UNSW
AUSTRALIA

SCHOOL OF ELECTRICAL ENGINEERING AND TELECOMMUNICATIONS

Qualitative Assessment Metrics For Transfer Learning

by

Joel Smith

Student ID: z5076397

ELEC4951 Research Thesis C - Report

Thesis submitted as a requirement for the degree

Bachelor of Engineering (Electrical Engineering)

Submitted: November 28, 2019
Supervisor: Dr. Beena Ahmed Assessor: Dr. Elias Aboutanios

Topic Title: Qualitative Assessment Metrics For Transfer Learning

Student Name: Joel Smith **Student ID: z5076397**

A. **Problem statement**

The nature of deep learning (and by extension, transfer learning) is that of a 'black-box', where it is unclear how to assess the performance of these applications beyond a few standard, high-level metrics; the most prevalent being accuracy. As a result, this inherently limits the ability to improve the system without a more detailed qualitative perspective. There is a need to develop qualitative assessment metrics to understand the performance of transfer learning applications. This would provide further insight into potential errors within the application and areas of improvement, beyond what is perceivable by high-level metrics such as accuracy.

B. **Objective**

To identify qualitative metrics that can be used to successfully evaluate the performance of transfer learning applications at a finer-grade level than accuracy. These metrics should be able to develop insight into improving the application and qualitatively explain higher-level metrics, such as accuracy.

C. **My solution**

Create a TL model from scratch to identify potential metrics to investigate.
Take the identified metrics from the previous phase and explore their capacity for providing qualitative assessment using a more sophisticated TL system built on a high-level, robust DL Python library called *Keras*.

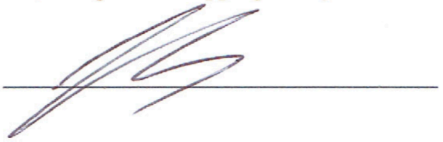
D. **Contributions** (at most one per line, most important first)

Metric 1: Neural Network Utilization
Metric 2: Activation Spectrum
Metric 3: Activation Range
Identified the existence of 'deactivated neurons' in DL and TL applications
Identified NNU trends on small-scale experiment
Developed 8 various DL models (intentionally ranging in complexity and performance)

E. **Suggestions for future work**

Create relationships between defined qualitative metrics and other high-level metrics from confusion matrix (accuracy was only confusion matrix metric explored)
Continue developing some of the unsuccessful tests, such as 'maximally activating features'
Investigate how the weights change before and after transfer and see if other qualitative metrics can be gleaned.

While I may have benefited from discussion with other people, I certify that this report is entirely my own work, except where appropriately documented acknowledgements are included.

Signature:  Date: 27 / 11 / 19

Pointers

5	Problem Statement
6	Objective

Theory (up to 5 most relevant ideas)

8	Deep Learning
13	Transfer Learning
15	Model Adaptation (using pre-trained feature extractor)
16	Existing Assessment Metrics (Confusion Matrix)
18, 20	Optimization strategies(Regularization, data augmentation)

Method of solution (up to 5 most relevant points)

25	"Investigation Phase" – developing TL model from scratch
26	Hyper-parameter sweep test
27	Transfer learning technique – Model Adaptation
29-31	"Exploration Phase" – Developing 8 models using *Keras*
35-36	Neural Network Utilization Test, Activation Spectrum Test, Activation Range Test.

Contributions (most important first)

46-48	Metric 1: Neural Network Utilization
48-50	Metric 2: Activation Spectrum
51-52	Metric 3: Activation Range
42	Identified the existence of 'deactivated neurons' in DL and TL applications
42-43	Identified NNU trends on small-scale experiment
41	Identified maximum activation trends on small-scale experiment

My work

29-32	System block diagrams/architectural diagrams
35-36	Description of assessment criteria used/metrics to be created
25-28, 29-34	Description of procedure (e.g. for experiments) (Divided into 2 phases)

Results

37-42, 45-52	Succinct presentation of results (2 phases)
41-44, 46-52	Analysis (2 phases)
42-44, 46-54	Significance of results (There is a lot of interweaving of these three 'Results' sections, not just isolated into three separate sections.)

Conclusion

55-59	Statement of whether the outcomes met the objectives
59-60	Suggestions for future research

Literature: (up to 5 most important references)

15, 29-34	[31] Sarkar, D. 2018
20-21	[33] C. Shorten and T. M. Khoshgoftaar. 2019
22-23, 28, 41	[15] A. Kensert, P. J. Harrison, 2019
18, 19, 25-26	[19] A. Ng. 2019
22-23, 54	[30] W. Samek, A. Binder, G. Montavon, K. Müller. 2016

Abstract

Advancements in machine learning and artificial intelligence continue to make new strides at an incredible pace, including the recent exploration of transfer learning; which at a basic premise promotes high-level, robust, learned features being shared across other features and tasks. This innovation allows applications with smaller datasets to produce significant results. However, the nature of deep learning (and by extension, transfer learning) is that of a 'black-box', where it is unclear how to assess the performance of these applications beyond a few standardised, high-level metrics; the most prevalent being accuracy. As a result, this inherently limits the ability to improve the system without a more detailed, qualitative perspective. This thesis develops qualitative assessment metrics to understand the performance of transfer learning applications. The three metrics developed stem from analysis at the activation level within neural networks. These metrics provide further insight into potential errors within the application and areas of improvement, beyond what is perceivable by high-level metrics such as accuracy.

Acknowledgements

I wish to thank all the people that have made contributions to the contents of this thesis, particularly, my supervisor Dr. Beena Ahmed, who has provided direction, guidance and insight throughout the entire project.

I would also like to thank the people who have supported me this year:

Julian and Juliette Smith, Jason and Abi Smith, Jordan and Millie Smith, Jenna and Dan Smith, Jemima Smith, Vanessa Long, Harry Kreicers, Oliver Mullins, Mathew Wood and Adrian Mena.

Abbreviations

ANN Artificial Neural Network

AR Activation Range

AS Activation Spectrum

CM Confusion Matrix

CNN Convolutional Neural Network

DL Deep Learning

DNN Deep Neural Network

FC Fully Connected Layers

LSTM Long Short Term Memory

ML Machine Learning

MLP Multilayer Perceptron

NLP Natural Language Processing

NNU Neural Network Utilization

ReLU Rectified Linear Unit

RNN Recurrent Neural Network

TL Transfer Learning

Contents

Chapter 1

Introduction

Advancements in machine learning and artificial intelligence continue to make new strides at an incredible pace, including the recent exploration of transfer learning (TL); which at a basic premise promotes high-level, robust, learned features being shared across other features and tasks. This innovation allows applications with smaller datasets to produce significant results.

1.1 Problem Definition

The nature of deep learning (and by extension, transfer learning) is that of a 'black-box', where it is unclear how to assess the performance of these applications beyond a few standard, high-level metrics; the most prevalent being accuracy. As a result, this inherently limits the ability to improve the system without a more detailed qualitative perspective.

There is a need to develop qualitative assessment metrics to understand the performance of transfer learning applications. This would provide further insight into potential errors within the application and areas of improvement, beyond what is perceivable by high-level metrics such as accuracy.

1.2 Thesis Objectives

The objective of this thesis is to identify qualitative metrics that can be used to successfully evaluate the performance of transfer learning applications at a finer-grade level than accuracy. These metrics should be able to develop insight into improving the application and qualitatively explain higher-level metrics, such as accuracy.

1.3 Summary Of Results

Three qualitative metrics were identified; Neural Network Utilization (NNU), Activation Spectrum (AS), and Activation Range (AR). Each of these metrics emerged from analysis at the activation level within a variety of different models.

NNU refers to the percentage of deactivated neurons within a network, alluding to the degree of utilization within the network. Results imply reducing NNU increases performance, which is the intuitive reaction to maximizing utilization of a network. Qualitatively, a model with high NNU can be improved by reassessing regularization methods and improving data quantity, such as by utilizing data augmentation techniques.

AS is a visualisation of the activations across the entire network in a frequency histogram. Results reinforce the implications of NNU, as well as discovering superior models depict right-skewed distributions. A model producing a poor AS could be improved by minimising NNU, followed by re-evaluating the activation function, weight distribution and using regularization methods such as L2.

AR is a visualisation of the maximum activations of individual layers within the network. Results imply consistent maximum activations between layers correlate to improved results. Similarly to AS, reassessing weight distribution and using regularization methods such as L2 could improve AR.

These metrics could be used as a foundational step in removing the 'black-box' nature of deep neural networks, highlighting their significance to the current research.

1.4 Thesis Outline

The thesis is divided into four chapters (excluding this chapter):

- Chapter 2 gives a greater context to the subject matter and concepts, providing the background theory to understand the problem. It contains the current literature, predominantly surrounding transfer learning applications, and highlights the lack of expression of qualitative results.
- Chapter 3 provides the methodology behind developing potential qualitative metrics. The solution was defined through two phases; an *investigation phase* and an *exploration phase*. The first phase investigates potential metrics using a small-scale TL system developed from scratch. The second phase takes the identified metrics from the previous phase and explores their capacity for providing qualitative assessment using a more sophisticated TL system built on a high-level, robust DL Python library called *Keras*.
- Chapter 4 evaluates and discusses the results found in chapter 3. This chapter provides the implications of the metrics to the broader research, particularly highlighting the qualitative nature of the metrics that help provide insight into improving TL systems.
- Chapter 5 provides conclusions to the thesis; what has and has not been achieved. It also highlights where future research could be conducted.

