample. This was the same sample pair for all visualizations, testing sample #159 (of 6000+) labeled benign and testing sample #165 infected. The only difference was the order in which the rows and columns were arranged. In all experiments, these two samples were predicted accurately, though not always at 100%. Plots of these visualizations and a discussion of them are found per CNN model over the following pages:

When comparing the Inception-V3 visualizations from the infected sample with the benign sample in the worst order (Figure 5.11 and Figure 5.12, respectively), visible is the concentration of pixels within the Salience map that are insignificant, with infected showing a higher concentration. ScoreCAM shows a definite direction toward the upper half of the grid for benign with a counter direction for the infected. GradCAM shows it uses the whole grid in making a

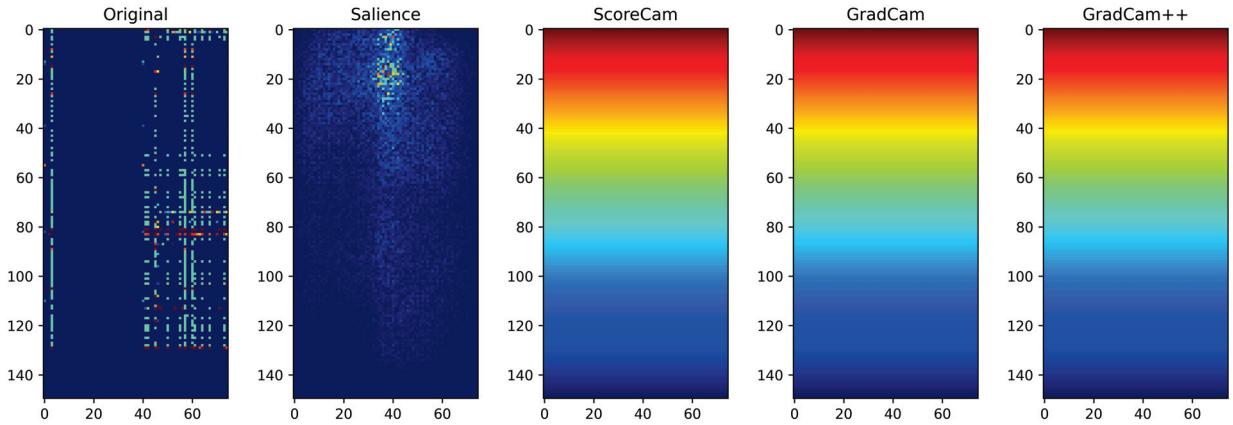


Figure 5.12: Inception-V3's Worst Order (Anticorrelated/Random-5) Infected Sample #165 Visualizations (Pred: 1)

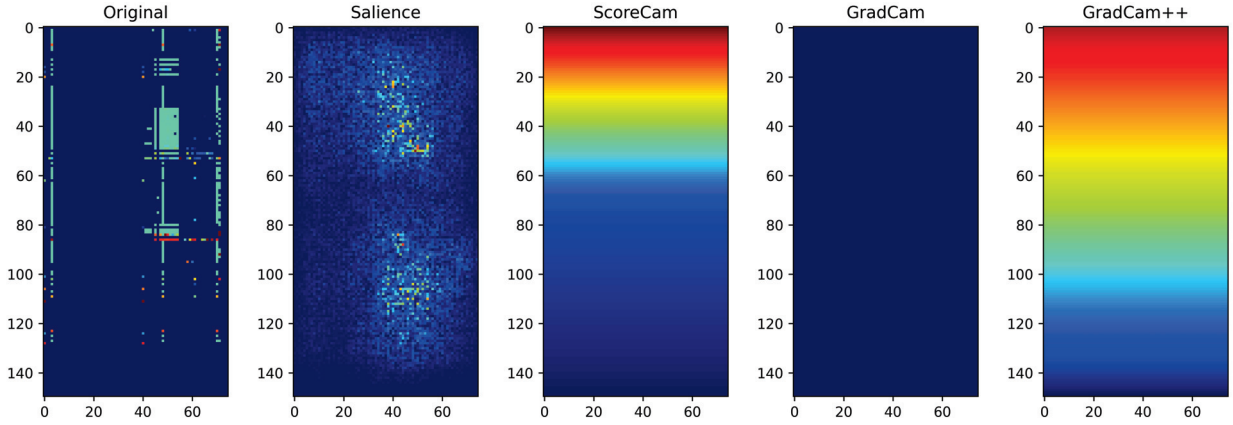


Figure 5.13: Inception-V3's Best Order (ABS-Correlated/Correlated) Benign Sample #159 Visualizations (Pred: 6.9e-10)

benign decision, while showing a similar response with the infected sample as ScoreCAM. GradCAM++ shows that the lower region has more influence over the benign decision than the infected sample.

In the best ordering for Inception-V3, the visualizations for the same samples are in Figure 5.13 and Figure 5.14. The Salience insignificant pattern is in clumps with benign comprising of two masses spread top to bottom, where as the infected sample has three. ScoreCAM produced what appears to be identical influence patterns for the infected and benign samples with the lower half having influence, whereas GradCAM informs the entire grid has some influence over the benign sample while in the infected, the lower half. GradCAM++ informs us that the lower half

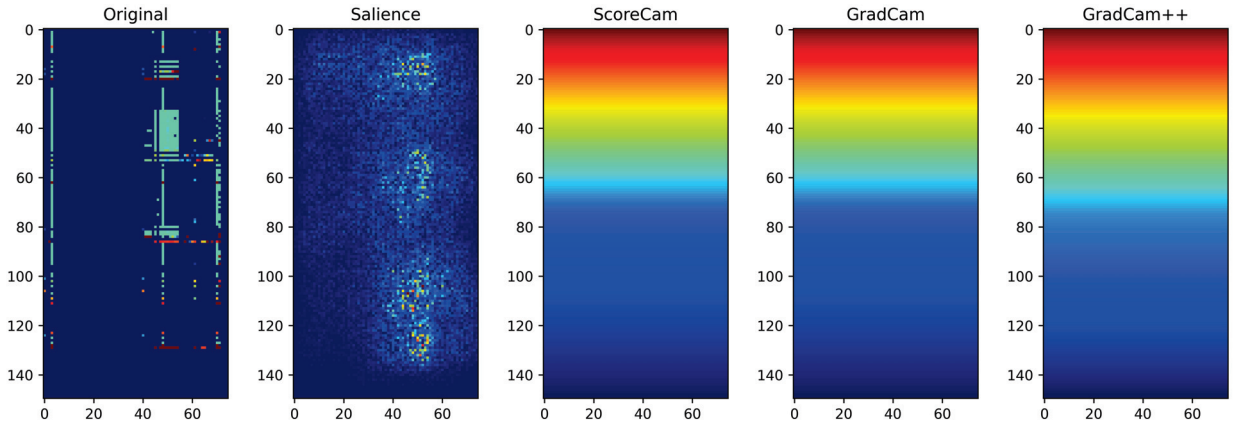


Figure 5.14: Inception-V3's Best Order (ABS-Correlated/Correlated) Infected Sample #165 Visualizations (Pred: 1)

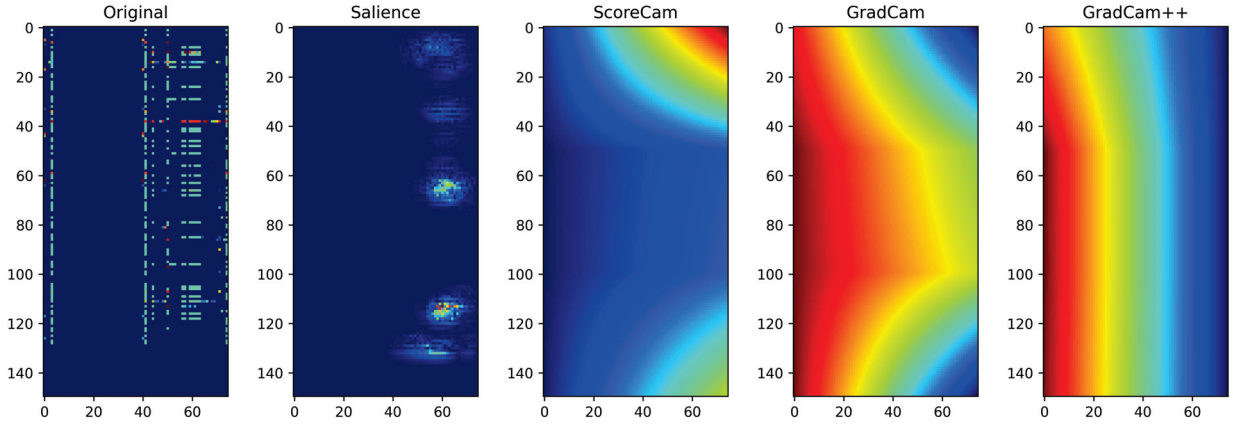


Figure 5.15: ResNet-18's Worst Order (Random-1/Original) Benign Sample #159 Visualizations (Pred: 5.9e-5)

influences both the benign and infected samples, but the infected sample more so.

Comparing the worst order with the best order for Inception-V3, the best order has the data element clumped together in blocks of matching values, whereas the worst order has the data elements dispersed. The difference between mAP percentages is spread of an 11% or a 89% improvement. The Salience plots clearly show that the denser the data clumps the more source data is insignificant as the best order allows the model to focus on the important elements. In addition the other visualizations have fewer similarities between the benign and infected samples in the worst order analyzed by Inception-V3.

Examining the ResNet-18 visualizations, they appear very different than those for Inception-

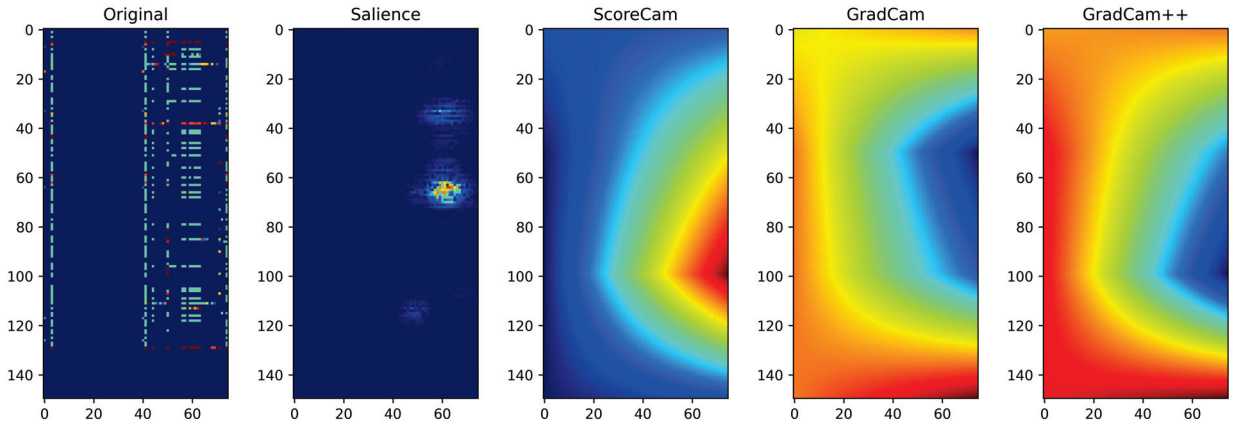


Figure 5.16: ResNet-18's Worst Order (Random-1/Original) Infected Sample #165 Visualizations (Pred: 0.9991)

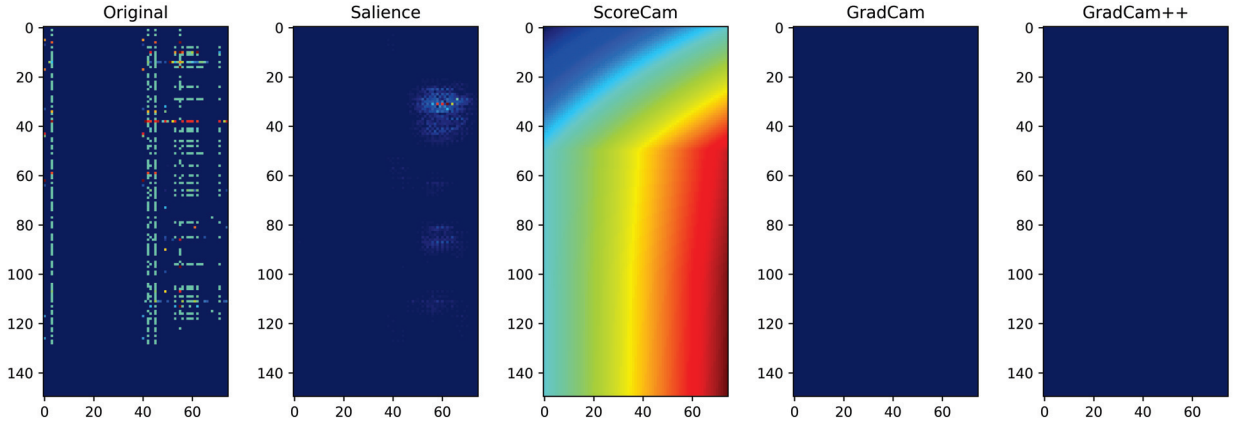


Figure 5.17: ResNet-18's Best Order (Random-1/Random-9) Benign Sample #159 Visualizations (Pred: 0.0132)

V3. This is visible within the Salience plot of the worst order with tight clusters of activation, the benign sample having five insignificance clusters, two of them densely packed and two lightly packed. The infected sample in Figure 5.16 has only three insignificant clusters, but one is very dense and one light. The infected clusters appear in similar locations to three of the benign clusters (Figure 5.15). The CAM plots appear to indicate different things: ScoreCAM and GradCam show almost exact opposite regions of the data having influence, with ScoreCAM favoring the left side of the data samples, and GradCAM favoring the right. GradCAM++ is also favoring the right side but in different regions. The infected and benign samples show influence from different regions of the data sample in all three CAM plots. Notably this experiment performed worst out of all of them, achieving only a 50.31% mAP score

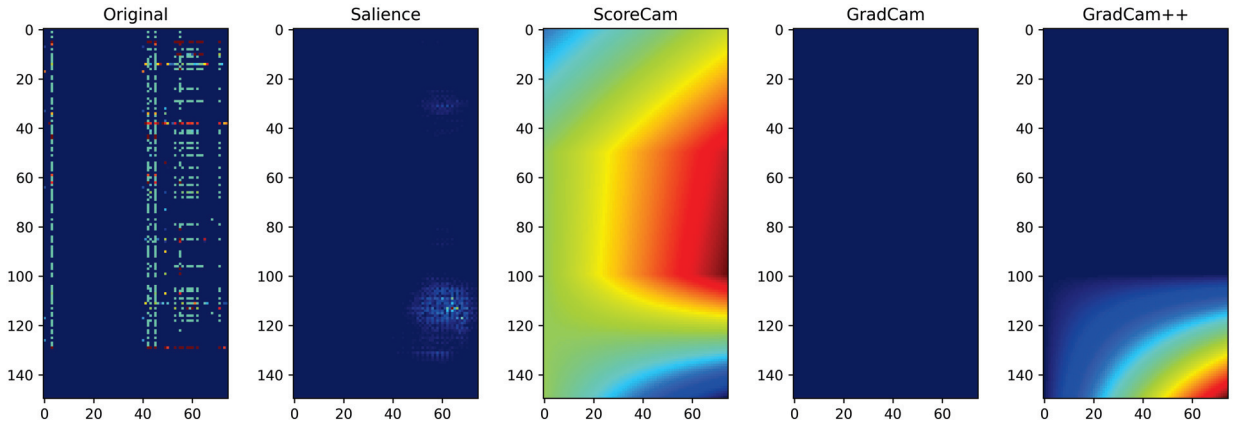


Figure 5.18: ResNet-18's Best Order (Random-1/Random-9) Infected Sample #165 Visualizations (Pred: 0.528)

The ResNet-18's best order performed quite the opposite manner, achieving a near-perfect mAP of 99.99%. The Salience plots are extremely sparse, showing that this network found that most data has a significant influence on the decision. The sparse insignificant clusters have a single local density in both the benign (Figure 5.17) and infected (Figure 5.18), samples, but in different regions of the sample. Again, the CAM plots show different things, with ScoreCAM finding the top left significant in both samples, including the bottom right in the infected sample. Most GradCAM and GradCAM++ plots show the entire sample has influence, except for GradCAM++ infected sample which finds the bottom right insignificant. Notably, all three CAM plots found very similar regions of the benign and infected samples relevant.

Contrasting ResNet's best and worst ordering schemes, in the best sample, the data patterns are slightly less contiguous. Something about ResNet-18's architecture very much appreciates the data slightly looser patterns, by 49.68% mAP points, or a 99.97% improvement. The Saliency plots' differences are quite extreme with dense insignificant regions with the worst ordering, but are very sparse and light clusters in the best ordering. Almost the opposite response is seen in the Inception-V3's Saliency plots. The CAM plots are also quite different with the benign and infected regions in the worst samples showing very different influence regions, whereas the influence patterns between the benign and infected samples are very similar in the best ordering.

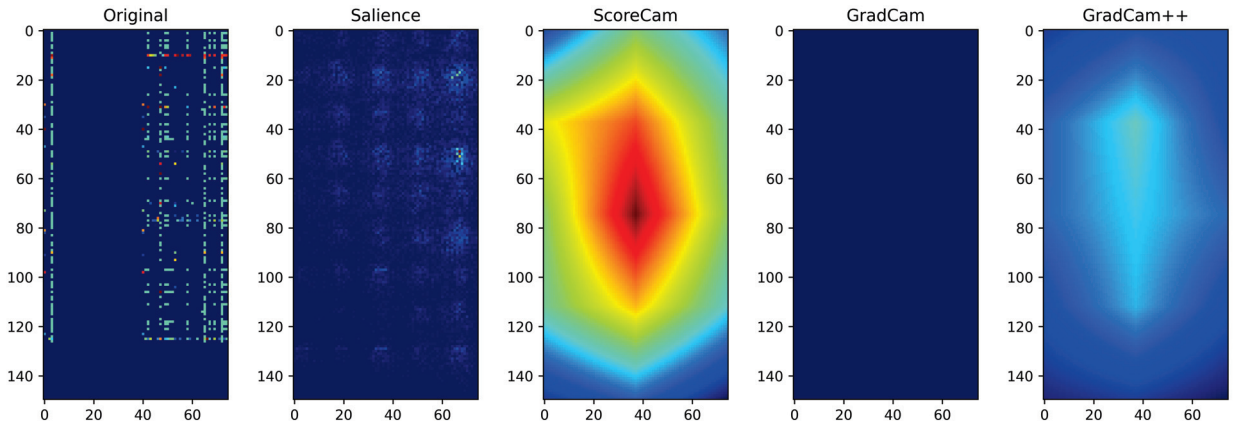


Figure 5.19: Xception's Worst Order (Random-7/Random-4) Benign Sample #159 Visualizations (Pred: 2.5e-13)

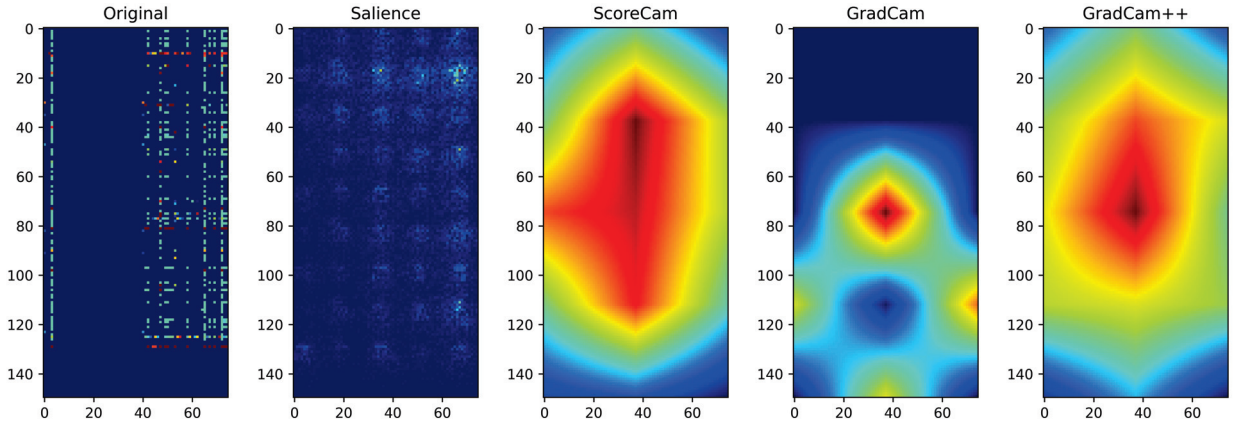


Figure 5.20: Xception's Worst Order (Random-7/Random-4) Infected Sample #165 Visualizations (Pred: 1)

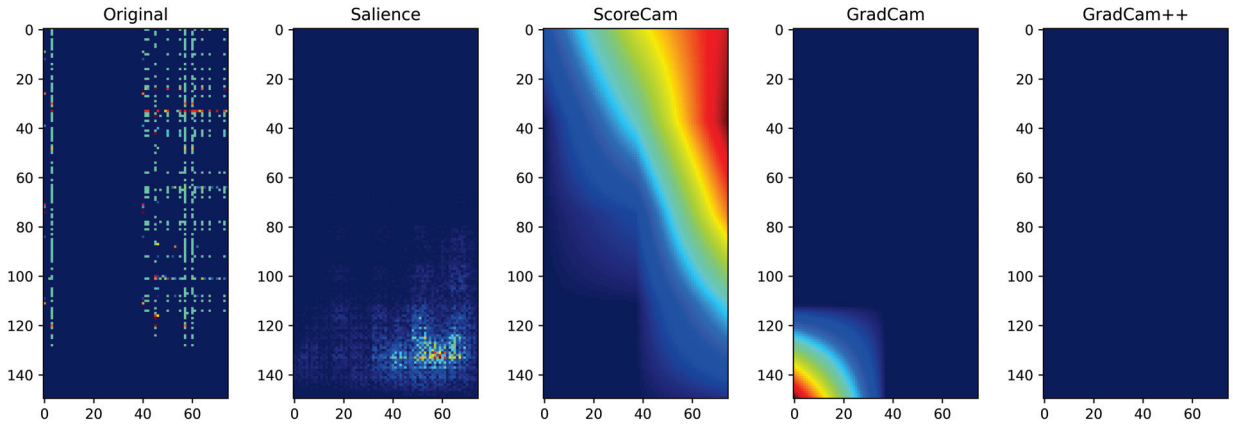


Figure 5.21: Xception's Best Order (Random-3/Random-5) Benign Sample #159 Visualizations (Pred: 0)

Xception also has distinct patterns within the visualizations, different from the previous two models. In the worst case, the Saliency of the benign (Figure 5.19) and infected (Figure 5.20) appear to have a grid like pattern with the infected sample having slightly more intense insignificant blocks within the influence grid. All CAM plots show some similarities, with the center of the sample being insignificant but each to a different degree. ScoreCAM has an extremely intense insignificant region with minor variations between the benign and infected samples. Benign is slightly less intense and has a slight extension going to the upper left, whereas the infected sample is more intense, particularly in the upper middle with a strong extension out to the middle left. GradCAM++ has its center insignificant region very dim in the benign sample, but very similar to ScoreCAM in the infected sample. GradCAM has no insignificant region for the benign sample, but the infected has a small but intense insignificant region in the center with three lower isolated lobes on the lower half edge.

The best order for the Xception's visualization differs from those of the previous two models, but the Saliency plots do have a similar grid-like pattern. The benign (Figure 5.21) has only a few very intense insignificant clusters in a region of the grid at the lower left of the data sample, whereas the infected (Figure 5.22) insignificant clusters are less intense and spread throughout the influence grid. The CAM plots between the benign and infected samples are almost identical, with most of the data sample having influence. ScoreCAM has a bar of insignificance intensely

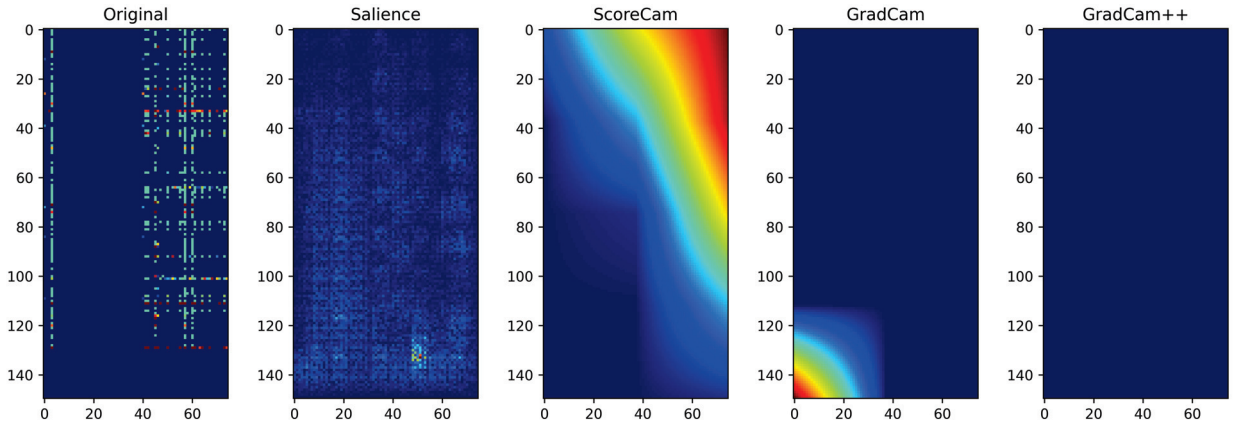


Figure 5.22: Xception's Best Order (Random-3/Random-5) Infected Sample #165 Visualizations (Pred: 1)

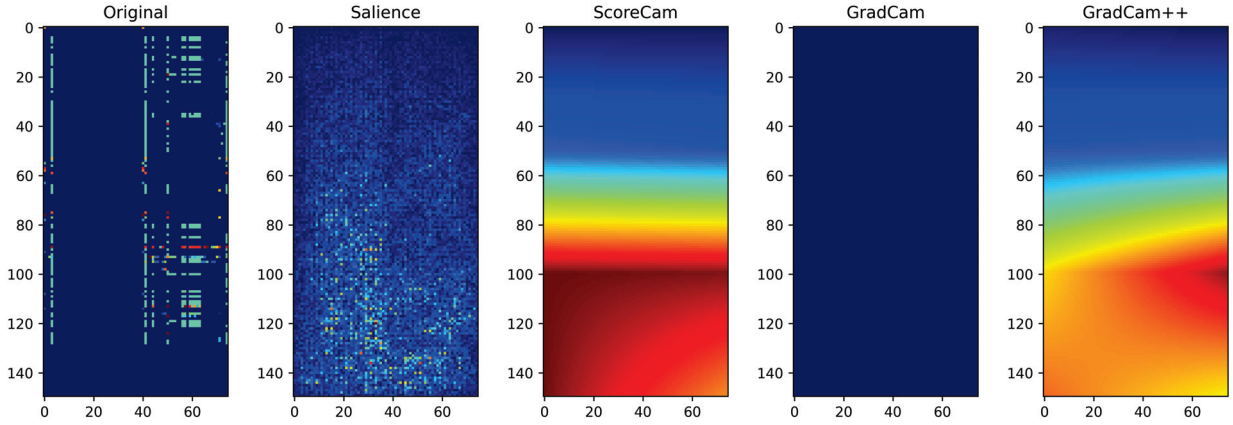


Figure 5.23: MobileNet's Worst Order (Alphanumeric/Original) Benign Sample #159 Visualizations (Pred: 2e-13)

crossing the top right corner, whereas GradCAM has a little but intense bubble on the lower left, and GradCAM++ has no insignificant region for either infection status.

Comparing the best and worst orderings, where there is a 3.6% spread but a 97.8% improvement. The differences between visualizations are stark. The data samples in the worst order appear more stochastic whereas, in the best order, the data has cross-like patterns with some breaks between. The Saliency plots for the best order have more intense but concentrated insignificant regions. The CAM plots for the best order are nearly identical between the infected and benign samples, whereas the plots have distinct characteristics between infection status for the worst order.

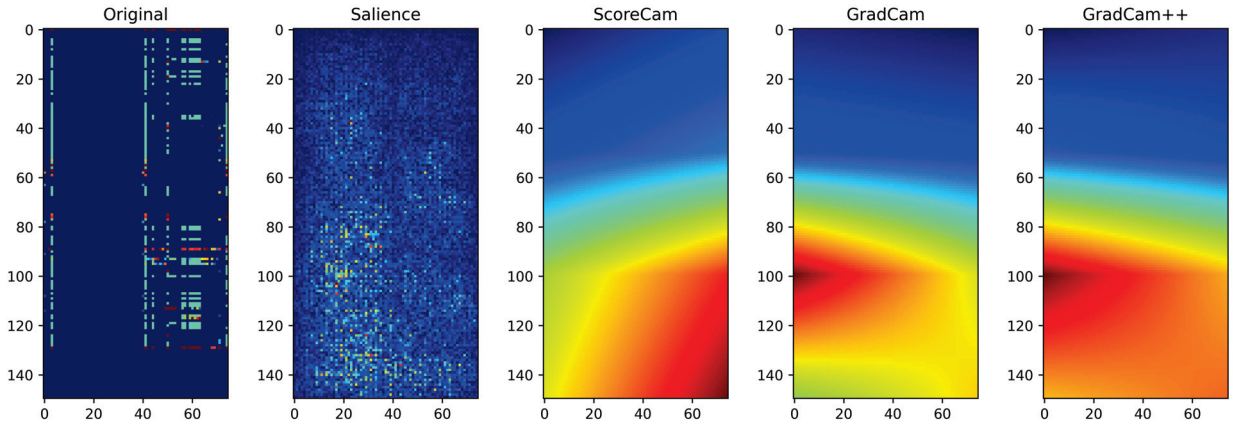


Figure 5.24: MobileNet's Worst Order (Alphanumeric/Original) Infected Sample #165 Visualizations (Pred: 1)

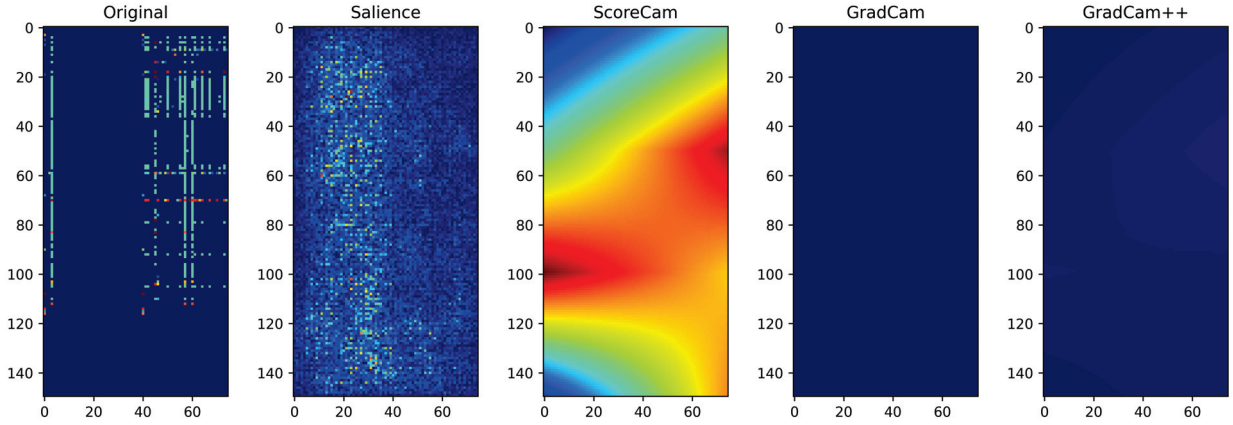


Figure 5.25: MobileNet's Best Order (Correlated/Random-5) Benign Sample #159 Visualizations (Pred: 7.7e-18)

Examining the MobileNet's visualizations, the Salience plots appear very *noisy* with scattered insignificant data points spread though out the sample. Both have the lower half more intense, but upper half is more sparse in the benign sample, (Figure 5.23) than the infected (Figure 5.24). The CAM plots all appear to show similar responses displaying that the lower half is insignificant. ScoreCAM is more so, with a deep intensity on the benign sample, whereas skewed a little to the right of infected sample. GradCAM's benign plot shows no insignificant region, whereas the infected sample shows insignificance in the lower half and is skewed to the middle left. The GradCAM++ graphs also have the lower half marked as insignificant but skewed right and left on the benign and infected sample respectively.

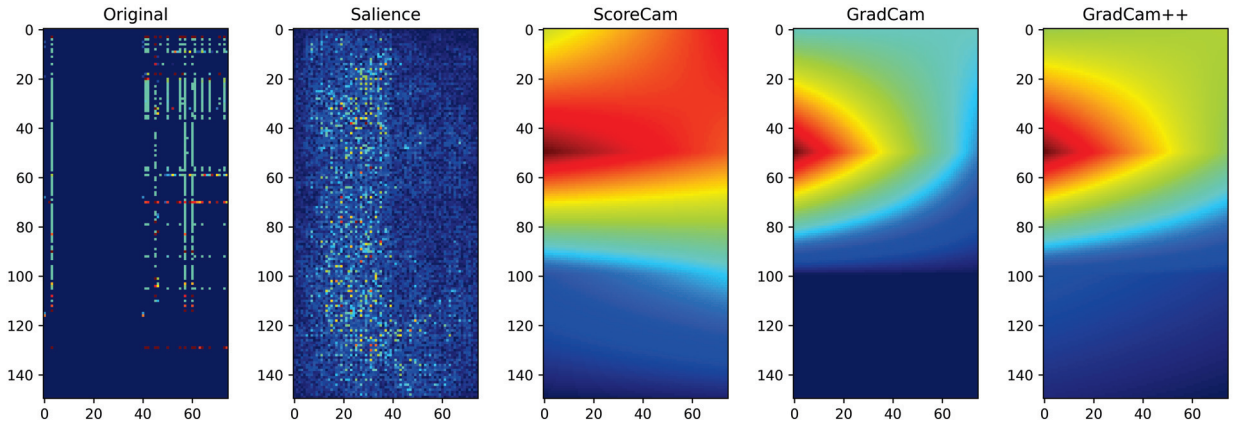


Figure 5.26: MobileNet's Best Order (Correlated/Random-5) Infected Sample #165 Visualizations (Pred: 1)

The graphs produced for the MobileNet's best ordering scheme are slightly different from the worst just reviewed. The Saliency plots are still noisy in appearance, but the insignificant data points crowd around the left side of the graph. The benign (Figure 5.25) and infected (Figure 5.26) appear nearly identical with the infected sample slightly more intense. ScoreCAM shows a strong band of insignificance horizontally across both samples, but the benign is a triangle shape, starting on the middle left and extending toward both right-hand corners. GradCAM and GradCAM++ share almost identical patterns with the benign sample having no insignificant region whereas the infected has a large bubble on the upper half of the left edge. The difference between the two is that GradCAM++ is more intense.

Comparing the best and worst orderings for MobileNet's experiments, it is obvious that the model performance correlates to the data aligning with the long side of the grid. The differences are stark with a mAP spread of 40.9% or an improvement of 99.56%. The Salience maps appear to share this alignment, with the best performing order aligned vertically along the longer row axis. There are distinct differences within the CAM plots when comparing the best and worst orderings but differentiating on to how that relates to performance is difficult to discern. Although the insignificant regions all appear to cover the same general area when relating to the infection status, GradCAM and GradCAM++ have slightly less overall intensity in the best order. The noticeable major variation between the order schemes is that GradCAM++ considers the entire

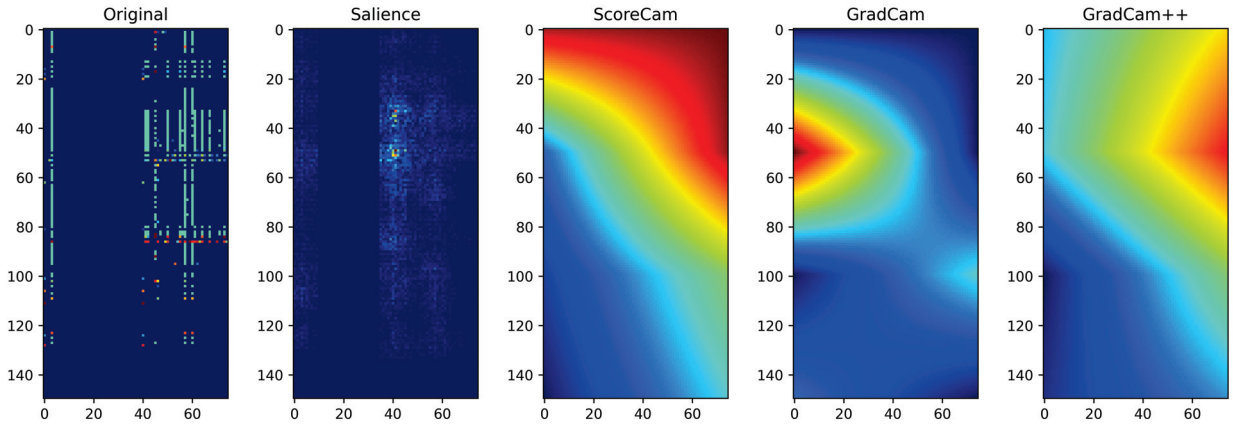


Figure 5.27: DenseNet-121's Worst Order (ABS-Correlated/Random-5) Benign Sample #159 Visualizations (Pred: 2.2e-10)

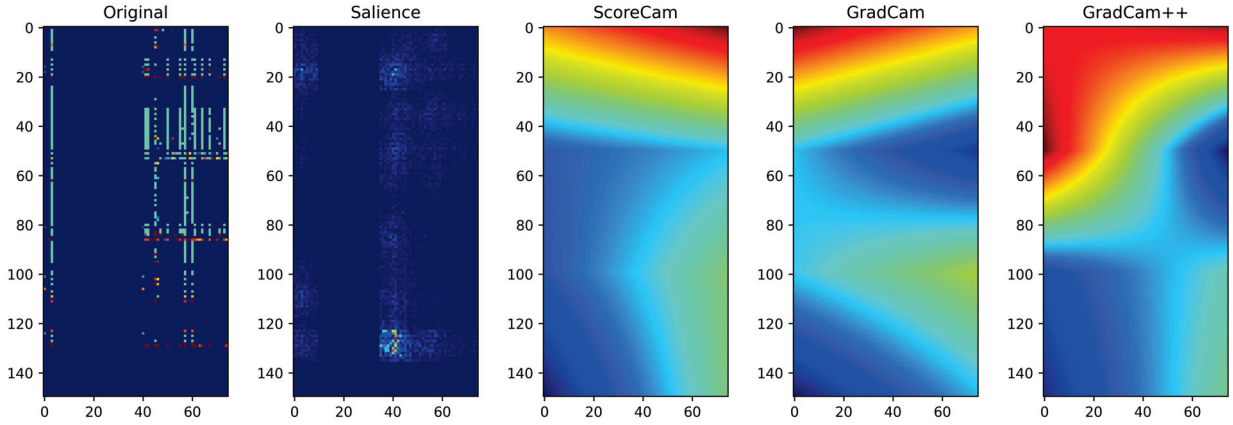


Figure 5.28: DenseNet-121's Worst Order (ABS-Correlated/Random-5) Infected Sample #165 Visualizations (Pred: 1)

benign sample a significant deciding influence for the best order.

DenseNet-121 has the highest performing worst order at 96.36%. Its Salience maps have a general but faint grid-like appearance with different bubbles of insignificant clusters over certain regions. The benign sample (Figure 5.27) has tighter clusters in the upper middle of the map whereas the infected (Figure 5.28) is more spread out with one intense cluster in the lower center. The CAM maps show similar regions of insignificance. The benign sample has them all on the upper half, but the ScoreCAM plot of the region is the most intense going from the top-left to the right side whereas the GradCAM++ plot is missing the top-left corner and GradCAM mostly comprises of an intense bubble on the upper-left edge. The infected sample has insignificant

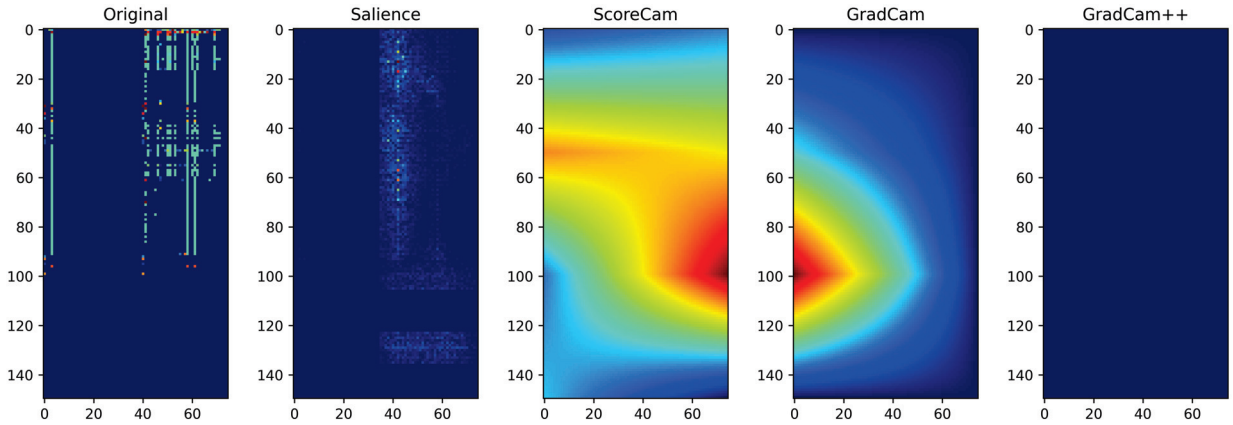


Figure 5.29: DenseNet-121's Best Order (VM-PID/Random-1) Benign Sample #159 Visualizations (Pred: 3.5e-20)

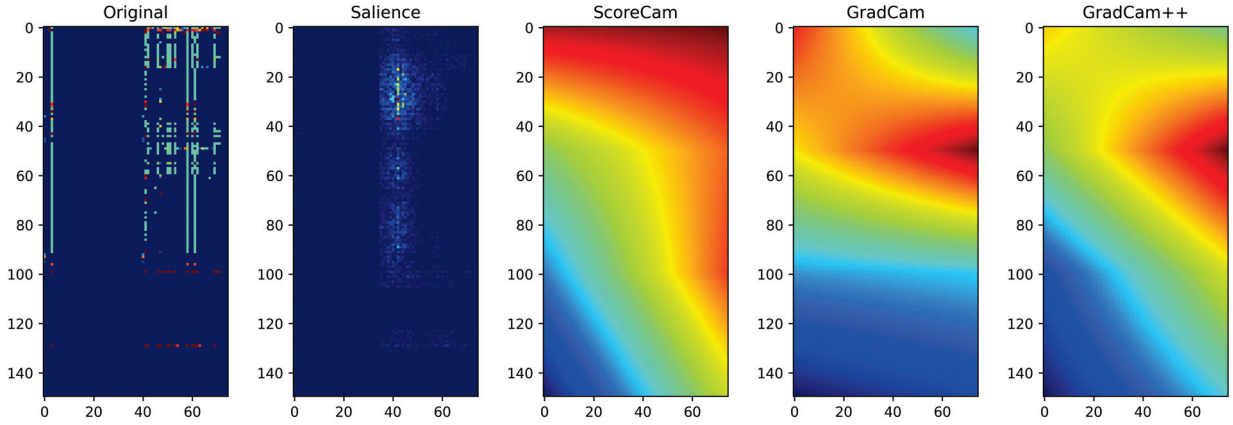


Figure 5.30: DenseNet-121's Best Order (VM-PID/Random-1) Infected Sample #165 Visualizations (Pred: 1)

regions in the same general area but is quite distinct in form from the benign sample. All three have a strong band on the top edge, with ScoreCAM leaning sightly right, GradCAM leaning slightly left, and GradCAM++ leaning heavily left. They also all have moderate shapes over the lower right edge. ScoreCAM is a semicircle, GradCAM is a triangle whose center vertex reaches the left side, and GradCAM++ is just a vertical bar.

The best order for DenseNet-121 has very similar Saliency maps between the infected status. Both the benign (Figure 5.29) and infected sample (Figure 5.30) have a band of clustered insignificant points going through the upper middle of the graph. The major difference is the infected are more intense and localized, whereas there is a general dispersion of lightly insignificant

clusters found through the benign Salience visual. The CAM maps show definite variation between the benign and infected samples. ScoreCAM has a large band of insignificance going from top the left-edge to the lower-right regardless of infection status, but the benign sample has a strong band of influence along the top. GradCAM has a strong bubble of influence on the lower-left edge of the benign sample, which shifts up and stretches over to the other side of the infected sample. GradCAM++ shows full influence on the benign sample, whereas a similar band of insignificance as the other CAMs, starting from the top left that becomes more intense towards the right edge.

DenseNet-121 has the smallest variation in the differences between best and worst orderings, with only a 3.51% mAP spread, but even here, when the best is 99.87%, near perfection (1 fail in 333), the improvement above 96% (1 fail in 25) is an order of magnitude better. The differences between data samples are almost indiscernible. Careful examination can reveal the worst ordering appears to have small blocks of contiguous data with erratic breaks whereas the best order also has contiguous blocks of data but are broken up at intermittent intervals. The Salience and CAM plots are similar by comparison, the major difference is that GradCAM++ has full influence on the best order benign sample. By comparison the other maps have similar areas, just shifts in regions and intensities.

5.2.9 Analyzing Malware Results Using MiCAM

** The material presented in this subsection will appeared in the proceedings on the 4th International Conference on Machine Learning and Soft Computing (MLSC 2023) in the article,* "MiCAM: Visualizing Feature Extraction Of Nonnatural Data", *co-authored with Ram Krishnan, Ph.D.*

As mentioned in the previous chapters, MiCAM was used to analyze the difference between the best and worst ordering schemes (Table 5.3) when searching for malware. Between the LeNet-5 MiCAM plots it is seen that the pooling layers have the most distinguishing characteristics. It is visible in Figure 5.31 which is divided by the best and worst ordering schemes.

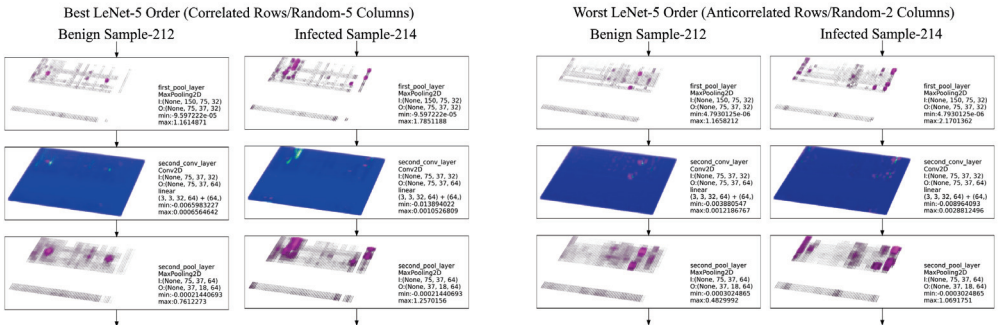


Figure 5.31: MiCAM Plots of the Lower Quarter for the LeNet-5 Best (left) and Worst (right) Ordering of Samples Benign #212 and Infected #214

Displayed is how the features are better defined in the pooling layers with the stronger intensities, and the range on the infected sample of the best order is noticeably larger in the second pooling layer than the worst order.

Within the ResNet-18 plots there are a number of items to take notice of in Figure 5.32. Several of the CAM plots are identifying clusters of data points that have some significance on the decision. In particular the B4 residue convolution layers and associated additions and activation layers, highlighted in yellow, perhaps point to particular data points the CNN identifies as maleficent or benign. Also noticed is that the features from the best ordering are distinct in the final pooling layers for the benign and infected samples, highlighted orange, but the worst order displays those layers as having similar features by comparison.

For brevity the DenseNet-121 MiCAM graph are not displayed but they are available at the Git site mentioned earlier. Things to note, the CAM plots most relatable to the source data are the last convolution stage before the first bottle neck stage. Visible are a number of highlighted pixels of interest for the different classifications. In particular noticed is a highlighted row within the best ordering scheme for an infected sample, perhaps informing us that there is an infected process on that row.

Figure 5.32: MiCAM Plots of Pooling and Last Convolution Layers for the ResNet-5 Best (left) and Worst (right) Ordering of Samples Benign #212 and Infected #214

Table 5.4: Pooling operations within Models

LeNet-5			Inception V3			Xception		
Layer #	**Type**	**Radius**	**Layer #**	**Type**	**Radius**	**Layer #**	**Type**	**Radius**
3	Max	1	10	Max	5	13	Max	7
6	Max	3	17	Max	7	23	Max	15
			27	Avg	7	33	Max	31
ResNet-18			50	Avg	11	123	Max	319
Layer #	**Type**	**Radius**	73	Avg	15			
61	Max	60	99	Max	19	**DenseNet-121**		
62	Avg	60	119	Avg	43	**Layer #**	**Type**	**Radius**
			151	Avg	67	6	Max	3
MobileNet			183	Avg	91	52	Avg	15
Layer #	**Type**	**Radius**	215	Avg	115	140	Avg	39
none			247	Max	139	312	Avg	87
			262	Avg	203			
			293	Avg	331			

5.2.10 Analyzing Malware Multiple ResNet-18 Models

One thing discovered during this research was that the ResNet models where susceptible to high degree in performance variance when changes were made within the grid ordering. Additional experimentation was performed and it was discovered that minor variations in model had implications to this noticeable change. Included in this subsection are an analysis of the results.

The first major impact noticed was that the learning rate has a huge implication in this variance. The initial experiments were with a learning rate 0 0.001 where the most variance was found between models in the experiments of Subsection 5.2.3. Following the initial published results, the learning rate was modified incrementally in the future experiments and it was discovered that a learning rate or 0.00001 provided significant improvement in the overall response of the network.

Analyzing the differences between the networks it was noticed that ResNet's use of pooling operations was different compared to other networks (Table 5.4). Most use a maximum pooling operation which caries through the most intense pixel from the initial input to represent the local-

Table 5.5: ResNet-18 variance at LR:1e-5 Mean AUC for Precision Recall Curves

CNN Architecture	All Options	Corr Rows%	ABS Corr Rows%	Anti Corr Rows%	Corr Cols%	ABS Corr Cols%	Anti Corr Cols%
ResNet-18	99.852	99.846	99.879	99.834	99.846	99.858	99.884
ResNet-18max	99.823	99.801	99.872	99.804	99.825	99.869	99.861
ResNet-18avg	99.840	99.850	99.914	99.811	99.837	99.833	99.842

Table 5.6: ResNet-18 variance at LR:1e-5 Improvement Over Average (Mean) Performance

CNN Architecture	100%- All Mean	Corr Rows%	ABS Corr Rows%	Anti Corr Rows%	Corr Cols%	ABS Corr Cols%	Anti Corr Cols%
ResNet-18	0.148	-.00006	.00027	-.00018	-.00006	.00006	.00032
ResNet-18max	0.173	.00011	.00075	-.00029	-.00015	.00030	.00021
ResNet-18avg	0.160	-.00022	.00047	-.00022	-.00015	-.00019	-.00010

ized region a the output, and these are normally scattered throughout the network. ResNet's use only a single pool pair, a maximum and average, concatenating the results at the end of the network. By theory this should amplify the less responsive pixels within the final extracted features set. It our hypothesis that it is this global amplification late in the network that recognizes the anticorrelation as a responsive order.

This was tested by creating three versions of the ResNet-18 model, each operating at at the learning rate of 0.00001. The first was the normal ResNet18 model. The second removed the average pooling so the results of using just the maximum as a pooling operation could be examined. The third did the opposite, so how the network performs toward order variation when just using the average pooling operation could be examined. They were labeled ResNet-18max and ResNet-18avg and the results are found in Table 5.5 with the percentage of improvement/degradation shown in Table 5.6.

It can be seen that the modification to the learning rate vastly improved performance of all modified ResNet-18 models, and it reduced the variance in performance with regards to grid order changes. The variance noticed between the models is attributable to changes in the pooling

layers which strongly relates to the responsiveness to the anticorrelation order. This leads us to several theoretical analysis explored in the next chapter.

5.3 IP-Traffic of Security Attacks: CIC-IDS-2017

5.3.1 Analyzing Malware Results Using MiCAM

** The material presented in this section will appeared in the proceedings on the 4th International Conference on Machine Learning and Soft Computing (MLSC 2023) in the article,* "MiCAM: Visualizing Feature Extraction Of Nonnatural Data", *co-authored with Ram Krishnan, Ph.D.*

This study is analyzing the affect that order has on nonnatural data, and one data set considered relevant is the CIC-IDS-2017 raw IP-traffic data. It poses a novel scenario to tests the hypothesis. As described previously in Subsection 4.1, 146 different columns ordering schemes were devised and compared to the performance results when using an order devised from the internet protocol specification. A shallow LeNet-5 CNN model (2 convolution and three dense layers) was trained, matching previously published research. In identifying the presence of maleficent actors, the results are found in Table 5.7 and in classification of the maleficent actors in Table 5.8. They include the PR curve mAP value for every non-random ordering scheme devised and a percentage of improvement over the mAP average of all the randomly generated schemes.

In detection tasks, Table 5.7, changes in grid order can make a significant difference because of the percentage improvement towards perfection. Like discussed when analyzing best and worst performing results analyzing malware using the Dense Network, Subsection 5.2.8, when failures are costly the range seen here is significant. A change from 99.45% (non random worst) to 99.7% (best) means almost a halving of the failures rate.

Counter our hypothesis, the ordering scheme derived when following the IP specification exceeded expectations, out performing all other ordering options. It had a 34.67% improvement

Table 5.7: Detection Analysis Results and Statistical Sample Counts By Class Order

Sample			**% Prec/Recall mAP**			**% Improve/Degrade**		
Random Average			99.54			0.0		
IP-Specification			**99.70**			**34.67**		
Order Sample Set	**Count**	**%**	**Corr**	**ABS**	**Anti**	**Corr**	**ABS**	**Anti**
Bot	1228	0.15	99.54	99.57	99.57	-0.36	6.98	6.57
DDoS	44918	5.53	99.60	99.64	99.55	14.19	22.21	1.69
DoS Hulk	5952	0.73	99.58	99.57	99.53	7.85	5.69	-3.04
DoS Slowhttptest	4216	0.52	99.54	99.58	99.56	-0.27	8.99	3.59
DoS slowloris	3872	0.47	99.45	99.55	99.66	-18.97	1.98	25.28
FTP-Patator	3974	0.49	99.58	99.54	99.52	8.93	0.34	-5.34
Infiltration	6	0.001	99.58	99.60	99.64	9.56	13.51	21.20
PortScan	158410	19.53	99.56	99.54	99.56	4.50	0.13	4.48
SSH-Patator	2978	0.36	99.59	99.56	99.54	9.81	3.71	-0.19
Web Attack-Brute Force	1363	0.16	99.61	99.56	99.47	14.67	5.36	-15.15
Web Attack-Sql Injection	12	0.001	99.61	99.59	99.54	15.33	10.08	0.83
Web Attack-XSS	625	0.077	99.58	99.48	99.51	8.79	-13.73	-6.55
Maleficient	227554	28.06	99.55	99.57	99.52	2.98	6.29	-4.36
Benign	583411	71.940	99.51	99.54	99.51	-6.38	-0.55	-6.80
Total	810965	100.0	99.59	99.58	99.57	10.94	9.63	6.47
Average Improvement	-	-	-	-	-	5.44	5.37	1.91

over the average. This shows the care to which IEEE specification was laid to logically organize the data packets as they relate to each other.

It is also interesting to note that the majority of ordering schemes devised around a statistical relationship between data bytes within subsets of the data also performed better than average. The surprise regarding the subsets was the correlation of the benign samples. Only two other correlation subsets showed a major degradation in performance compared to the random average, and those sample sizes were less than one percent of the total samples. The benign correlation had 70% of the samples, but resulted in more than a 6% degradation. Focusing on benign samples to find maleficent actors proved detrimental. These findings support our hypothesis that statistical correlation does produce a better than average precision, as long as the data subset that the correlation is taken from has enough maleficent samples.

It's also notable that although anticorrelation ordering did have some significant improvement for some subsets, the majority of the subsets showed a poorer performance. Absolute value of correlation produced only one significantly detrimental ordering using a subset, which comprised of less than 1/10th of 1% of the total samples, so appears to be a relativity safe when using with a shallow network.

When analyzing classification task some results match what was the found in detection. IP specification performed the best again, and both correlation and absolute value of the correlation perform better than average using most data subset, while anticorrelation showed the least average benefit. In contrast there appears to be a maximum performance in classification tasks with an apparent limit of a 98.7%. This means the small variance effected by order is negligible compared the difference toward perfection. Another difference is that using correlation from the whole data set appeared to hamper classification, while anticorrelation supported it.

To analyze the differences between the best and worst ordering schemes for detection, they were selected for processing with MiCAM diagrams their performance details are included in

Table 5.8: Classification Analysis Results and Statistical Sample Counts By Class Order

Sample			**% Prec/Recall mAP**			**% Improve/Degrade**		
Random Average			98.34			0.0		
IP-Specification			**98.68**			**.0035**		
Order Sample Set	**Count**	**%**	**Corr**	**ABS**	**Ant**	**Corr**	**ABS**	**Anti**
Bot	1228	0.151	98.52	98.55	98.31	0.0018	0.0021	-0.0003
DDoS	44918	5.539	98.50	98.51	98.24	.0016	.0017	-.0011
DoS Hulk	5952	0.734	98.43	98.48	98.32	.0009	.0014	-.0002
DoS Slowhttptest	4216	0.520	98.42	98.46	98.45	.0008	.0012	.0011
DoS slowloris	3872	0.477	98.35	98.60	98.46	.0001	.0027	.0012
FTP-Patator	3974	0.490	98.58	98.32	98.11	.0024	-.0002	-.0023
Infiltration	6	0.001	98.39	98.52	98.39	.0005	.0018	.0005
PortScan	158410	19.534	98.28	98.45	98.31	-.0006	.0011	-.0003
SSH-Patator	2978	0.367	98.09	98.17	98.34	-.0025	-.0018	.0000
Web Attack-Brute Force	1363	0.168	98.44	98.30	98.32	.0010	-.0005	-.0002
Web Attack-Sql Injection	12	0.001	98.56	98.35	98.30	.0022	.0001	-.0005
Web Attack-XSS	625	0.077	98.42	98.36	98.28	.0008	.0002	-.0006
Maleficient	227554	28.060	98.42	98.54	98.45	.0008	.0020	.0011
Benign	583411	71.940	98.56	98.48	98.35	.0022	.0014	.0000
Total	810965	100.0	98.27	98.28	98.57	-.0007	-.0006	.0023
Average Improvement	-	-	-	-	-	.0007	.0009	.0000

Table 5.9: Best and Worst Ordering Schemes For Maleficent IP-Traffic

CNN Architecture	**Best Column Order**	**mAP Score**	**Worst Column Order**	**mAP Score**
Lenet-5 *10 epoch*	IP Specification	99.703%	Random-40	99.445%

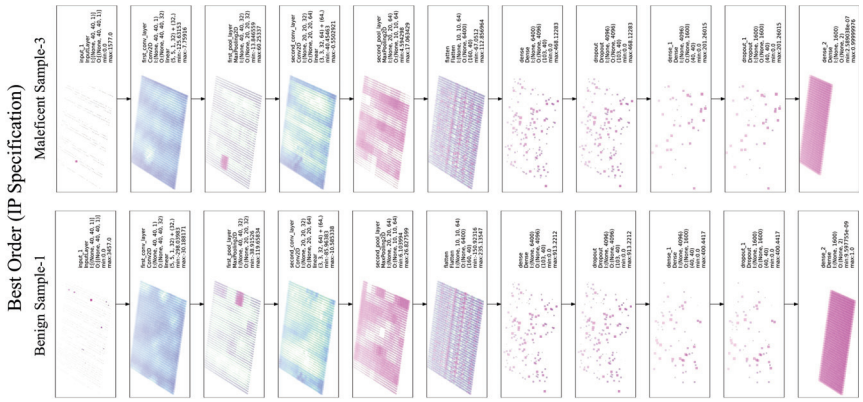


Figure 5.33: MiCAM Plots of LeNet-5 Analyzing Best Order (IP Spec) IP Packets with Benign and Maleficent Packages

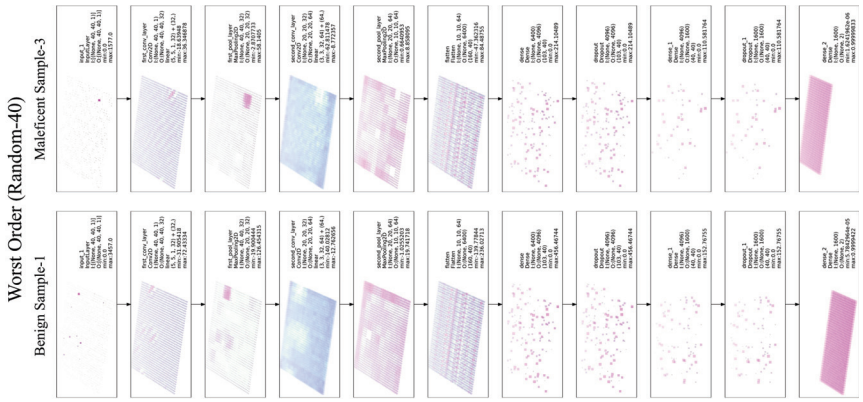


Figure 5.34: MiCAM Plots of LeNet-5 Analyzing Worst Order (Random-40) IP Packets with Benign and Maleficent Packages

Table 5.9.

Examining the MiCAM plots, in Figures 5.33 and 5.34, it is visible how the best order has a wider range, with the peak negative values showing very distinct regions within the convolutional layers. Also in several layers, both orders show the first quarter of the sample is significant in finding the maleficent sample's attack vector, while several areas within the packet are identified significant in the benign.

CHAPTER 6: ANALYSIS AND CONCLUSION

The hypothesis of this research is to show that data order does matter for CNN performance. It also offers that be duplicating images properties with data ordering, using correlation as a basis, can help performance. Through experimentation, several models were found, including the latest benchmark, DenseNet, to improve performance when that order is based on the data sets statistical correlation.

This research shows that axis order always matters, and using correlation as a basis for ordering the independent smaller axis does appear to improve identification performance over the average regardless of the data or model used. All of the models performed well when the independent shorter axis was ordered by correlation, but it was not always the preferred order.

6.1 Malware and Model Performance

** The material presented in this section previously appeared in the proceedings of the International Conference on Secure Knowledge Management in the Artificial Intelligence Era (SKM 2021) in the articles,* "Analyzing CNN Model Performance Sensitivity to the Ordering of Non-Natural Data" *and in the journal, Information Systems Frontiers (2022) ,* "Visualizing CNN Models' Sensitivity to Nonnatural Data Order", *co-authored with Ram Krishnan, Ph.D.*

There is relevance to proper data ordering when preparing data for CNN, regardless of the model. Even with a model that mattered least, DenseNet-121, an improvement of 96.4% toward perfection is still noticed between the tested extremes. With every model, there was a handful of ordering schemes that achieved close to the best performing score, whereas the worst order was always an outlier. Notably there were 252 unique ordering schemes. This leads us to hypothesize that every model has many ordering schemes that nearly approach the optimal performance, but the poorly performing ones are rare in comparison.

Ordering performance is model-dependent. What one model considered a good order on an axis, other models may not. Usually some performance improvement was seen when using either correlation or anticorrelation algorithm for an ordering scheme, but amount relevant was model-dependent. It was also axis dependent, the shorter axis appreciated having the correlation or anticorrelation function as an ordering scheme, while the longer axis might not.

With malware data, different models behaved differently. On average Xception produced the best results and had a few options in order schemes that achieved the second-highest mark for a model. One of the schemes happens to include regular correlation on the shorter axis, and all correlation schemes performed better than average on the longer axis. This seems to indicate that Xception may be easy to tune. DenseNet-121 also performed very well with only a slightly lower peak mAP rating in two-thirds of the training time. Correlation along the longer axis is one of the peak ordering schemes for DenseNet-121; thus it may also be easy to tune. Surprisingly only Lenet-5 didn't like the anti-correlated columns on the shorter independent axis. It is suspected that it has to do with the lack of total width only two levels of convolution layers provides.

ResNet-18 produced the widest range of results, both the best and worst, followed by MobileNet. On average, ResNet-18 appears to like the anticorrelated on both axes, almost opposite behavior than the other models. MobileNet appears to perform well with any proposed correlation order schemes for the shorter axis, but performs very poorly on the longer axis. It is suspected that MobileNet's responsiveness to one axis over the other and the visualization of that responsiveness has to do with how features are extracted from the convolution layers within the model. This is in contrast to the maximum pooling operations found in most models.

Modifying ResNet-18's model did have an effect on order performance, which showed that the feature extraction process has an affect on which order performs best. Most networks a maximum pool operation integrated within their stages through out the network. They identify the peaks within filter response. Since most networks use these as feature extractors earlier within the network, those patterns are highly localized. ResNet-18 uses convolutions as feature extractors

earlier in the network, and MobileNet uses only convolutions and doesn't use any pooling operation. It is these structural differences in feature extraction that are the reason ordering performance behaves as differently.

Although Inception-V3 was faster at training than both Xception and DenseNet-121, its average performance was outperformed by MobileNet, which produced better results in half the training time. Inception-V3's architecture seems to lend to a more intuitive understanding of the visualizations produced.

6.2 Maleficent IP-Traffic and Statistical Subsets

** Some of the material presented in this section will appeared in the proceedings on the 4th International Conference on Machine Learning and Soft Computing (MLSC 2023) in the article,* "MiCAM: Visualizing Feature Extraction Of Nonnatural Data", *co-authored with Ram Krishnan, Ph.D.*

This research again shows that order has an affect on CNN performance. In this case though, the structural order as defined by the IP specification proved best. It shows the organization and care that IEEE put forward in defining the internet protocol. There is obviously some order to the IP-specification that provides the proper patterns for CNN to identify. It is also important to note that correlation did improve performance over the average of random orderings in detection tasks, even for the majority of cases where only a subset of maleficent data set was used to generate ordering statistics.

Classification tasks appear to have limitations in performance that ordering has little affect on. The improvements made by changing the order didn't significantly improve performance. IP specification again performed best, but that improvement was negligible over the average random order. Correlation also improves over the average in the majority of cases, but insignificantly.

6.3 Visualizations and MiCAM

** The material presented in this section previously appeared in the journal, Information Systems Frontiers (2022) ,* "Visualizing CNN Models' Sensitivity to Nonnatural Data Order", *and will appeared in the proceedings on the 4th International Conference on Machine Learning and Soft Computing (MLSC 2023) in the article,* "MiCAM: Visualizing Feature Extraction Of Nonnatural Data", *co-authored with Ram Krishnan, Ph.D.*

Visualizations provide some insight into what the CNNs are doing with nonnatural data, but deciphering the plots from these deeper models on the malware data is not as intuitive as it is when examining visualizations from sequential models with images. It was not possible to identify what particular data points within the sample made up a malware feature. In general, Salience and ScoreCAM plots contain the most distinct and unique plots per experiment scenario. By contrast GradCAM creates the same plot, an empty rectangle, for more than one-third of the scenarios. GradCAM and GradCAM++, in one of the best scenarios each, create the same empty plot for both the positive and negative samples. They appear unable to display the distinguishing features the CNNs are using to make decisions.

ScoreCAM appears the most descriptive. The best ordering schemes of every model show the plots between benign and maleficent are more similar than the ScoreCAM plots in the worst ordering schemes. This indicates that ScoreCAM identifies that similar features are used in decisions in better ordering schemes, whereas the worst ordering schemes show distinct or divergent features. In general the visualizations provide some sense to the distinctness the various CNN models generate features from the different orderings, but lack in clarity as to which portions of the original data grid had an influence on those features.

In contrast MiCAM diagrams offer more detail regarding feature extraction within the CNN models. They visually expose the layers, allowing the user to further understand the intensities of features extracted within the CNN structure. It was what directed this research in identifying the underlying structure capabilities within ResNet, allowing further comparison how minor vari-

ation in a model or process can affect feature extraction. MiCAM offers an additional tool for engineers as they tailor CNN models to nonnatural cybersecurty applications.

6.4 Final Analysis

This research clearly shows that order affects CNN performance. Using correlation as the basis for order is shown to have a significant opportunity to be a preferred order, but it may not be optimal. Structural orders that are arbitrarily defined have proven to achieve poor results, but those structural orders that the are diligently defined through rigorous organizational procedure could provide an optimal order.

As a result of this study a methodology is proposed for identifying a preferred model and ordering for novel grids of nonnatural data:

- Identify an initial, usually structurally defined, ordering for the grids.
- Test any available model with this ordering for a limited number of epochs and select several of the best performing models.
- Use the methodology detailed within this manuscript to generate several ordering options from statistics, and include a dozen random orders for a baseline.
- Test all of the options with chosen models and select best performer.

The sample sets used in generating the statistics for order generation don't have to comprise of the entire data set but must include substantial examples of the positive (maleficent) outcomes.

After processing a model, analyzing how the different ordering schemes behave can provide additional insight. Large variance in performance between the ordering schemes, like in the initial ResNet experiments, may be indicative of an improper parameter such as learning rate.

Current visualizations tools are not very supportive in nonnatural analysis scenarios, but it was the use of the new tool, MiCAM, that enabled several milestones within this research. In particular it was the MiCAM diagrams that pointed in the directing of examining the feature extraction process as the source of order related performance changes. It also displays examples of localizing pixel elements that are associated with a benign or maleficent classes.

This concludes this manuscript. Following is an Appendix that details CNN model definitions and the bibliography.

Appendices

APPENDIX A: LENET MODEL DETAILS

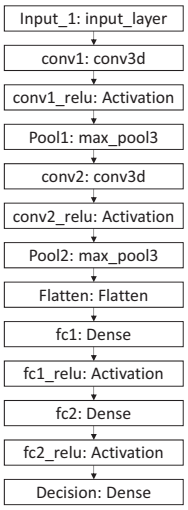


Figure A.1: Metric Malware Analysis LeNet-5 CNN Model

Table A.1: Process/Metric Malware Analysis LeNet-5 Model Details

LeNet-5	Layer		Input Size			Filter						Param
Stage	No.	Name	Hght	Wdth	Dpth	Cnt	Hght	Wdth	Dpth	Strd	Pad	Count
Convolution-1	1	conv1	150	75	1	32	3	3	1	1	1	288
Convolution-1	2	conv1_relu	150	75	32							32
Pool-1	3	pool1	150	75	32	0	2	2	1	2	1	0
Convolution-2	4	conv2	75	38	32	64	3	3	32	1	1	18,432
Convolution-2	5	conv2_relu	75	38	64							64
Pool-2	6	pool2	75	38	64	0	2	2	1	2	1	0
	7	flatten	38	19	64							0
Fully Connected-1	8	fc1	1	1	46,208	1024						47,316,992
Fully Connected-1	9	conv3_relu	1	1	1024							0
Fully Connected-2	10	fc2	1	1	1024	512						524,288
Fully Connected-2	11	conv3_relu	1	1	512							0
Decision	12	dense	1	1	512	2						1024

Table A.2: IP Maleficent Analysis LeNet-5 Model Details

LeNet-5	Layer		Input Size			Filter						Param
Stage	No.	Name	Hght	Wdth	Dpth	Cnt	Hght	Wdth	Dpth	Strd	Pad	Count
Convolution-1	1	conv1	40	40	1	32	3	3	3	1	1	320
Convolution-1	2	conv1_relu	40	40	32							0
Pool-1	3	pool1	40	40	32	0	2	2	1	2	1	0
Convolution-2	4	conv2	20	20	32	64	3	3	32	1	1	18,496
Convolution-2	5	conv2_relu	20	20	64							0
Pool-2	6	pool2	20	20	64	0	2	2	1	2	1	0
	7	flatten	10	10	64							0
Fully Connected-1	8	fc1	1	1	6400	4096						26,214,400
Fully Connected-1	9	conv3_relu	1	1	4096							0
	10	droput	1	1	4096							0
Fully Connected-2	11	fc2	1	1	4096	1600						6,553,600
Fully Connected-2	12	conv3_relu	1	1	1600							0
	13	droput	1	1	1600							0
Decision	14	dense	1	1	1600	2						3200