Chapter 2

Background

2.1 Background Theory

2.1.1 Deep Learning

Deep learning (DL) is defined as a class of machine learning (ML) techniques that exploit many layers of non-linear information processing for supervised or unsupervised feature extraction and transformation, and pattern analysis and classification. The main differential of DL to traditional ML is the multiple levels (or layers) of representation and abstraction, which translates to superior performance as data quantity increases (see figure 2.1). This advent has begun impacting a wide range of research domains since 2006 [6].

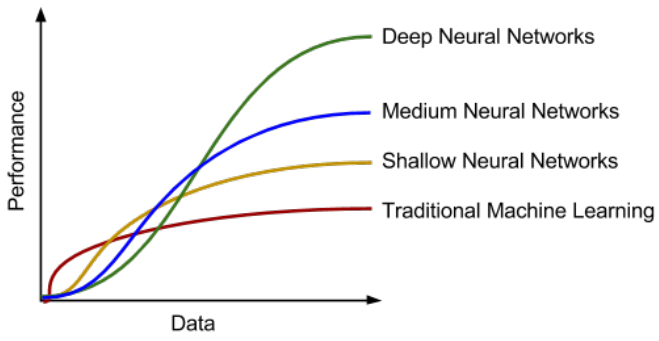


Figure 2.1: Advantage of DL over ML and NN [3].

Deep Neural Networks

A deep neural network (DNN) is the archetypal DL framework that is inspired by the neural networks of the human brain to learn, recognise and classify patterns. They are modelled as multilayer perceptrons (MLP) or "fully connected layers" (FCs), containing an input layer, several hidden layers, and an output layer (see figure 2.2). They are feed-forward systems, where the units in one layer are connected unidirectionally to the following layers to perform non-linear transformations to the input data [26]. A DNN's performance is traditionally assessed on its accuracy of classification, generally on a test dataset not used in training.

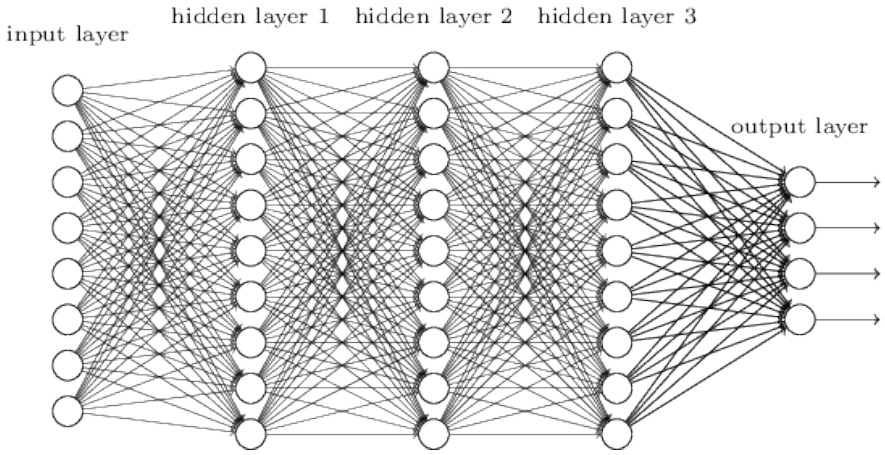


Figure 2.2: MLP model of a 4-layer DNN [20].

Modelling An L-layer DNN

In an L-layer DNN, each hidden unit, j, in layer, l, has an activation function (see activation functions), such as the sigmoid or logistic function seen in equation 2.1, to map its total input, x_j, from the layer below, $l+1$, to the scalar state, y_j. It is then connected to all the hidden units of the layer above, $l-1$, to repeat the activation process.

$$y_i = sigmoid(x_j) = \frac{1}{1+e^{-x_j}}, \qquad x_j = b_j + \sum_i y_i w_i \tag{2.1}$$

where b_j is the bias of unit j, i is an index over units in the layer below, and w_{ij} is the weight on a connection to unit j from unit i in the layer below. [12]

Classification occurs at the output layer, L, by calculating the class probability, p_j, using the 'softmax' non-linearity:

$$p_j = \frac{e^{x_j}}{\sum_k e^{x_k}} = softmax_s(x_j) \tag{2.2}$$

where k is an index over all classes.

The DNN can then be discriminatively trained by calculating the discrepancy between the target outputs and the actual outputs produced for each training case and make slight adjustments within the system's parameters as it backpropagates through the layers; accomplished by using a cost function between the desired probabilities, d_j, and the outputs of the softmax, p_j, seen in equation 2.3.

$$C = -\sum_j d_j \log p_j \tag{2.3}$$

Convolutional Neural Networks

The Convolutional Neural Network (CNN) is the predominant DL architecture utilized in the current research, as it has found particular success in image recognition [10, 15, 23, 41], and recently other domains, such as natural language processing (NLP) [9, 39] and speech [1]. The premise of a CNN is the extraction of convolved features, or 'feature maps', which are created by passing a filter matrix or 'kernel' over the input and producing a dot product of the resultant multiplications (see figure 2.3).

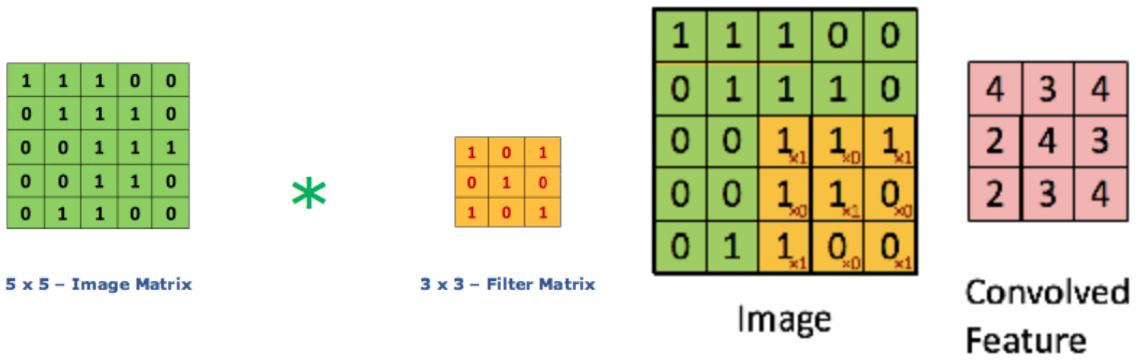


Figure 2.3: CNN feature map convolution [37].

There are additional variables as to how this feature map is created by the convolution of the filter matrix over the input matrix. **Stride** dictates how many pixels the filter matrix slides over the input matrix after each multiplication. **Padding** allows the input matrix to be padded with zeroes around the border to apply filters to the bordering elements (which is called *wide convolution*). **Depth** corresponds to the number of filters used for the convolution operation.

After each convolution operation, the feature maps are passed through an activation function, usually a Rectified Linear Unit (ReLU) function (see activation functions). This parallels the activation of the neurons in the DNN through the sigmoid function in equation 2.1.

Pooling (or subsampling in figure 2.4) is used after convolutional layers to reduce the dimensionality of each feature map while retaining the important information. Pooling summarises the feature map data, depending on the type of pooling desired, such as max, average [41], sum etc. In the case of *Max Pooling*, which is the most common [23, 31, 39], a spatial neighbourhood is defined (such as a 2 x 2 window) and the largest element of that window is maintained in the reduced matrix. Different pooling techniques have different effects but are generally used to reduce the number of parameters and computations, therefore controlling overfitting.

Finally, the last convolution layer feeds into a traditional MLP or FC layer (or layers), followed by a softmax function, similar to 2.1.1. These are added to the end of CNNs to utilise their functionality for classification [10, 31]. Figure 2.4 summarises the typical CNN structure.

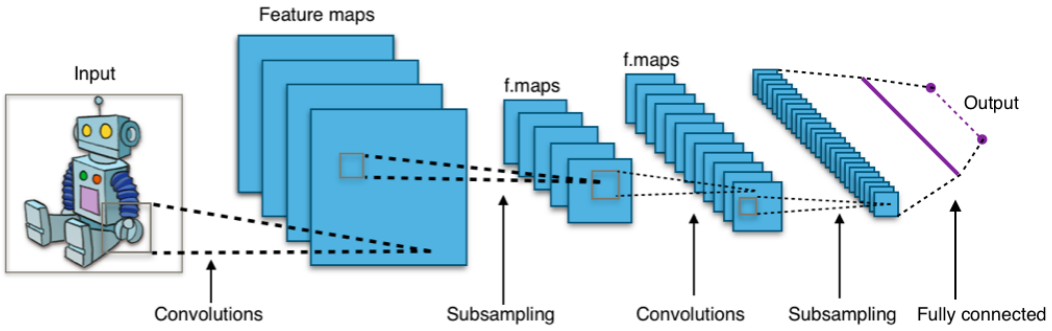
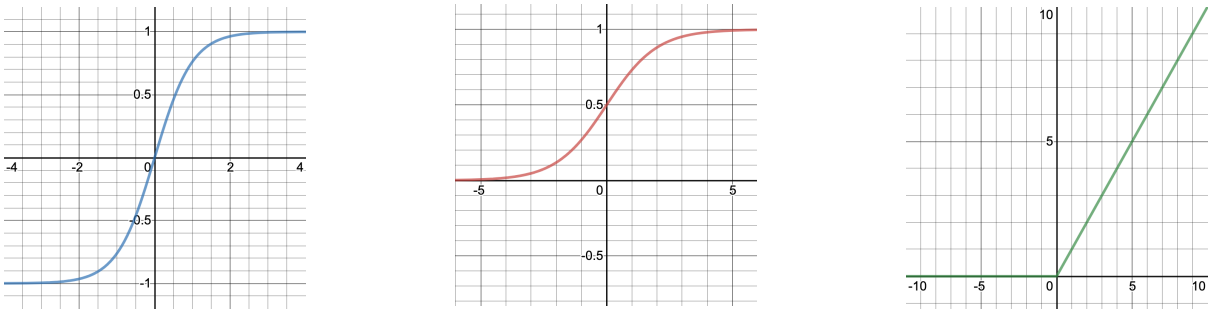


Figure 2.4: Typical CNN model structure [31].

Activation Functions

The three main activation functions that are commonly used are tanh, sigmoid and ReLU (see figure 2.5).



(a) Tanh Activation
$y = tanh(x)$

(b) Sigmoid Activation
$y = \frac{1}{1+e^{-x_j}}$

(c) ReLU Activation
$y = max(0, x)$

Figure 2.5: Three main activation functions.

The ReLU is the most commonly used activation function for intermediate activations as it is the least computationally expensive [7, 31, 33, 41]. Sigmoids tend to be used at the output layer for classification. Tanh is seldom used.

Recurrent Neural Networks

There are other various architectures beside DNNs and CNNs, including Recurrent Neural Networks (RNNs). RNNs use a sequence of inputs by looping internally rather than using a feed-forward structure (see figure 2.6). For this reason, they are useful for complex data that is formed in a sequence of inputs, such as speech or text. A well-known and high-performing RNN is the Long Short Term Memory (LSTM) network, which is an RNN which specialises in learning long-term dependencies [22].

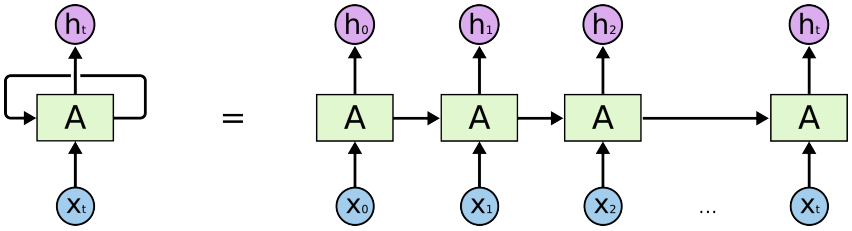


Figure 2.6: An 'unrolled' cycle within an RNN [22].

2.1.2 Transfer Learning

Definition Of Transfer Learning

Transfer learning (TL) is a machine learning technique which Andrew Ng in his NIPS16 tutorial remarked is 'a future key driver of machine learning success' (see figure 2.7)[18]. Transfer learning involves all methods that utilize any auxiliary resources (data, model, labels, etc.) to enhance model learning for the target task [38]. This can be understood by a human's ability to retain and reuse previously learned knowledge in related, but different tasks, such as a classical pianist learning jazz piano [2, 31].

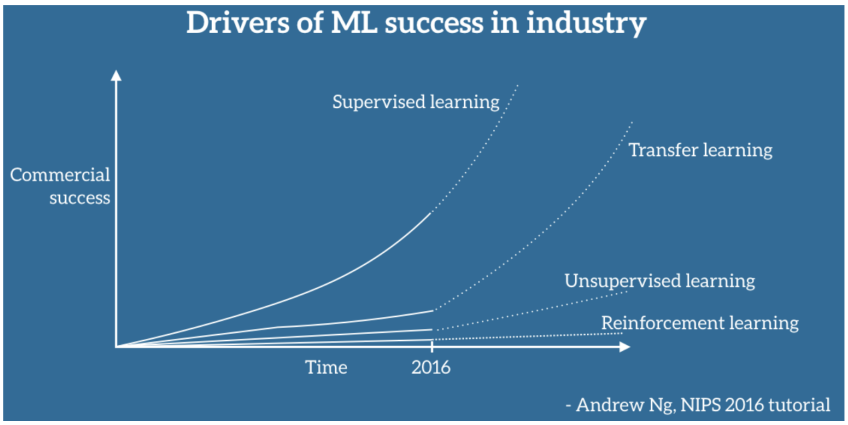


Figure 2.7: Drivers of ML industrial success according to Andrew Ng [28].

Transfer Learning Performance Over Traditional Machine Learning

Assessing the effectiveness of transfer learning performance over traditional machine learning techniques is generally summarized in three high-level ways [32]:

1. higher start - indicating improved performance at the initial points.
2. higher slope - showing more rapid growth of performance.
3. higher asymptote - leading to an improved final performance.

As shown in figure 2.8, these above reasons are indicative of the advantage of new tasks relying on previously learned tasks, instead of in isolation. Traditional ML uses single-task learning, where knowledge is not retained or accumulated, whereas the knowledge accumulated in TL allows the learning process to be faster, more accurate and needing less training data to achieve success.

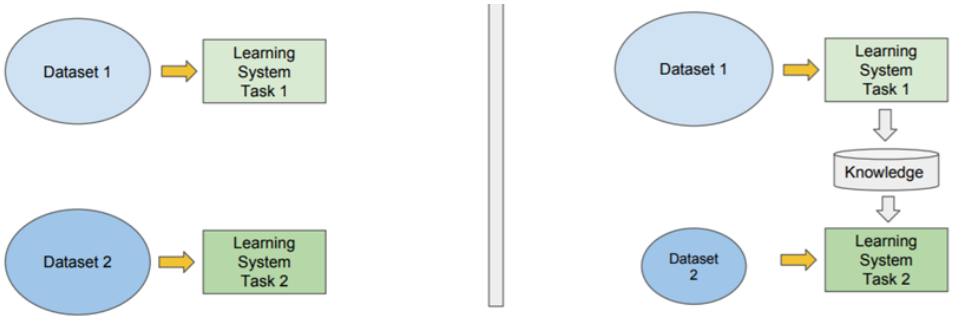


Figure 2.8: Traditional ML (left) vs. Transfer Learning (right) [31].

Relationship of Transfer Learning And Deep Learning

The paradigmatic shift brought on by deep learning changed everything, including transfer learning. A major component of this impact, as mentioned in subsection 2.1.1, was the strong modelling power via a multi-level hierarchy. This provided several advantages, including the ability to learn high-level features, hierarchical parameter sharing, simple feature learning without any labelled data, and learned models can be well adapted to specific tasks with little supervised training.

For these reasons, deep learning provides a graceful framework for transfer learning, launching off the major research direction depicted in the NIPS95 workshop, "Learning to Learn" [4, 38], with high-level, robust, previously-learned features being shared across other features and tasks.

Transfer learning methods

Many different methods emerged with transfer learning and can be differentiated into four categories of applications, where *data* (or feature domain) and *task* are two conditional factors:

	Same Tasks	**Different Tasks**
Same Domains	Model adaptation, incremental learning, model transfer	Multitask learning
Different Domains	Co-training, heterogeneous transfer learning, generalized distillation	Analogy learning

Table 2.1: Categories of transfer learning methods.

Table 2.1 refers to several transfer learning methods, but is not exhaustive; there are a large number of methods devised by many authors [7, 10, 29, 40]. Appropriate methods are expressions of transfer learning being applied under specific contexts, hence these methods are driven by the previously mentioned conditional factors of *data* and *task*. To illustrate these differences, two different transfer learning methods will be explored below.

Model Adaptation This is considered the simplest and most common method, where the source and target share the same domains, but the model is adapted to meet a change of *data* distribution. One example is a DNN model completely trained with data from a high resource, such as adult English speech, and has its last layer adapted to the target condition or data with significantly fewer resources, such as dysarthric speech [35]; domains share enough feature similarities to provide substantial results using model adaptation. With the internal weights regularized by the high resource learning, this information can be retained and reused powerfully for the low-resource domain.

Figure 2.9 shows this technique practically with the higher resource domain network being used as a pre-trained feature extractor for the lower resource domain network. The layers of the pre-trained feature extractor are either frozen or finely-tuned when training with the target data. Finely-tuned layers tend to produce better performing results [10, 15].