APPENDIX B: INCEPTION NET V3 MODEL DETAILS

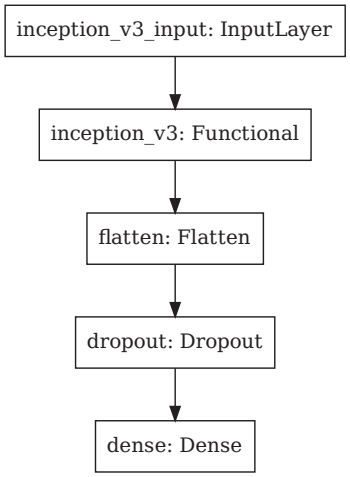


Figure B.1: Inception CNN Model Decision Layers

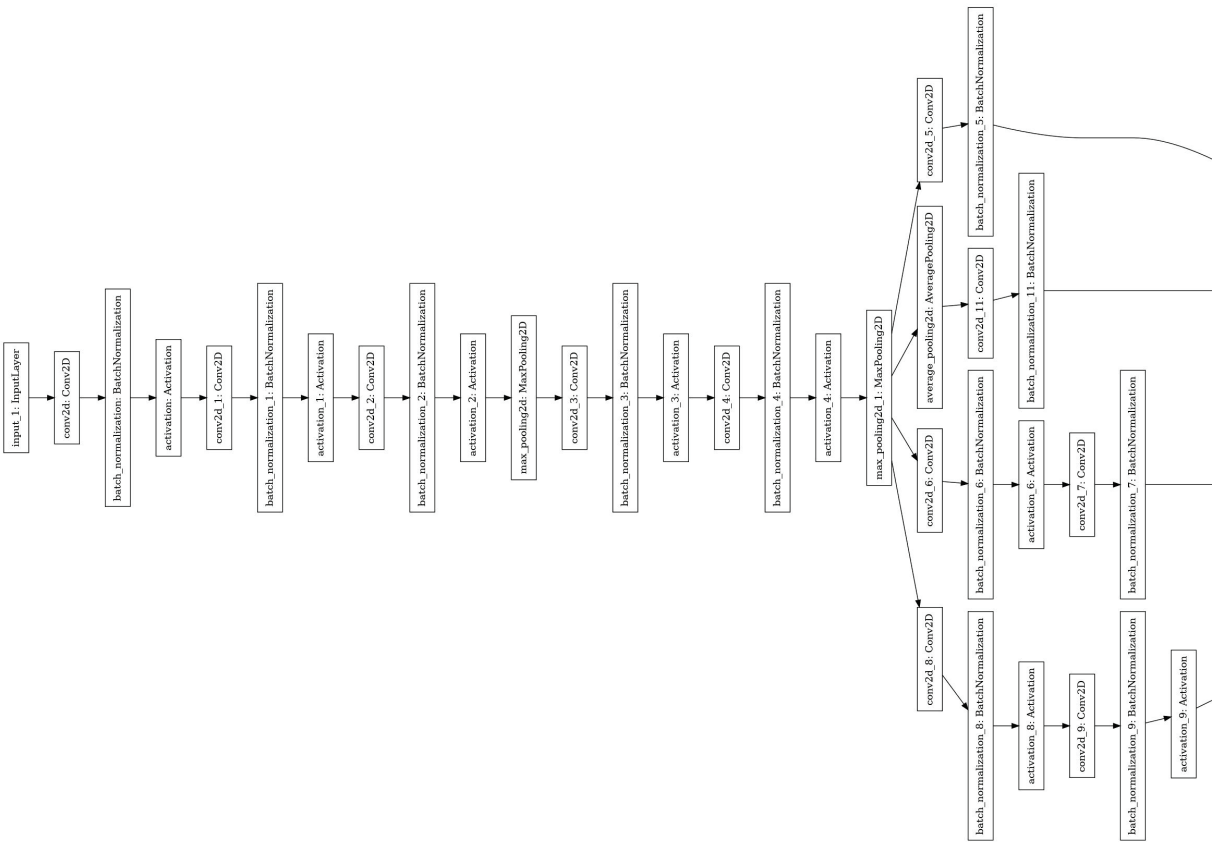


Figure B.2: Inception CNN Model Functional Layers: Page 1 of 7

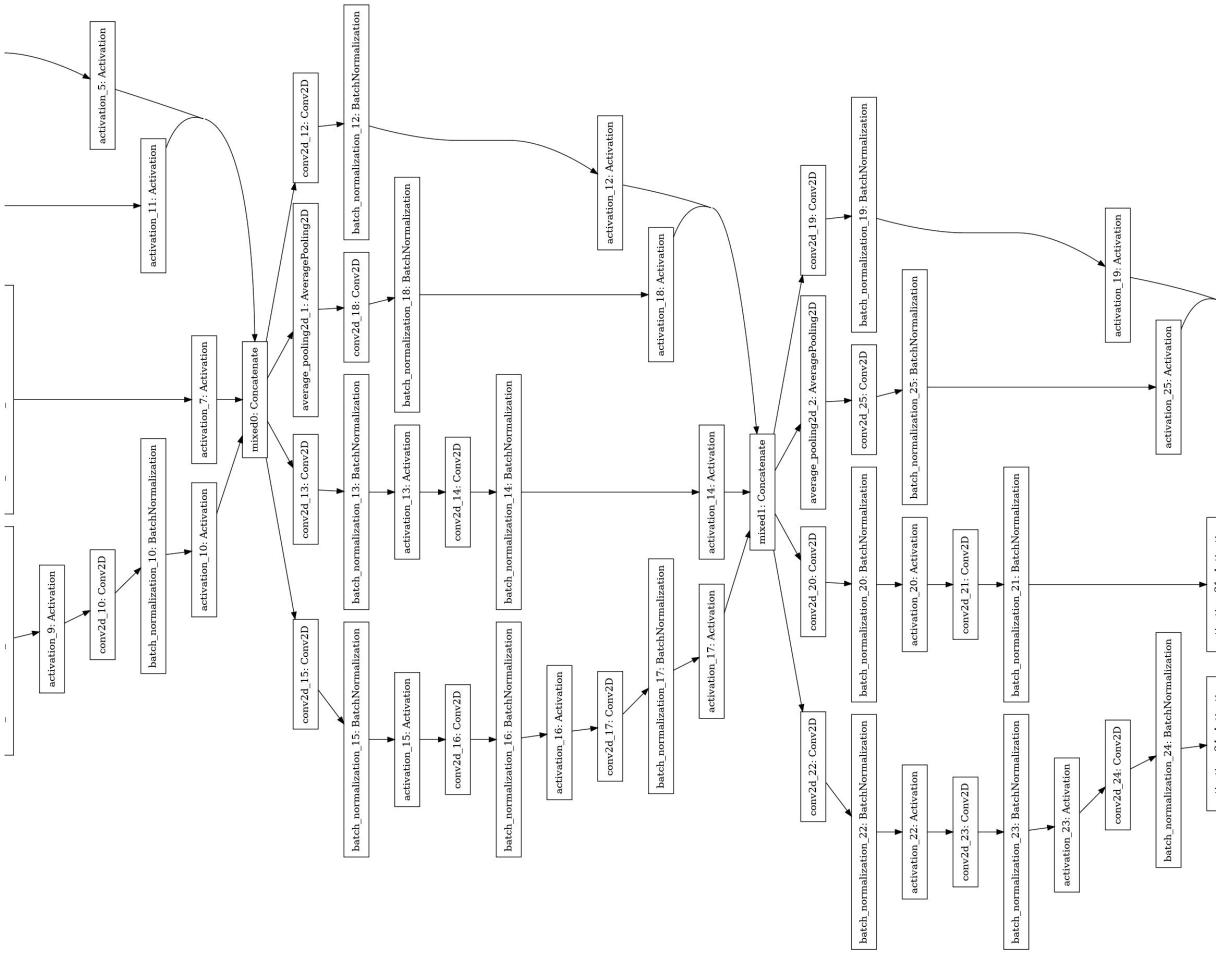


Figure B.3: Inception CNN Model Functional Layers: Page 2 of 7

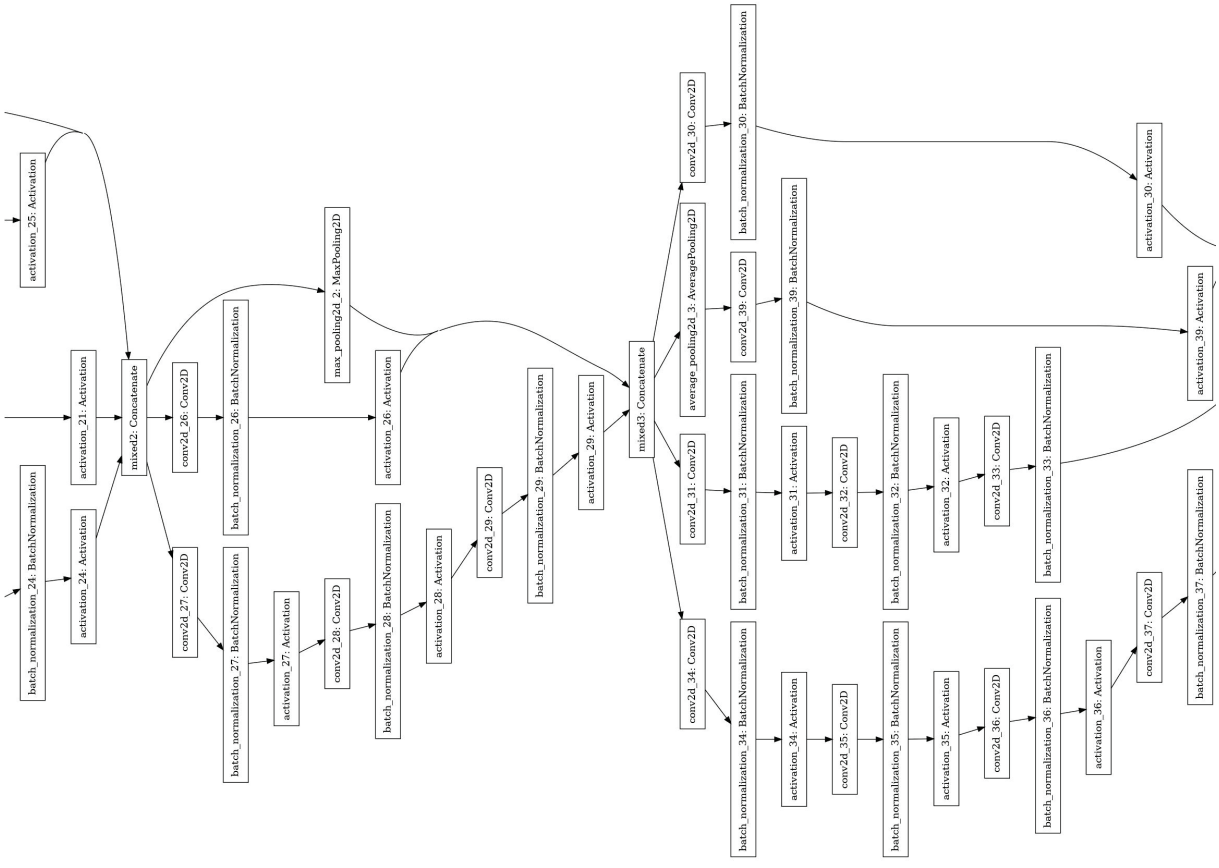


Figure B.4: Inception CNN Model Functional Layers: Page 3 of 7

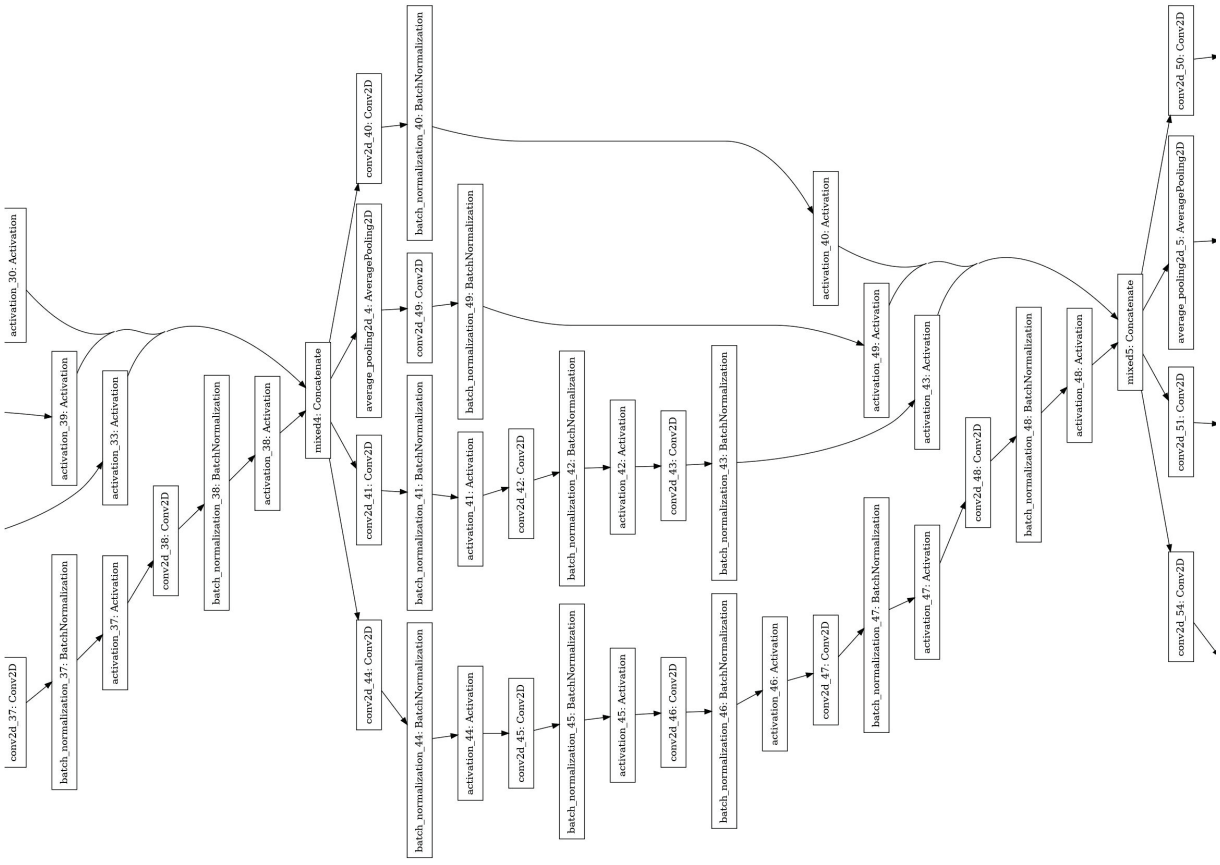


Figure B.5: Inception CNN Model Functional Layers: Page 4 of 7

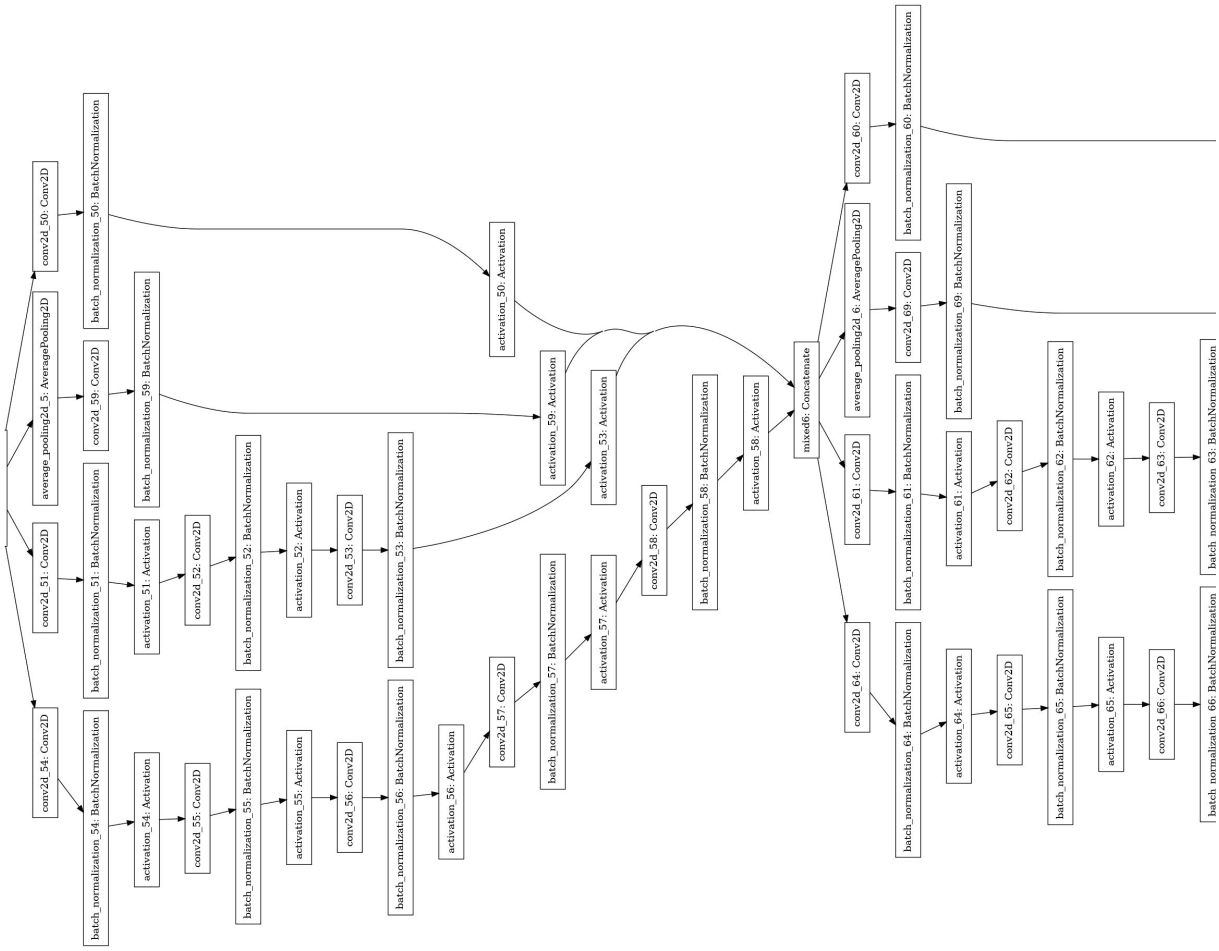


Figure B.6: Inception CNN Model Functional Layers: Page 5 of 7

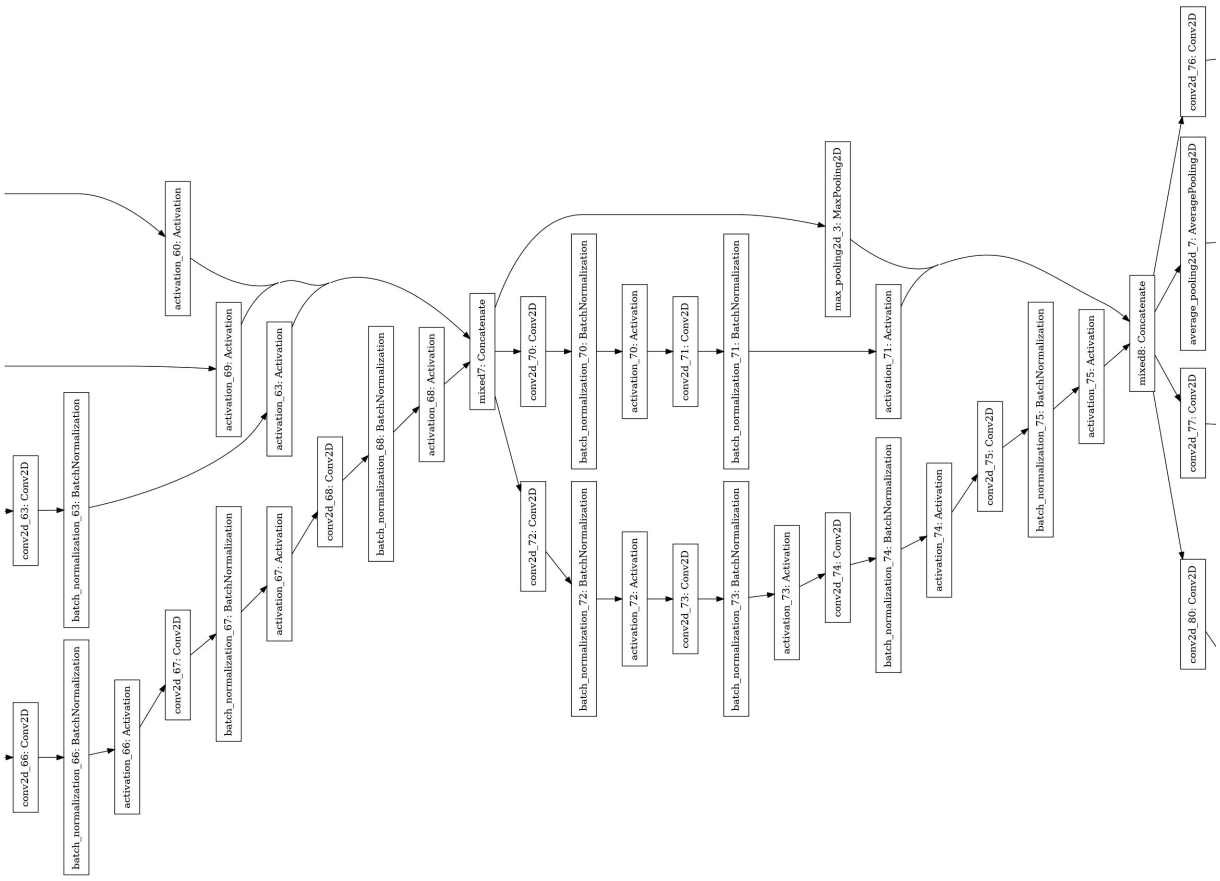


Figure B.7: Inception CNN Model Functional Layers: Page 6 of 7

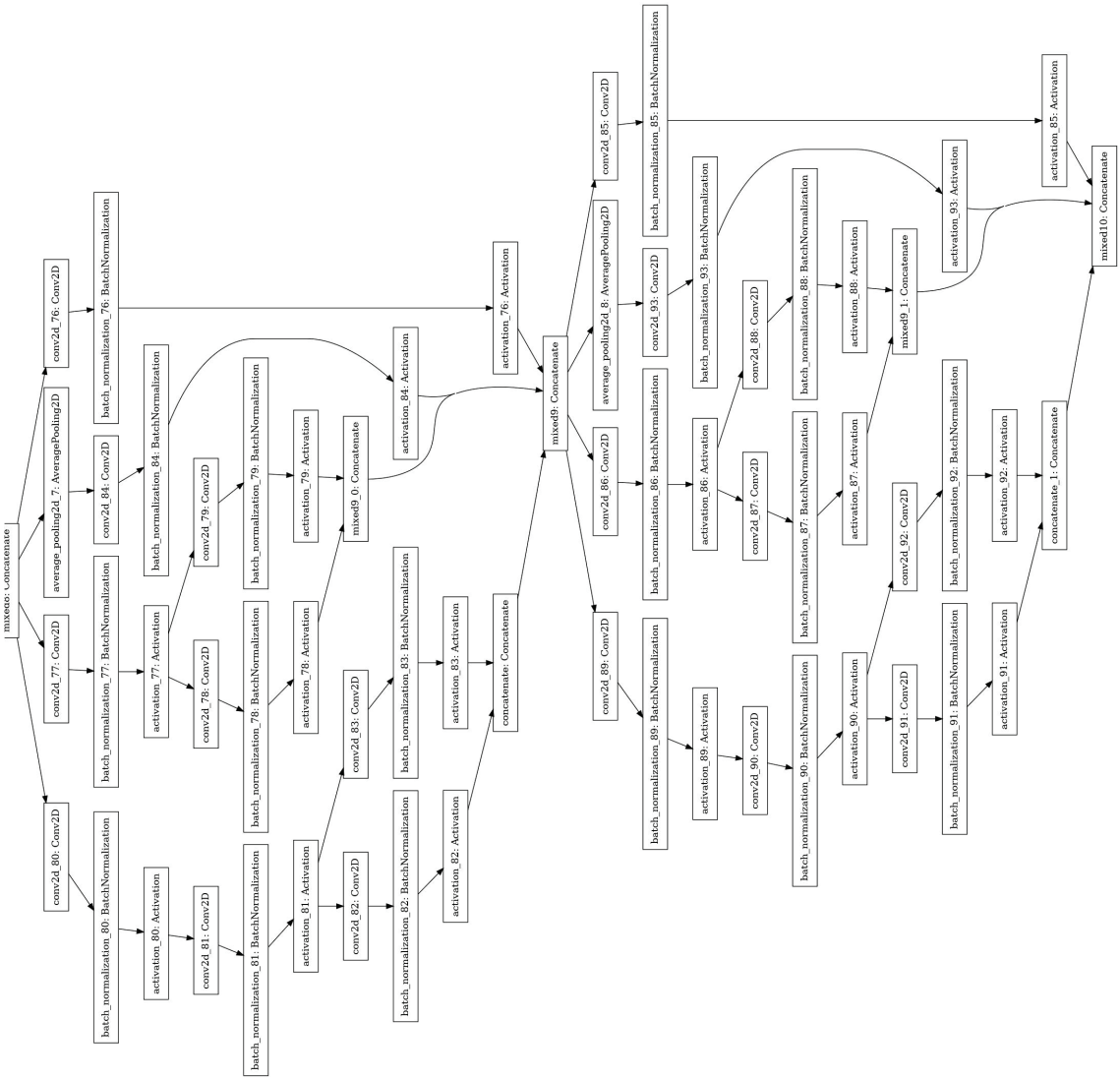


Figure B.8: Inception CNN Model Functional Layers: Page 7 of 7

Table B.1: Inception V3 Details

Inception	Layer		Input Size			Filter						Param
Stage	No.	Name	Height	Width	Depth	Count	Hght	Wdth	Dpth	Strd	Pad	Cnt
Pre-feature A	1	conv2d	150	75	1	32	3	3	1	2	0	288
Pre-feature A	2	batch_norm	74	37	32							96
Pre-feature A	3	activation	74	37	32							0
Pre-feature A	4	conv2d_1	74	37	32	32	3	3	32	1	0	9,216
Pre-feature A	5	batch_norm_1	72	35	32							96
Pre-feature A	6	activation_1	72	35	32							0
Pre-feature A	7	conv2d_2	72	35	32	64	3	3	32	1	1	18,432
Pre-feature A	8	batch_norm_2	72	35	64							192
Pre-feature A	9	activation_2	72	35	64							0
	10	max_pool2d	72	35	64	0	3	3	1	2	0	0
Pre-feature B	11	conv2d_3	35	17	64	80	1	1	64	1	0	5,120
Pre-feature B	12	batch_norm_3	35	17	80							240
Pre-feature B	13	activation_3	35	17	80							0
Pre-feature B	14	conv2d_4	35	17	80	192	3	3	80	1	0	138,240
Pre-feature B	15	batch_norm_4	33	15	192							576

(*table continues*)

Table B.1: Inception V3 Details

Inception	Layer		Input Size			Filter						Param
Stage	No.	Name	Height	Width	Depth	Count	Hght	Wdth	Dpth	Strd	Pad	Cnt
Pre-feature B	16	activation_4	33	15	192							0
	17	max_pool2d_1	33	15	192	0	3	3	1	2	0	0
Incept A1	18	conv2d_8	16	7	192	64	1	1	192	1	0	12,288
Incept A1	19	batch_norm_8	16	7	64							192
Incept A1	20	activation_8	16	7	64							0
Incept A1	21	conv2d_6	16	7	192	48	1	1	192	1	0	9,216
Incept A1	22	conv2d_9	16	7	64	96	3	3	64	1	1	55,296
Incept A1	23	batch_norm_6	16	7	96							144
Incept A1	24	batch_norm_9	16	7	48							288
Incept A1	25	activation_6	16	7	96							0
Incept A1	26	activation_9	16	7	48							0
Incept A1	27	average_pool2d	16	7	192	0	3	3	1	1	1	0
Incept A1	28	conv2d_5	16	7	192	64	1	1	192	1	0	12,288
Incept A1	29	conv2d_7	16	7	48	64	5	5	48	1	2	76,800
Incept A1	30	conv2d_10	16	7	96	96	3	3	96	1	1	82,944

(*table continues*)

Table B.1: Inception V3 Details

Inception	Layer		Input Size			Filter						Param
Stage	No.	Name	Height	Width	Depth	Count	Hght	Wdth	Dpth	Strd	Pad	Cnt
Incept A1	31	conv2d_11	16	7	192	32	1	1	192	1	0	6,144
Incept A1	32	batch_norm_5	16	7	64							192
Incept A1	33	batch_norm_7	16	7	64							192
Incept A1	34	batch_norm_10	16	7	96							288
Incept A1	35	batch_norm_11	16	7	32							96
Incept A1	36	activation_5	16	7	64							0
Incept A1	37	activation_7	16	7	64							0
Incept A1	38	activation_10	16	7	96							0
Incept A1	39	activation_11	16	7	32							0
	40	mixed0	16	7	256							0
Incept A2	41	conv2d_15	16	7	256	64	1	1	256	1	0	16,384
Incept A2	42	batch_norm_15	16	7	64							192
Incept A2	43	activation_15	16	7	64							0
Incept A2	44	conv2d_13	16	7	256	48	1	1	64	1	0	12,288
Incept A2	45	conv2d_16	16	7	64	96	3	3	48	1	1	55,296

(*table continues*)

Table B.1: Inception V3 Details

Inception	Layer		Input Size			Filter						Param
Stage	No.	Name	Height	Width	Depth	Count	Hght	Wdth	Dpth	Strd	Pad	Cnt
Incept A2	46	batch_norm_13	16	7	48							144
Incept A2	47	batch_norm_16	16	7	96							288
Incept A2	48	activation_13	16	7	48							0
Incept A2	49	activation_16	16	7	96							0
Incept A2	50	average_pool2d_1	16	7	256	0	3	3	1	1	1	0
Incept A2	51	conv2d_12	16	7	256	64	1	1	256	1	0	16,384
Incept A2	52	conv2d_14	16	7	48	64	5	5	64	1	2	76,800
Incept A2	53	conv2d_17	16	7	96	96	3	3	64	1	1	82,944
Incept A2	54	conv2d_18	16	7	256	64	1	1	96	1	0	16,384
Incept A2	55	batch_norm_12	16	7	64							192
Incept A2	56	batch_norm_14	16	7	64							192
Incept A2	57	batch_norm_17	16	7	96							288
Incept A2	58	batch_norm_18	16	7	64							192
Incept A2	59	activation_12	16	7	64							0
Incept A2	60	activation_14	16	7	64							0

(*table continues*)

Table B.1: Inception V3 Details

Inception	Layer		Input Size			Filter						Param
Stage	No.	Name	Height	Width	Depth	Count	Hght	Wdth	Dpth	Strd	Pad	Cnt
Incept A2	61	activation_17	16	7	96							0
Incept A2	62	activation_18	16	7	64							0
	63	mixed1	16	7	288							0
Incept A3	64	conv2d_22	16	7	288	64	1	1	288	1	0	18,432
Incept A3	65	batch_norm_22	16	7	64							192
Incept A3	66	activation_22	16	7	64							0
Incept A3	67	conv2d_20	16	7	288	48	1	1	288	1	0	13,824
Incept A3	68	conv2d_23	16	7	64	96	3	3	64	1	1	55,296
Incept A3	69	batch_norm_20	16	7	96							144
Incept A3	70	batch_norm_23	16	7	48							288
Incept A3	71	activation_20	16	7	96							0
Incept A3	72	activation_23	16	7	48							0
Incept A3	73	average_pool2d_2	16	7	288	0	3	3	1	1	1	0
Incept A3	74	conv2d_19	16	7	288	64	1	1	288	1	0	18,432
Incept A3	75	conv2d_21	16	7	48	64	5	5	64	1	2	76,800

(*table continues*)

Table B.1: Inception V3 Details

Inception	Layer		Input Size			Filter						Param
Stage	No.	Name	Height	Width	Depth	Count	Hght	Wdth	Dpth	Strd	Pad	Cnt
Incept A3	76	conv2d_24	16	7	96	96	3	3	64	1	1	82,944
Incept A3	77	conv2d_25	16	7	288	64	1	1	96	1	0	18,432
Incept A3	78	batch_norm_19	16	7	64							192
Incept A3	79	batch_norm_21	16	7	64							192
Incept A3	80	batch_norm_24	16	7	64							288
Incept A3	81	batch_norm_25	16	7	96							192
Incept A3	82	activation_19	16	7	64							0
Incept A3	83	activation_21	16	7	64							0
Incept A3	84	activation_24	16	7	64							0
Incept A3	85	activation_25	16	7	96							0
	86	mixed2	16	7	288							0
Incept B	87	conv2d_27	16	7	288	64	1	1	288	1	0	18,432
Incept B	88	batch_norm_27	16	7	64							192
Incept B	89	activation_27	16	7	64							0
Incept B	90	conv2d_28	16	7	64	96	3	3	64	1	1	55,296

(*table continues*)

Table B.1: Inception V3 Details

Inception	Layer		Input Size			Filter						Param
Stage	No.	Name	Height	Width	Depth	Count	Hght	Wdth	Dpth	Strd	Pad	Cnt
Incept B	91	batch_norm_28	16	7	96							288
Incept B	92	activation_28	16	7	96							0
Incept B	93	conv2d_26	16	7	288	384	3	3	96	2	0	995,328
Incept B	94	conv2d_29	7	3	96	96	3	3	96	2	0	82,944
Incept B	95	batch_norm_26	7	3	384							1,152
Incept B	96	batch_norm_29	7	3	96							288
Incept B	97	activation_26	7	3	384							0
Incept B	98	activation_29	7	3	96							0
Incept B	99	max_pool2d_2	7	3	288	0	3	3	1	2	0	0
	100	mixed3	7	3	768							0
Incept C1	101	conv2d_34	7	3	768	128	1	1	768	1	0	98,304
Incept C1	102	batch_norm_34	7	3	128							384
Incept C1	103	activation_34	7	3	128							0
Incept C1	104	conv2d_35	7	3	128	128	7	1	128	1	3	114,688
Incept C1	105	batch_norm_35	7	3	128							384

(*table continues*)

Table B.1: Inception V3 Details

Inception	Layer		Input Size			Filter						Param
Stage	No.	Name	Height	Width	Depth	Count	Hght	Wdth	Dpth	Strd	Pad	Cnt
Incept C1	106	activation_35	7	3	128							0
Incept C1	107	conv2d_31	7	3	768	128	1	1	768	1	0	98,304
Incept C1	108	conv2d_36	7	3	128	128	1	7	128	1	3	114,688
Incept C1	109	batch_norm_31	7	3	128							384
Incept C1	110	batch_norm_36	7	3	128							384
Incept C1	111	activation_31	7	3	128							0
Incept C1	112	activation_36	7	3	128							0
Incept C1	113	conv2d_32	7	3	128	128	1	7	128	1	3	114,688
Incept C1	114	conv2d_37	7	3	128	128	7	1	128	1	3	114,688
Incept C1	115	batch_norm_32	7	3	128							384
Incept C1	116	batch_norm_37	7	3	128							384
Incept C1	117	activation_32	7	3	128							0
Incept C1	118	activation_37	7	3	128							0
Incept C1	119	average_pool2d_3	7	3	768	0	3	3	1	1	1	0
Incept C1	120	conv2d_30	7	3	768	192	1	1	768	1	0	147,456

(*table continues*)

Table B.1: Inception V3 Details

Inception	Layer		Input Size			Filter						Param
Stage	No.	Name	Height	Width	Depth	Count	Hght	Wdth	Dpth	Strd	Pad	Cnt
Incept C1	121	conv2d_33	7	3	128	192	7	1	128	1	3	172,032
Incept C1	122	conv2d_38	7	3	128	192	1	7	128	1	3	172,032
Incept C1	123	conv2d_39	7	3	768	192	1	1	768	1	0	147,456
Incept C1	124	batch_norm_30	7	3	192							576
Incept C1	125	batch_norm_33	7	3	192							576
Incept C1	126	batch_norm_38	7	3	192							576
Incept C1	127	batch_norm_39	7	3	192							576
Incept C1	128	activation_30	7	3	192							0
Incept C1	129	activation_33	7	3	192							0
Incept C1	130	activation_38	7	3	192							0
Incept C1	131	activation_39	7	3	192							0
	132	mixed4	7	3	768							0
Incept C2	133	conv2d_44	7	3	768	160	1	1	768	1	0	122,880
Incept C2	134	batch_norm_44	7	3	160							480
Incept C2	135	activation_44	7	3	160							0

(*table continues*)

Table B.1: Inception V3 Details

Inception	Layer		Input Size			Filter						Param
Stage	No.	Name	Height	Width	Depth	Count	Hght	Wdth	Dpth	Strd	Pad	Cnt
Incept C2	136	conv2d_45	7	3	160	160	7	1	160	1	3	179,200
Incept C2	137	batch_norm_45	7	3	160							480
Incept C2	138	activation_45	7	3	160							0
Incept C2	139	conv2d_41	7	3	768	160	1	1	768	1	0	122,880
Incept C2	140	conv2d_46	7	3	160	160	1	7	160	1	3	179,200
Incept C2	141	batch_norm_41	7	3	160							480
Incept C2	142	batch_norm_46	7	3	160							480
Incept C2	143	activation_41	7	3	160							0
Incept C2	144	activation_46	7	3	160							0
Incept C2	145	conv2d_42	7	3	160	160	1	7	160	1	3	179,200
Incept C2	146	conv2d_47	7	3	160	160	7	1	160	1	3	179,200
Incept C2	147	batch_norm_42	7	3	160							480
Incept C2	148	batch_norm_47	7	3	160							480
Incept C2	149	activation_42	7	3	160							0
Incept C2	150	activation_47	7	3	160							0

(*table continues*)

Table B.1: Inception V3 Details

Inception	Layer		Input Size			Filter						Param
Stage	No.	Name	Height	Width	Depth	Count	Hght	Wdth	Dpth	Strd	Pad	Cnt
Incept C2	151	average_pool2d_4	7	3	768	0	3	3	1	1	1	0
Incept C2	152	conv2d_40	7	3	768	192	1	1	768	1	0	147,456
Incept C2	153	conv2d_43	7	3	160	192	7	1	160	1	3	215,040
Incept C2	154	conv2d_48	7	3	160	192	1	7	160	1	3	215,040
Incept C2	155	conv2d_49	7	3	768	192	1	1	768	1	0	147,456
Incept C2	156	batch_norm_40	7	3	192							576
Incept C2	157	batch_norm_43	7	3	192							576
Incept C2	158	batch_norm_48	7	3	192							576
Incept C2	159	batch_norm_49	7	3	192							576
Incept C2	160	activation_40	7	3	192							0
Incept C2	161	activation_43	7	3	192							0
Incept C2	162	activation_48	7	3	192							0
Incept C2	163	activation_49	7	3	192							0
	164	mixed5	7	3	768							0
Incept C3	165	conv2d_54	7	3	768	160	1	1	768	1	0	122,880

(*table continues*)

Table B.1: Inception V3 Details

Inception	Layer		Input Size			Filter						Param
Stage	No.	Name	Height	Width	Depth	Count	Hght	Wdth	Dpth	Strd	Pad	Cnt
Incept C3	166	batch_norm_54	7	3	160							480
Incept C3	167	activation_54	7	3	160							0
Incept C3	168	conv2d_55	7	3	160	160	7	1	160	1	3	179,200
Incept C3	169	batch_norm_55	7	3	160							480
Incept C3	170	activation_55	7	3	160							0
Incept C3	171	conv2d_51	7	3	768	160	1	1	768	1	0	122,880
Incept C3	172	conv2d_56	7	3	160	160	1	7	160	1	3	179,200
Incept C3	173	batch_norm_51	7	3	160							480
Incept C3	174	batch_norm_56	7	3	160							480
Incept C3	175	activation_51	7	3	160							0
Incept C3	176	activation_56	7	3	160							0
Incept C3	177	conv2d_52	7	3	160	160	1	7	160	1	3	179,200
Incept C3	178	conv2d_57	7	3	160	160	7	1	160	1	3	179,200
Incept C3	179	batch_norm_52	7	3	160							480
Incept C3	180	batch_norm_57	7	3	160							480

(*table continues*)

Table B.1: Inception V3 Details

Inception	Layer		Input Size			Filter						Param
Stage	No.	Name	Height	Width	Depth	Count	Hght	Wdth	Dpth	Strd	Pad	Cnt
Incept C3	181	activation_52	7	3	160							0
Incept C3	182	activation_57	7	3	160							0
Incept C3	183	average_pool2d_5	7	3	768	0	3	3	1	1	1	0
Incept C3	184	conv2d_50	7	3	768	192	1	1	768	1	0	147,456
Incept C3	185	conv2d_53	7	3	160	192	7	1	160	1	3	215,040
Incept C3	186	conv2d_58	7	3	160	192	1	7	160	1	3	215,040
Incept C3	187	conv2d_59	7	3	768	192	1	1	768	1	0	147,456
Incept C3	188	batch_norm_50	7	3	192							576
Incept C3	189	batch_norm_53	7	3	192							576
Incept C3	190	batch_norm_58	7	3	192							576
Incept C3	191	batch_norm_59	7	3	192							576
Incept C3	192	activation_50	7	3	192							0
Incept C3	193	activation_53	7	3	192							0
Incept C3	194	activation_58	7	3	192							0
Incept C3	195	activation_59	7	3	192							0

(*table continues*)

Table B.1: Inception V3 Details

Inception	Layer		Input Size			Filter						Param
Stage	No.	Name	Height	Width	Depth	Count	Hght	Wdth	Dpth	Strd	Pad	Cnt
	196	mixed6	7	3	768							0
Incept C4	197	conv2d_64	7	3	768	192	1	1	768	1	0	147,456
Incept C4	198	batch_norm_64	7	3	192							576
Incept C4	199	activation_64	7	3	192							0
Incept C4	200	conv2d_65	7	3	192	192	7	1	160	1	3	258,048
Incept C4	201	batch_norm_65	7	3	192							576
Incept C4	202	activation_65	7	3	192							0
Incept C4	203	conv2d_61	7	3	768	192	1	1	768	1	0	147,456
Incept C4	204	conv2d_66	7	3	192	192	1	7	160	1	3	258,048
Incept C4	205	batch_norm_61	7	3	192							576
Incept C4	206	batch_norm_66	7	3	192							576
Incept C4	207	activation_61	7	3	192							0
Incept C4	208	activation_66	7	3	192							0
Incept C4	209	conv2d_62	7	3	192	192	1	7	160	1	3	258,048
Incept C4	210	conv2d_67	7	3	192	192	7	1	160	1	3	258,048

(*table continues*)

Table B.1: Inception V3 Details

Inception	Layer		Input Size			Filter						Param
Stage	No.	Name	Height	Width	Depth	Count	Hght	Wdth	Dpth	Strd	Pad	Cnt
Incept C4	211	batch_norm_62	7	3	192							576
Incept C4	212	batch_norm_67	7	3	192							576
Incept C4	213	activation_62	7	3	192							0
Incept C4	214	activation_67	7	3	192							0
Incept C4	215	average_pool2d_6	7	3	768	0	3	3	1	1	1	0
Incept C4	216	conv2d_60	7	3	768	192	1	1	768	1	0	147,456
Incept C4	217	conv2d_63	7	3	192	192	7	1	192	1	3	258,048
Incept C4	218	conv2d_68	7	3	192	192	1	7	192	1	3	258,048
Incept C4	219	conv2d_69	7	3	768	192	1	1	768	1	0	147,456
Incept C4	220	batch_norm_60	7	3	192							576
Incept C4	221	batch_norm_63	7	3	192							576
Incept C4	222	batch_norm_68	7	3	192							576
Incept C4	223	batch_norm_69	7	3	192							576
Incept C4	224	activation_60	7	3	192							0
Incept C4	225	activation_63	7	3	192							0

(*table continues*)

Table B.1: Inception V3 Details

Inception	Layer		Input Size			Filter						Param
Stage	No.	Name	Height	Width	Depth	Count	Hght	Wdth	Dpth	Strd	Pad	Cnt
Incept C4	226	activation_68	7	3	192							0
Incept C4	227	activation_69	7	3	192							0
	228	mixed7	7	3	768							0
Incept D	229	conv2d_72	7	3	768	192	1	1	768	1	0	147,456
Incept D	230	batch_norm_72	7	3	192							576
Incept D	231	activation_72	7	3	192							0
Incept D	232	conv2d_73	7	3	192	192	1	7	192	1	3	258,048
Incept D	233	batch_norm_73	7	3	192							576
Incept D	234	activation_73	7	3	192							0
Incept D	235	conv2d_70	7	3	768	192	1	1	768	1	0	147,456
Incept D	236	conv2d_74	7	3	192	192	1	7	192	1	3	258,048
Incept D	237	batch_norm_70	7	3	192							576
Incept D	238	batch_norm_74	7	3	192							576
Incept D	239	activation_70	7	3	192							0
Incept D	240	activation_74	7	3	192							0

(*table continues*)

Table B.1: Inception V3 Details

Inception	Layer		Input Size			Filter						Param
Stage	No.	Name	Height	Width	Depth	Count	Hght	Wdth	Dpth	Strd	Pad	Cnt
Incept D	241	conv2d_71	7	3	192	320	3	3	192	2	0	552,960
Incept D	242	conv2d_75	3	1	192	192	3	3	320	2	0	331,776
Incept D	243	batch_norm_71	3	1	320							960
Incept D	244	batch_norm_75	3	1	192							576
Incept D	245	activation_71	3	1	320							0
Incept D	246	activation_75	3	1	192							0
Incept D	247	max_pool2d_3	3	1	768	0	3	3	1	2	0	0
	248	mixed8	3	1	1,280							0
Incept E1	249	conv2d_80	3	1	1,280	448	1	1	1,280	1	0	573,440
Incept E1	250	batch_norm_80	3	1	448							1344
Incept E1	251	activation_80	3	1	448							0
Incept E1	252	conv2d_77	3	1	1,280	384	1	1	1,280	1	0	491,520
Incept E1	253	conv2d_81	3	1	448	384	3	3	384	2	0	1,548,288
Incept E1	254	batch_norm_77	3	1	384							1,152
Incept E1	255	batch_norm_81	3	1	384							1,152

(*table continues*)