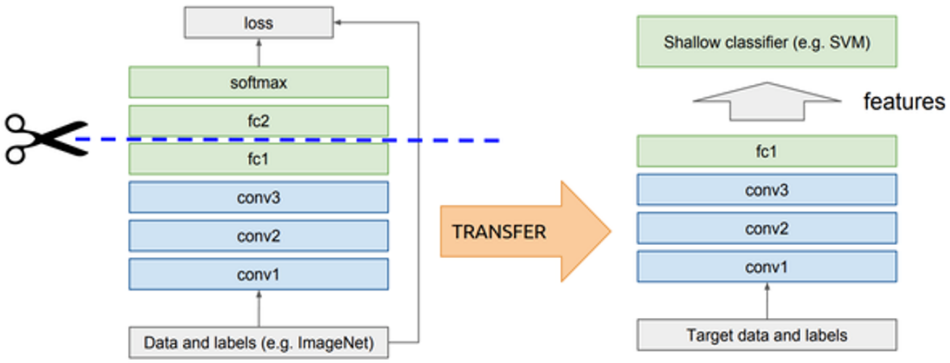


Figure 2.9: Model adaptation shown using a pre-trained model as a feature extractor, adapted into another domain with a shallow classifier [31].

Multitask Learning *Multitask Learning* utilises the similar feature spaces of the source and target domains, but where the *task* labels are significantly different. This is a popular method for cross-lingual applications [8, 11, 13]. The target and source share learning across a single level of the network; sharing feature extraction, sharing hidden layers, but having the softmax layer task-dependent (see figure 2.10) [2].

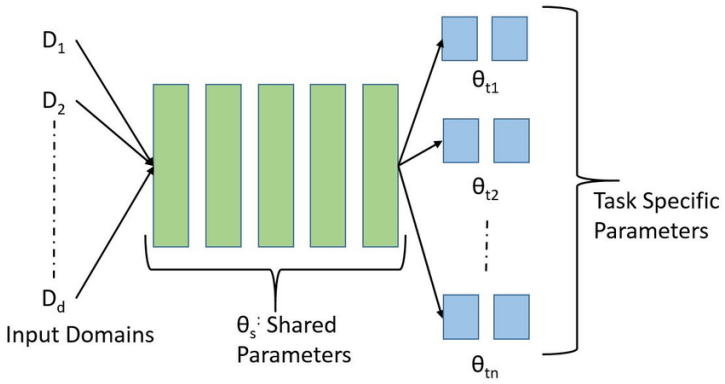


Figure 2.10: Example model of multitask learning [24].

2.2 Assessing Performance Of DL And TL Applications

2.2.1 Existing Assessment Metrics

Confusion Matrix Metrics

The majority of all standardised metrics used to assess both DL and TL applications stem from the Confusion Matrix (see table 2.2).

	Positive Predicted	**Negative Predicted**
Positive Actual	True Positive (TP)	False Negative (FN)
Negative Actual	False Positive (FP)	True Negative (TN)

Table 2.2: Confusion Matrix

From these values, several standardised, high-level metrics can be calculated.

Accuracy is the most common metric used in all applications for assessing performance. It is very high-level and provides little insight as to where the issues lie within systems.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.4}$$

Recall is the ratio of correctly classified positive examples to the total positive examples. High recall indicates the class is correctly recognized.

$$Recall = \frac{TP}{TP + FN} \tag{2.5}$$

Precision is the ratio of correctly classified positive examples to the total positive predictions.

$$Precision = \frac{TP}{TP + FP} \tag{2.6}$$

F1 Score conveys the balance between precision and recall.

$$F1\ Score = \frac{2.Precision.Recall}{Precision + Recall} \tag{2.7}$$

These metrics and more (Cohen's Kappa, misclassification rate, null error rate etc.) all use various combinations of the confusion matrix to create high-level insight into a system's classification ability. However, none of them provides a substantial qualitative understanding of how the system reached these values.

The majority of the literature compares a TL application's performance to a traditional DL baseline model to show performance improvement, as accuracy typically will improve. There is a need for additional metrics to provide finer-grade insight into the performance of TL applications beyond what is available from these high-level metrics. This is outlined in section 2.3.

2.2.2 Optimizing Performance Strategies

Regularization

Regularization provides additional efficiency and superior results, particularly as it reduces overfitting. Two common regularization techniques are L2 regularization and dropout.

L2 relies on the assumption that a network is simpler with smaller weights. Thus, it modifies the cost function by adding an additional term, penalizing the square values of weights, driving weights to be smaller [16, 19]. This is shown in equation 2.8 below.

$$J_{regularized} = -\frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\sum_{l=1}^{L}||w^{[l]}||_F^2 \tag{2.8}$$

Figure 2.11: L2 regularization where a second term (with λ) is added to the cost function [19].

Dropout will randomly eliminate neurons at a specified probability, setting that neuron to 0 for that iteration, allowing the network to operate on a subset of itself during training (see figure 2.12). This promotes the network to not rely too heavily on any individual neuron. This is a common method employed to improve performance by reducing overfitting (see figure 2.13) in related works [21, 33, 41]).

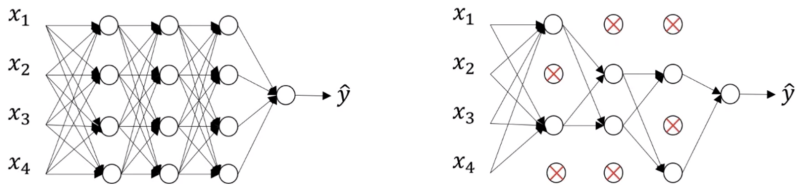


Figure 2.12: Dropout regularization visualization [19].

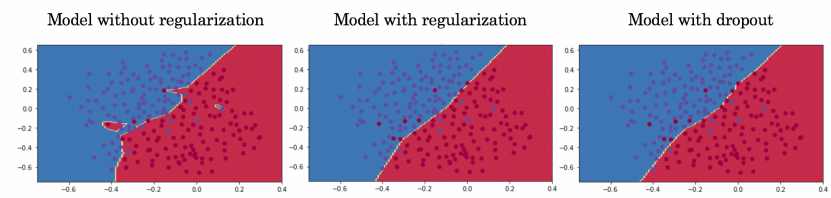


Figure 2.13: Difference between no regularization (left), L2 regularization (centre), and dropout (right) in reducing overfitting [19].

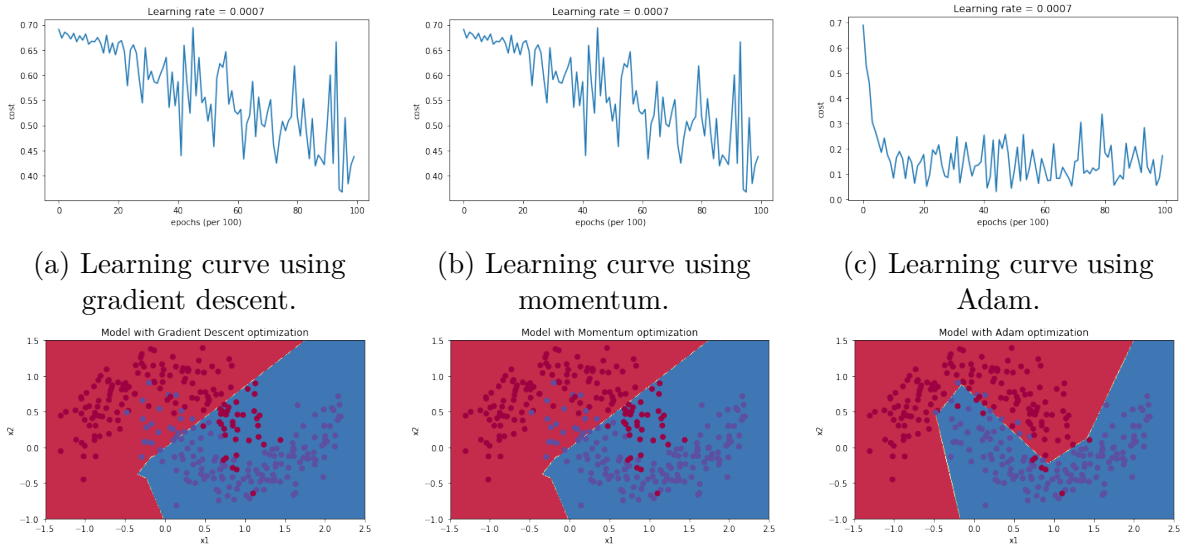
Optimization Algorithms

Optimization algorithms are used to minimize the cost or loss produced by the cost function (see equation 2.3 for an example cost function) that will lead to optimal solutions faster and produce better-performing systems (see figure 2.14). The three common optimization algorithms are gradient descent with momentum, root mean square propagation (RMSprop), and the Adam optimization algorithm.

Gradient descent with momentum computes an exponentially weighted average of the gradients which is then used to adjust weights. This, in turn, produces more direct steps towards the local minima as vertical oscillations are reduced by using exponentially weighted averages [7].

RMSprop utilizes a similar idea as momentum but uses division by the root mean square of the derivatives, in hopes to maximise learning speed in the horizontal plane and minimise vertical oscillations [31].