Table B.1: Inception V3 Details

Inception	Layer		Input Size			Filter						Param
Stage	No.	Name	Height	Width	Depth	Count	Hght	Wdth	Dpth	Strd	Pad	Cnt
Incept E1	256	activation_77	3	1	384							0
Incept E1	257	activation_81	3	1	384							0
Incept E1	258	conv2d_78	3	1	384	384	1	3	384	1	0	442,368
Incept E1	259	conv2d_79	3	1	384	384	3	1	384	1	0	442,368
Incept E1	260	conv2d_82	3	1	384	384	1	3	384	1	0	442,368
Incept E1	261	conv2d_83	3	1	384	384	3	1	384	1	0	442,368
Incept E1	262	average_pool2d_7	3	1	1,280	0	3	3	1	1	1	0
Incept E1	263	conv2d_76	3	1	1,280	320	1	1	1,280	1	0	409,600
Incept E1	264	batch_norm_78	3	1	320							1,152
Incept E1	265	batch_norm_79	3	1	384							1,152
Incept E1	266	batch_norm_82	3	1	384							1,152
Incept E1	267	batch_norm_83	3	1	384							1,152
Incept E1	268	conv2d_84	3	1	1,280	192	1	1	1,280	1	0	245,760
Incept E1	269	batch_norm_76	3	1	320							960
Incept E1	270	activation_78	3	1	320							0

(*table continues*)

Table B.1: Inception V3 Details

Inception	Layer		Input Size			Filter						Param
Stage	No.	Name	Height	Width	Depth	Count	Hght	Wdth	Dpth	Strd	Pad	Cnt
Incept E1	271	activation_79	3	1	384							0
Incept E1	272	activation_82	3	1	384							0
Incept E1	273	activation_83	3	1	384							0
Incept E1	274	batch_norm_84	3	1	192							576
Incept E1	275	activation_76	3	1	320							0
Incept E1	276	mixed9_0	3	1	768							0
Incept E1	277	concatenate	3	1	768							0
Incept E1	278	activation_84	3	1	192							0
	279	mixed9	3	1	2,048							0
Incept E2	280	conv2d_89	3	1	2,048	448	1	1	2,048	1	0	917,504
Incept E2	281	batch_norm_89	3	1	448							1,344
Incept E2	282	activation_89	3	1	448							0
Incept E2	283	conv2d_86	3	1	2,048	384	1	1	2,048	1	0	786,432
Incept E2	284	conv2d_90	3	1	448	384	3	3	448	2	0	1,548,288
Incept E2	285	batch_norm_86	3	1	384							1,152

(*table continues*)

Table B.1: Inception V3 Details

Inception	Layer		Input Size			Filter						Param
Stage	No.	Name	Height	Width	Depth	Count	Hght	Wdth	Dpth	Strd	Pad	Cnt
Incept E2	286	batch_norm_90	3	1	384							1,152
Incept E2	287	activation_86	3	1	384							0
Incept E2	288	activation_90	3	1	384							0
Incept E2	289	conv2d_87	3	1	384	384	1	3	384	1	0	442,368
Incept E2	290	conv2d_88	3	1	384	384	3	1	384	1	0	442,368
Incept E2	291	conv2d_91	3	1	384	384	1	3	384	1	0	442,368
Incept E2	292	conv2d_92	3	1	384	384	3	1	384	1	0	442,368
Incept E2	293	average_pool2d_8	3	1	2,048	0	3	3	1	1	1	0
Incept E2	294	conv2d_85	3	1	2,048	320	1	1	2,048	1	0	655,360
Incept E2	295	batch_norm_87	3	1	384							1,152
Incept E2	296	batch_norm_88	3	1	384							1,152
Incept E2	297	batch_norm_91	3	1	384							1,152
Incept E2	298	batch_norm_92	3	1	384							1,152
Incept E2	299	conv2d_93	3	1	2,048	192	1	1	2,048	1	0	393,216
Incept E2	300	batch_norm_85	3	1	320							960

(*table continues*)

Table B.1: Inception V3 Details

Inception	Layer		Input Size			Filter						Param
Stage	No.	Name	Height	Width	Depth	Count	Hght	Wdth	Dpth	Strd	Pad	Cnt
Incept E2	301	activation_87	3	1	384							0
Incept E2	302	activation_88	3	1	384							0
Incept E2	303	activation_91	3	1	384							0
Incept E2	304	activation_92	3	1	384							0
Incept E2	305	batch_norm_93	3	1	192							576
Incept E2	306	activation_85	3	1	320							0
Incept E2	307	mixed9_1	3	1	768							0
Incept E2	308	concatenate_1	3	1	768							0
Incept E2	309	activation_93	3	1	192							0
	310	mixed10	3	1	2048							0
	311	flatten	3	1	2048							0
	312	dropout	1	1	6144							0
Decision	313	dense	1	1	6144							12,290

APPENDIX C: RESNET-18 MODEL DETAILS

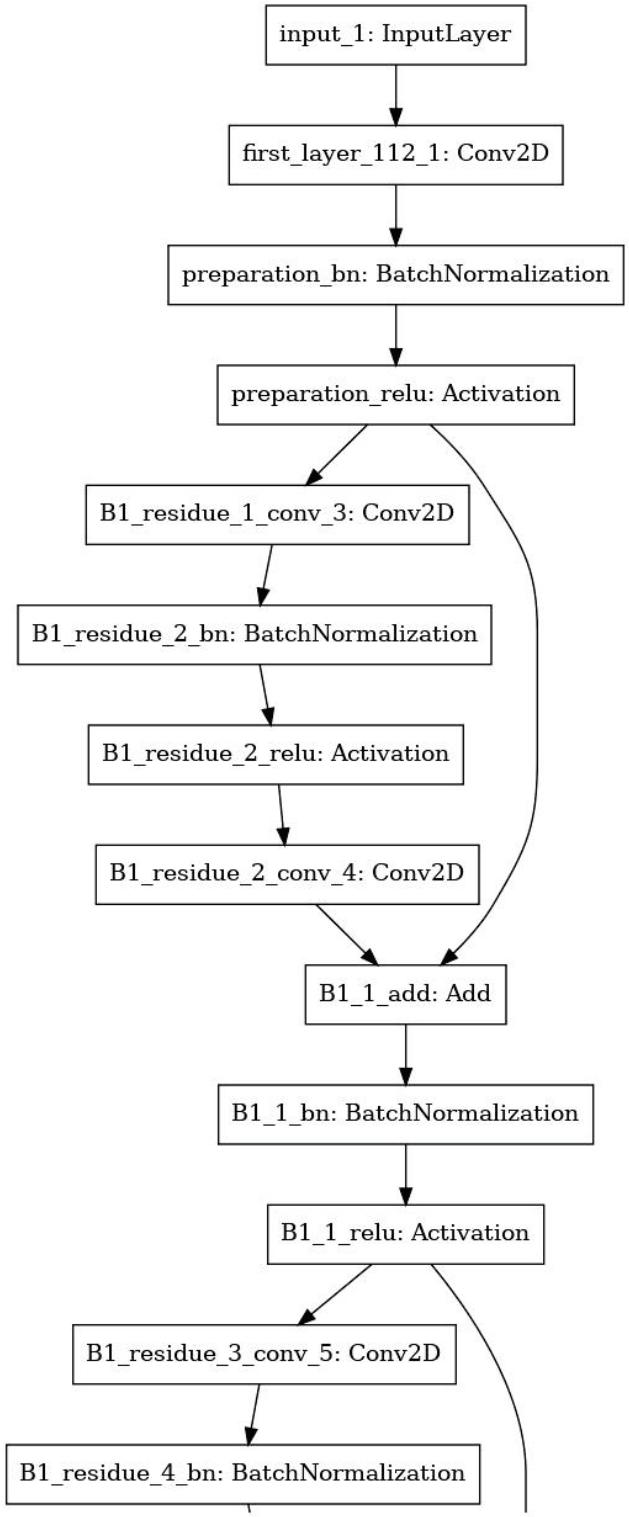


Figure C.1: ResNet18 CNN Model Functional Layers: Page 1 of 5

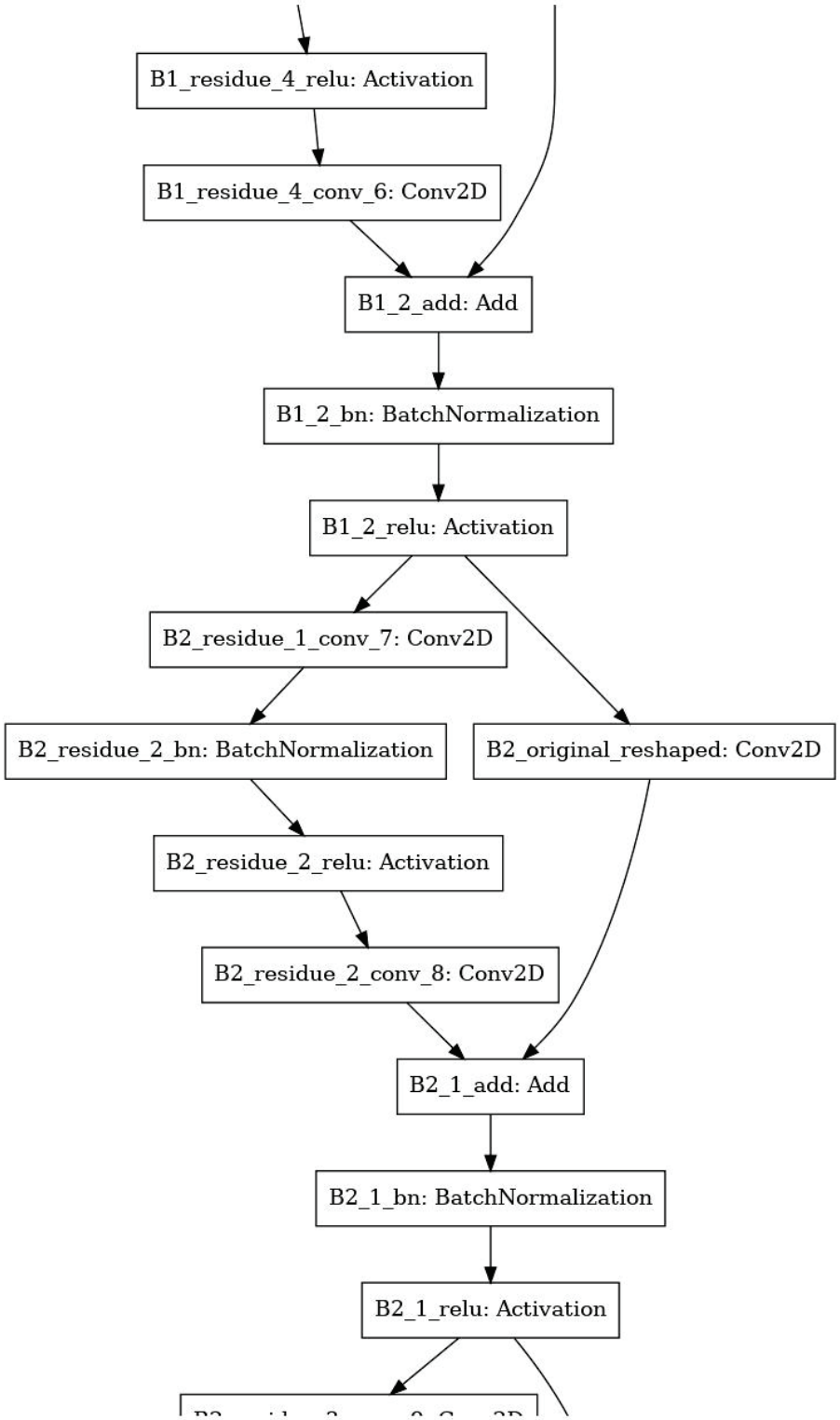


Figure C.2: ResNet18 CNN Model Functional Layers: Page 2 of 5

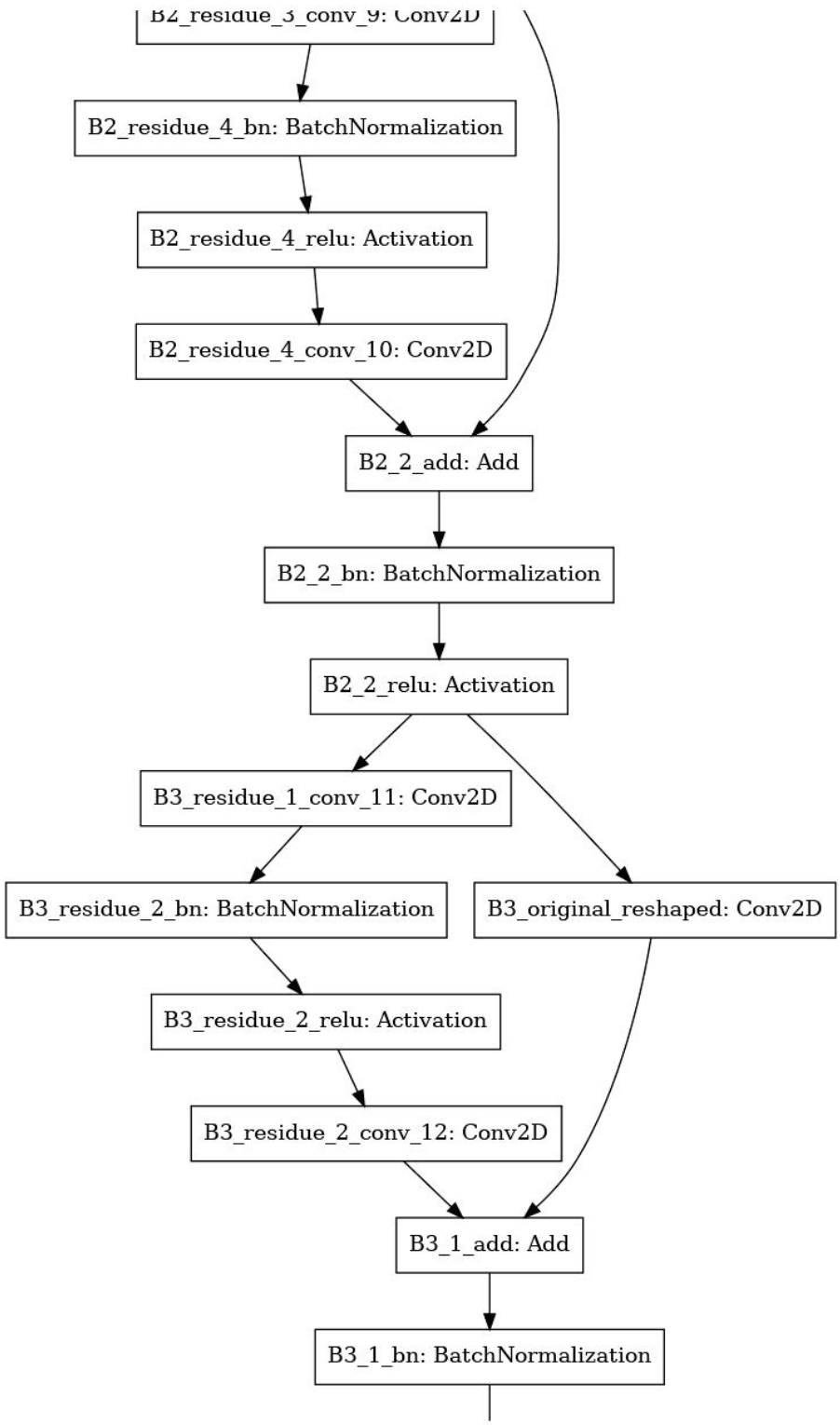


Figure C.3: ResNet18 CNN Model Functional Layers: Page 3 of 5

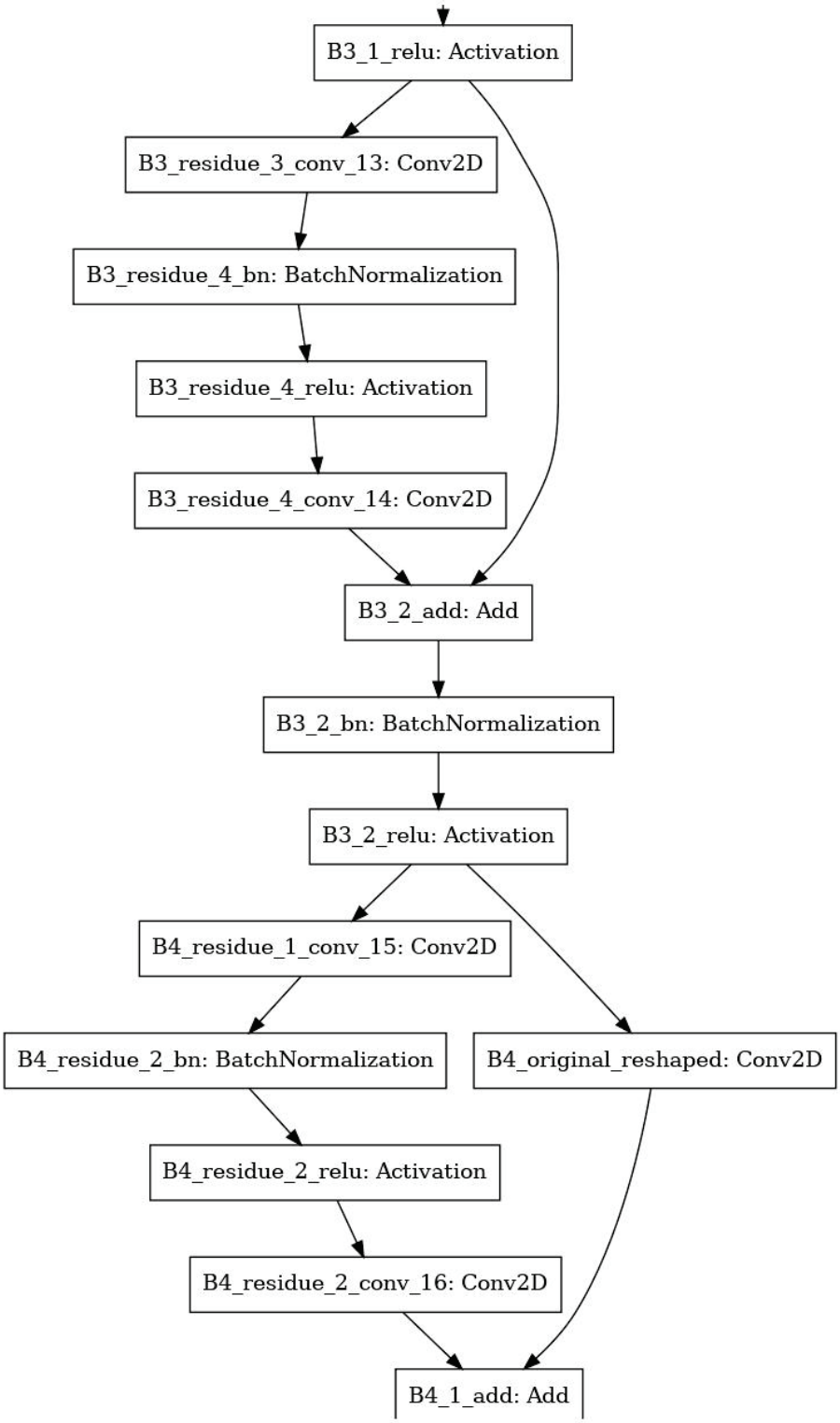


Figure C.4: ResNet18 CNN Model Functional Layers: Page 4 of 5

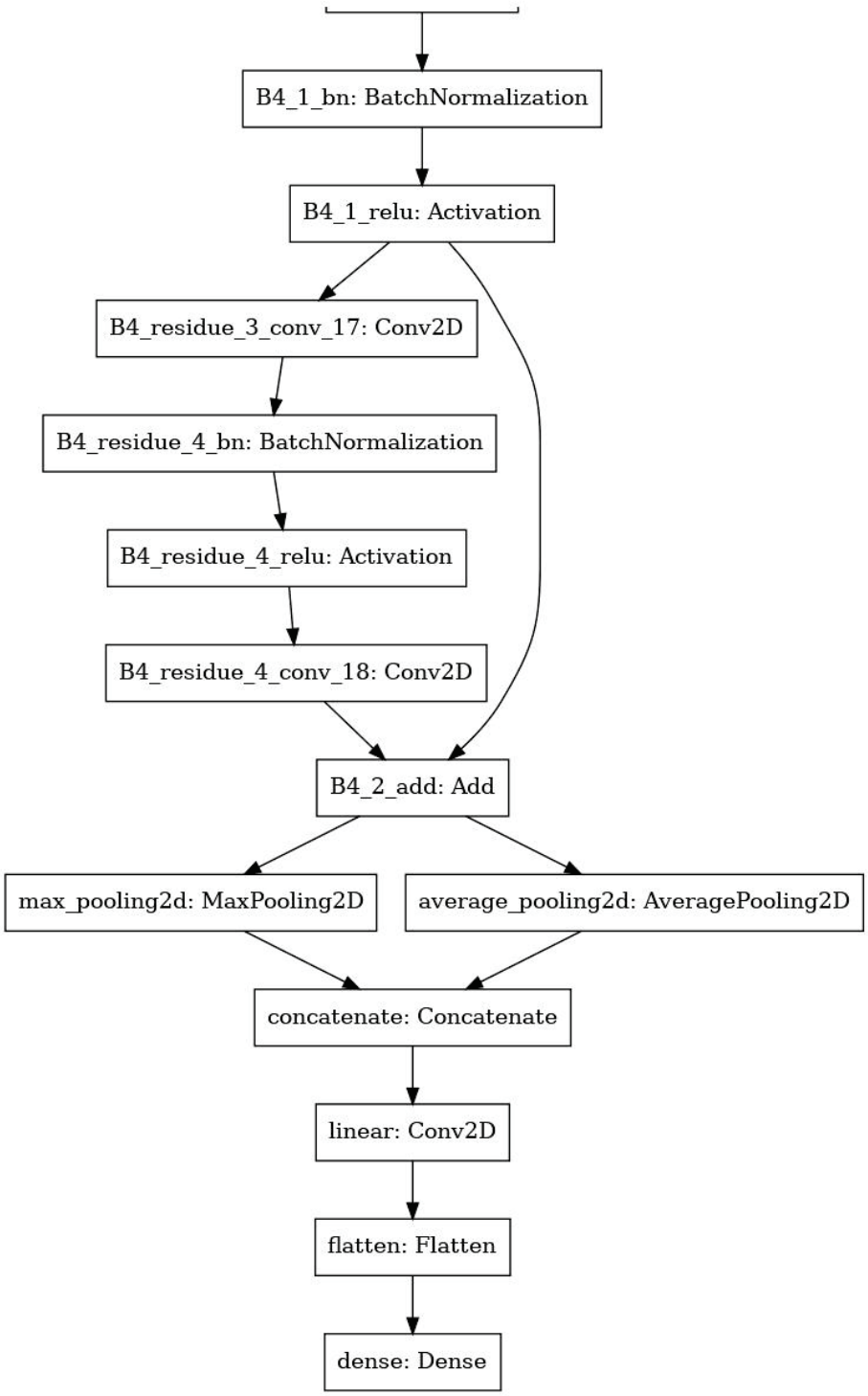


Figure C.5: ResNet18 CNN Model Functional Layers: Page 5 of 5

Table C.1: ResNet-18 Details

ResNET-18	Layer		Input Size			Filter						Param
Stage	No.	Name	Hght	Wdth	Dpth	Cnt	Hght	Wdth	Dpth	Strd	Pad	Count
Prefetch	1	first_layer_112_1	150	75	1	64	3	3	1	1	1	640
Prefetch	2	preparation_bn	150	75	64							256
Prefetch	3	preparation_relu	150	75	64							0
Residual Stage-1	4	B1_residue_1_conv_3	150	75	64	64	3	3	64	1	1	36,928
Residual Stage-1	5	B1_residue_2_bn	150	75	64							256
Residual Stage-1	6	B1_residue_2_relu	150	75	64							0
Residual Stage-1	7	B1_residue_2_conv_4	150	75	64	64	3	3	64	1	1	36,928
Join-1	8	B1_1_add	150	75	64							0
Join-1	9	B1_1_bn	150	75	64							256
Join-1	10	B1_1_relu	150	75	64							0
Residual Stage-2	11	B1_residue_3_conv_5	150	75	64	64	3	3	64	1	1	36,928
Residual Stage-2	12	B1_residue_4_bn	150	75	64							256
Residual Stage-2	13	B1_residue_4_relu	150	75	64							0
Residual Stage-2	14	B1_residue_4_conv_6	150	75	64	64	3	3	64	1	1	36,928
Join-2	15	B1_2_add	150	75	64							0

(*table continues*)

Table C.1: ResNet-18 Details

ResNET-18	Layer		Input Size			Filter						Param
Stage	No.	Name	Hght	Wdth	Dpth	Cnt	Hght	Wdth	Dpth	Strd	Pad	Count
Join-2	16	B1_2_bn	150	75	64							256
Join-2	17	B1_2_relu	150	75	64							0
Residual Stage-3	18	B2_residue_1_conv_7	150	75	64	128	3	3	64	2	0	73,856
Residual Stage-3	19	B2_residue_2_bn	75	38	128							512
Residual Stage-3	20	B2_residue_2_relu	75	38	128							0
Residual Link-3	21	B2_original_reshaped	150	75	64	128	1	1	128	2	0	8,320
Residual Stage-3	22	B2_residue_2_conv_8	75	38	128	128	3	3	128	1	1	14,7584
Join-3	23	B2_1_add	75	38	128							0
Join-3	24	B2_1_bn	75	38	128							512
Join-3	25	B2_1_relu	75	38	128							0
Residual Stage-4	26	B2_residue_3_conv_9	75	38	128	128	3	3	128	1	1	147,584
Residual Stage-4	27	B2_residue_4_bn	75	38	128							512
Residual Stage-4	28	B2_residue_4_relu	75	38	128							0
Residual Stage-4	29	B2_residue_4_conv_10	75	38	128	128	3	3	128	1	1	14,7584
Join-4	30	B2_2_add	75	38	128							0

(*table continues*)

Table C.1: ResNet-18 Details

ResNET-18	Layer		Input Size			Filter						Param
Stage	No.	Name	Hght	Wdth	Dpth	Cnt	Hght	Wdth	Dpth	Strd	Pad	Count
Join-4	31	B2_2_bn	75	38	128							512
Join-4	32	B2_2_relu	75	38	128							0
Residual Stage-5	33	B3_residue_1_conv_11	75	38	128	256	3	3	128	2	0	295,168
Residual Stage-5	34	B3_residue_2_bn	38	19	256							1024
Residual Stage-5	35	B3_residue_2_relu	38	19	256							0
Residual Link-5	36	B3_original_reshaped	75	38	128	256	1	2	256	2	0	33,024
Residual Stage-5	37	B3_residue_2_conv_12	38	19	256	256	3	3	256	1	1	590,080
Join-5	38	B3_1_add	38	19	256							0
Join-5	39	B3_1_bn	38	19	256							1024
Join-5	40	B3_1_relu	38	19	256							0
Residual Stage-6	41	B3_residue_3_conv_13	38	19	256	256	3	3	256	1	1	590,080
Residual Stage-6	42	B3_residue_4_bn	38	19	256							1024
Residual Stage-6	43	B3_residue_4_relu	38	19	256							0
Residual Stage-6	44	B3_residue_4_conv_14	38	19	256	256	3	3	256	1	1	590,080
Join-6	45	B3_2_add	38	19	256							0

(*table continues*)

Table C.1: ResNet-18 Details

ResNET-18	Layer		Input Size			Filter						Param
Stage	No.	Name	Hght	Wdth	Dpth	Cnt	Hght	Wdth	Dpth	Strd	Pad	Count
Join-6	46	B3_2_bn	38	19	256							1024
Join-6	47	B3_2_relu	38	19	256							0
Residual Stage-7	48	B4_residue_1_conv_15	38	19	256	512	3	3	256	2	0	1,180,160
Residual Stage-7	49	B4_residue_2_bn	19	10	512							2048
Residual Stage-7	50	B4_residue_2_relu	19	10	512							0
Residual Link-7	51	B4_original_reshaped	38	19	256	512	1	2	512	2	0	131,584
Residual Stage-7	52	B4_residue_2_conv_16	19	10	512	512	3	3	512	1	1	2,359,808
Join-7	53	B4_1_add	19	10	512							0
Join-7	54	B4_1_bn	19	10	512							2048
Join-7	55	B4_1_relu	19	10	512							0
Residual Stage-8	56	B4_residue_3_conv_17	19	10	512	512	3	3	512	1	1	2,359,808
Residual Stage-8	57	B4_residue_4_bn	19	10	512							2048
Residual Stage-8	58	B4_residue_4_relu	19	10	512							0
Residual Stage-8	59	B4_residue_4_conv_18	19	10	512	512	3	3	512	1	1	2,359,808
Join-8	60	B4_2_add	19	10	512							0

(*table continues*)

Table C.1: ResNet-18 Details

ResNET-18	Layer		Input Size			Filter						Param
Stage	No.	Name	Hght	Wdth	Dpth	Cnt	Hght	Wdth	Dpth	Strd	Pad	Count
Final Pooling	61	max_pooling2d	19	10	512	0	4	4	512	4	0	0
Final Pooling	62	average_pooling2d	19	10	512	0	4	4	512	4	0	0
Final Pooling	63	concatenate	4	2	1024							0
	64	linear	4	2	1,024	10	1	1	1024	10	0	10,250
	65	flatten	4	2	10							0
Decision	66	dense	80	1	1	2						162

APPENDIX D: XCEPTION NET MODEL DETAILS

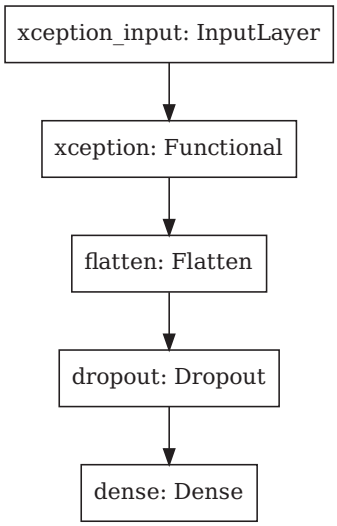


Figure D.1: Xcept CNN Model Decision Layers

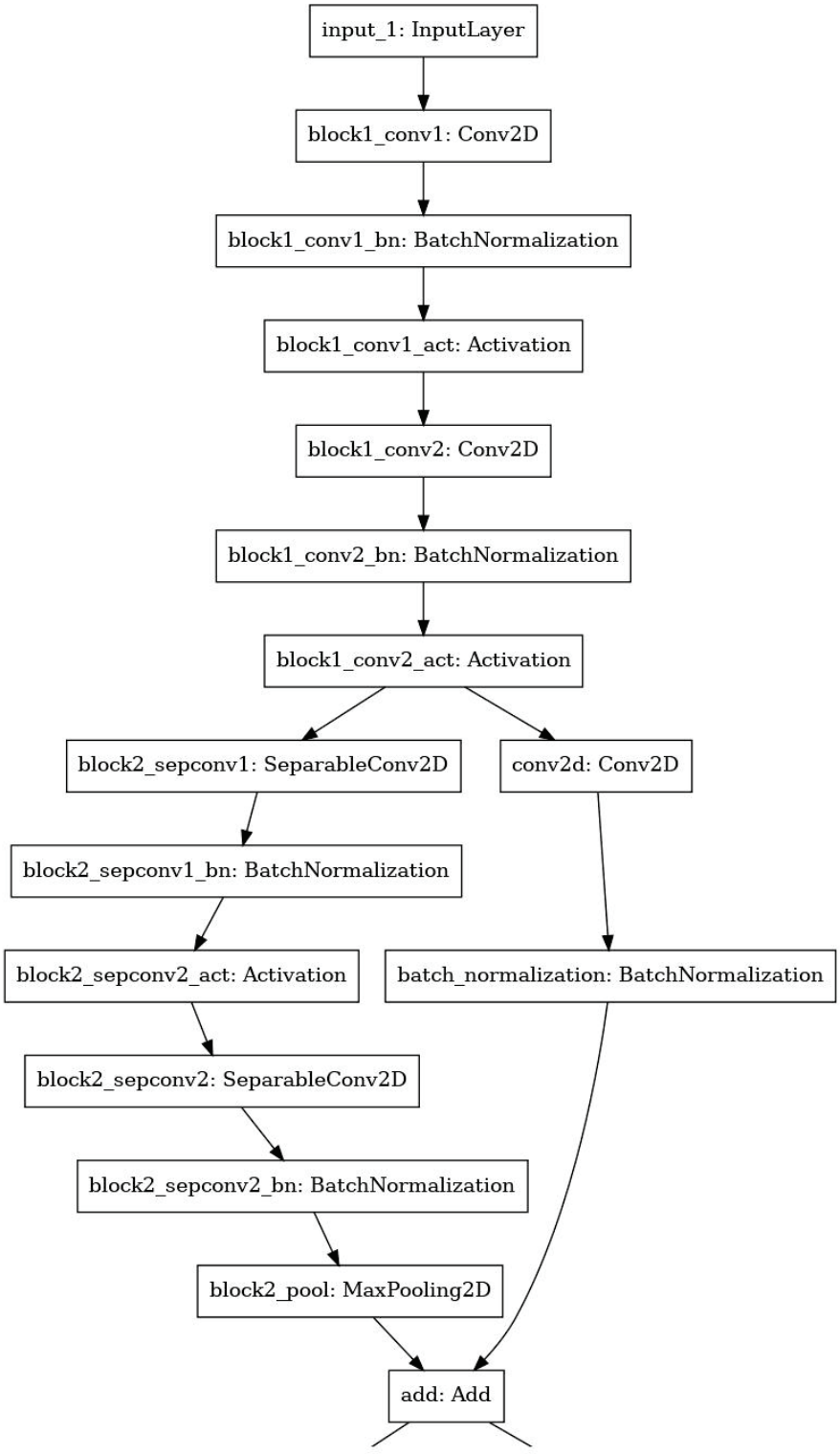


Figure D.2: Xcept CNN Model Functional Layers: Page 1 of 9

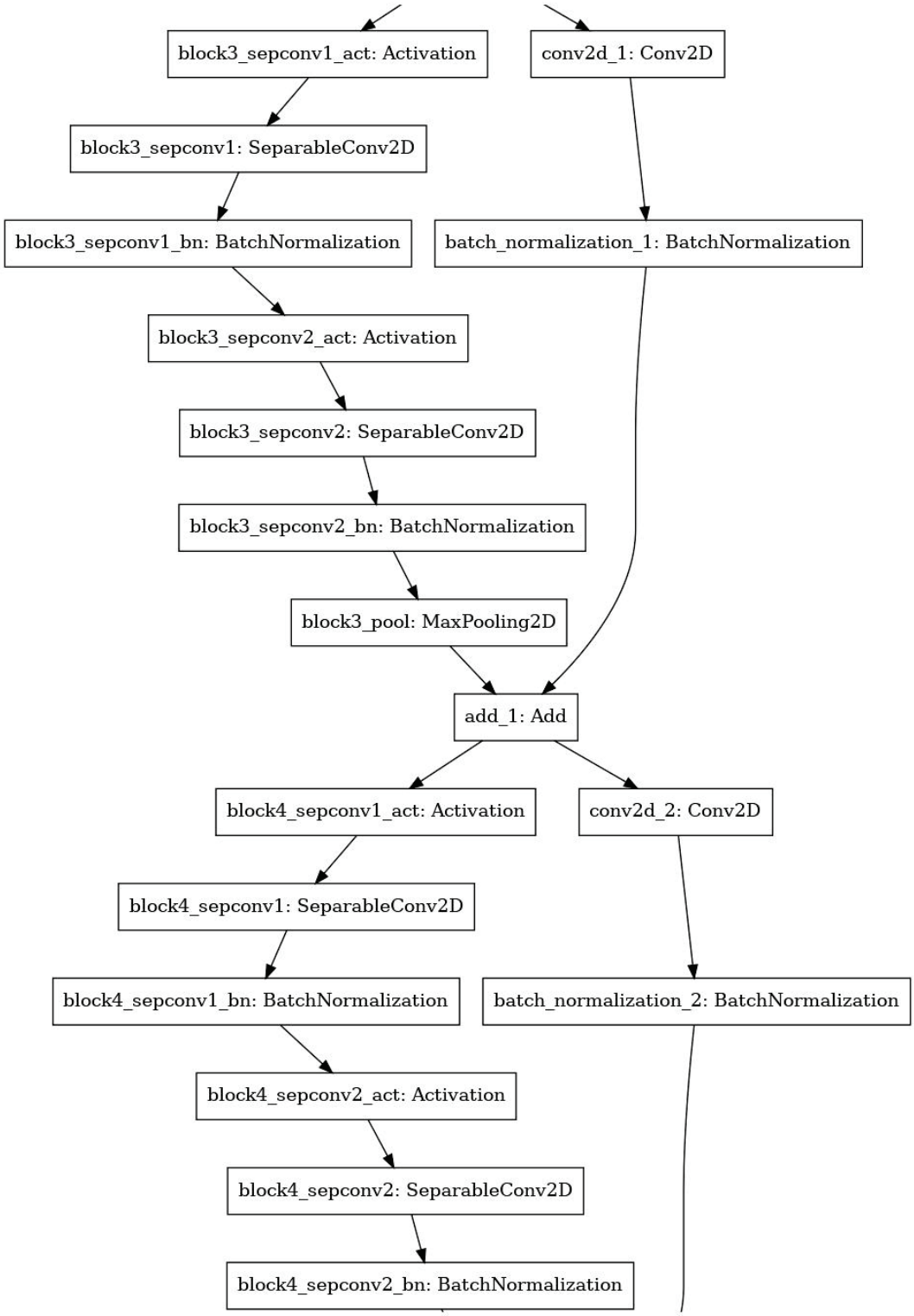


Figure D.3: Xcept CNN Model Functional Layers: Page 2 of 9

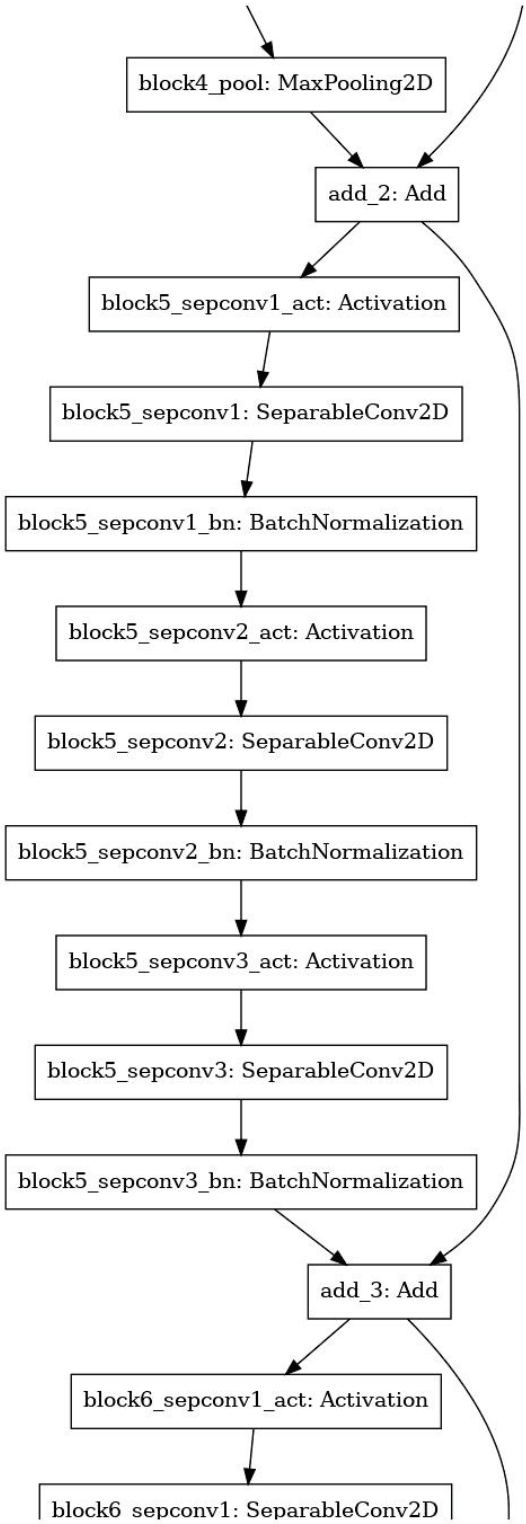


Figure D.4: Xcept CNN Model Functional Layers: Page 3 of 9

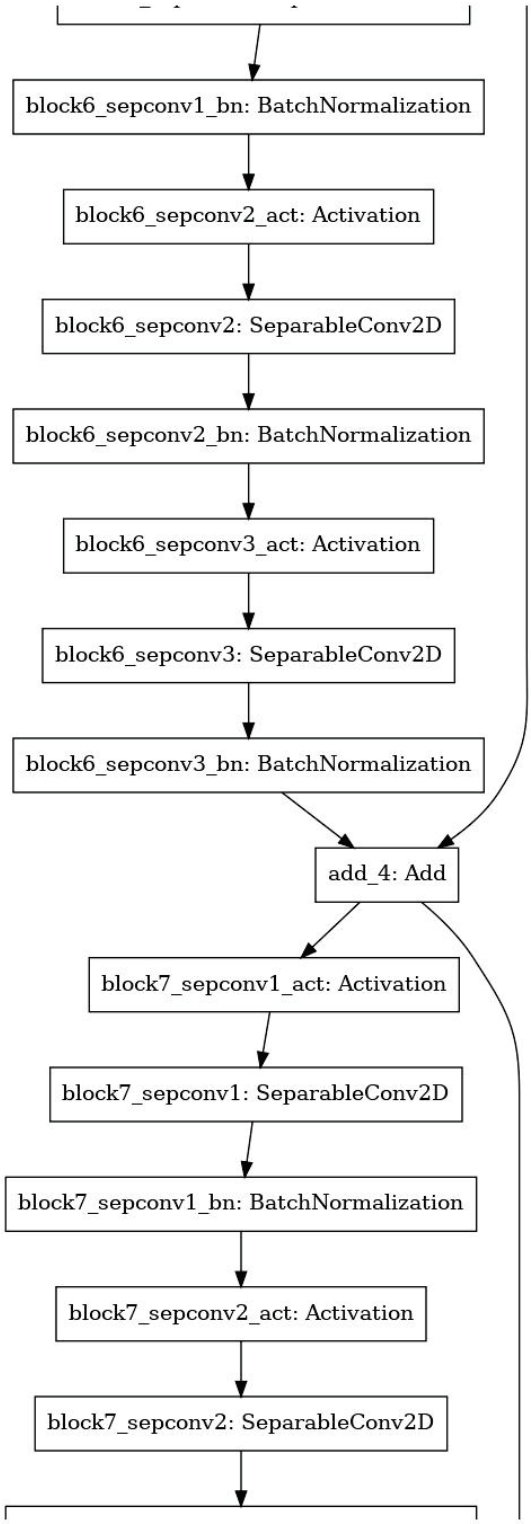


Figure D.5: Xcept CNN Model Functional Layers: Page 4 of 9

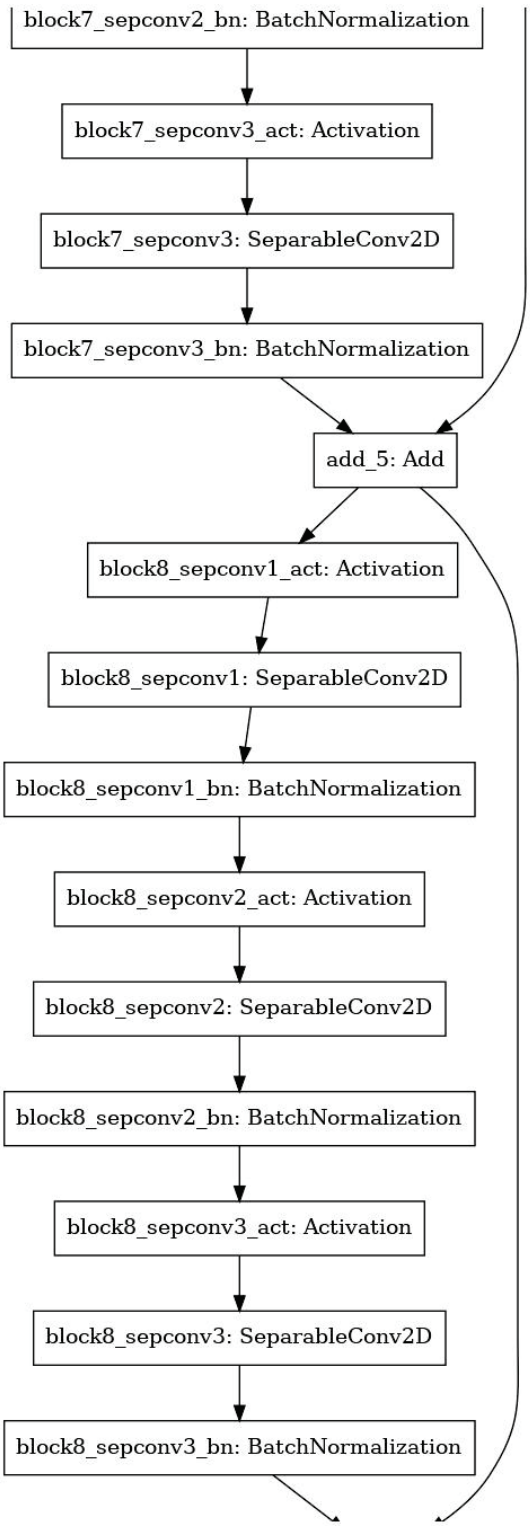


Figure D.6: Xcept CNN Model Functional Layers: Page 5 of 9

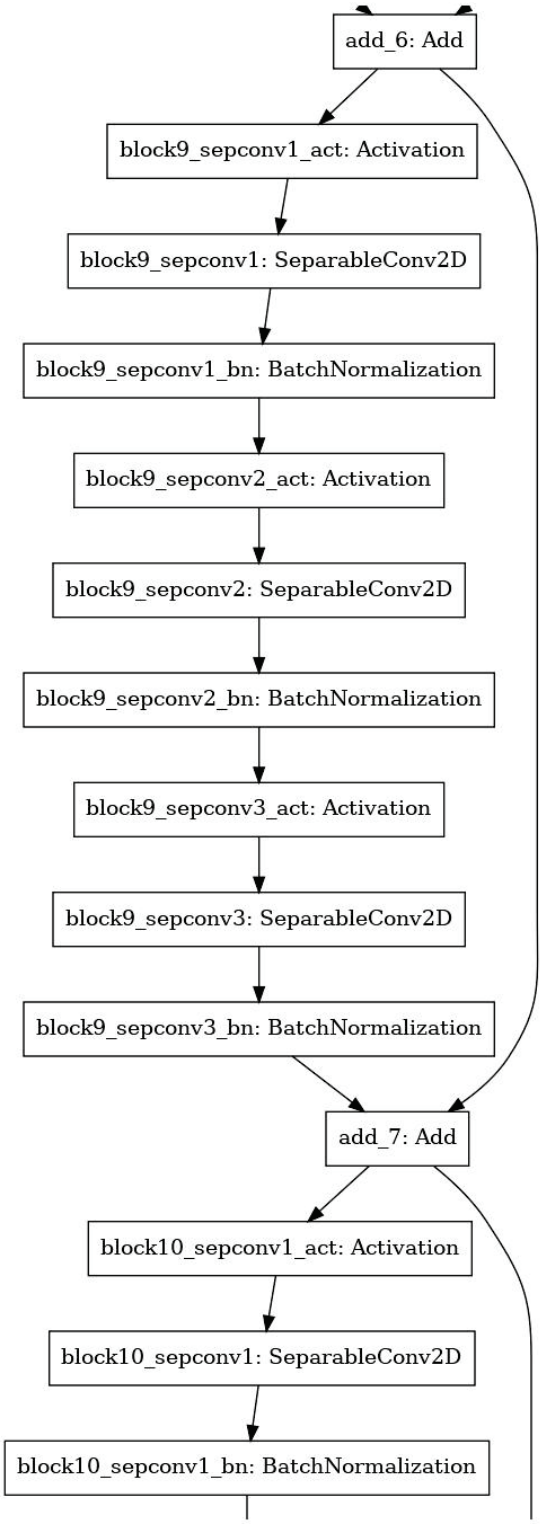


Figure D.7: Xcept CNN Model Functional Layers: Page 6 of 9

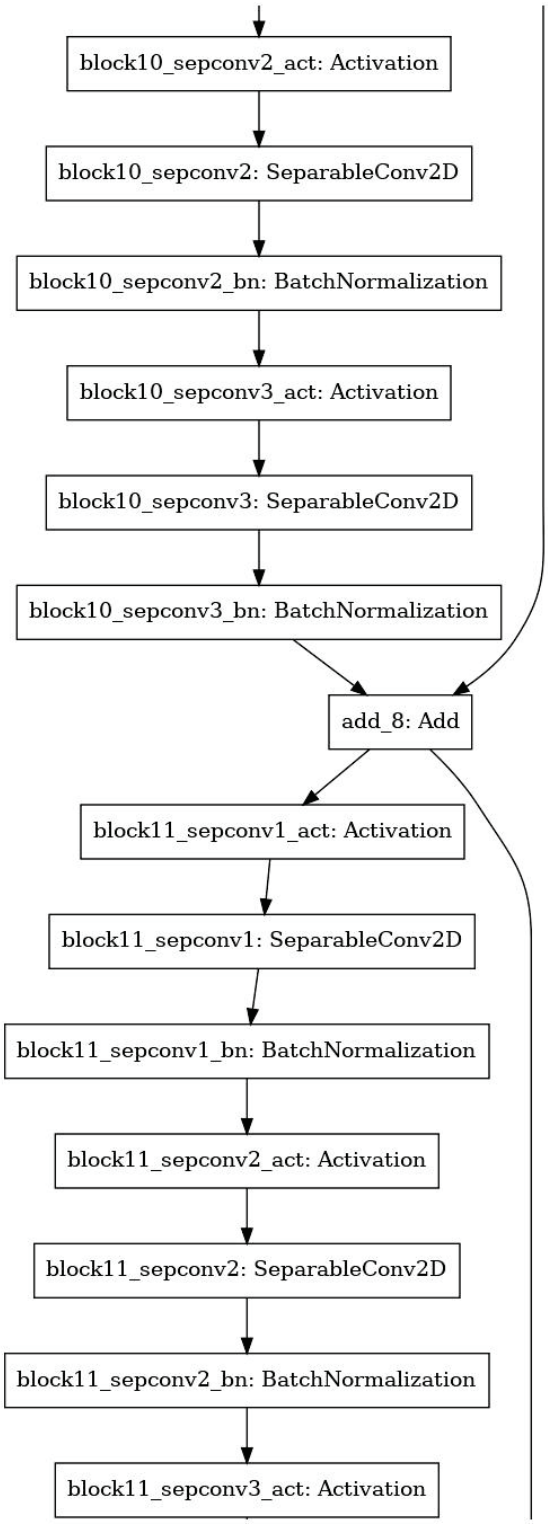


Figure D.8: Xcept CNN Model Functional Layers: Page 7 of 9

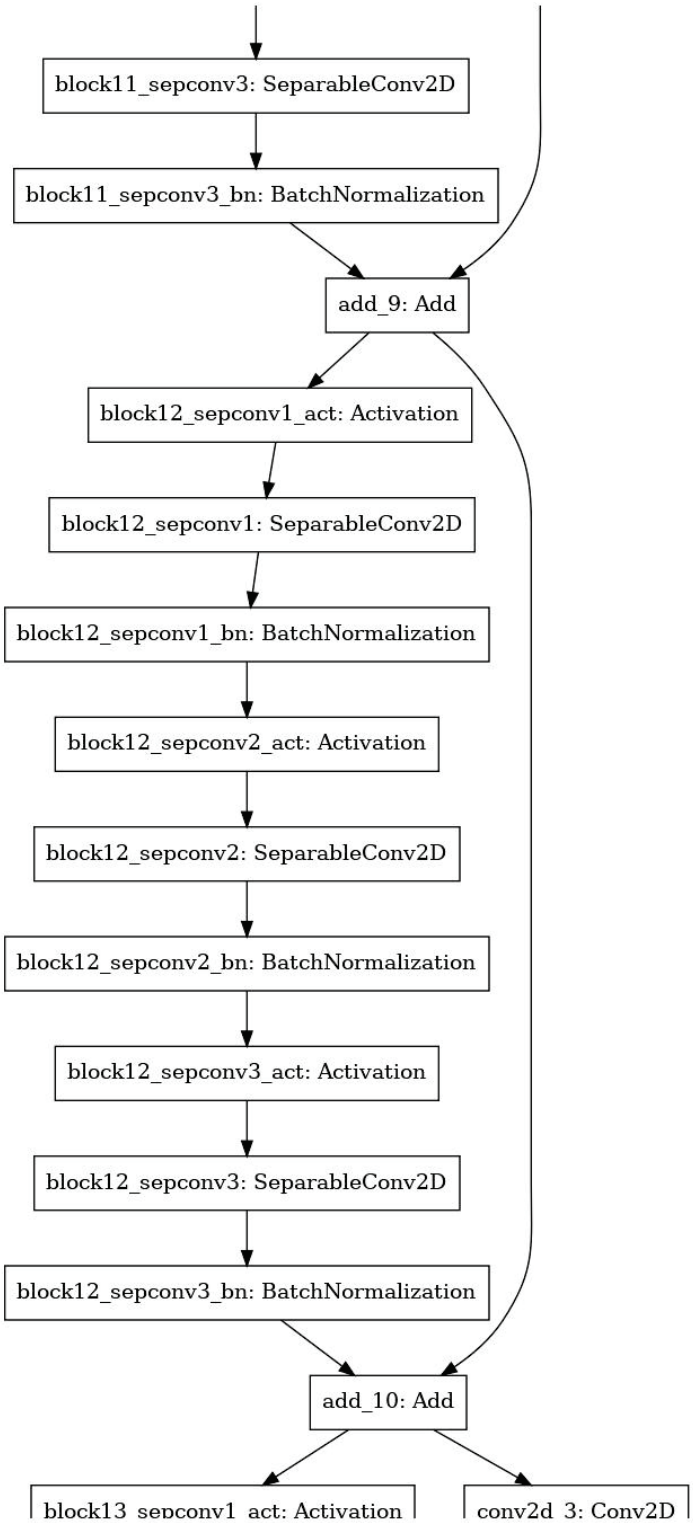


Figure D.9: Xcept CNN Model Functional Layers: Page 8 of 9

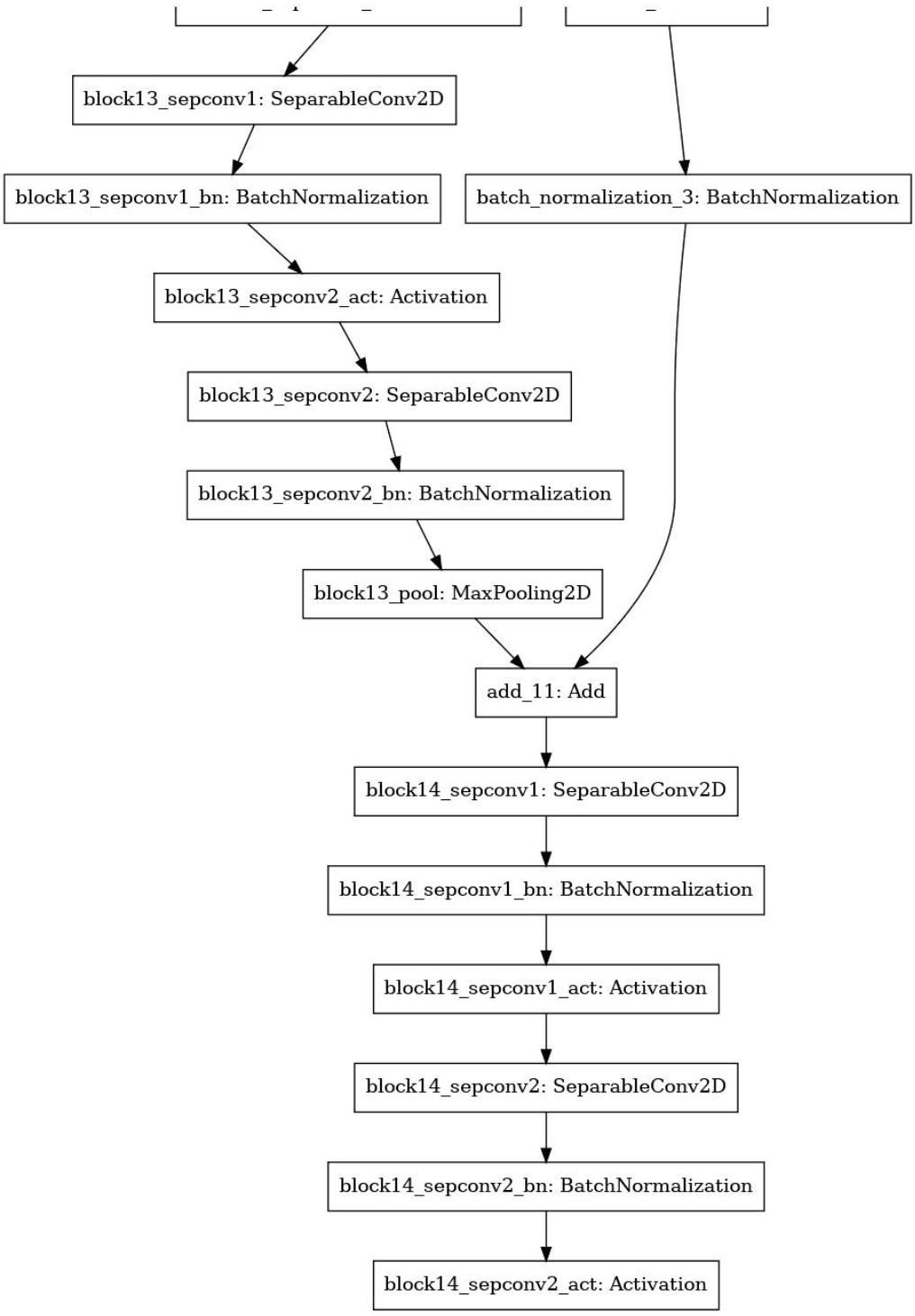


Figure D.10: Xcept CNN Model Functional Layers: Page 9 of 9

Table D.1: Xception Net Details

Xception	Layer		Input Size			Filter						Param
Stage	No.	Name	Hght	Wdth	Dpth	Cnt	Hght	Wdth	Dpth	Strd	Pad	Count
Prefetch Conv-1	1	block1_conv1	150	75	1	32	3	3	1	2	0	288
Prefetch Conv-1	2	block1_conv1_bn	74	37	32							128
Prefetch Conv-1	3	block1_conv1_act	74	37	32							
Prefetch Conv-1	4	block1_conv2	74	37	32	64	3	3	32	1	0	18,432
Prefetch Conv-1	5	block1_conv2_bn	72	35	64							256
Prefetch Conv-1	6	block1_conv2_act	72	35	64							
Depth Sep-2	7	block2_sepconv1	72	35	64	128	3	3	1	1	1	8,768
Depth Sep-2	8	block2_sepconv1_bn	72	35	128							512
Depth Sep-2	9	block2_sepconv2_act	72	35	128							
Depth Sep-2	10	block2_sepconv2	72	35	128	128	3	3	1	1	1	17,536
Depth Sep-2	11	block2_sepconv2_bn	72	35	128							512
Width Bypass-2	12	conv2d	72	35	64	128	1	1	128	2	0	8,192
Width Bypass-2	13	block2_pool	72	35	128		3	3	1	2	0	
Width Bypass-2	14	batch_normalization	36	18	128							512
	15	add	36	18	128							

(*table continues*)

Table D.1: Xception Net Details

Xception	Layer		Input Size			Filter						Param
Stage	No.	Name	Hght	Wdth	Dpth	Cnt	Hght	Wdth	Dpth	Strd	Pad	Count
Depth Sep-3	16	block3_sepconv1_act	36	18	128							
Depth Sep-3	17	block3_sepconv1	36	18	128	256	3	3	1	1	1	33,920
Depth Sep-3	18	block3_sepconv1_bn	36	18	256							1,024
Depth Sep-3	19	block3_sepconv2_act	36	18	256							
Depth Sep-3	20	block3_sepconv2	36	18	256	256	3	3	1	1	1	67,840
Width Bypass-3	21	block3_sepconv2_bn	36	18	256							1,024
Width Bypass-3	22	conv2d_1	36	18	256	256	1	1	256	2	0	32,768
Width Bypass-3	23	block3_pool	36	18	256		3	3	1	2	0	
	24	batch_normalization_1	18	9	256							1,024
	25	add_1	18	9	256							
Depth Sep-4	26	block4_sepconv1_act	18	9	256							
Depth Sep-4	27	block4_sepconv1	18	9	256	728	3	3	1	1	1	188,672
Depth Sep-4	28	block4_sepconv1_bn	18	9	728							2,912
Depth Sep-4	29	block4_sepconv2_act	18	9	728							
Depth Sep-4	30	block4_sepconv2	18	9	728	728	3	3	1	1	1	536,536

(*table continues*)

Table D.1: Xception Net Details

Xception	Layer		Input Size			Filter						Param
Stage	No.	Name	Hght	Wdth	Dpth	Cnt	Hght	Wdth	Dpth	Strd	Pad	Count
Width Bypass-4	31	block4_sepconv2_bn	18	9	728							2,912
Width Bypass-4	32	conv2d_2	18	9	256	728	1	1	256	2	0	186,368
Width Bypass-4	33	block4_pool	18	9	728		3	3	1	2	0	
	34	batch_normalization_2	9	5	728							2,912
	35	add_2	9	5	728							
Depth Sep-5	36	block5_sepconv1_act	9	5	728							
Depth Sep-5	37	block5_sepconv1	9	5	728	728	3	3	1	1	1	536,536
Depth Sep-5	38	block5_sepconv1_bn	9	5	728							2,912
Depth Sep-5	39	block5_sepconv2_act	9	5	728							
Depth Sep-5	40	block5_sepconv2	9	5	728	728	3	3	1	1	1	536,536
Depth Sep-5	41	block5_sepconv2_bn	9	5	728							2,912
Depth Sep-5	42	block5_sepconv3_act	9	5	728							
Depth Sep-5	43	block5_sepconv3	9	5	728	728	1	1	728	1	0	536,536
Depth Sep-5	44	block5_sepconv3_bn	9	5	728							2,912
	45	add_3	9	5	728							

(*table continues*)

Table D.1: Xception Net Details

Xception	Layer		Input Size			Filter						Param
Stage	No.	Name	Hght	Wdth	Dpth	Cnt	Hght	Wdth	Dpth	Strd	Pad	Count
Depth Sep-6	46	block6_sepconv1_act	9	5	728							
Depth Sep-6	47	block6_sepconv1	9	5	728	728	3	3	1	1	1	536,536
Depth Sep-6	48	block6_sepconv1_bn	9	5	728							2,912
Depth Sep-6	49	block6_sepconv2_act	9	5	728							
Depth Sep-6	50	block6_sepconv2	9	5	728	728	3	3	1	1	1	536,536
Depth Sep-6	51	block6_sepconv2_bn	9	5	728							2,912
Depth Sep-6	52	block6_sepconv3_act	9	5	728							
Depth Sep-6	53	block6_sepconv3	9	5	728	728	1	1	728	1	0	536,536
Depth Sep-6	54	block6_sepconv3_bn	9	5	728							2,912
	55	add_4	9	5	728							
Depth Sep-7	56	block7_sepconv1_act	9	5	728							
Depth Sep-7	57	block7_sepconv1	9	5	728	728	3	3	1	1	1	536,536
Depth Sep-7	58	block7_sepconv1_bn	9	5	728							2,912
Depth Sep-7	59	block7_sepconv2_act	9	5	728							
Depth Sep-7	60	block7_sepconv2	9	5	728	728	3	3	1	1	1	536,536

(*table continues*)

Table D.1: Xception Net Details

Xception	Layer		Input Size			Filter						Param
Stage	No.	Name	Hght	Wdth	Dpth	Cnt	Hght	Wdth	Dpth	Strd	Pad	Count
Depth Sep-7	61	block7_sepconv2_bn	9	5	728							2,912
Depth Sep-7	62	block7_sepconv3_act	9	5	728							
Depth Sep-7	63	block7_sepconv3	9	5	728	728	1	1	728	1	0	536,536
Depth Sep-7	64	block7_sepconv3_bn	9	5	728							2,912
	65	add_5	9	5	728							
Depth Sep-8	66	block8_sepconv1_act	9	5	728							
Depth Sep-8	67	block8_sepconv1	9	5	728	728	3	3	1	1	1	536,536
Depth Sep-8	68	block8_sepconv1_bn	9	5	728							2,912
Depth Sep-8	69	block8_sepconv2_act	9	5	728							
Depth Sep-8	70	block8_sepconv2	9	5	728	728	3	3	1	1	1	536,536
Depth Sep-8	71	block8_sepconv2_bn	9	5	728							2,912
Depth Sep-8	72	block8_sepconv3_act	9	5	728							
Depth Sep-8	73	block8_sepconv3	9	5	728	728	1	1	728	1	0	536,536
Depth Sep-8	74	block8_sepconv3_bn	9	5	728							2,912
	75	add_6	9	5	728							

(*table continues*)

Table D.1: Xception Net Details

Xception	Layer		Input Size			Filter						Param
Stage	No.	Name	Hght	Wdth	Dpth	Cnt	Hght	Wdth	Dpth	Strd	Pad	Count
Depth Sep-9	76	block9_sepconv1_act	9	5	728							
Depth Sep-9	77	block9_sepconv1	9	5	728	728	3	3	1	1	1	536,536
Depth Sep-9	78	block9_sepconv1_bn	9	5	728							2,912
Depth Sep-9	79	block9_sepconv2_act	9	5	728							
Depth Sep-9	80	block9_sepconv2	9	5	728	728	3	3	1	1	1	536,536
Depth Sep-9	81	block9_sepconv2_bn	9	5	728							2,912
Depth Sep-9	82	block9_sepconv3_act	9	5	728							
Depth Sep-9	83	block9_sepconv3	9	5	728	728	1	1	728	1	0	536,536
Depth Sep-9	84	block9_sepconv3_bn	9	5	728							2,912
	85	add_7	9	5	728							
Depth Sep-10	86	block10_sepconv1_act	9	5	728							
Depth Sep-10	87	block10_sepconv1	9	5	728	728	3	3	1	1	1	536,536
Depth Sep-10	88	block10_sepconv1_bn	9	5	728							2,912
Depth Sep-10	89	block10_sepconv2_act	9	5	728							
Depth Sep-10	90	block10_sepconv2	9	5	728	728	3	3	1	1	1	536,536

(*table continues*)

Table D.1: Xception Net Details

Xception	Layer		Input Size			Filter						Param
Stage	No.	Name	Hght	Wdth	Dpth	Cnt	Hght	Wdth	Dpth	Strd	Pad	Count
Depth Sep-10	91	block10_sepconv2_bn	9	5	728							2,912
Depth Sep-10	92	block10_sepconv3_act	9	5	728							
Depth Sep-10	93	block10_sepconv3	9	5	728	728	1	1	728	1	0	536,536
Depth Sep-10	94	block10_sepconv3_bn	9	5	728							2,912
	95	add_8	9	5	728							
Depth Sep-11	96	block11_sepconv1_act	9	5	728							
Depth Sep-11	97	block11_sepconv1	9	5	728	728	3	3	1	1	1	536,536
Depth Sep-11	98	block11_sepconv1_bn	9	5	728							2,912
Depth Sep-11	99	block11_sepconv2_act	9	5	728							
Depth Sep-11	100	block11_sepconv2	9	5	728	728	3	3	1	1	1	536,536
Depth Sep-11	101	block11_sepconv2_bn	9	5	728							2,912
Depth Sep-11	102	block11_sepconv3_act	9	5	728							
Depth Sep-11	103	block11_sepconv3	9	5	728	728	1	1	728	1	0	536,536
Depth Sep-11	104	block11_sepconv3_bn	9	5	728							2,912
	105	add_9	9	5	728							

(*table continues*)

Table D.1: Xception Net Details

Xception	Layer		Input Size			Filter						Param
Stage	No.	Name	Hght	Wdth	Dpth	Cnt	Hght	Wdth	Dpth	Strd	Pad	Count
Depth Sep-12	106	block12_sepconv1_act	9	5	728							
Depth Sep-12	107	block12_sepconv1	9	5	728	728	3	3	1	1	1	536,536
Depth Sep-12	108	block12_sepconv1_bn	9	5	728							2,912
Depth Sep-12	109	block12_sepconv2_act	9	5	728							
Depth Sep-12	110	block12_sepconv2	9	5	728	728	3	3	1	1	1	536,536
Depth Sep-12	111	block12_sepconv2_bn	9	5	728							2,912
Depth Sep-12	112	block12_sepconv3_act	9	5	728							
Depth Sep-12	113	block12_sepconv3	9	5	728	728	1	1	728	1	0	536,536
Depth Sep-12	114	block12_sepconv3_bn	9	5	728							2,912
	115	add_10	9	5	728							
Depth Sep-13	116	block13_sepconv1_act	9	5	728							
Depth Sep-13	117	block13_sepconv1	9	5	728	728	3	3	1	1	1	536,536
Depth Sep-13	118	block13_sepconv1_bn	9	5	728							2,912
Depth Sep-13	119	block13_sepconv2_act	9	5	728							
Depth Sep-13	120	block13_sepconv2	9	5	728	1,024	3	3	1	1	1	752,024

(*table continues*)

Table D.1: Xception Net Details

Xception	Layer		Input Size			Filter						Param
Stage	No.	Name	Hght	Wdth	Dpth	Cnt	Hght	Wdth	Dpth	Strd	Pad	Count
Depth Sep-13	121	block13_sepconv2_bn	9	5	1,024							4,096
Width Bypass-13	122	conv2d_3	9	5	728	1,024	1	1	728	2	1	745,472
Width Bypass-13	123	block13_pool	9	5	728		3	3	1	2	1	
Width Bypass-13	124	batch_normalization_3	5	3	1,024							4,096
	125	add_11	5	3	1,024							
Depth Sep-14	126	block14_sepconv1	5	3	1,024	1,536	3	3	1	1	1	1,582,080
Depth Sep-14	127	block14_sepconv1_bn	5	3	1,536							6,144
Depth Sep-14	128	block14_sepconv1_act	5	3	1,536							
Depth Sep-14	129	block14_sepconv2	5	3	1,536	2,048	3	3	1	1	1	3,159,552
Depth Sep-14	130	block14_sepconv2_bn	5	3	2,048							8,192
Depth Sep-14	131	block14_sepconv2_act	5	3	2,048							
	132	flatten	5	3	2,048							
	133	dropout	30,720	1	1							
Decision	134	dense	30,720	1	1							61,442

APPENDIX E: MOBILENET MODEL DETAILS

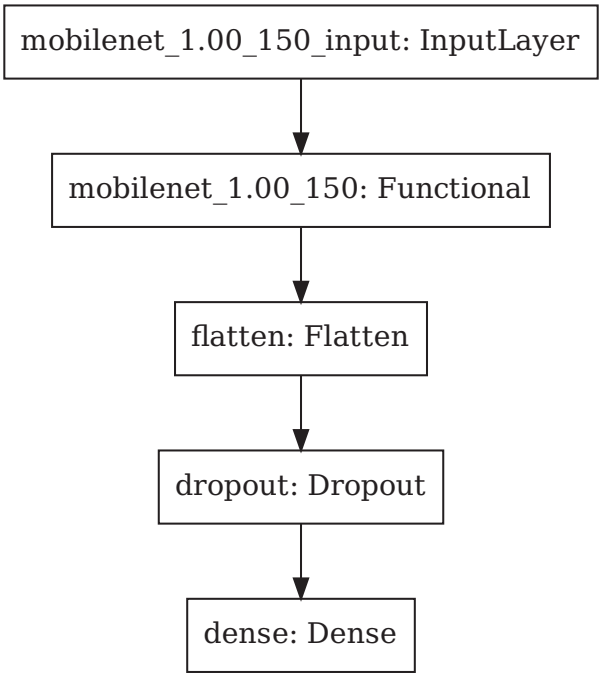


Figure E.1: MobileNet CNN Model Decision Layers

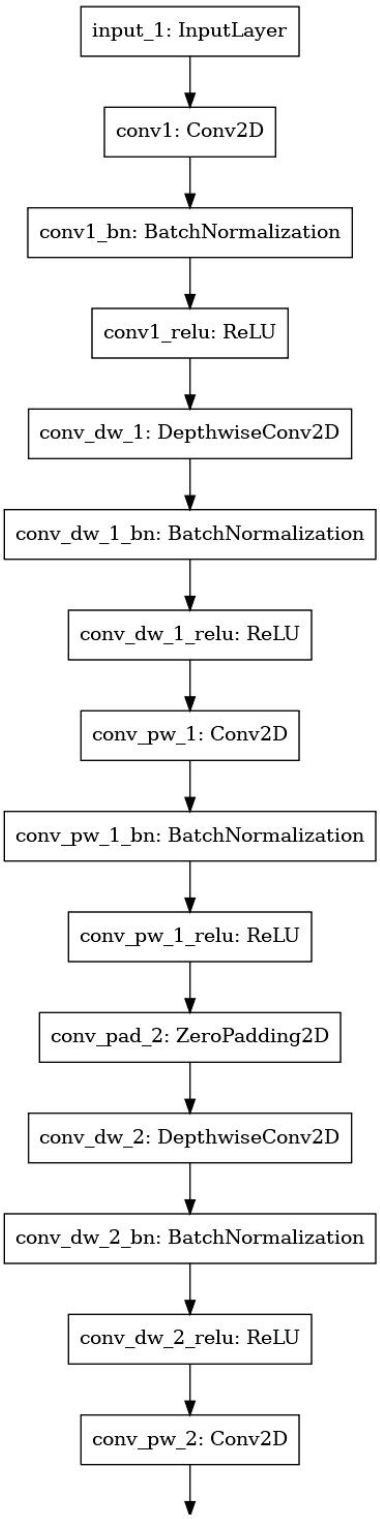


Figure E.2: MobileNet CNN Model Functional Layers: Page 1 of 6

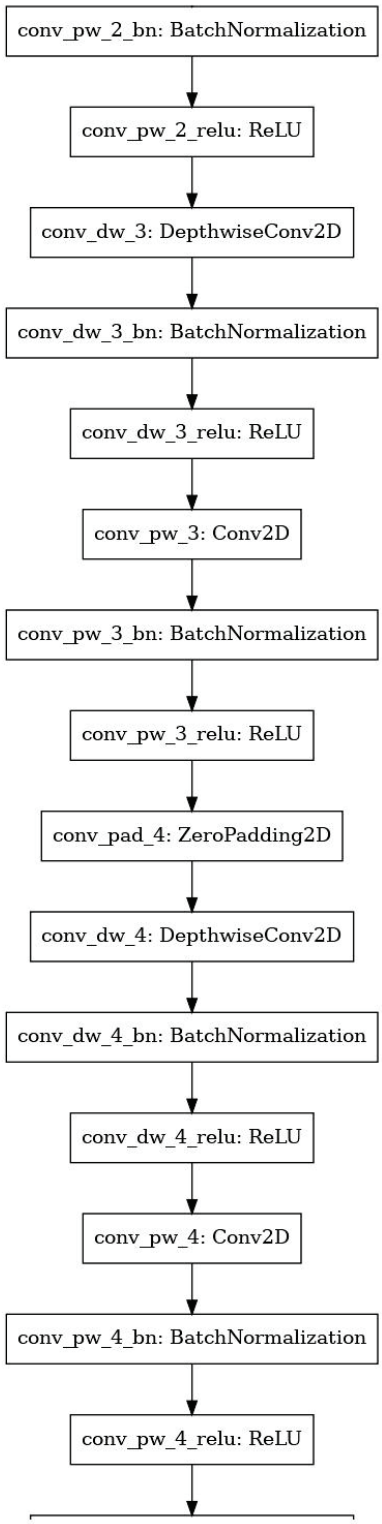


Figure E.3: MobileNet CNN Model Functional Layers: Page 2 of 6

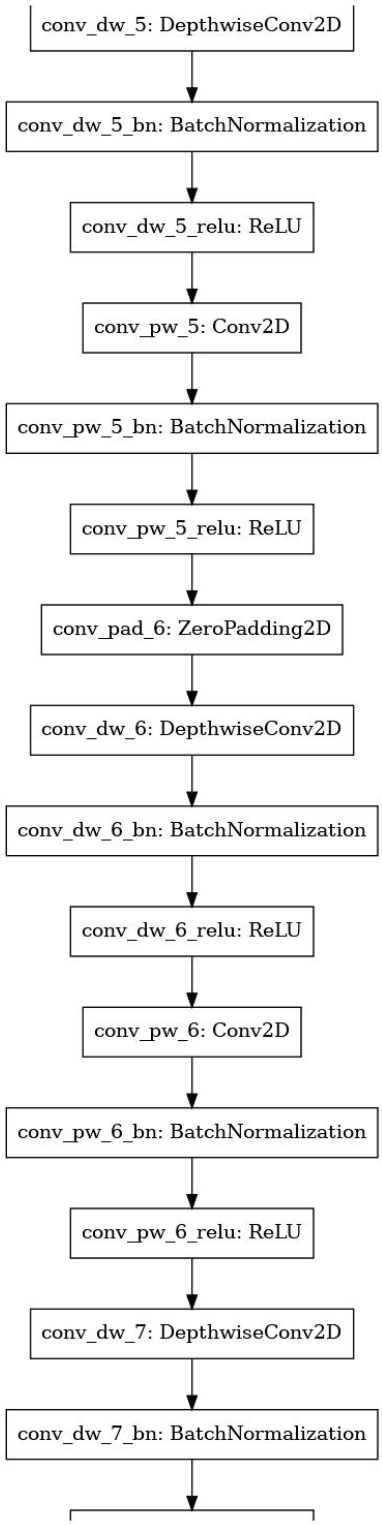


Figure E.4: MobileNet CNN Model Functional Layers: Page 3 of 6

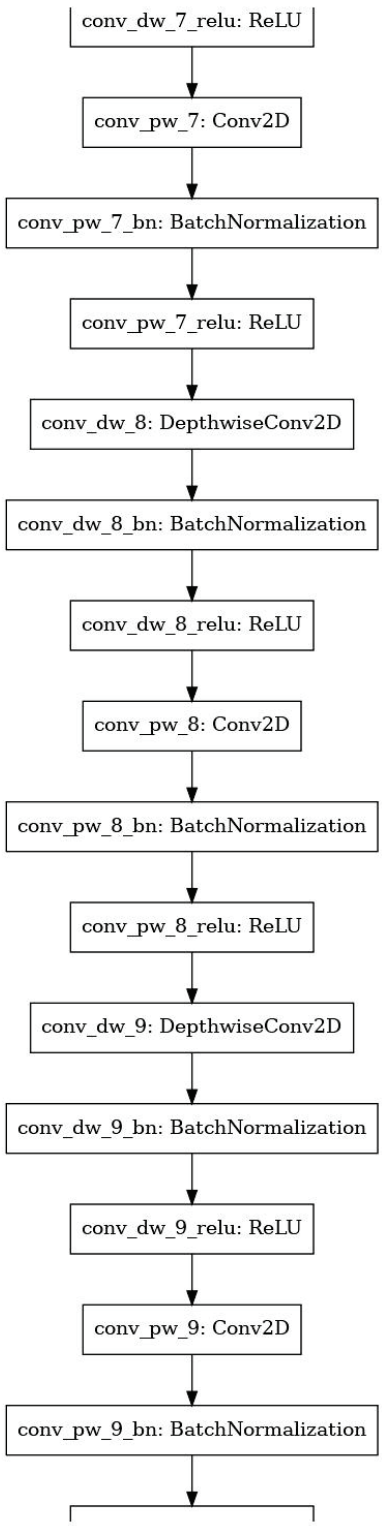


Figure E.5: MobileNet CNN Model Functional Layers: Page 4 of 6

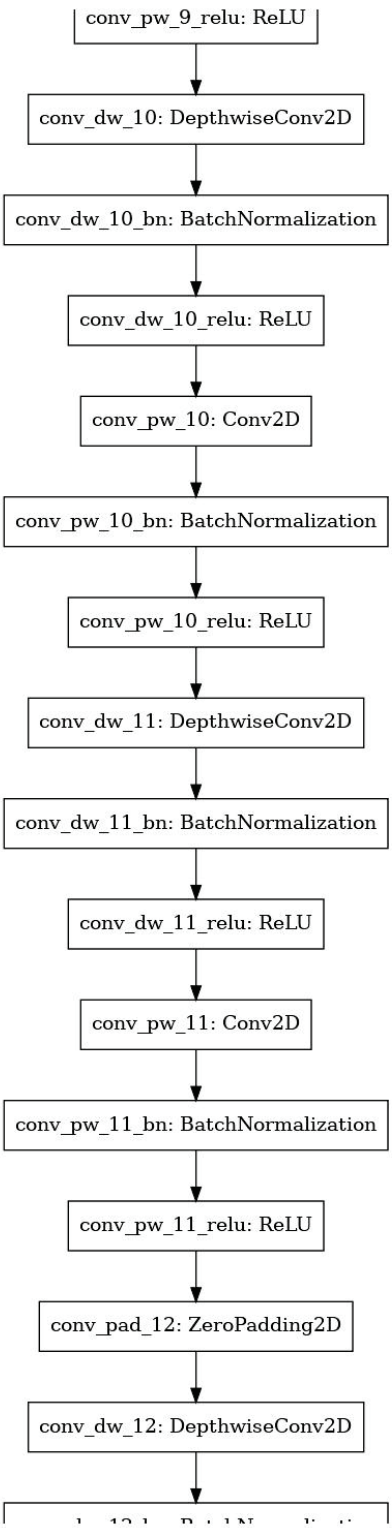


Figure E.6: MobileNet CNN Model Functional Layers: Page 5 of 7

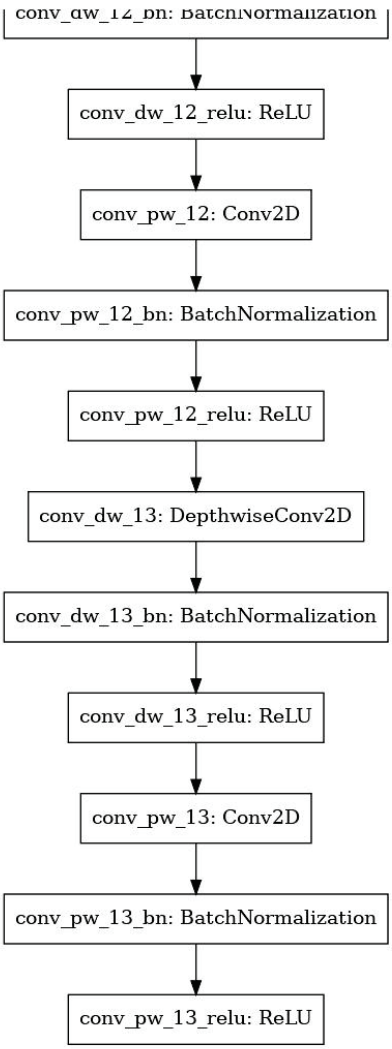


Figure E.7: MobileNet CNN Model Functional Layers: Page 6 of 6

Table E.1: MobileNet Details

MobileNet	Layer		Input Size			Filter						Param
Stage	No.	Name	Hght	Wdth	Dpth	Cnt	Hght	Wdth	Dpth	Strd	Pad	Count
Prefetch	1	conv1	150	75	1	32	3	3	3	2	1	288
Prefetch	2	conv1_bn	75	38	32							128
Prefetch	3	conv1_relu	75	38	32							
Depth Wise-1	4	conv_dw_1	75	38	32	32	3	3	1	1	1	288
Depth Wise-1	5	conv_dw_1_bn	75	38	32							128
Depth Wise-1	6	conv_dw_1_relu	75	38	32							
Point Wise-1	7	conv_pw_1	75	38	32	64	1	1	32	1	1	2,048
Point Wise-1	8	conv_pw_1_bn	75	38	64							256
Point Wise-1	9	conv_pw_1_relu	75	38	64							
	10	conv_pad_2	75	38	64	1	1	1	0	1	0	
Depth Wise-2	11	conv_dw_2	76	39	64	64	3	3	1	2	0	576
Depth Wise-2	12	conv_dw_2_bn	37	19	64							256
Depth Wise-2	13	conv_dw_2_relu	37	19	64							
Point Wise-2	14	conv_pw_2	37	19	64	128	1	1	64	1	0	8,192
Point Wise-2	15	conv_pw_2_bn	37	19	128							512