Adam combines both momentum and RMSprop to obtain the benefits of both. The Adam algorithm is most commonly used [23, 31, 33].



(a) Learning curve using gradient descent.

(b) Learning curve using momentum.

(c) Learning curve using Adam.

(d) Classification using gradient descent.

(e) Classification using momentum.

(f) Classification using Adam.

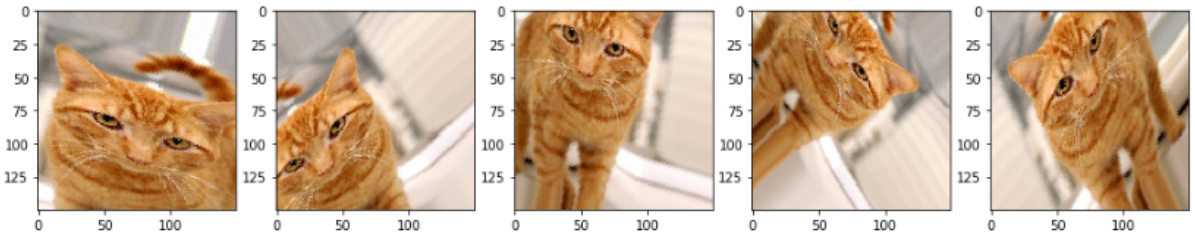
Figure 2.14: Comparison of optimization algorithms [19].

Data Augmentation

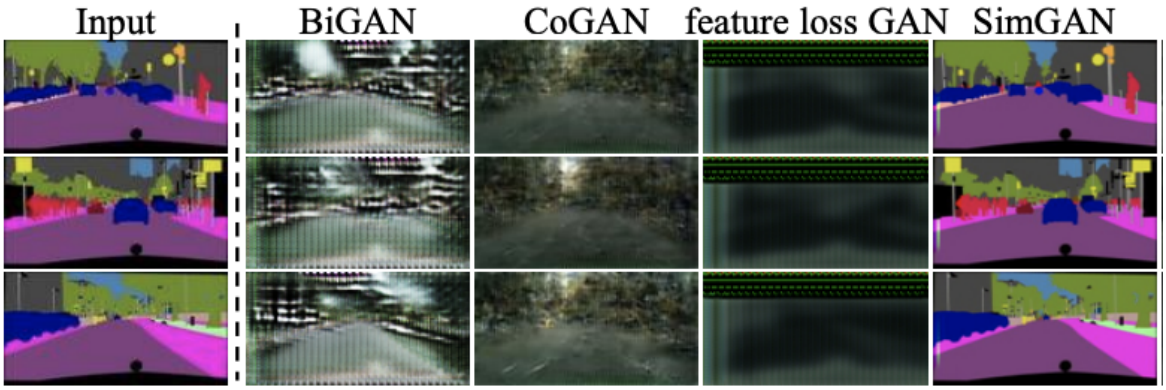
Data augmentation is a technique used to create additional, varied data samples from an existing dataset. This is a vital strategy for improving application performance for low resource systems, as these systems have difficulty generalising validation and test sets. The main techniques fall under the two categories of *data warping* and *oversampling*.

Data warping directly augments the input data to the model in the data space while maintaining the target label [33]. Many applications incorporate warping with geometric and colour transformations (see figure 2.15a), random erasing, adversarial training, and neural style transfer [15, 23, 41].

Oversampling techniques create synthetic instances and add them to the training set. This includes mixing images, feature space augmentations, and generative adversarial networks (see figure 2.15b) [23, 42].



(a) Data warping example used in section 3.2 utilizing geometric transformations.



(b) Oversampling example using generative adversarial networks (GANs) [42].

Figure 2.15: Data augmentation examples.

The two categories do not form a mutually exclusive dichotomy and are represented in the taxonomy in figure 2.16.

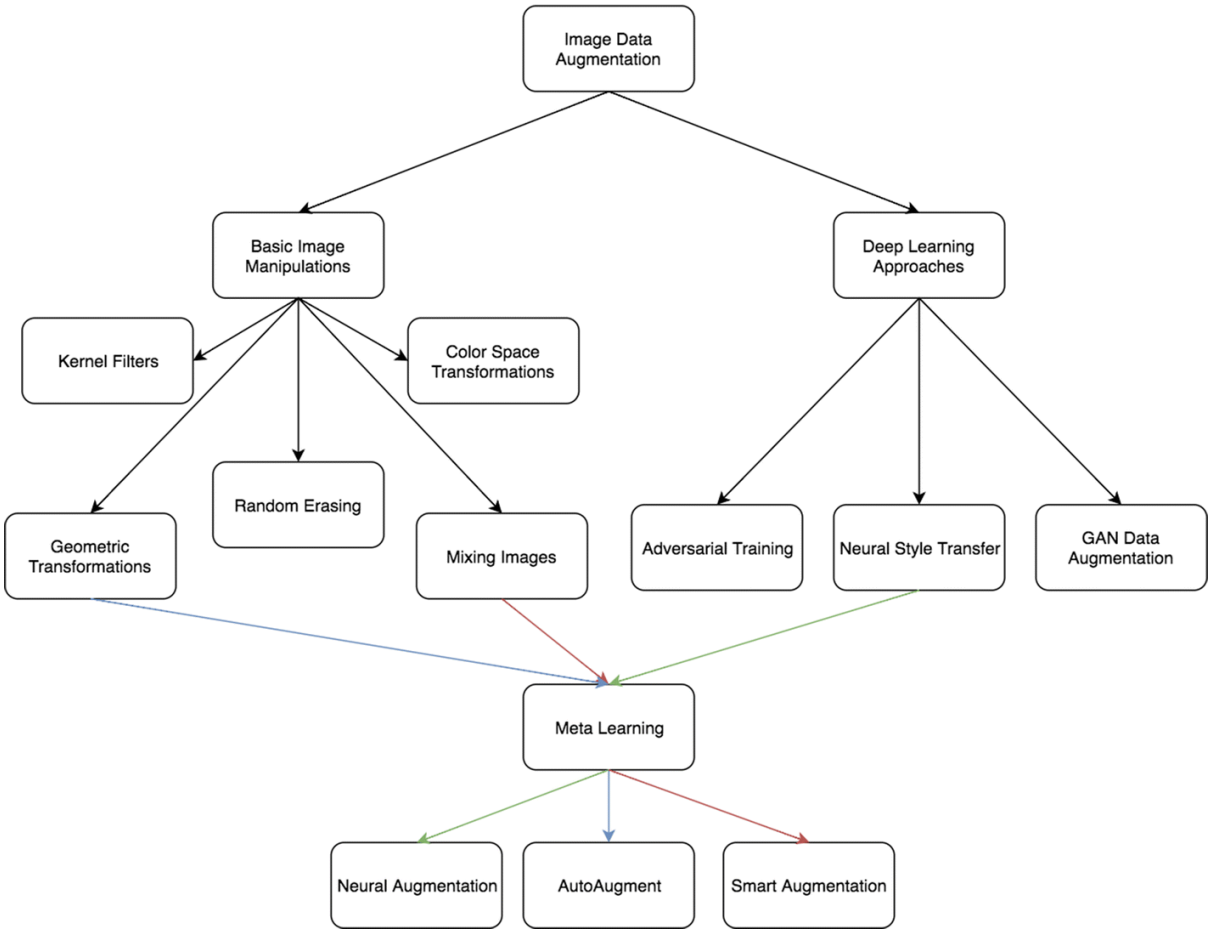


Figure 2.16: Taxonomy of image data augmentation techniques [33].

Data Quality And Quantity

The reason TL and data augmentation are successful in improving the performance of applications is the intuitive understanding of increasing data quantity generally increases the performance of DL applications. However, the quality of data is an important contributor to an application's ability to generalize in its classification. This is perhaps the greatest challenge in *Big Data* and DL [17]. Therefore, by building more substantial and higher-quality datasets, applications can be further optimized.

2.3 Knowledge Gap In Literature

As mentioned in the introduction, the large majority of related works assess performance by high-level metrics; almost always accuracy [10, 23, 41]. Since successful classification is the main objective with many applications, accuracy is the most logical metric to use when assessing if the application has achieved its purpose. However, this lacks the qualitative insight into the reasons behind an application's success (or failures), providing little direction of how to improve performance (see figure 2.17).

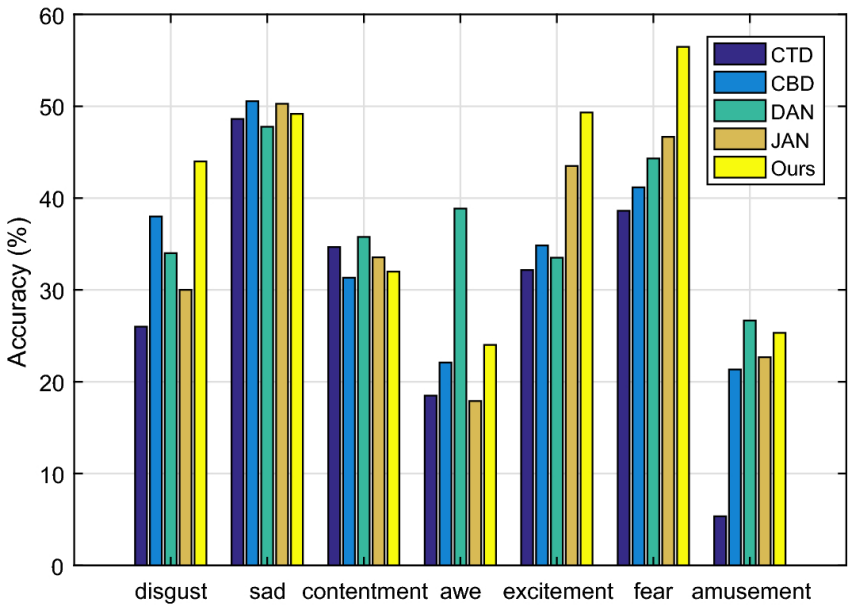


Figure 2.17: Typical example of a TL application using a comparison of accuracies between competing models to assess performance in classifying emotion[10].

Many papers comment on the 'black-box' nature of neural networks and some postulate ways of understanding the internals qualitatively by visualizing the internal activations (see figures 2.18 and 2.19) [15, 30]. These methods tend to reach conclusions that earlier layers show patterns of feature recognition whereas deeper layers tended to present no clear indications of trends. However, these methods do not develop specific metrics that can be employed to assess these relationships.

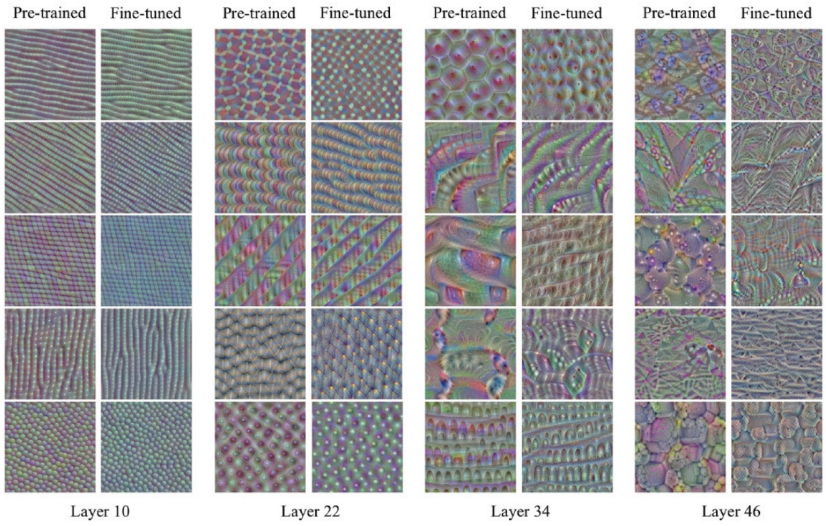


Figure 2.18: Example of seeking qualitative understanding of 'black-box' neural network by seeing maximized activations of internal layers [15].

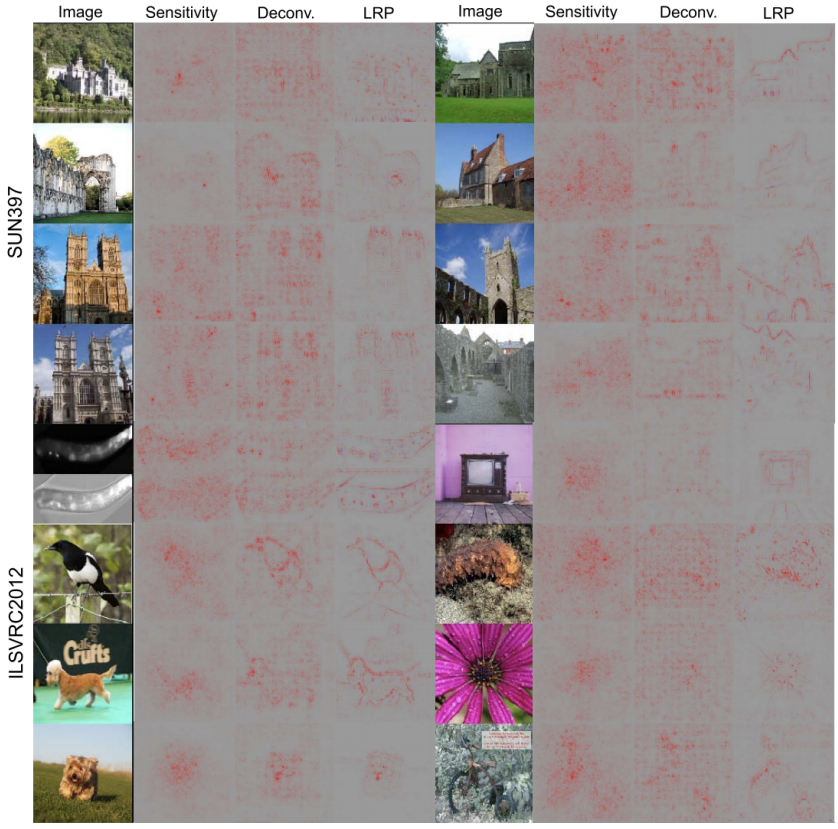


Figure 2.19: Example of seeking qualitative understanding of 'black-box' neural network by using heatmaps [30].

2.4 A Brief Discussion On Changes To Thesis

It should be noted that the thesis' problem statement, objectives, and therefore, methodologies have all shifted since trimester one. In trimester one, a similar goal was established in identifying a qualitative understanding of TL, however, under the specific domain of Automatic Speech Recognition (ASR) (see figure 2.20). The specific metric that was focused on heavily was '%WER' or Word Error Rate, which is a specific form of accuracy within ASR applications.

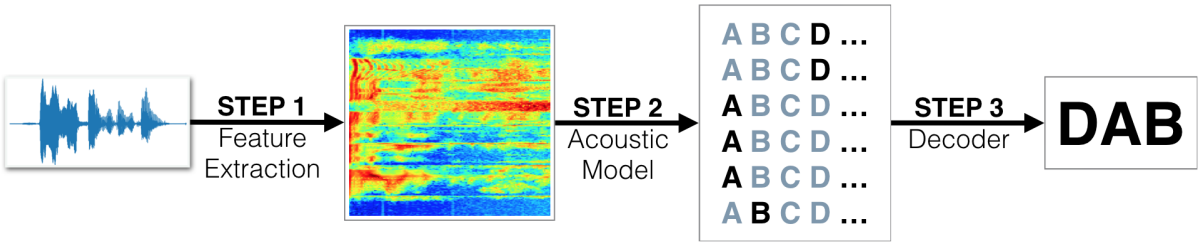


Figure 2.20: High-level view of ASR applications [36].

However, since trimester one, the thesis has developed in a more generalised direction of identifying qualitative metrics that could be applied to any domain. In addition to this, the various experiments of developing ASR models were beginning to be too time-consuming and beyond the scope of the thesis, as it drew away from the focus of qualitative assessment. Hence, the thesis has evolved away from ASR.

Chapter 3

Identified Metrics

The qualitative assessment metrics for transfer learning were developed in two distinct phases; investigation and exploration.

1. The *investigation phase* involved developing a TL model from scratch to attempt to identify potential metrics to investigate.

2. The *exploration phase* involved taking the potential metrics identified in the investigation phase and evaluating them within a much more robust, extensive model. This was accomplished by developing a model using *Keras*, which is a high-level DL Python library.

This chapter will illustrate the problems to be solved in each particular phase and the methodology to solve these problems.

3.1 Investigation Phase

3.1.1 Motivation

Problem: A working model is required

Potential metrics can only be defined if a working model exists. Hence the first problem to be addressed in this phase is to create a working DL application. After creating this, a TL application will then need to be created. Finally, after a working TL application exists, potential metrics could hypothetically be extracted via extensive experimentation.

Solution: create a working TL model from scratch

For this phase, it is significantly more advantageous to build a working TL model from scratch, as this allows more manipulability of the model, which is crucial for this phase of attempting to understand the 'black-box' nature. Also, it reinforces the theoretical principles behind DNNs by creating the methods from scratch (forward and backpropagation, activation functions etc.), providing insight into which areas to investigate.

3.1.2 Method

Architecture

The methodology behind this phase was to create a simple model to experiment with. Hence, more complex modelling, such as CNN, RNN and LSTM networks were all ignored for the sake of simplicity. Instead, a 4-layer DNN was implemented, with ReLU activation for all layers except for the output which uses a sigmoid activation (see figure 3.1).

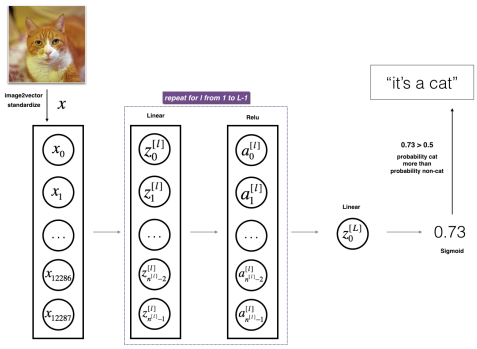


Figure 3.1: Simple DNN architecture for the *investigation phase* [19].