(*table continues*)

Table E.1: MobileNet Details

MobileNet	Layer		Input Size			Filter						Param
Stage	No.	Name	Hght	Wdth	Dpth	Cnt	Hght	Wdth	Dpth	Strd	Pad	Count
Point Wise-2	16	conv_pw_2_relu	37	19	128							
Depth Wise-3	17	conv_dw_3	37	19	128	128	3	3	1	1	1	1,152
Depth Wise-3	18	conv_dw_3_bn	37	19	128							512
Depth Wise-3	19	conv_dw_3_relu	37	19	128							
Point Wise-3	20	conv_pw_3	37	19	128	128	1	1	128	1	0	16,384
Point Wise-3	21	conv_pw_3_bn	37	19	128							512
Point Wise-3	22	conv_pw_3_relu	37	19	128							
	23	conv_pad_4	37	19	128	1	1	1	1	1	1	
Depth Wise-4	24	conv_dw_4	38	20	128	128	3	3	1	2	0	1,152
Depth Wise-4	25	conv_dw_4_bn	18	9	128							512
Depth Wise-4	26	conv_dw_4_relu	18	9	128							
Point Wise-4	27	conv_pw_4	18	9	128	256	1	1	128	1	0	32,768
Point Wise-4	28	conv_pw_4_bn	18	9	256							1,024
Point Wise-4	29	conv_pw_4_relu	18	9	256							
Depth Wise-5	30	conv_dw_5	18	9	256	256	3	3	1	1	1	2,304

(*table continues*)

Table E.1: MobileNet Details

MobileNet	Layer		Input Size			Filter						Param
Stage	No.	Name	Hght	Wdth	Dpth	Cnt	Hght	Wdth	Dpth	Strd	Pad	Count
Depth Wise-5	31	conv_dw_5_bn	18	9	256							1,024
Depth Wise-5	32	conv_dw_5_relu	18	9	256							
Point Wise-5	33	conv_pw_5	18	9	256	256	1	1	256	1	0	65,536
Point Wise-5	34	conv_pw_5_bn	18	9	256							1,024
Point Wise-5	35	conv_pw_5_relu	18	9	256							
	36	conv_pad_6	18	9	256	1	1	1	1	1	1	
Depth Wise-6	37	conv_dw_6	19	10	256	256	3	3	1	2	0	2,304
Depth Wise-6	38	conv_dw_6_bn	9	4	256							1,024
Depth Wise-6	39	conv_dw_6_relu	9	4	256							
Point Wise-6	40	conv_pw_6	9	4	256	512	1	1	256	1	0	131,072
Point Wise-6	41	conv_pw_6_bn	9	4	512							2,048
Point Wise-6	42	conv_pw_6_relu	9	4	512							
Depth Wise-7	43	conv_dw_7	9	4	512	512	3	3	1	1	1	4,608
Depth Wise-7	44	conv_dw_7_bn	9	4	512							2,048
Depth Wise-7	45	conv_dw_7_relu	9	4	512							

(*table continues*)

Table E.1: MobileNet Details

MobileNet	Layer		Input Size			Filter						Param
Stage	No.	Name	Hght	Wdth	Dpth	Cnt	Hght	Wdth	Dpth	Strd	Pad	Count
Point Wise-7	46	conv_pw_7	9	4	512	512	1	1	512	1	0	262,144
Point Wise-7	47	conv_pw_7_bn	9	4	512							2,048
Point Wise-7	48	conv_pw_7_relu	9	4	512							
Depth Wise-8	49	conv_dw_8	9	4	512	512	3	3	1	1	1	4,608
Depth Wise-8	50	conv_dw_8_bn	9	4	512							2,048
Depth Wise-8	51	conv_dw_8_relu	9	4	512							
Point Wise-8	52	conv_pw_8	9	4	512	512	1	1	512	1	0	262,144
Point Wise-8	53	conv_pw_8_bn	9	4	512							2,048
Point Wise-8	54	conv_pw_8_relu	9	4	512							
Depth Wise-9	55	conv_dw_9	9	4	512	512	3	3	1	1	1	4,608
Depth Wise-9	56	conv_dw_9_bn	9	4	512							2,048
Depth Wise-9	57	conv_dw_9_relu	9	4	512							
Point Wise-9	58	conv_pw_9	9	4	512	512	1	1	512	1	0	262,144
Point Wise-9	59	conv_pw_9_bn	9	4	512							2,048
Point Wise-9	60	conv_pw_9_relu	9	4	512							

(*table continues*)

Table E.1: MobileNet Details

MobileNet	Layer		Input Size			Filter						Param
Stage	No.	Name	Hght	Wdth	Dpth	Cnt	Hght	Wdth	Dpth	Strd	Pad	Count
Depth Wise-10	61	conv_dw_10	9	4	512	512	3	3	1	1	1	4,608
Depth Wise-10	62	conv_dw_10_bn	9	4	512							2,048
Depth Wise-10	63	conv_dw_10_relu	9	4	512							
Point Wise-10	64	conv_pw_10	9	4	512	512	1	1	512	1	0	262,144
Point Wise-10	65	conv_pw_10_bn	9	4	512							2,048
Point Wise-10	66	conv_pw_10_relu	9	4	512							
Depth Wise-11	67	conv_dw_11	9	4	512	512	3	3	1	1	1	4,608
Depth Wise-11	68	conv_dw_11_bn	9	4	512							2,048
Depth Wise-11	69	conv_dw_11_relu	9	4	512							
Point Wise-11	70	conv_pw_11	9	4	512	512	1	1	512	1	0	262,144
Point Wise-11	71	conv_pw_11_bn	9	4	512							2,048
Point Wise-11	72	conv_pw_11_relu	9	4	512							
	73	conv_pad_12	9	4	512	1	1	1	1	1	1	
Depth Wise-12	74	conv_dw_12	10	5	512	512	3	3	1	2	0	4,608
Depth Wise-12	75	conv_dw_12_bn	4	2	512							2,048

(*table continues*)

Table E.1: MobileNet Details

MobileNet	Layer		Input Size			Filter						Param
Stage	No.	Name	Hght	Wdth	Dpth	Cnt	Hght	Wdth	Dpth	Strd	Pad	Count
Depth Wise-12	76	conv_dw_12_relu	4	2	512							
Point Wise-12	77	conv_pw_12	4	2	512	1,024	1	1	512	1	0	524,288
Point Wise-12	78	conv_pw_12_bn	4	2	1,024							4,096
Point Wise-12	79	conv_pw_12_relu	4	2	1,024							
Depth Wise-13	80	conv_dw_13	4	2	1,024	1,024	3	3	1	1	1	9,216
Depth Wise-13	81	conv_dw_13_bn	4	2	1,024							4,096
Depth Wise-13	82	conv_dw_13_relu	4	2	1,024							
Point Wise-13	83	conv_pw_13	4	2	1,024	1,024	1	1	512	1	0	1,048,576
Point Wise-13	83	conv_pw_13_bn	4	2	1,024							4,096
Point Wise-13	83	conv_pw_13_relu	4	2	1,024							
	83	flatten	8,192	1	1							
Decision	83	dropout	8,192	1	1							
Decision	83	dense	8,192	1	1	2						16,386