Baseline Hyper-Parameters

Hyper-Parameter	Value
Input Layer	64 by 64 pixel RGB image becomes $64*64*3 = 12288$ neurons
Hidden Layers	Three hidden layers with neurons 20, 7, 5 respectively
Output Layer	1 neuron
Learning Rate	0.0075
Cost Function	Cross Entropy Criterion
Activation Function	Rectified Linear (Sigmoid Output)

Table 3.1: Baseline hyper-parameters for both cat and dog classifiers. Baseline parameter choices were modelled from Ng's models [19].

Datasets

This model is a unary image classifier which will be used with a 'high-resource' dataset and a 'low-resource' dataset to perform model adaptation. Ng's *cat vs. non-cat* dataset was used as the 'high-resource' domain for the feature extractor, and a *dog vs. non-dog* dataset that was self-curated from *Kaggle*'s *Natural Images* dataset was used as the 'low-resource' domain (see figure 3.2)[27].

Cat vs. non-cat is a curation of cats as the primary class and other objects as the anti-class. There are 209 images in the training set, and 50 images the in testing set. *Dog vs. non-dog* is identical to the cat dataset, but with a dog as the primary class and much fewer resources with 62 images in the training set, and 30 images in the testing set.

Small datasets were utilised to reduce computational costs as a tradeoff to classification accuracy, as high accuracy was not a priority of this phase of the thesis. Image classification was chosen as the domain space as there are substantial resources for development and many examples of successful TL in the literature [10, 15, 23, 31, 41].

Figure 3.2: Three samples from dog dataset.

Hyper-Parameter Sweep Tests

Four hyperparameter sweep tests were developed to assess models under structural variation. These sweeps were done in anticipation that trends would reveal themselves that would ideally forge the potential qualitative metrics. The first sweep test is a learning rate sweep of 50 rates in the range of $[10^{-4}, 1)$ across a logarithmic distribution (as best results tend to emerge exponentially towards zero). The other three sweep tests were for the three hidden layers, sweeping across a variable number of neurons per layer, with each sweep ranging from 1 to 50 neurons. In total, a model would be trained on 200 different sets of hyper-parameters and produced 200 sets of results per dataset.

Transfer Learning Technique

Model adaptation is used to take the cat classifier (domain with more resources) as the feature extractor and transferred to a dog classifier (domain with fewer resources) as shown in figure 3.3; see section chapter 2 for theory. The significant similarity between dogs and cats provides a substantial problem space to explore transfer learning using this method.

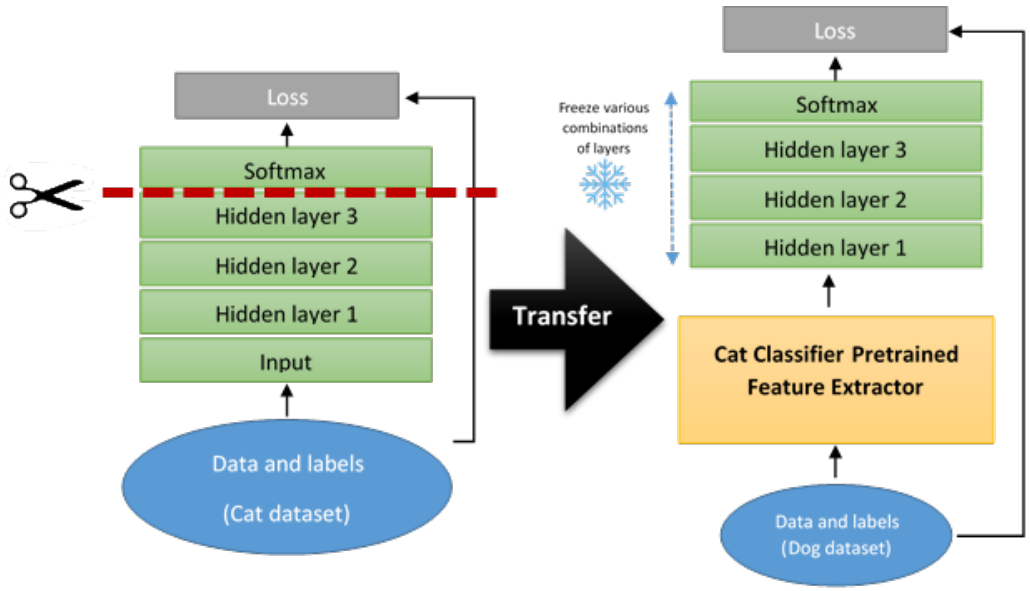


Figure 3.3: TL method for the *investigation phase*.

The cat classifier is trained to maximum accuracy using the baseline hyperparameters in table 3.1. The output layer is removed and fed into the first hidden layer of the dog classifier, which uses the same baseline parameters in table 3.1, effectively achieving model adaptation. This model is then fed through the hyper-parameter sweep tests explained previously, allowing qualitative comparisons between sweep results of different models; particularly the baseline cat, baseline dog, and TL model.

The final variation to the model adaptation are various permutations of layers are frozen and then run through the sweep tests to see how freezing particular layers of the target domain (dog classifier) responded to the sweep tests.

Specific Investigative Metric Tests

In addition to the sweep tests for investigation, the following are specific tests developed for drawing out potential metrics.

The first specific test is maximum activation, which determines the input image that produces the maximum activation for any given neuron. This test was developed to show a correlation between some neurons being considered 'cat-neurons' and others being 'non-cat neurons', similar to work done by Kensert et al. [15].

Another test was to analyse particular neuron's activation variations across training in response to specific inputs. This effectively 'reverse engineers' the DNN as the internals are being analysed in response to a known output. This attempted to find qualitative metrics from activation trends.

3.2 Exploration Phase

3.2.1 Motivation

Problem: Potential metrics need to be solidified

The results of the investigation phase developed several potential metrics to explore, but the nature of the experiment produced results that were inconclusive due to the small-scale experimentation.

Solution: Create a large-scale experiment

Hence, by leveraging high-level DL Python libraries, such as *Keras*, highly optimized and powerful TL models can be developed and used with large datasets. Using these models, the identified potential metrics can be substantially explored, thereby understanding their implications for improving applications. The basis for this experiment is found in Sarkar's work on TL [31].

3.2.2 Method

Macro-structure Models Architecture Overview

Five models are created, each increasing with complexity from the previous to ideally increase in performance (i.e. accuracy). Table 3.2 outlines the high-level description of each model and figures 3.4 and 3.5 show the architectural schematics. The specific structural differences of each are outlined in greater detail in 'Model Parameters'. The five models are referred to as 'macro-structure models' or 'macro-models' as each model has a significant structural difference to each predecessor (denoted as models 1, 2, 3, 4 and 5). The models were based off Sarkar's work [31].

Model 1	Simple CNN model, with three convolutional layers and max-pooling for automatic-extraction of features from images. This model has no regularization. See figure 3.4 for the overall architecture.
Model 2	The same simple CNN model as model 1, but model utilises image augmentation via geometric transformations to reduce overfitting. This model also uses regularization via dropout. See chapter 2 for more details on image augmentation and regularization. See figure 3.4 for the overall architecture.
Model 3	This model uses TL by leveraging a pre-trained CNN model as a feature extractor into a shallow fully-connected classifier (see model adaptation). The pre-trained model is outlined in the VGG-16 section below. All layers within the pre-trained model are frozen. This model uses dropout. See figure 3.5 for the overall architecture.
Model 4	The same TL model as model 3, but this model utilises image augmentation. See figure 3.5 for the overall architecture.
Model 5	The same TL model as model 4, but this model finely-tunes the weights of the feature-extractor instead of freezing them. See figure 3.5 for the overall architecture.

Table 3.2: Macro-structure models which increase in complexity.

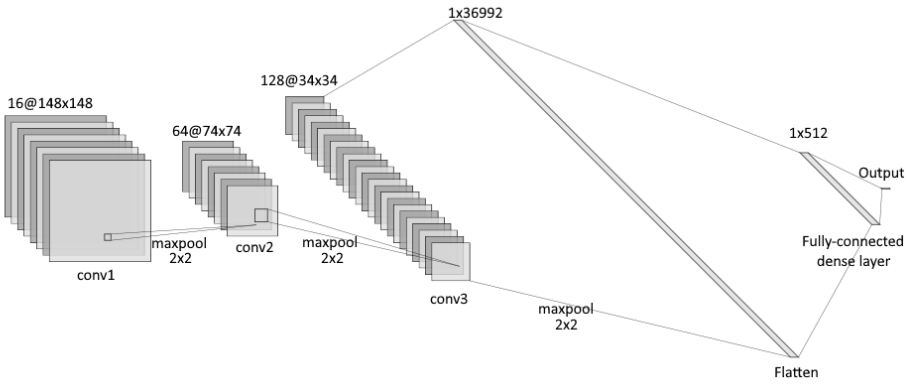


Figure 3.4: Models 1 and 2 architecture schematic.