APPENDIX F: DENSENET-121 MODEL DETAILS

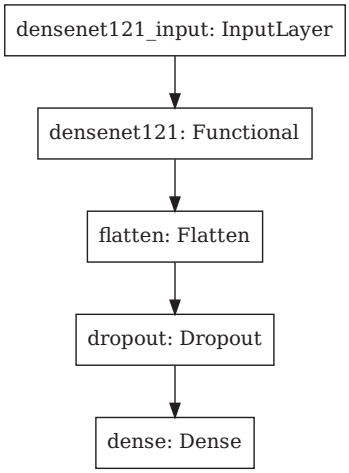


Figure F.1: DenseNet-121 CNN Model Decision Layers

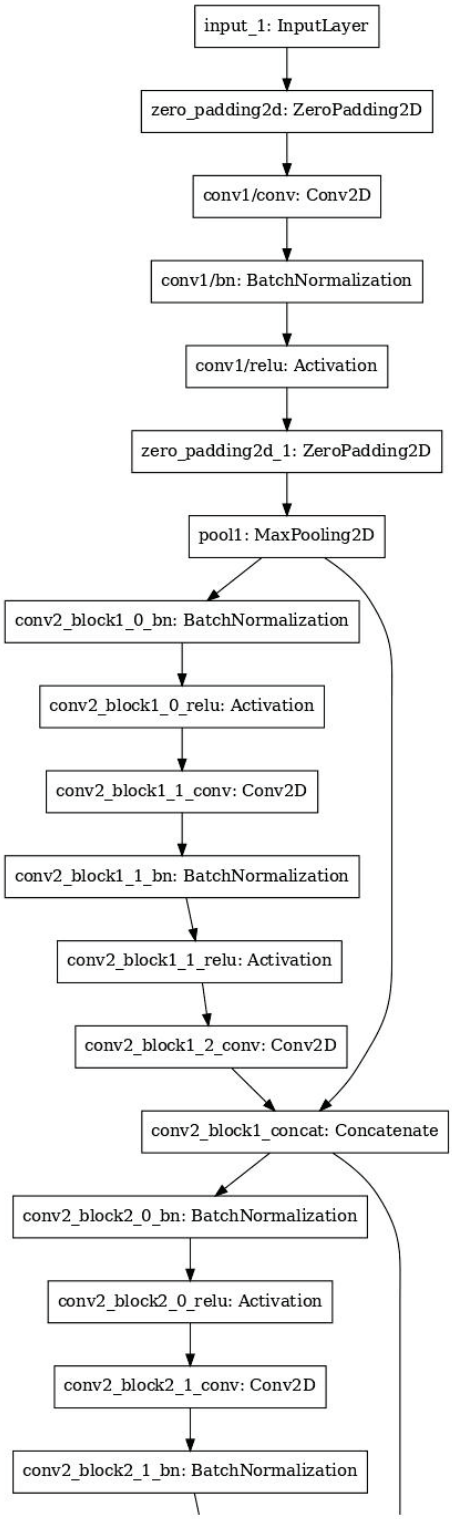


Figure F.2: DenseNet-121 CNN Model Functional Layers: Page 1 of 24

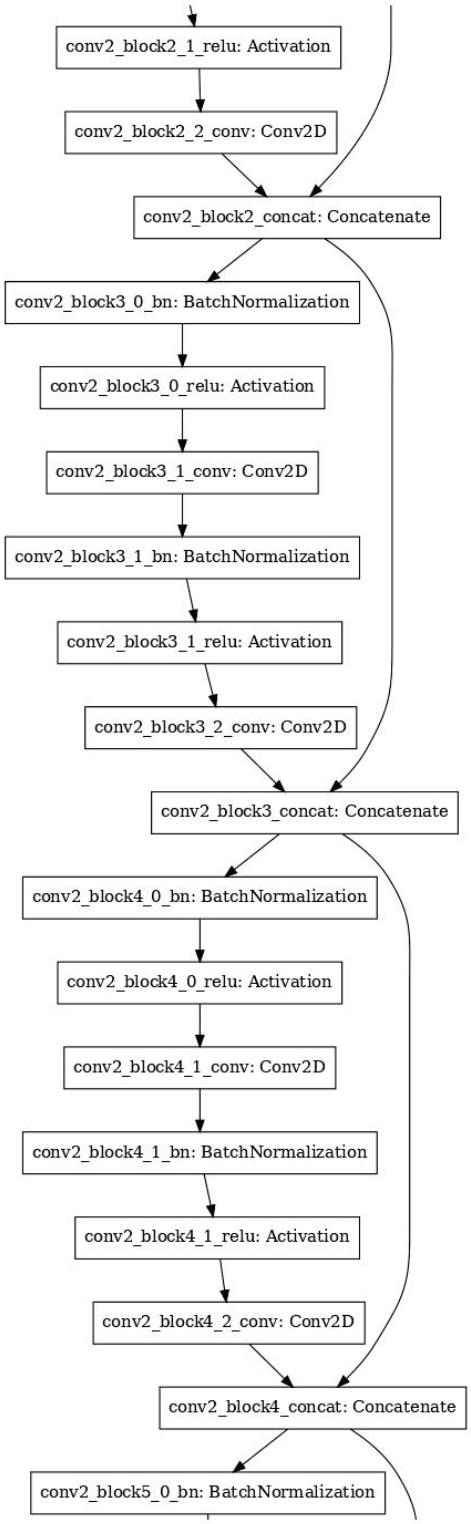


Figure F.3: DenseNet-121 CNN Model Functional Layers: Page 2 of 24

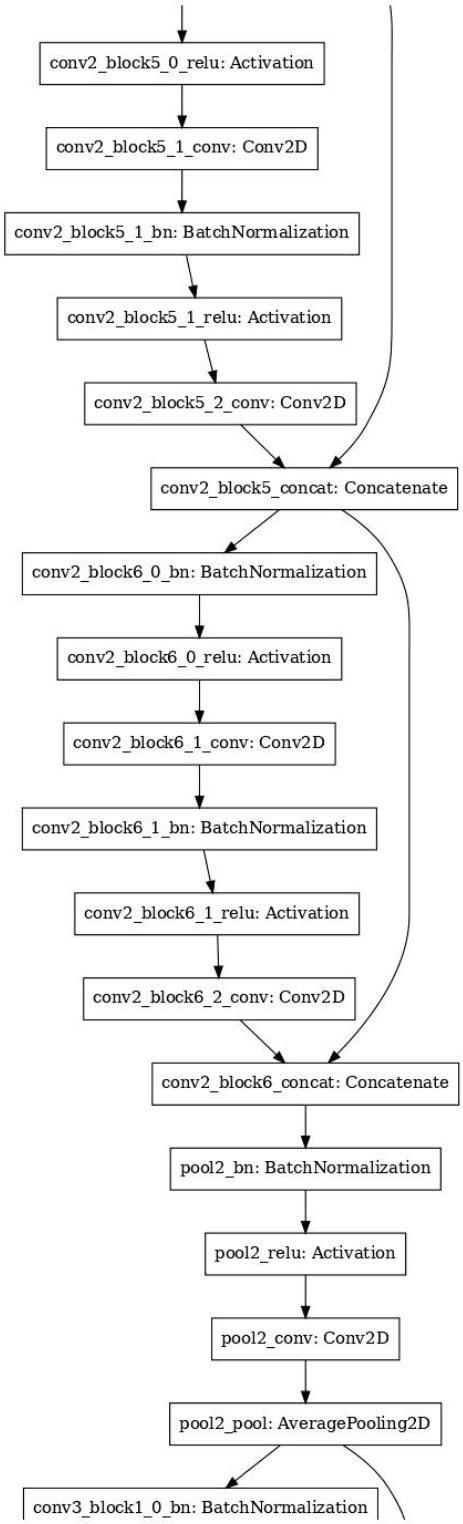


Figure F.4: DenseNet-121 CNN Model Functional Layers: Page 3 of 24

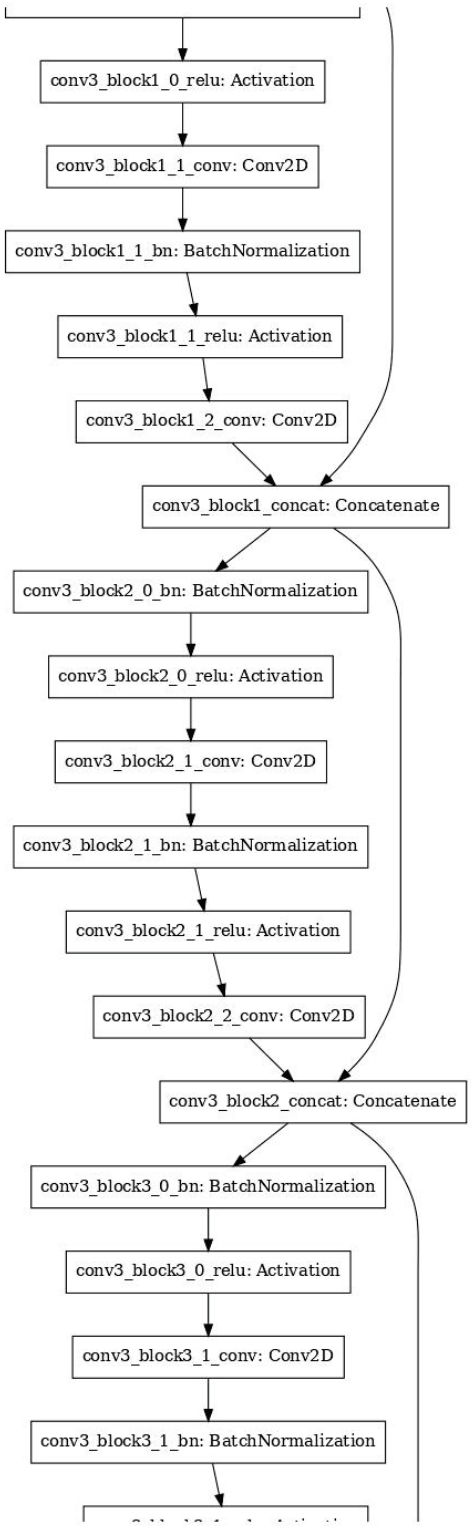


Figure F.5: DenseNet-121 CNN Model Functional Layers: Page 4 of 24

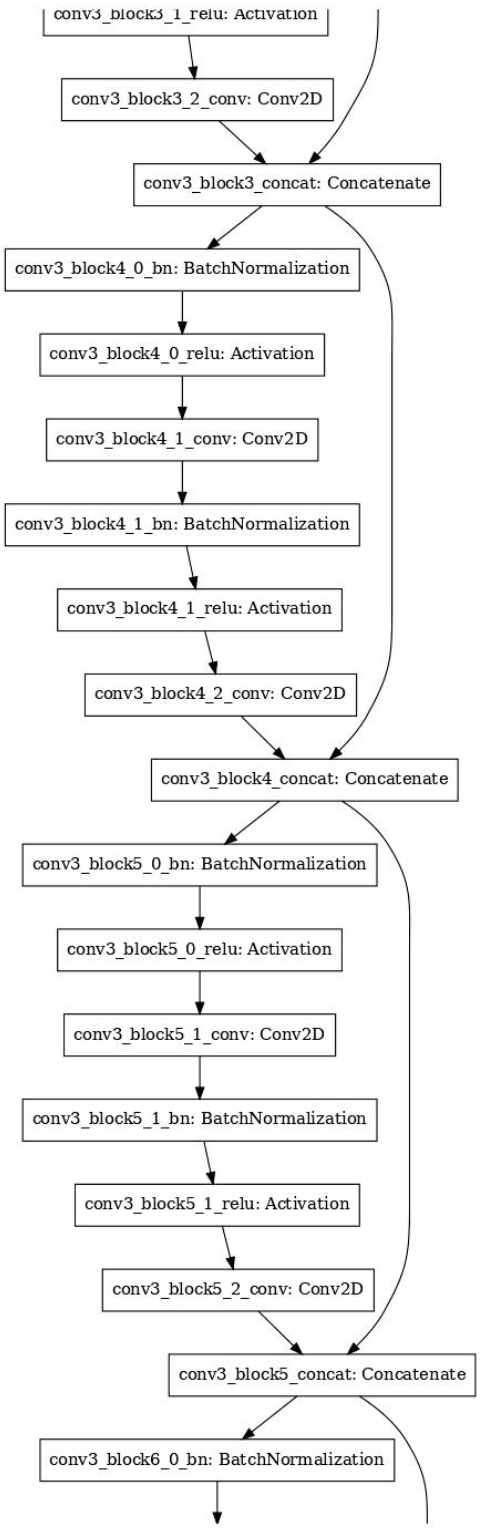


Figure F.6: DenseNet-121 CNN Model Functional Layers: Page 5 of 24

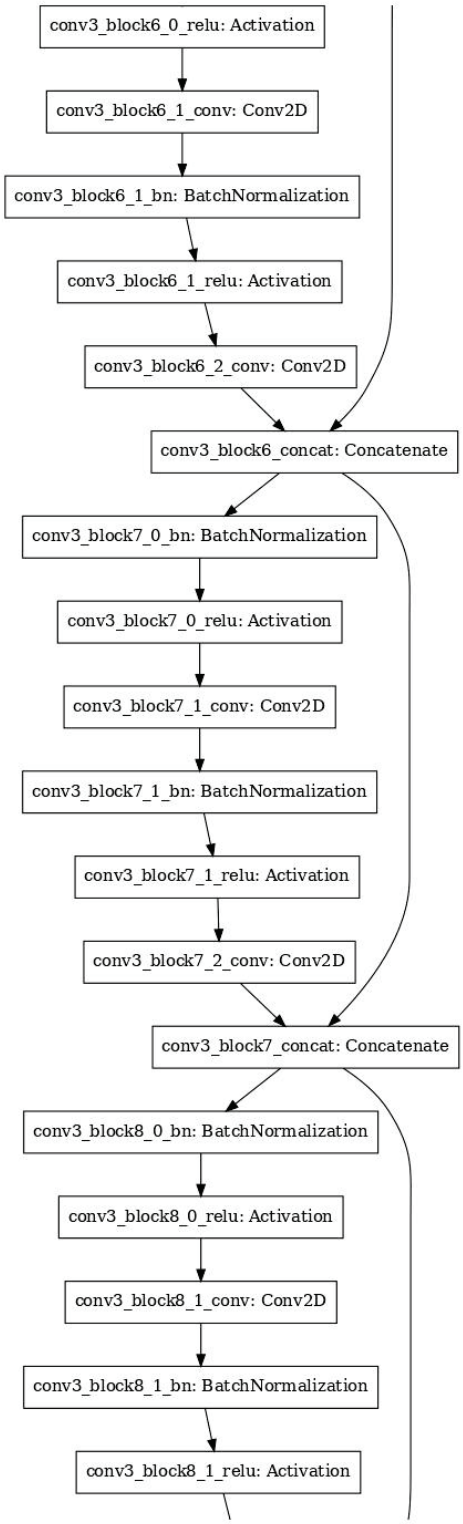


Figure F.7: DenseNet-121 CNN Model Functional Layers: Page 6 of 24

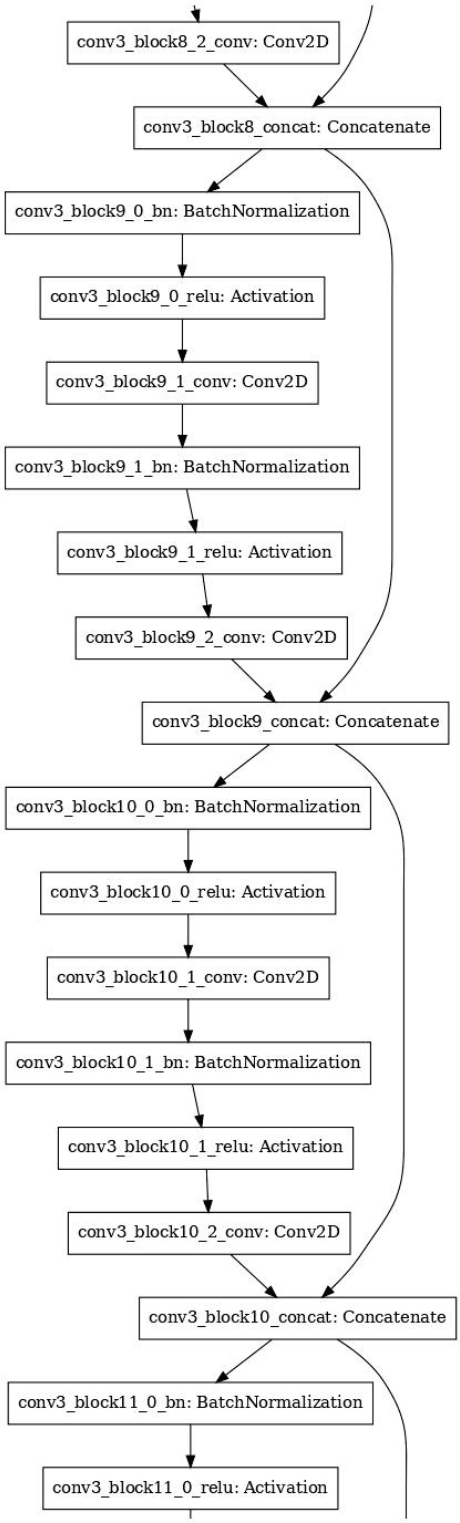


Figure F.8: DenseNet-121 CNN Model Functional Layers: Page 7 of 24

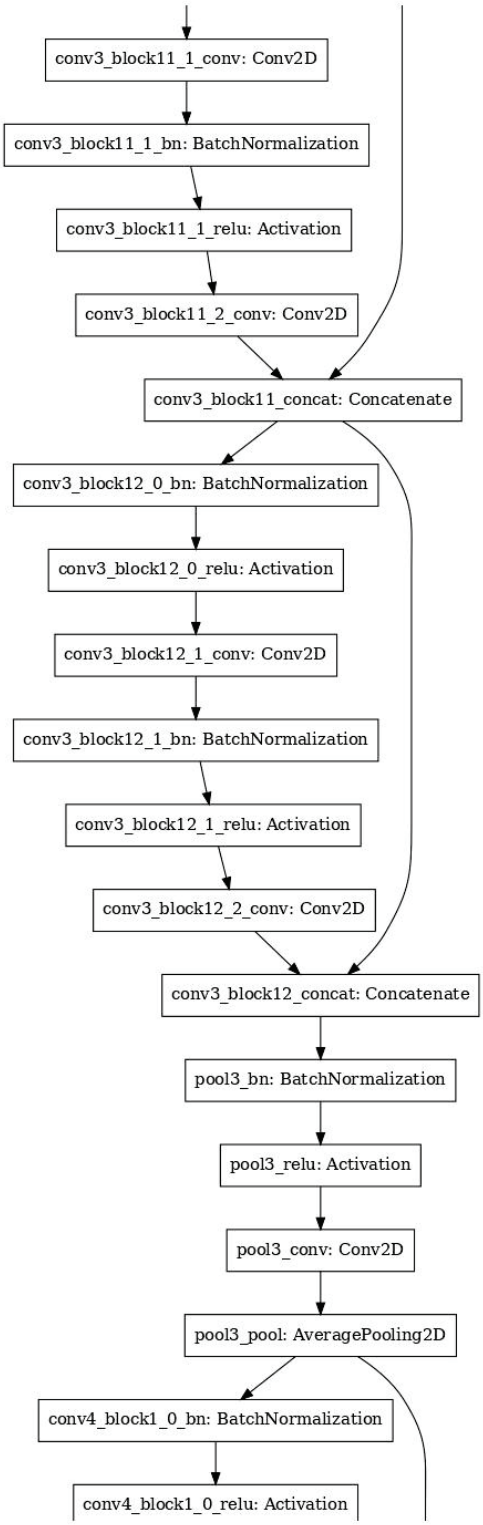


Figure F.9: DenseNet-121 CNN Model Functional Layers: Page 8 of 24

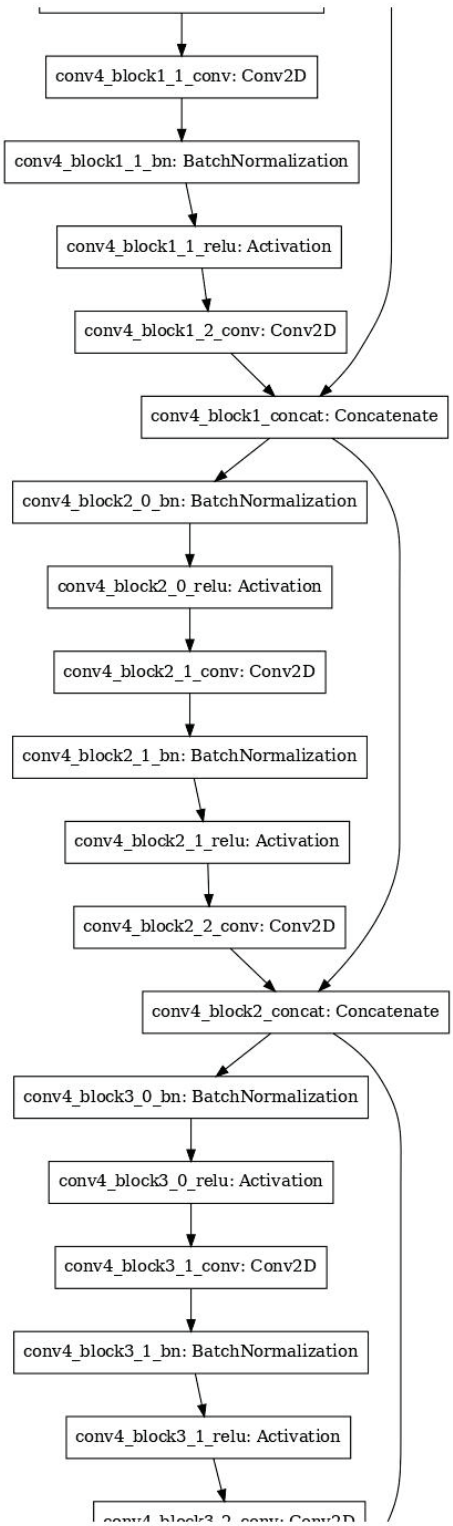


Figure F.10: DenseNet-121 CNN Model Functional Layers: Page 9 of 24

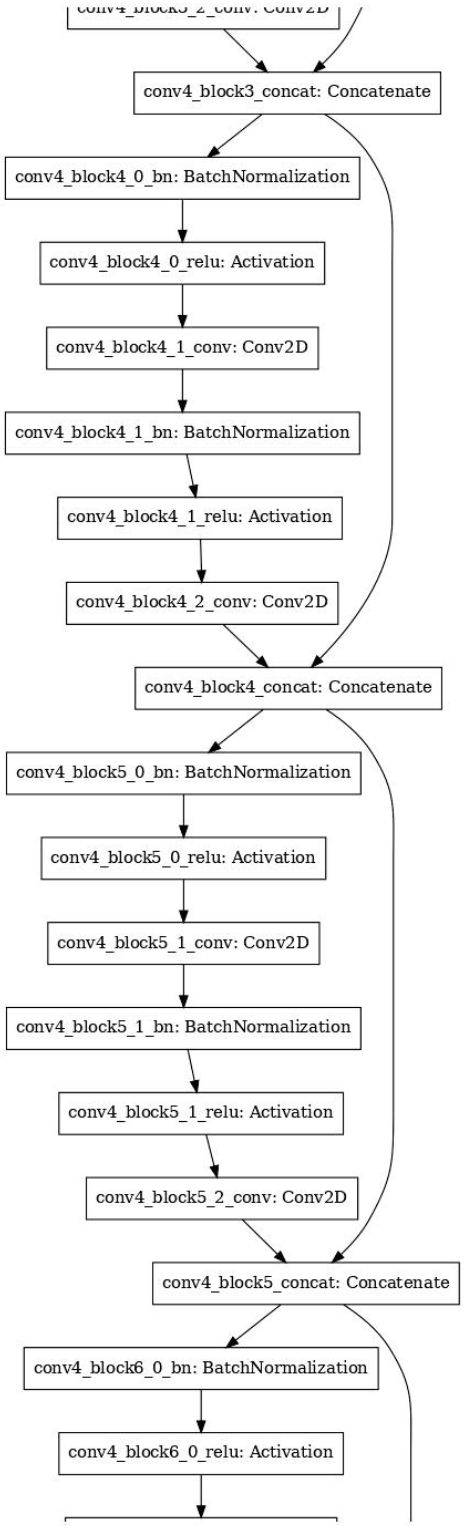


Figure F.11: DenseNet-121 CNN Model Functional Layers: Page 10 of 24

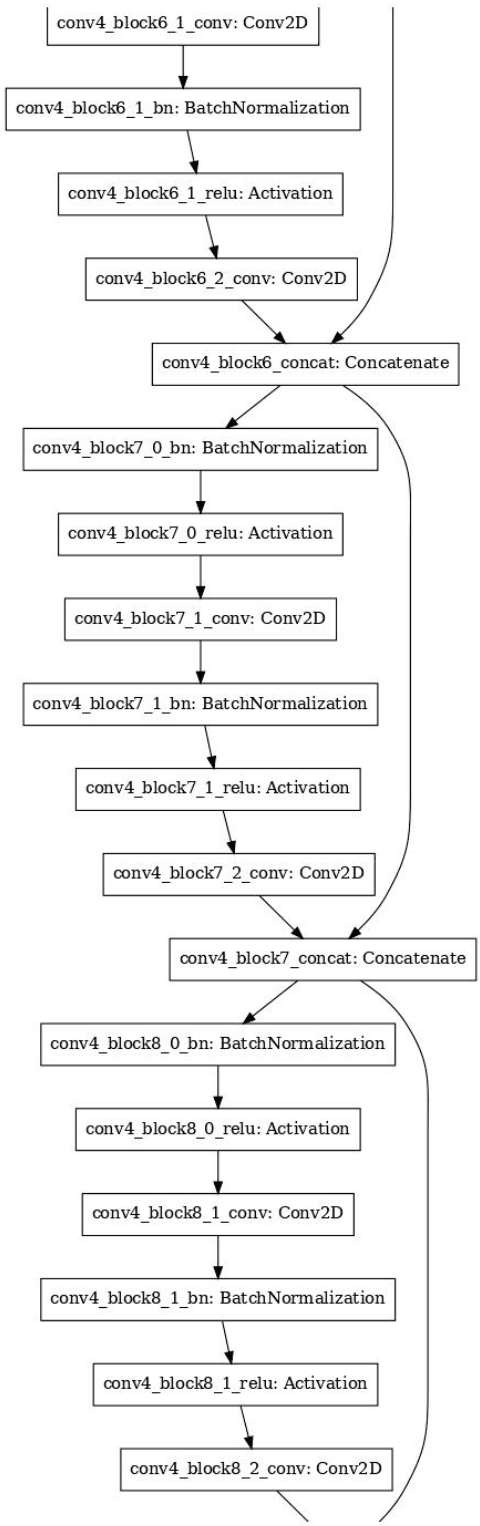


Figure F.12: DenseNet-121 CNN Model Functional Layers: Page 11 of 24

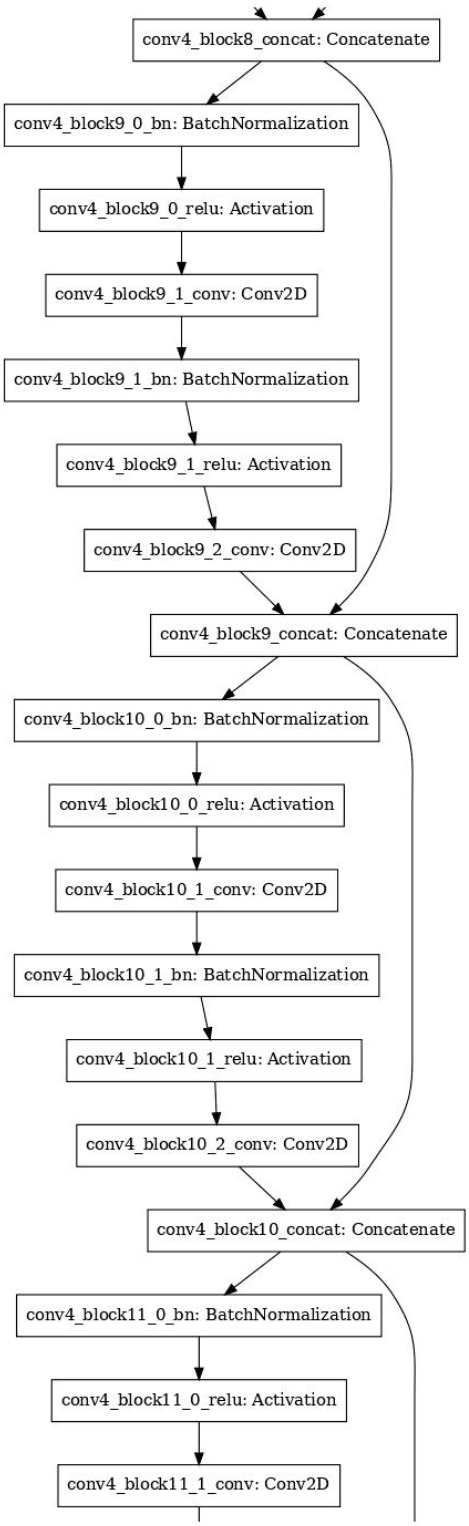


Figure F.13: DenseNet-121 CNN Model Functional Layers: Page 12 of 24

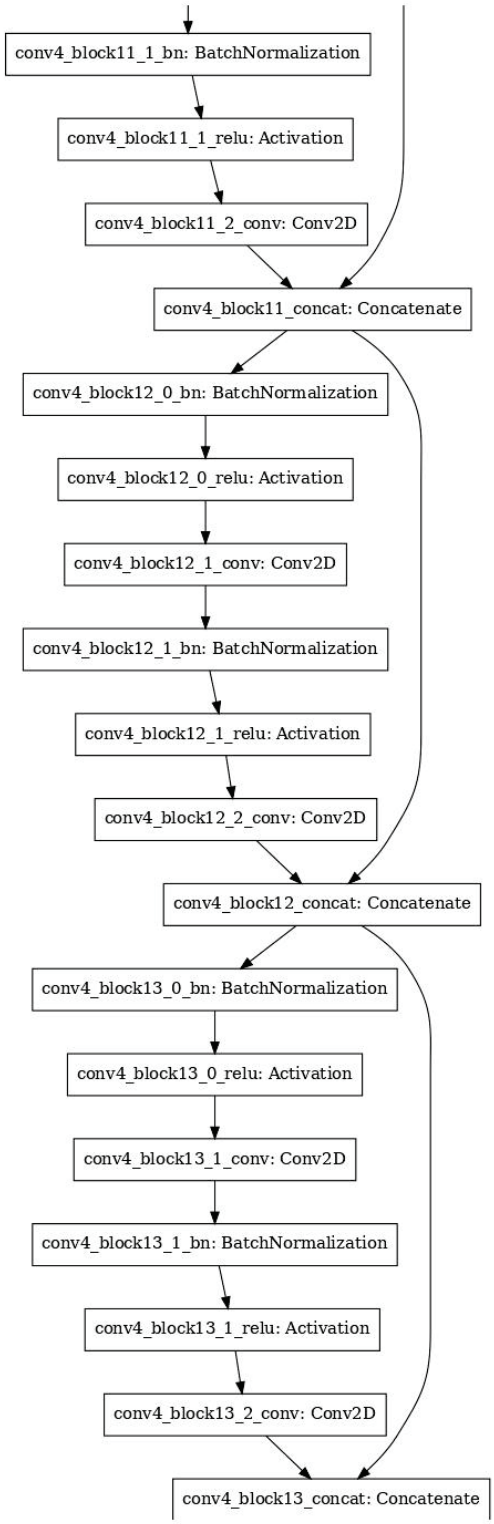


Figure F.14: DenseNet-121 CNN Model Functional Layers: Page 13 of 24

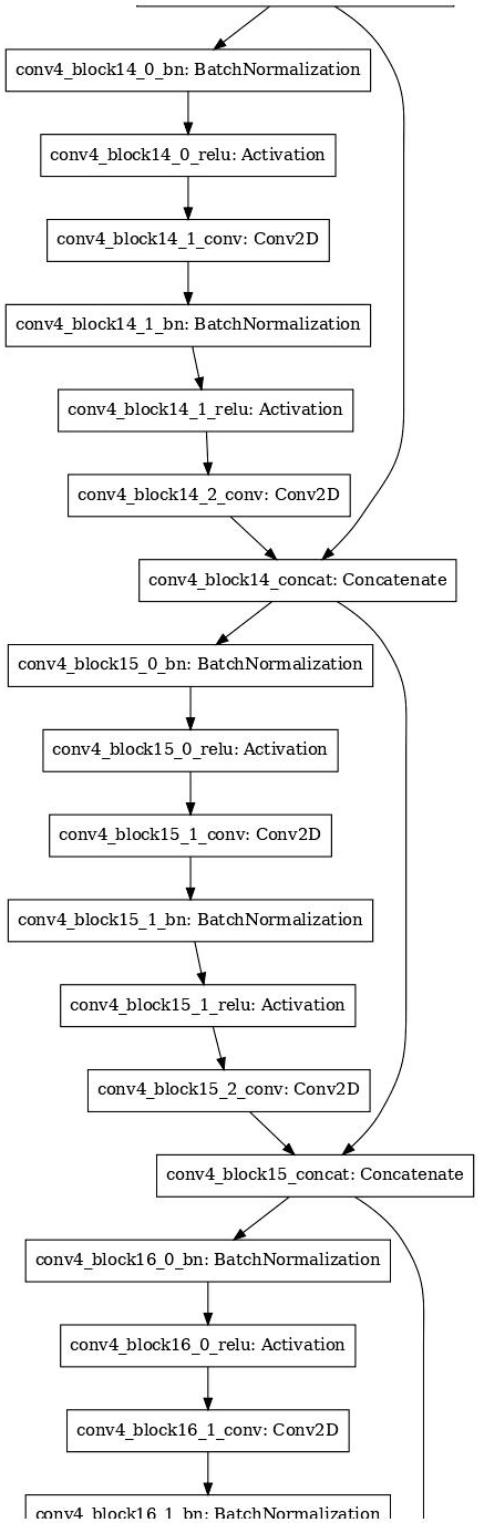


Figure F.15: DenseNet-121 CNN Model Functional Layers: Page 14 of 24

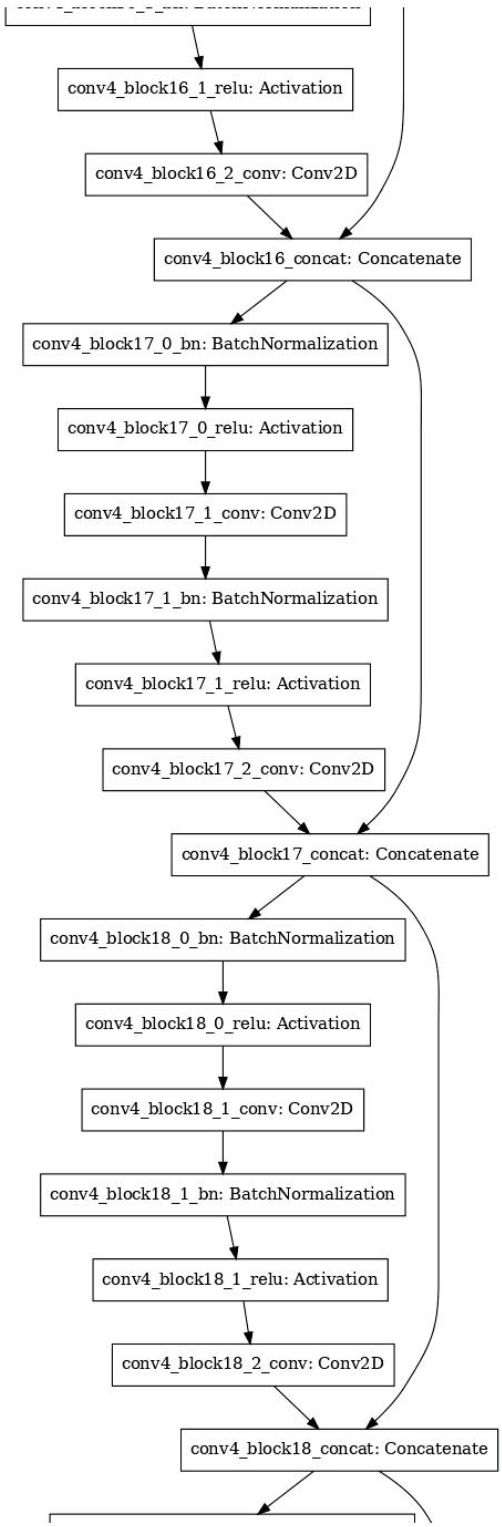


Figure F.16: DenseNet-121 CNN Model Functional Layers: Page 15 of 24

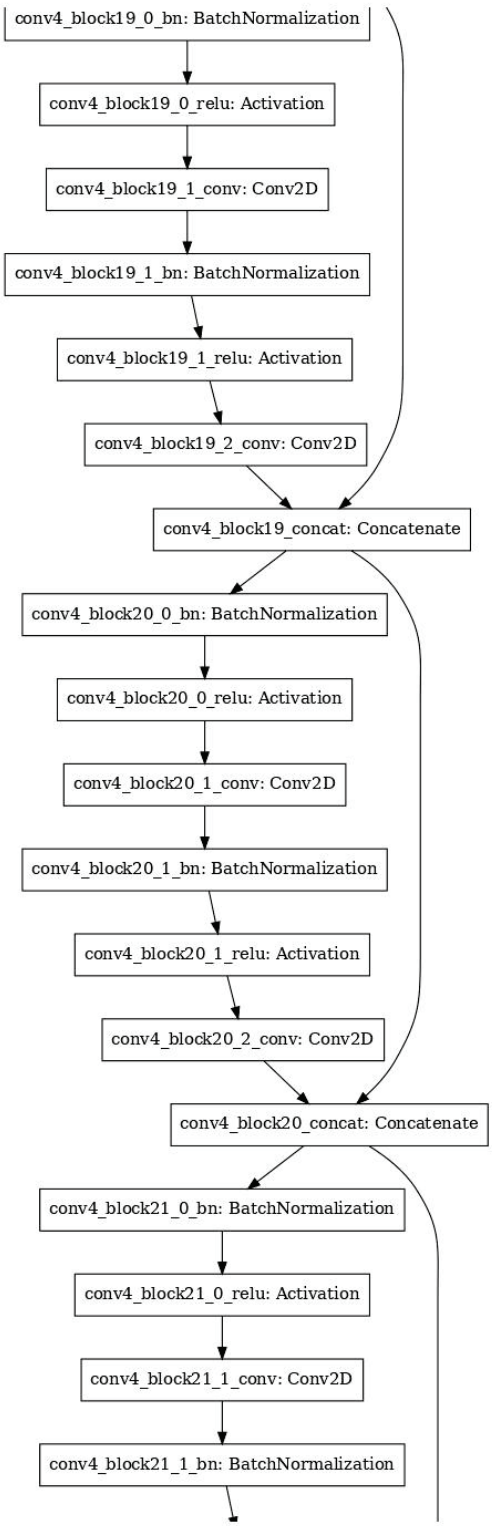


Figure F.17: DenseNet-121 CNN Model Functional Layers: Page 16 of 24

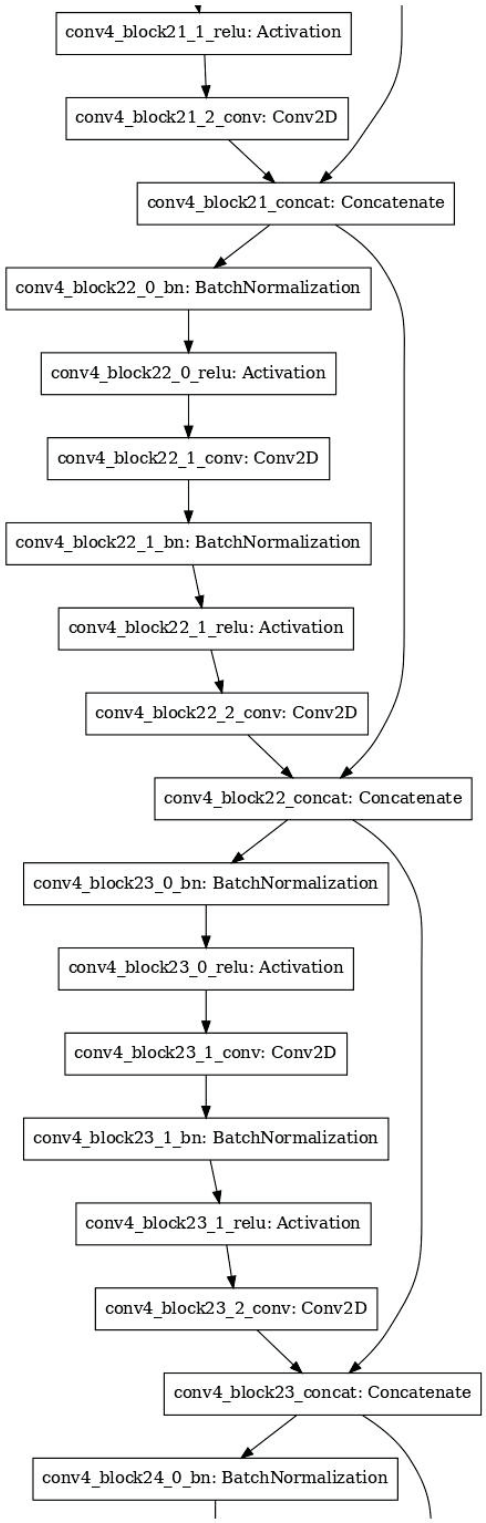


Figure F.18: DenseNet-121 CNN Model Functional Layers: Page 17 of 24

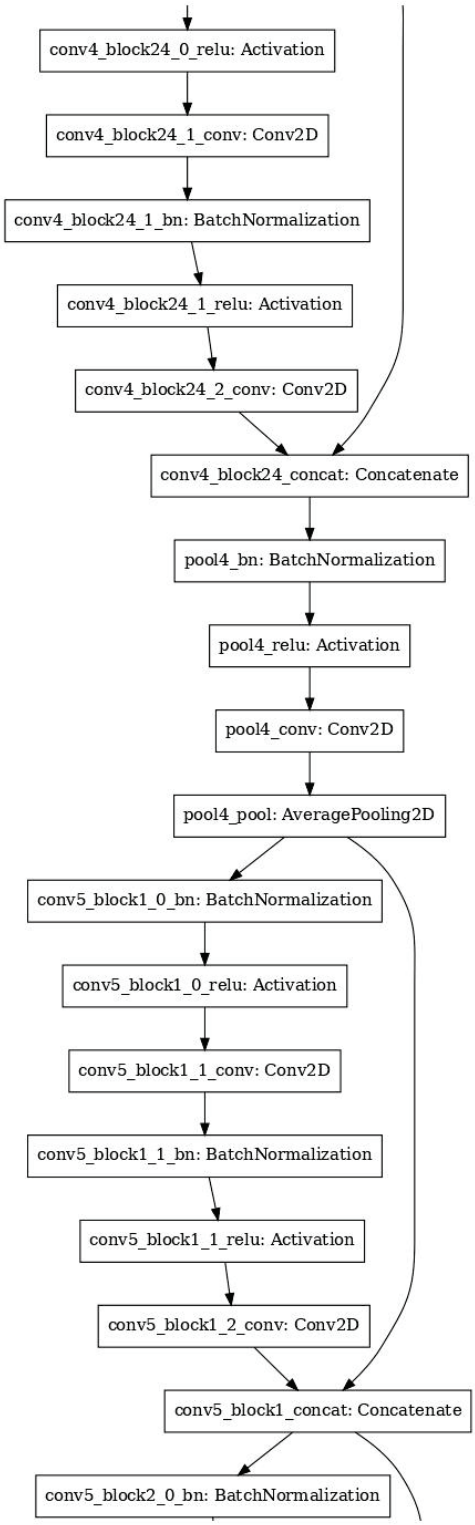


Figure F.19: DenseNet-121 CNN Model Functional Layers: Page 18 of 24

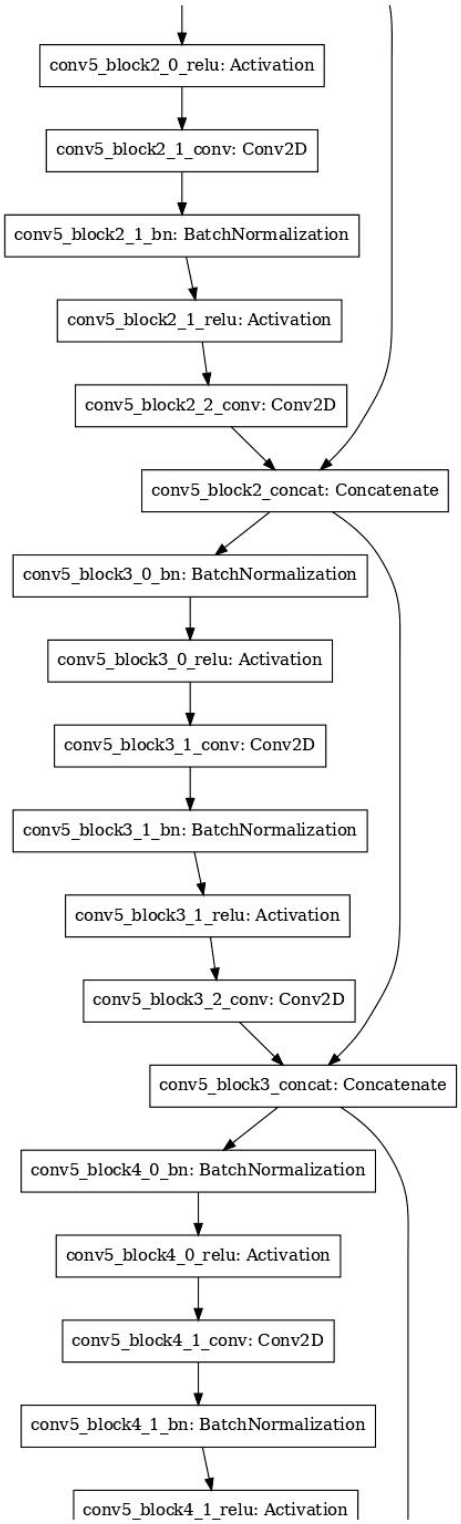


Figure F.20: DenseNet-121 CNN Model Functional Layers: Page 19 of 24

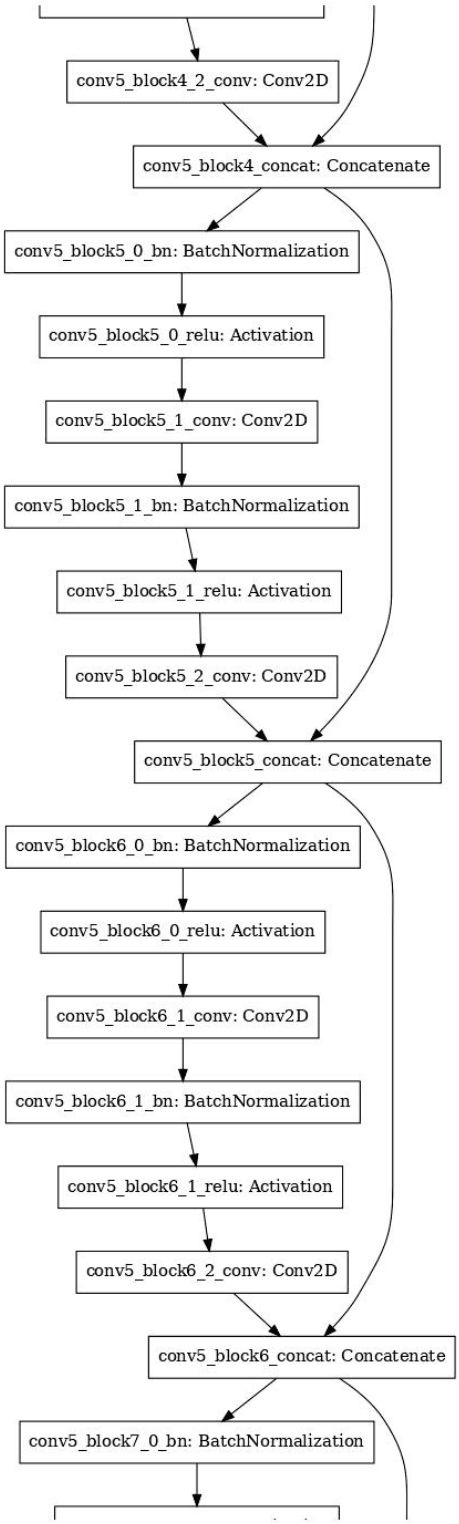


Figure F.21: DenseNet-121 CNN Model Functional Layers: Page 20 of 24

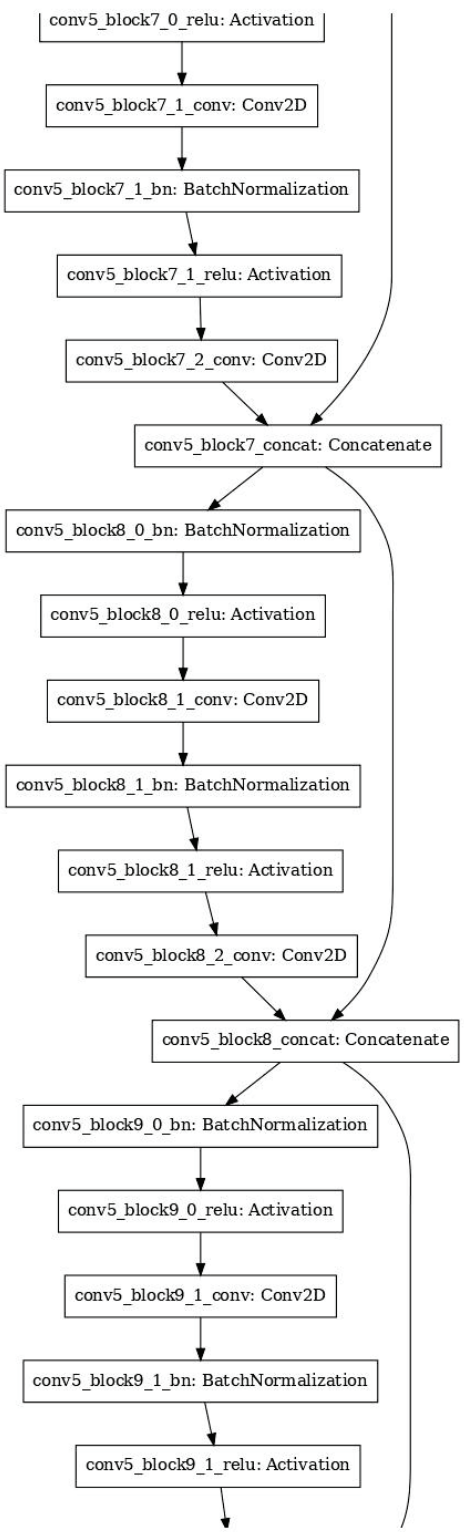


Figure F.22: DenseNet-121 CNN Model Functional Layers: Page 21 of 24

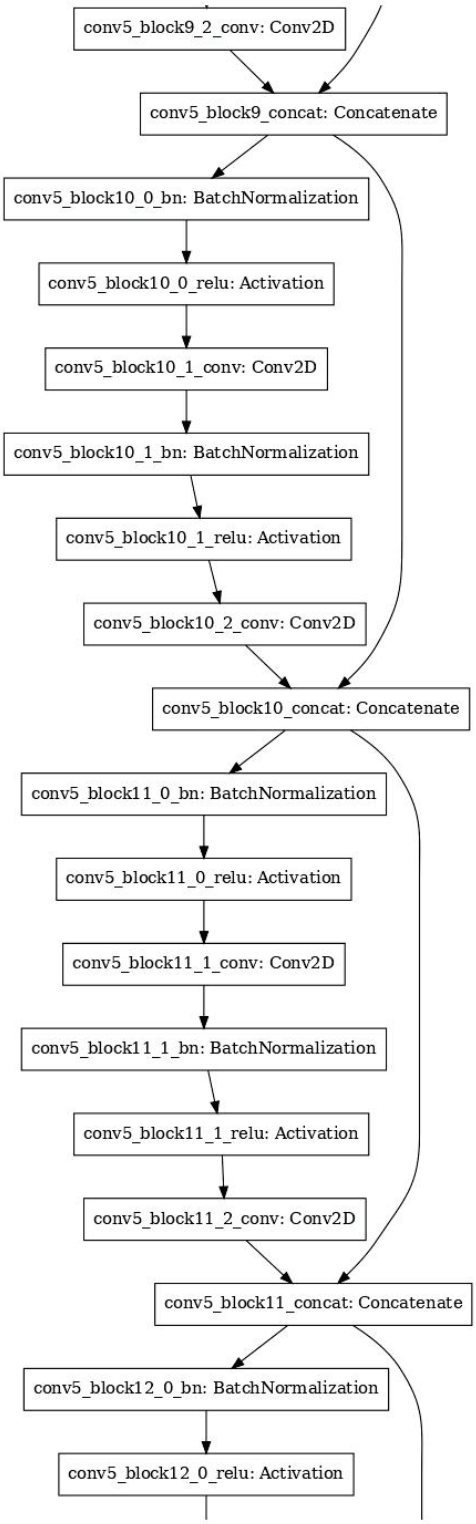


Figure F.23: DenseNet-121 CNN Model Functional Layers: Page 22 of 24

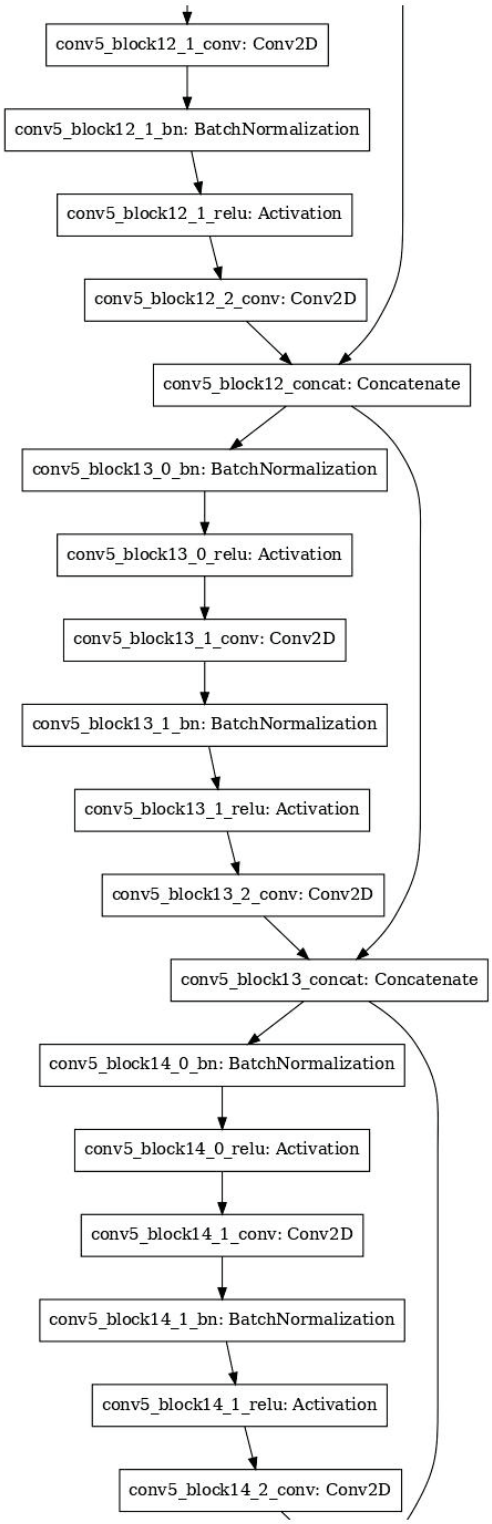


Figure F.24: DenseNet-121 CNN Model Functional Layers: Page 23 of 24

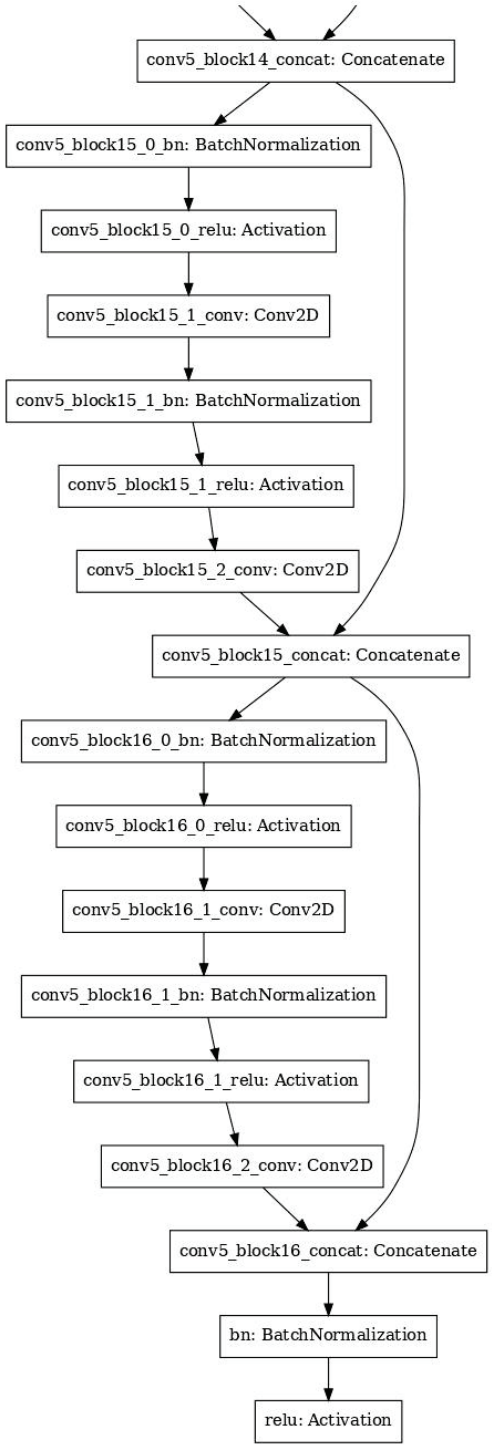


Figure F.25: DenseNet-121 CNN Model Functional Layers: Page 24 of 24

Table F.1: DenseNet-121 Details

Dense	Layer		Input Size			Filter						Param
Stage	No.	Name	Hght	Wdth	Dpth	Cnt	Hght	Wdth	Dpth	Strd	Pad	Count
Pre-Stage	1	zero_padding2d	150	75	1	1	3	3	1	2	3	
Pre-Stage	2	conv1/conv	156	81	1	64	7	7	1	2	0	3,136
Pre-Stage	3	conv1/bn	75	38	64							256
Pre-Stage	4	conv1/relu	75	38	64							
Pre-Stage	5	zero_padding2d_1	75	38	64	1	1	1	1	2	1	
	6	pool1	77	40	64	1	3	3	1	2	0	
Stage-1	7	conv2_block1_0_bn	38	19	64							256
Stage-1	8	conv2_block1_0_relu	38	19	64							
Stage-1	9	conv2_block1_1_conv	38	19	64	128	1	1	64	1	0	8,192
Stage-1	10	conv2_block1_1_bn	38	19	128							512
Stage-1	11	conv2_block1_1_relu	38	19	128							
Stage-1	12	conv2_block1_2_conv	38	19	128	32	3	3	128	1	1	36,864
Stage-1	13	conv2_block1_concat	38	19	96							
Stage-1	14	conv2_block2_0_bn	38	19	96							384
Stage-1	15	conv2_block2_0_relu	38	19	96							

(*table continues*)

Table F.1: DenseNet-121 Details

Dense	Layer		Input Size			Filter						Param
Stage	No.	Name	Hght	Wdth	Dpth	Cnt	Hght	Wdth	Dpth	Strd	Pad	Count
Stage-1	16	conv2_block2_1_conv	38	19	96	128	1	1	96	1	0	12,288
Stage-1	17	conv2_block2_1_bn	38	19	128							512
Stage-1	18	conv2_block2_1_relu	38	19	128							
Stage-1	19	conv2_block2_2_conv	38	19	128	32	3	3	128	1	1	36,864
Stage-1	20	conv2_block2_concat	38	19	128							
Stage-1	21	conv2_block3_0_bn	38	19	128							512
Stage-1	22	conv2_block3_0_relu	38	19	128							
Stage-1	23	conv2_block3_1_conv	38	19	128	128	1	1	128	1	0	16,384
Stage-1	24	conv2_block3_1_bn	38	19	128							512
Stage-1	25	conv2_block3_1_relu	38	19	128							
Stage-1	26	conv2_block3_2_conv	38	19	128	32	3	3	128	1	1	36,864
Stage-1	27	conv2_block3_concat	38	19	160							
Stage-1	28	conv2_block4_0_bn	38	19	160							640
Stage-1	29	conv2_block4_0_relu	38	19	160							
Stage-1	30	conv2_block4_1_conv	38	19	160	128	1	1	160	1	0	20,480

(*table continues*)

Table F.1: DenseNet-121 Details

Dense	Layer		Input Size			Filter						Param
Stage	No.	Name	Hght	Wdth	Dpth	Cnt	Hght	Wdth	Dpth	Strd	Pad	Count
Stage-1	31	conv2_block4_1_bn	38	19	128							512
Stage-1	32	conv2_block4_1_relu	38	19	128							
Stage-1	33	conv2_block4_2_conv	38	19	128	32	3	3	128	1	1	36,864
Stage-1	34	conv2_block4_concat	38	19	192							
Stage-1	35	conv2_block5_0_bn	38	19	192							768
Stage-1	36	conv2_block5_0_relu	38	19	192							
Stage-1	37	conv2_block5_1_conv	38	19	192	128	1	1	192	1	0	24,576
Stage-1	38	conv2_block5_1_bn	38	19	128							512
Stage-1	39	conv2_block5_1_relu	38	19	128							
Stage-1	40	conv2_block5_2_conv	38	19	128	32	3	3	128	1	1	36,864
Stage-1	41	conv2_block5_concat	38	19	224							
Stage-1	42	conv2_block6_0_bn	38	19	224							896
Stage-1	43	conv2_block6_0_relu	38	19	224							
Stage-1	44	conv2_block6_1_conv	38	19	224	128	1	1	224	1	0	28,672
Stage-1	45	conv2_block6_1_bn	38	19	128							512

(*table continues*)

Table F.1: DenseNet-121 Details

Dense	Layer		Input Size			Filter						Param
Stage	No.	Name	Hght	Wdth	Dpth	Cnt	Hght	Wdth	Dpth	Strd	Pad	Count
Stage-1	46	conv2_block6_1_relu	38	19	128							
Stage-1	47	conv2_block6_2_conv	38	19	128	32	3	3	128	1	1	36,864
Stage-1	48	conv2_block6_concat	38	19	256							
Stage-1	49	pool2_bn	38	19	256							1,024
Stage-1	50	pool2_relu	38	19	256							
Stage-1	51	pool2_conv	38	19	256	128	1	1	256	1	0	32,768
	52	pool2_pool	38	19	128	1	2	2	1	2	0	
Stage-2	53	conv3_block1_0_bn	19	9	128							512
Stage-2	54	conv3_block1_0_relu	19	9	128							
Stage-2	55	conv3_block1_1_conv	19	9	128	128	1	1	128	1	0	16,384
Stage-2	56	conv3_block1_1_bn	19	9	128							512
Stage-2	57	conv3_block1_1_relu	19	9	128							
Stage-2	58	conv3_block1_2_conv	19	9	128	32	3	3	128	1	1	36,864
Stage-2	59	conv3_block1_concat	19	9	160							
Stage-2	60	conv3_block2_0_bn	19	9	160							640

(*table continues*)

Table F.1: DenseNet-121 Details

Dense	Layer		Input Size			Filter						Param
Stage	No.	Name	Hght	Wdth	Dpth	Cnt	Hght	Wdth	Dpth	Strd	Pad	Count
Stage-2	61	conv3_block2_0_relu	19	9	160							
Stage-2	62	conv3_block2_1_conv	19	9	160	128	1	1	160	1	0	20,480
Stage-2	63	conv3_block2_1_bn	19	9	128							512
Stage-2	64	conv3_block2_1_relu	19	9	128							
Stage-2	65	conv3_block2_2_conv	19	9	128	32	3	3	128	1	1	36,864
Stage-2	66	conv3_block2_concat	19	9	192							
Stage-2	67	conv3_block3_0_bn	19	9	192							768
Stage-2	68	conv3_block3_0_relu	19	9	192							
Stage-2	69	conv3_block3_1_conv	19	9	192	128	1	1	192	1	0	24,576
Stage-2	70	conv3_block3_1_bn	19	9	128							512
Stage-2	71	conv3_block3_1_relu	19	9	128							
Stage-2	72	conv3_block3_2_conv	19	9	128	32	3	3	128	1	1	36,864
Stage-2	73	conv3_block3_concat	19	9	224							
Stage-2	74	conv3_block4_0_bn	19	9	224							896
Stage-2	75	conv3_block4_0_relu	19	9	224							

(*table continues*)

Table F.1: DenseNet-121 Details

Dense	Layer		Input Size			Filter						Param
Stage	No.	Name	Hght	Wdth	Dpth	Cnt	Hght	Wdth	Dpth	Strd	Pad	Count
Stage-2	76	conv3_block4_1_conv	19	9	224	128	1	1	224	1	0	28,672
Stage-2	77	conv3_block4_1_bn	19	9	128							512
Stage-2	78	conv3_block4_1_relu	19	9	128							
Stage-2	79	conv3_block4_2_conv	19	9	128	32	3	3	128	1	1	36,864
Stage-2	80	conv3_block4_concat	19	9	256							
Stage-2	81	conv3_block5_0_bn	19	9	256							1,024
Stage-2	82	conv3_block5_0_relu	19	9	256							
Stage-2	83	conv3_block5_1_conv	19	9	256	128	1	1	256	1	0	32,768
Stage-2	84	conv3_block5_1_bn	19	9	128							512
Stage-2	85	conv3_block5_1_relu	19	9	128							
Stage-2	86	conv3_block5_2_conv	19	9	128	32	3	3	128	1	1	36,864
Stage-2	87	conv3_block5_concat	19	9	288							
Stage-2	88	conv3_block6_0_bn	19	9	288							1,152
Stage-2	89	conv3_block6_0_relu	19	9	288							
Stage-2	90	conv3_block6_1_conv	19	9	288	128	1	1	288	1	0	36,864

(*table continues*)

Table F.1: DenseNet-121 Details

Dense	Layer		Input Size			Filter						Param
Stage	No.	Name	Hght	Wdth	Dpth	Cnt	Hght	Wdth	Dpth	Strd	Pad	Count
Stage-2	91	conv3_block6_1_bn	19	9	128							512
Stage-2	92	conv3_block6_1_relu	19	9	128							
Stage-2	93	conv3_block6_2_conv	19	9	128	32	3	3	128	1	1	36,864
Stage-2	94	conv3_block6_concat	19	9	320							
Stage-2	95	conv3_block7_0_bn	19	9	320							1,280
Stage-2	96	conv3_block7_0_relu	19	9	320							
Stage-2	97	conv3_block7_1_conv	19	9	320	128	1	1	320	1	0	40,960
Stage-2	98	conv3_block7_1_bn	19	9	128							512
Stage-2	99	conv3_block7_1_relu	19	9	128							
Stage-2	100	conv3_block7_2_conv	19	9	128	32	3	3	128	1	1	36,864
Stage-2	101	conv3_block7_concat	19	9	352							
Stage-2	102	conv3_block8_0_bn	19	9	352							1,408
Stage-2	103	conv3_block8_0_relu	19	9	352							
Stage-2	104	conv3_block8_1_conv	19	9	352	128	1	1	352	1	0	45,056
Stage-2	105	conv3_block8_1_bn	19	9	128							512

(*table continues*)

Table F.1: DenseNet-121 Details

Dense	Layer		Input Size			Filter						Param
Stage	No.	Name	Hght	Wdth	Dpth	Cnt	Hght	Wdth	Dpth	Strd	Pad	Count
Stage-2	106	conv3_block8_1_relu	19	9	128							
Stage-2	107	conv3_block8_2_conv	19	9	128	32	3	3	128	1	1	36,864
Stage-2	108	conv3_block8_concat	19	9	384							
Stage-2	109	conv3_block9_0_bn	19	9	384							1,536
Stage-2	110	conv3_block9_0_relu	19	9	384							
Stage-2	111	conv3_block9_1_conv	19	9	384	128	1	1	384	1	0	49,152
Stage-2	112	conv3_block9_1_bn	19	9	128							512
Stage-2	113	conv3_block9_1_relu	19	9	128							
Stage-2	114	conv3_block9_2_conv	19	9	128	32	3	3	128	1	1	36,864
Stage-2	115	conv3_block9_concat	19	9	416							
Stage-2	116	conv3_block10_0_bn	19	9	416							1,664
Stage-2	117	conv3_block10_0_relu	19	9	416							
Stage-2	118	conv3_block10_1_conv	19	9	416	128	1	1	416	1	0	53,248
Stage-2	119	conv3_block10_1_bn	19	9	128							512
Stage-2	120	conv3_block10_1_relu	19	9	128							

(*table continues*)

Table F.1: DenseNet-121 Details

Dense	Layer		Input Size			Filter						Param
Stage	No.	Name	Hght	Wdth	Dpth	Cnt	Hght	Wdth	Dpth	Strd	Pad	Count
Stage-2	121	conv3_block10_2_conv	19	9	128	32	3	3	128	1	1	36,864
Stage-2	122	conv3_block10_concat	19	9	448							
Stage-2	123	conv3_block11_0_bn	19	9	448							1,792
Stage-2	124	conv3_block11_0_relu	19	9	448							
Stage-2	125	conv3_block11_1_conv	19	9	448	128	1	1	448	1	0	57,344
Stage-2	126	conv3_block11_1_bn	19	9	128							512
Stage-2	127	conv3_block11_1_relu	19	9	128							
Stage-2	128	conv3_block11_2_conv	19	9	128	32	3	3	128	1	1	36,864
Stage-2	129	conv3_block11_concat	19	9	480							
Stage-2	130	conv3_block12_0_bn	19	9	480							1,920
Stage-2	131	conv3_block12_0_relu	19	9	480							
Stage-2	132	conv3_block12_1_conv	19	9	480	128	1	1	480	1	0	61,440
Stage-2	133	conv3_block12_1_bn	19	9	128							512
Stage-2	134	conv3_block12_1_relu	19	9	128							
Stage-2	135	conv3_block12_2_conv	19	9	128	32	3	3	128	1	1	36,864

(*table continues*)

Table F.1: DenseNet-121 Details

Dense	Layer		Input Size			Filter						Param
Stage	No.	Name	Hght	Wdth	Dpth	Cnt	Hght	Wdth	Dpth	Strd	Pad	Count
Stage-2	136	conv3_block12_concat	19	9	512							
Stage-2	137	pool3_bn	19	9	512							2,048
Stage-2	138	pool3_relu	19	9	512							
Stage-2	139	pool3_conv	19	9	512	256	1	1	256	1	0	131,072
	140	pool3_pool	19	9	256	1	2	2	1	2	0	
Stage-3	141	conv4_block1_0_bn	9	4	256							1,024
Stage-3	142	conv4_block1_0_relu	9	4	256							
Stage-3	143	conv4_block1_1_conv	9	4	256	128	1	1	256	1	0	32,768
Stage-3	144	conv4_block1_1_bn	9	4	128							512
Stage-3	145	conv4_block1_1_relu	9	4	128							
Stage-3	146	conv4_block1_2_conv	9	4	128	32	3	3	128	1	1	36,864
Stage-3	147	conv4_block1_concat	9	4	288							
Stage-3	148	conv4_block2_0_bn	9	4	288							1,152
Stage-3	149	conv4_block2_0_relu	9	4	288							
Stage-3	150	conv4_block2_1_conv	9	4	288	128	1	1	288	1	0	36,864

(*table continues*)

Table F.1: DenseNet-121 Details

Dense	Layer		Input Size			Filter						Param
Stage	No.	Name	Hght	Wdth	Dpth	Cnt	Hght	Wdth	Dpth	Strd	Pad	Count
Stage-3	151	conv4_block2_1_bn	9	4	128							512
Stage-3	152	conv4_block2_1_relu	9	4	128							
Stage-3	153	conv4_block2_2_conv	9	4	128	32	3	3	128	1	1	36,864
Stage-3	154	conv4_block2_concat	9	4	320							
Stage-3	155	conv4_block3_0_bn	9	4	320							1,280
Stage-3	156	conv4_block3_0_relu	9	4	320							
Stage-3	157	conv4_block3_1_conv	9	4	320	128	1	1	320	1	0	40,960
Stage-3	158	conv4_block3_1_bn	9	4	128							512
Stage-3	159	conv4_block3_1_relu	9	4	128							
Stage-3	160	conv4_block3_2_conv	9	4	128	32	3	3	128	1	1	36,864
Stage-3	161	conv4_block3_concat	9	4	352							
Stage-3	162	conv4_block4_0_bn	9	4	352							1,408
Stage-3	163	conv4_block4_0_relu	9	4	352							
Stage-3	164	conv4_block4_1_conv	9	4	352	128	1	1	352	1	0	45,056
Stage-3	165	conv4_block4_1_bn	9	4	128							512

(*table continues*)

Table F.1: DenseNet-121 Details

Dense	Layer		Input Size			Filter						Param
Stage	No.	Name	Hght	Wdth	Dpth	Cnt	Hght	Wdth	Dpth	Strd	Pad	Count
Stage-3	166	conv4_block4_1_relu	9	4	128							
Stage-3	167	conv4_block4_2_conv	9	4	128	32	3	3	128	1	1	36,864
Stage-3	168	conv4_block4_concat	9	4	384							
Stage-3	169	conv4_block5_0_bn	9	4	384							1,536
Stage-3	170	conv4_block5_0_relu	9	4	384							
Stage-3	171	conv4_block5_1_conv	9	4	384	128	1	1	384	1	0	49,152
Stage-3	172	conv4_block5_1_bn	9	4	128							512
Stage-3	173	conv4_block5_1_relu	9	4	128							
Stage-3	174	conv4_block5_2_conv	9	4	128	32	3	3	128	1	1	36,864
Stage-3	175	conv4_block5_concat	9	4	416							
Stage-3	176	conv4_block6_0_bn	9	4	416							1,664
Stage-3	177	conv4_block6_0_relu	9	4	416							
Stage-3	178	conv4_block6_1_conv	9	4	416	128	1	1	416	1	0	53,248
Stage-3	179	conv4_block6_1_bn	9	4	128							512
Stage-3	180	conv4_block6_1_relu	9	4	128							

(*table continues*)

Table F.1: DenseNet-121 Details

Dense	Layer		Input Size			Filter						Param
Stage	No.	Name	Hght	Wdth	Dpth	Cnt	Hght	Wdth	Dpth	Strd	Pad	Count
Stage-3	181	conv4_block6_2_conv	9	4	128	32	3	3	128	1	1	36,864
Stage-3	182	conv4_block6_concat	9	4	448							
Stage-3	183	conv4_block7_0_bn	9	4	448							1,792
Stage-3	184	conv4_block7_0_relu	9	4	448							
Stage-3	185	conv4_block7_1_conv	9	4	448	128	1	1	448	1	0	57,344
Stage-3	186	conv4_block7_1_bn	9	4	128							512
Stage-3	187	conv4_block7_1_relu	9	4	128							
Stage-3	188	conv4_block7_2_conv	9	4	128	32	3	3	128	1	1	36,864
Stage-3	189	conv4_block7_concat	9	4	480							
Stage-3	190	conv4_block8_0_bn	9	4	480							1,920
Stage-3	191	conv4_block8_0_relu	9	4	480							
Stage-3	192	conv4_block8_1_conv	9	4	480	128	1	1	480	1	0	61,440
Stage-3	193	conv4_block8_1_bn	9	4	128							512
Stage-3	194	conv4_block8_1_relu	9	4	128							
Stage-3	195	conv4_block8_2_conv	9	4	128	32	3	3	128	1	1	36,864

(*table continues*)

Table F.1: DenseNet-121 Details

Dense	Layer		Input Size			Filter						Param
Stage	No.	Name	Hght	Wdth	Dpth	Cnt	Hght	Wdth	Dpth	Strd	Pad	Count
Stage-3	196	conv4_block8_concat	9	4	512							
Stage-3	197	conv4_block9_0_bn	9	4	512							2,048
Stage-3	198	conv4_block9_0_relu	9	4	512							
Stage-3	199	conv4_block9_1_conv	9	4	512	128	1	1	512	1	0	65,536
Stage-3	200	conv4_block9_1_bn	9	4	128							512
Stage-3	201	conv4_block9_1_relu	9	4	128							
Stage-3	202	conv4_block9_2_conv	9	4	128	32	3	3	128	1	1	36,864
Stage-3	203	conv4_block9_concat	9	4	544							
Stage-3	204	conv4_block10_0_bn	9	4	544							2,176
Stage-3	205	conv4_block10_0_relu	9	4	544							
Stage-3	206	conv4_block10_1_conv	9	4	544	128	1	1	544	1	0	69,632
Stage-3	207	conv4_block10_1_bn	9	4	128							512
Stage-3	208	conv4_block10_1_relu	9	4	128							
Stage-3	209	conv4_block10_2_conv	9	4	128	32	3	3	128	1	1	36,864
Stage-3	210	conv4_block10_concat	9	4	576							

(*table continues*)

Table F.1: DenseNet-121 Details

Dense	Layer		Input Size			Filter						Param
Stage	No.	Name	Hght	Wdth	Dpth	Cnt	Hght	Wdth	Dpth	Strd	Pad	Count
Stage-3	211	conv4_block11_0_bn	9	4	576							2,304
Stage-3	212	conv4_block11_0_relu	9	4	576							
Stage-3	213	conv4_block11_1_conv	9	4	576	128	1	1	576	1	0	73,728
Stage-3	214	conv4_block11_1_bn	9	4	128							512
Stage-3	215	conv4_block11_1_relu	9	4	128							
Stage-3	216	conv4_block11_2_conv	9	4	128	32	3	3	128	1	1	36,864
Stage-3	217	conv4_block11_concat	9	4	608							
Stage-3	218	conv4_block12_0_bn	9	4	608							2,432
Stage-3	219	conv4_block12_0_relu	9	4	608							
Stage-3	220	conv4_block12_1_conv	9	4	608	128	1	1	608	1	0	77,824
Stage-3	221	conv4_block12_1_bn	9	4	128							512
Stage-3	222	conv4_block12_1_relu	9	4	128							
Stage-3	223	conv4_block12_2_conv	9	4	128	32	3	3	128	1	1	36,864
Stage-3	224	conv4_block12_concat	9	4	640							
Stage-3	225	conv4_block13_0_bn	9	4	640							2,560

(*table continues*)

Table F.1: DenseNet-121 Details

Dense	Layer		Input Size			Filter						Param
Stage	No.	Name	Hght	Wdth	Dpth	Cnt	Hght	Wdth	Dpth	Strd	Pad	Count
Stage-3	226	conv4_block13_0_relu	9	4	640							
Stage-3	227	conv4_block13_1_conv	9	4	640	128	1	1	640	1	0	81,920
Stage-3	228	conv4_block13_1_bn	9	4	128							512
Stage-3	229	conv4_block13_1_relu	9	4	128							
Stage-3	230	conv4_block13_2_conv	9	4	128	32	3	3	128	1	1	36,864
Stage-3	231	conv4_block13_concat	9	4	672							
Stage-3	232	conv4_block14_0_bn	9	4	672							2,688
Stage-3	233	conv4_block14_0_relu	9	4	672							
Stage-3	234	conv4_block14_1_conv	9	4	672	128	1	1	672	1	0	86,016
Stage-3	235	conv4_block14_1_bn	9	4	128							512
Stage-3	236	conv4_block14_1_relu	9	4	128							
Stage-3	237	conv4_block14_2_conv	9	4	128	32	3	3	128	1	1	36,864
Stage-3	238	conv4_block14_concat	9	4	704							
Stage-3	239	conv4_block15_0_bn	9	4	704							2,816
Stage-3	240	conv4_block15_0_relu	9	4	704							

(*table continues*)

Table F.1: DenseNet-121 Details

Dense	Layer		Input Size			Filter						Param
Stage	No.	Name	Hght	Wdth	Dpth	Cnt	Hght	Wdth	Dpth	Strd	Pad	Count
Stage-3	241	conv4_block15_1_conv	9	4	704	128	1	1	704	1	0	90,112
Stage-3	242	conv4_block15_1_bn	9	4	128							512
Stage-3	243	conv4_block15_1_relu	9	4	128							
Stage-3	244	conv4_block15_2_conv	9	4	128	32	3	3	128	1	1	36,864
Stage-3	245	conv4_block15_concat	9	4	736							
Stage-3	246	conv4_block16_0_bn	9	4	736							2,944
Stage-3	247	conv4_block16_0_relu	9	4	736							
Stage-3	248	conv4_block16_1_conv	9	4	736	128	1	1	736	1	0	94,208
Stage-3	249	conv4_block16_1_bn	9	4	128							512
Stage-3	250	conv4_block16_1_relu	9	4	128							
Stage-3	251	conv4_block16_2_conv	9	4	128	32	3	3	128	1	1	36,864
Stage-3	252	conv4_block16_concat	9	4	768							
Stage-3	253	conv4_block17_0_bn	9	4	768							3,072
Stage-3	254	conv4_block17_0_relu	9	4	768							
Stage-3	255	conv4_block17_1_conv	9	4	768	128	1	1	768	1	0	98,304

(*table continues*)

Table F.1: DenseNet-121 Details

Dense	Layer		Input Size			Filter						Param
Stage	No.	Name	Hght	Wdth	Dpth	Cnt	Hght	Wdth	Dpth	Strd	Pad	Count
Stage-3	256	conv4_block17_1_bn	9	4	128							512
Stage-3	257	conv4_block17_1_relu	9	4	128							
Stage-3	258	conv4_block17_2_conv	9	4	128	32	3	3	128	1	1	36,864
Stage-3	259	conv4_block17_concat	9	4	800							
Stage-3	260	conv4_block18_0_bn	9	4	800							3,200
Stage-3	261	conv4_block18_0_relu	9	4	800							
Stage-3	262	conv4_block18_1_conv	9	4	800	128	1	1	800	1	0	102,400
Stage-3	263	conv4_block18_1_bn	9	4	128							512
Stage-3	264	conv4_block18_1_relu	9	4	128							
Stage-3	265	conv4_block18_2_conv	9	4	128	32	3	3	128	1	1	36,864
Stage-3	266	conv4_block18_concat	9	4	832							
Stage-3	267	conv4_block19_0_bn	9	4	832							3,328
Stage-3	268	conv4_block19_0_relu	9	4	832							
Stage-3	269	conv4_block19_1_conv	9	4	832	128	1	1	832	1	0	106,496
Stage-3	270	conv4_block19_1_bn	9	4	128							512

(*table continues*)

Table F.1: DenseNet-121 Details

Dense	Layer		Input Size			Filter						Param
Stage	No.	Name	Hght	Wdth	Dpth	Cnt	Hght	Wdth	Dpth	Strd	Pad	Count
Stage-3	271	conv4_block19_1_relu	9	4	128							
Stage-3	272	conv4_block19_2_conv	9	4	128	32	3	3	128	1	1	36,864
Stage-3	273	conv4_block19_concat	9	4	864							
Stage-3	274	conv4_block20_0_bn	9	4	864							3,456
Stage-3	275	conv4_block20_0_relu	9	4	864							
Stage-3	276	conv4_block20_1_conv	9	4	864	128	1	1	864	1	0	110,592
Stage-3	277	conv4_block20_1_bn	9	4	128							512
Stage-3	278	conv4_block20_1_relu	9	4	128							
Stage-3	279	conv4_block20_2_conv	9	4	128	32	3	3	128	1	1	36,864
Stage-3	280	conv4_block20_concat	9	4	896							
Stage-3	281	conv4_block21_0_bn	9	4	896							3,584
Stage-3	282	conv4_block21_0_relu	9	4	896							
Stage-3	283	conv4_block21_1_conv	9	4	896	128	1	1	896	1	0	114,688
Stage-3	284	conv4_block21_1_bn	9	4	128							512
Stage-3	285	conv4_block21_1_relu	9	4	128							

(*table continues*)

Table F.1: DenseNet-121 Details

Dense	Layer		Input Size			Filter						Param
Stage	No.	Name	Hght	Wdth	Dpth	Cnt	Hght	Wdth	Dpth	Strd	Pad	Count
Stage-3	286	conv4_block21_2_conv	9	4	128	32	3	3	128	1	1	36,864
Stage-3	287	conv4_block21_concat	9	4	928							
Stage-3	288	conv4_block22_0_bn	9	4	928							3,712
Stage-3	289	conv4_block22_0_relu	9	4	928							
Stage-3	290	conv4_block22_1_conv	9	4	928	128	1	1	928	1	0	118,784
Stage-3	291	conv4_block22_1_bn	9	4	128							512
Stage-3	292	conv4_block22_1_relu	9	4	128							
Stage-3	293	conv4_block22_2_conv	9	4	128	32	3	3	128	1	1	36,864
Stage-3	294	conv4_block22_concat	9	4	960							
Stage-3	295	conv4_block23_0_bn	9	4	960							3,840
Stage-3	296	conv4_block23_0_relu	9	4	960							
Stage-3	297	conv4_block23_1_conv	9	4	960	128	1	1	960	1	0	122,880
Stage-3	298	conv4_block23_1_bn	9	4	128							512
Stage-3	299	conv4_block23_1_relu	9	4	128							
Stage-3	300	conv4_block23_2_conv	9	4	128	32	3	3	128	1	1	36,864

(*table continues*)

Table F.1: DenseNet-121 Details

Dense	Layer		Input Size			Filter						Param
Stage	No.	Name	Hght	Wdth	Dpth	Cnt	Hght	Wdth	Dpth	Strd	Pad	Count
Stage-3	301	conv4_block23_concat	9	4	992							
Stage-3	302	conv4_block24_0_bn	9	4	992							3,968
Stage-3	303	conv4_block24_0_relu	9	4	992							
Stage-3	304	conv4_block24_1_conv	9	4	992	128	1	1	992	1	0	126,976
Stage-3	305	conv4_block24_1_bn	9	4	128							512
Stage-3	306	conv4_block24_1_relu	9	4	128							
Stage-3	307	conv4_block24_2_conv	9	4	128	32	3	3	128	1	1	36,864
Stage-3	308	conv4_block24_concat	9	4	1,024							
Stage-3	309	pool4_bn	9	4	1,024							4,096
Stage-3	310	pool4_relu	9	4	1,024							
Stage-3	311	pool4_conv	9	4	1,024	512	1	1	1024	1	0	524,288
	312	pool4_pool	9	4	512	1	2	2	1	2	0	
Stage-4	313	conv5_block1_0_bn	4	2	512							2,048
Stage-4	314	conv5_block1_0_relu	4	2	512							
Stage-4	315	conv5_block1_1_conv	4	2	512	128	1	1	512	1	0	65,536

(*table continues*)

Table F.1: DenseNet-121 Details

Dense	Layer		Input Size			Filter						Param
Stage	No.	Name	Hght	Wdth	Dpth	Cnt	Hght	Wdth	Dpth	Strd	Pad	Count
Stage-4	316	conv5_block1_1_bn	4	2	128							512
Stage-4	317	conv5_block1_1_relu	4	2	128							
Stage-4	318	conv5_block1_2_conv	4	2	128	32	3	3	128	1	1	36,864
Stage-4	319	conv5_block1_concat	4	2	544							
Stage-4	320	conv5_block2_0_bn	4	2	544							2,176
Stage-4	321	conv5_block2_0_relu	4	2	544							
Stage-4	322	conv5_block2_1_conv	4	2	544	128	1	1	544	1	0	69,632
Stage-4	323	conv5_block2_1_bn	4	2	128							512
Stage-4	324	conv5_block2_1_relu	4	2	128							
Stage-4	325	conv5_block2_2_conv	4	2	128	32	3	3	128	1	1	36,864
Stage-4	326	conv5_block2_concat	4	2	576							
Stage-4	327	conv5_block3_0_bn	4	2	576							2,304
Stage-4	328	conv5_block3_0_relu	4	2	576							
Stage-4	329	conv5_block3_1_conv	4	2	576	128	1	1	576	1	0	73,728
Stage-4	330	conv5_block3_1_bn	4	2	128							512

(*table continues*)

Table F.1: DenseNet-121 Details

Dense	Layer		Input Size			Filter						Param
Stage	No.	Name	Hght	Wdth	Dpth	Cnt	Hght	Wdth	Dpth	Strd	Pad	Count
Stage-4	331	conv5_block3_1_relu	4	2	128							
Stage-4	332	conv5_block3_2_conv	4	2	128	32	3	3	128	1	1	36,864
Stage-4	333	conv5_block3_concat	4	2	608							
Stage-4	334	conv5_block4_0_bn	4	2	608							2,432
Stage-4	335	conv5_block4_0_relu	4	2	608							
Stage-4	336	conv5_block4_1_conv	4	2	608	128	1	1	608	1	0	77,824
Stage-4	337	conv5_block4_1_bn	4	2	128							512
Stage-4	338	conv5_block4_1_relu	4	2	128							
Stage-4	339	conv5_block4_2_conv	4	2	128	32	3	3	128	1	1	36,864
Stage-4	340	conv5_block4_concat	4	2	640							
Stage-4	341	conv5_block5_0_bn	4	2	640							2,560
Stage-4	342	conv5_block5_0_relu	4	2	640							
Stage-4	343	conv5_block5_1_conv	4	2	640	128	1	1	640	1	0	81,920
Stage-4	344	conv5_block5_1_bn	4	2	128							512
Stage-4	345	conv5_block5_1_relu	4	2	128							

(*table continues*)

Table F.1: DenseNet-121 Details

Dense	Layer		Input Size			Filter						Param
Stage	No.	Name	Hght	Wdth	Dpth	Cnt	Hght	Wdth	Dpth	Strd	Pad	Count
Stage-4	346	conv5_block5_2_conv	4	2	128	32	3	3	128	1	1	36,864
Stage-4	347	conv5_block5_concat	4	2	672							
Stage-4	348	conv5_block6_0_bn	4	2	672							2,688
Stage-4	349	conv5_block6_0_relu	4	2	672							
Stage-4	350	conv5_block6_1_conv	4	2	672	128	1	1	672	1	0	86,016
Stage-4	351	conv5_block6_1_bn	4	2	128							512
Stage-4	352	conv5_block6_1_relu	4	2	128							
Stage-4	353	conv5_block6_2_conv	4	2	128	32	3	3	128	1	1	36,864
Stage-4	354	conv5_block6_concat	4	2	704							
Stage-4	355	conv5_block7_0_bn	4	2	704							2,816
Stage-4	356	conv5_block7_0_relu	4	2	704							
Stage-4	357	conv5_block7_1_conv	4	2	704	128	1	1	704	1	0	90,112
Stage-4	358	conv5_block7_1_bn	4	2	128							512
Stage-4	359	conv5_block7_1_relu	4	2	128							
Stage-4	360	conv5_block7_2_conv	4	2	128	32	3	3	128	1	1	36,864

(*table continues*)

Table F.1: DenseNet-121 Details

Dense	Layer		Input Size			Filter						Param
Stage	No.	Name	Hght	Wdth	Dpth	Cnt	Hght	Wdth	Dpth	Strd	Pad	Count
Stage-4	361	conv5_block7_concat	4	2	736							
Stage-4	362	conv5_block8_0_bn	4	2	736							2,944
Stage-4	363	conv5_block8_0_relu	4	2	736							
Stage-4	364	conv5_block8_1_conv	4	2	736	128	1	1	736	1	0	94,208
Stage-4	365	conv5_block8_1_bn	4	2	128							512
Stage-4	366	conv5_block8_1_relu	4	2	128							
Stage-4	367	conv5_block8_2_conv	4	2	128	32	3	3	128	1	1	36,864
Stage-4	368	conv5_block8_concat	4	2	768							
Stage-4	369	conv5_block9_0_bn	4	2	768							3,072
Stage-4	370	conv5_block9_0_relu	4	2	768							
Stage-4	371	conv5_block9_1_conv	4	2	768	128	1	1	768	1	0	98,304
Stage-4	372	conv5_block9_1_bn	4	2	128							512
Stage-4	373	conv5_block9_1_relu	4	2	128							
Stage-4	374	conv5_block9_2_conv	4	2	128	32	3	3	128	1	1	36,864
Stage-4	375	conv5_block9_concat	4	2	800							

(*table continues*)

Table F.1: DenseNet-121 Details

Dense	Layer		Input Size			Filter						Param
Stage	No.	Name	Hght	Wdth	Dpth	Cnt	Hght	Wdth	Dpth	Strd	Pad	Count
Stage-4	376	conv5_block10_0_bn	4	2	800							3,200
Stage-4	377	conv5_block10_0_relu	4	2	800							
Stage-4	378	conv5_block10_1_conv	4	2	800	128	1	1	800	1	0	102,400
Stage-4	379	conv5_block10_1_bn	4	2	128							512
Stage-4	380	conv5_block10_1_relu	4	2	128							
Stage-4	381	conv5_block10_2_conv	4	2	128	32	3	3	128	1	1	36,864
Stage-4	382	conv5_block10_concat	4	2	832							
Stage-4	383	conv5_block11_0_bn	4	2	832							3,328
Stage-4	384	conv5_block11_0_relu	4	2	832							
Stage-4	385	conv5_block11_1_conv	4	2	832	128	1	1	832	1	0	106,496
Stage-4	386	conv5_block11_1_bn	4	2	128							512
Stage-4	387	conv5_block11_1_relu	4	2	128							
Stage-4	388	conv5_block11_2_conv	4	2	128	32	3	3	128	1	1	36,864
Stage-4	389	conv5_block11_concat	4	2	864							
Stage-4	390	conv5_block12_0_bn	4	2	864							3,456

(*table continues*)

Table F.1: DenseNet-121 Details

Dense	Layer		Input Size			Filter						Param
Stage	No.	Name	Hght	Wdth	Dpth	Cnt	Hght	Wdth	Dpth	Strd	Pad	Count
Stage-4	391	conv5_block12_0_relu	4	2	864							
Stage-4	392	conv5_block12_1_conv	4	2	864	128	1	1	864	1	0	110,592
Stage-4	393	conv5_block12_1_bn	4	2	128							512
Stage-4	394	conv5_block12_1_relu	4	2	128							
Stage-4	395	conv5_block12_2_conv	4	2	128	32	3	3	128	1	1	36,864
Stage-4	396	conv5_block12_concat	4	2	896							
Stage-4	397	conv5_block13_0_bn	4	2	896							3,584
Stage-4	398	conv5_block13_0_relu	4	2	896							
Stage-4	399	conv5_block13_1_conv	4	2	896	128	1	1	896	1	0	114,688
Stage-4	400	conv5_block13_1_bn	4	2	128							512
Stage-4	401	conv5_block13_1_relu	4	2	128							
Stage-4	402	conv5_block13_2_conv	4	2	128	32	3	3	128	1	1	36,864
Stage-4	403	conv5_block13_concat	4	2	928							
Stage-4	404	conv5_block14_0_bn	4	2	928							3,712
Stage-4	405	conv5_block14_0_relu	4	2	928							

(*table continues*)

Table F.1: DenseNet-121 Details

Dense	Layer		Input Size			Filter						Param
Stage	No.	Name	Hght	Wdth	Dpth	Cnt	Hght	Wdth	Dpth	Strd	Pad	Count
Stage-4	406	conv5_block14_1_conv	4	2	928	128	1	1	928	1	0	118,784
Stage-4	407	conv5_block14_1_bn	4	2	128							512
Stage-4	408	conv5_block14_1_relu	4	2	128							
Stage-4	409	conv5_block14_2_conv	4	2	128	32	3	3	128	1	1	36,864
Stage-4	410	conv5_block14_concat	4	2	960							
Stage-4	411	conv5_block15_0_bn	4	2	960							3,840
Stage-4	412	conv5_block15_0_relu	4	2	960							
Stage-4	413	conv5_block15_1_conv	4	2	960	128	1	1	960	1	0	122,880
Stage-4	414	conv5_block15_1_bn	4	2	128							512
Stage-4	415	conv5_block15_1_relu	4	2	128							
Stage-4	416	conv5_block15_2_conv	4	2	128	32	3	3	128	1	1	36,864
Stage-4	417	conv5_block15_concat	4	2	992							
Stage-4	418	conv5_block16_0_bn	4	2	992							3,968
Stage-4	419	conv5_block16_0_relu	4	2	992							
Stage-4	420	conv5_block16_1_conv	4	2	992	128	1	1	992	1	0	126,976

(*table continues*)

Table F.1: DenseNet-121 Details

Dense	Layer		Input Size			Filter						Param
Stage	No.	Name	Hght	Wdth	Dpth	Cnt	Hght	Wdth	Dpth	Strd	Pad	Count
Stage-4	421	conv5_block16_1_bn	4	2	128							512
Stage-4	422	conv5_block16_1_relu	4	2	128							
Stage-4	423	conv5_block16_2_conv	4	2	128	32	3	3	128	1	1	36,864
Stage-4	424	conv5_block16_concat	4	2	1,024							
Stage-4	425	bn	4	2	1,024							4,096
Stage-4	426	relu	4	2	1,024							
	427	flatten	4	2	1,024							
Decision	428	dropout	8,192	1	1							
Decision	429	dense	8,192	1	1	2						16,386

BIBLIOGRAPHY

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[2] A. Elhassouny and F. Smarandache, "Trends in deep convolutional neural networks architectures: a review," in *2019 International Conference of Computer Science and Renewable Energies (ICCSRE)*, 2019, pp. 1–8.

[3] X. Zhao, L. Gao, Z. Chen, B. Zhang, W. Liao, and X. Yang, "An entropy and mrf model-based cnn for large-scale landsat image classification," *IEEE Geoscience and Remote Sensing Letters*, vol. 16, no. 7, pp. 1145–1149, 2019.

[4] S. B. Avula, S. J. Badri, and G. Reddy P, "A novel forest fire detection system using fuzzy entropy optimized thresholding and stn-based cnn," in *2020 International Conference on COMmunication Systems NETworkS (COMSNETS)*, 2020, pp. 750–755.

[5] J. Y. Lee and F. Dernoncourt, "Sequential short-text classification with recurrent and convolutional neural networks," *CoRR*, vol. abs/1603.03827, 2016. [Online]. Available: http://arxiv.org/abs/1603.03827

[6] L. Deng, G. Hinton, and B. Kingsbury, "New types of deep neural network learning for speech recognition and related applications: an overview," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, May 2013, pp. 8599–8603.

[7] P. Mobadersany, S. Yousefi, M. Amgad, D. A. Gutman, J. S. Barnholtz-Sloan, J. E. Velázquez Vega, D. J. Brat, and L. A. D. Cooper, "Predicting cancer outcomes from histology and genomics using convolutional networks," *Proceedings of the National Academy of Sciences*, vol. 115, no. 13, pp. E2970–E2979, 2018. [Online]. Available: https://www.pnas.org/content/115/13/E2970

[8] F. van Wyk, Y. Wang, A. Khojandi, and N. Masoud, "Real-time sensor anomaly detection and identification in automated vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 3, pp. 1264–1276, 2020.

[9] Y. Zhang, X. Chen, L. Jin, X. Wang, and D. Guo, "Network intrusion detection: Based on deep hierarchical network and original flow data," *IEEE Access*, vol. 7, pp. 37 004–37 016, 2019.

[10] C. Liu, L. Dai, W. Cui, and T. Lin, "A byte-level cnn method to detect dns tunnels," in *2019 IEEE 38th International Performance Computing and Communications Conference (IPCCC)*, 2019, pp. 1–8.

[11] M. Abdelsalem, R. Krishnan, Y. Huang, and R. Sandu, "Malware detection in cloud infrastructure using convolutional neural networks," *IEE 11th International Conference on Cloud Computing*, 2018.

[12] Y. Hu, D. Zhang, G. Cao, and Q. Pan, "Network data analysis and anomaly detection using cnn technique for industrial control systems security," in *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, 2019, pp. 593–597.

[13] F. H. Perrin, "Methods of appraising photographic systems: Part i - historical review," *Journal of the SMPTE*, vol. 69, no. 3, pp. 151–156, 1960.

[14] L. P. Horwitz and G. L. Shelton, "Pattern recognition using autocorrelation," *Proceedings of the IRE*, vol. 49, no. 1, pp. 175–185, 1961.

[15] A. Arcese, P. Mengert, and E. Trombini, "Image detection through bipolar correlation," *IEEE Transactions on Information Theory*, vol. 16, no. 5, pp. 534–541, 1970.

[16] D. H. Hubel and T. N. Wiesel, "Receptive fields and functional architecture in two nonstriate visual areas (18 and 19) of the cat." *J Neurophysiol*, vol. 28, pp. 229–289, Mar. 1965.

[17] T. Serre, L. Wolf, and T. Poggio, "Object recognition with features inspired by visual cortex," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 2, 2005, pp. 994–1000 vol. 2.

[18] H. J. Bremermann, "Pattern recognition, functionals, and entropy," *IEEE Transactions on Biomedical Engineering*, vol. BME-15, no. 3, pp. 201–207, 1968.

[19] F. Rosenblatt, "Principles of neurodynamics. perceptrons and the theory of brain mechanisms," *American Journal of Psychology*, vol. 76, p. 705, 1963.

[20] K. Fukushima, S. Miyake, and T. Ito, "Neocognitron: A neural network model for a mechanism of visual pattern recognition," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-13, no. 5, pp. 826–834, 1983.

[21] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, Oct 1986. [Online]. Available: https://doi.org/10.1038/323533a0

[22] P. J. Werbos, "Building and understanding adaptive systems: A statistical/numerical approach to factory automation and brain research," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 17, no. 1, pp. 7–20, 1987.

[23] B. Hussain and M. Kabuka, "A novel feature recognition neural network and its application to character recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 1, pp. 98–106, 1994.

[24] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.

[25] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[26] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” 2014.

[27] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” 2015.

[28] F. Chollet, “Xception: Deep learning with depthwise separable convolutions,” 2017.

[29] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, “Mobilenets: Efficient convolutional neural networks for mobile vision applications,” 2017.

[30] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, “Densely connected convolutional networks,” 2018.

[31] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, “Grad-cam: Visual explanations from deep networks via gradient-based localization,” *International Journal of Computer Vision*, vol. 128, no. 2, pp. 336 – 359, Oct 2019. [Online]. Available: http://dx.doi.org/10.1007/s11263-019-01228-7

[32] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, “Learning deep features for discriminative localization,” 2015.

[33] D. Erhan, Y. Bengio, A. Courville, and P. Vincent, “Visualizing higher-layer features of a deep network,” *University of Montreal*, vol. 1341, no. 3, p. 1, 2009.

[34] M. T. Ribeiro, S. Singh, and C. Guestrin, “" why should i trust you?" explaining the predictions of any classifier,” in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016, pp. 1135–1144.

[35] P.-T. Jiang, C.-B. Zhang, Q. Hou, M.-M. Cheng, and Y. Wei, “Layercam: Exploring hierarchical class activation maps for localization,” *IEEE Transactions on Image Processing*, vol. 30, pp. 5875–5888, 2021.

[36] A. Chattopadhay, A. Sarkar, P. Howlader, and V. N. Balasubramanian, "Grad-cam++: Generalized gradient-based visual explanations for deep convolutional networks," *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, Mar 2018. [Online]. Available: http://dx.doi.org/10.1109/WACV.2018.00097

[37] H. Wang, Z. Wang, M. Du, F. Yang, Z. Zhang, S. Ding, P. Mardziel, and X. Hu, "Score-cam: Score-weighted visual explanations for convolutional neural networks," 2020.

[38] M. B. Muhammad and M. Yeasin, "Eigen-CAM: Class activation map using principal components," in *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, jul 2020. [Online]. Available: https://doi.org/10.1109%2Fijcnn48605.2020.9206626

[39] W. Lihao and D. Yanni, "A fault diagnosis method of tread production line based on convolutional neural network," in *2018 IEEE 9th International Conference on Software Engineering and Service Science (ICSESS)*, Nov 2018, pp. 987–990.

[40] E. Golinko, T. Sonderman, and X. Zhu, "Learning convolutional neural networks from ordered features of generic data," in *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, Dec 2018, pp. 897–900.

[41] M. Smith, J. Ingram, C. Lamb, T. Draelos, J. Doak, J. Aimone, and C. James, "Dynamic analysis of executables to detect and characterize malware," in *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, Dec 2018, pp. 16–22.

[42] S. Tobiyama, Y. Yamaguchi, H. Shimada, T. Ikuse, and T. Yagi, "Malware detection with deep neural network using process behavior," in *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, vol. 2, June 2016, pp. 577–582.

[43] A. McDole, M. Abdelsalam, M. Gupta, and S. Mittal, "Analyzing cnn based behavioural malware detection techniques on cloud iaas," in *Cloud Computing – CLOUD 2020*, Q. Zhang, Y. Wang, and L.-J. Zhang, Eds. Cham: Springer International Publishing, 2020, pp. 64–79.

[44] J. C. Kimmel, A. D. Mcdole, M. Abdelsalam, M. Gupta, and R. Sandhu, "Recurrent neural networks based online behavioural malware detection techniques for cloud infrastructure," *IEEE Access*, vol. 9, pp. 68 066–68 080, 2021.

[45] W. Jiang and L. Bruton, "Lossless color image compression using chromatic correlation," in *Proceedings DCC'99 Data Compression Conference (Cat. No. PR00096)*, 1999, pp. 533–.

[46] G. Liu and F. Zhao, "An efficient compression algorithm for hyperspectral images based on correlation coefficients adaptive three dimensional wavelet zerotree coding," in *2007 IEEE International Conference on Image Processing*, vol. 2, 2007, pp. II – 341–II – 344.

[47] Q. Wang and Y. Shen, "A jpeg2000 and nonlinear correlation measurement based method to enhance hyperspectral image compression," in *2005 IEEE Instrumentationand Measurement Technology Conference Proceedings*, vol. 3, 2005, pp. 2009–2011.

[48] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," 2014.

VITA

Randy Klepetko graduated from Texas A&M University in May of 1990 with a Bachelors in Computer Science. Since, he has enjoyed a 30 year career in a broad range of engineering and digital fields. He continued his education with courses in electrical engineering and electronics at Texas A&M University, San Antonio College, and University at San Antonio during the 1990's and 2000's, saturating his knowledge in audio and video engineering protocols, methods and techniques. In 2017 he returned to academia at the University of Texas at San Antonio to enhance his competency regarding digital security where he was encouraged to join the University's Center for Security and Privacy Enhanced Cloud Computing (C-SPECC), receiving his Masters in May 2022 and scheduled PhD completion in December 2022. His research is focused on using artificial intelligence to resolve digital security issues.