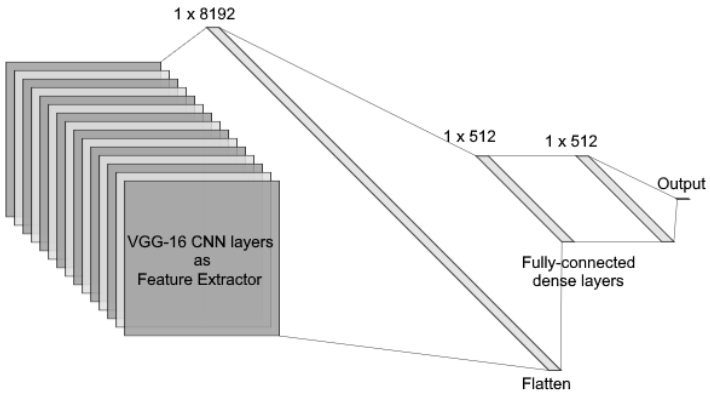


Figure 3.5: Models 3, 4 and 5 architecture schematic.

Micro-structure Models Architecture Overview

Upon creating these five models with macro-structural changes, the best of these models, model 5, was taken and altered to create micro-structural changes (denoted as models 5a, 5b, 5c). Table 3.3 outlines the high-level descriptions of the micro-structural changes and figures 3.6 and 3.7 show the architectural schematics.

Model 5a	This model altered the number of neurons per layer in the shallow FC classifier. See figure 3.6 for the overall architecture.
Model 5b	This model added an additional FC layer to the shallow FC classifier. See figure 3.7 for the overall architecture.
Model 5c	This model altered the learning rate of training. See figure 3.5 for the overall architecture.

Table 3.3: Micro-structure models.

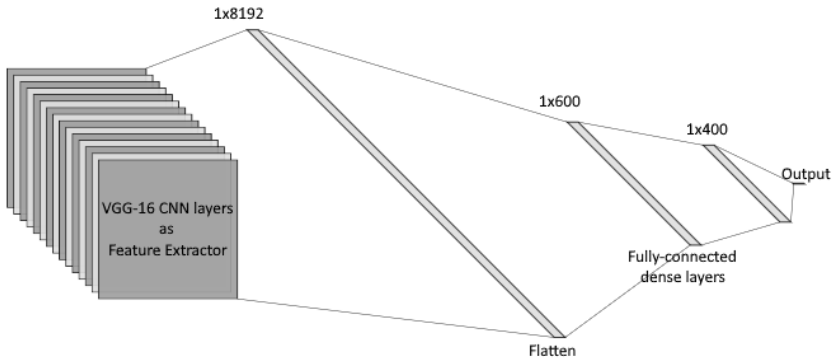


Figure 3.6: Model 5a architecture schematic.

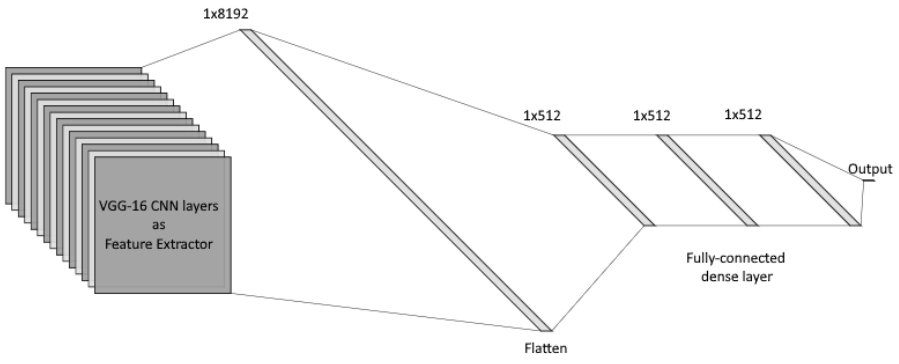


Figure 3.7: Model 5b architecture schematic.

VGG-16 - Pre-trained Feature Extractor

The VGG-16 model is a 16-layer (convolution and fully connected) network built on the ImageNet database [5]. It is a state-of-the-art system that relies on conventional CNN architecture with substantially increased depth. The system was trained for 1000 different classes, using 1.3 million images for training which took approximately three weeks on a high-performance system for training a single network [34]. It is generalised for a wide range of tasks and datasets, which makes it an ideal candidate for a fully-realised pre-trained feature extractor for dogs and cats. See figure 3.8 for architecture.

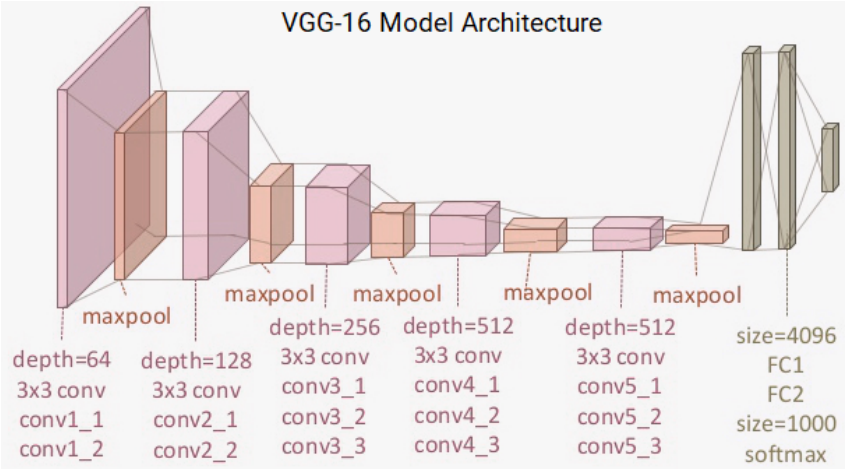


Figure 3.8: VGG-16 Model Architecture [31].

Model Parameters

The code for creating all models and detailed model summaries outlining specific layer properties can be found in Appendix 3.

The following tables outline the parameter and structural choices behind each model for the macro-structure tests. Hyper-parameters such as learning rate vary from model to model. This was experimented by trial-and-error to achieve certain accuracy thresholds, which is suitable when choosing hyper-parameters [19]. The same optimizer, cost function and activation functions were used across all models, seen in table 3.4. Hyper-parameter choice was modelled from Sarkar's work [31].

Component	Value
Optimizer	RMSprop
Cost Function	Cross Entropy Criterion
Activation Function	Rectified Linear (Sigmoid Output)

Table 3.4: Common design decisions across all models

Component	Value
Learning Rate	0.001
Image Augmentation	No

Table 3.5: Model 1 Parameters - Simple CNN

Component	Value
Learning Rate	0.0001
Regularization	Dropout of 0.3 for FC layers
Image Augmentation	Yes

Table 3.6: Model 2 Parameters - Simple CNN with image augmentation and regularization

Component	Value
Learning Rate	0.00001
Regularization	Dropout of 0.3 for FC layers
Image Augmentation	No
Transfer Learning Layers	Frozen

Table 3.7: Model 3 Parameters - TL using VGG-16

Component	Value
Learning Rate	0.00002
Regularization	Dropout of 0.3 for FC layers
Image Augmentation	Yes
Transfer Learning Layers	Frozen

Table 3.8: Model 4 Parameters - TL using VGG-16 and image augmentation

Component	Value
Learning Rate	0.00001
Regularization	Dropout of 0.3 for FC layers
Image Augmentation	Yes
Transfer Learning Layers	Fine-tune

Table 3.9: Model 5 Parameters - TL using VGG-16, image augmentation and fine-tuning

Altered Component	Value
Number of Neurons	FC1 and FC2 changed to 600 and 400 neurons respectively

Table 3.10: Model 5a Parameters - Alter model 5 neurons per FC layer

Altered Component	Value
Number of FC Layers	Two (1 x 512) FC layers changed to three (1x512) layers

Table 3.11: Model 5b Parameters - Alter model 5 number of FC layers

Altered Component	Value
Learning rate	0.00001 changed to 0.00005

Table 3.12: Model 5c Parameters - Alter model 5 learning rate.

Domain

Similar to the investigation phase domain, the famous *Dogs vs. Cat* dataset is used [14]. The *Kaggle* dataset contains 25,000 images of dogs and cats (12,500 per class) for a binary image classification application. A subset of these images was curated for training, validating and testing the models to replicate TL conditions of minimal resources, shown in table 3.13 below.

Training	Validation	Testing
3000	1000	1000

Table 3.13: Subset of *Dogs vs. Cats* dataset used for models, each set equally represented by both classes.

The VGG-16 feature extractor was trained on the *ImageNet* dataset, which is an extremely large visual database, containing over 14 million hand-annotated images, with over 20,000 classes [5]. This makes the VGG-16 model an ideal candidate as the 'high-resource' domain for model adaptation.

Neural Network Utilization Test

The inception of Neural Network Utilization (NNU) is found in 4.1.5.

Utilizing the powerful modelling capability of *Keras*, a model object with a variable number of outputs can be instantiated, such that the predict method in the Keras library can return the activations of a custom selection of individual layers of the trained model as *numpy* arrays in response to the test dataset. To obtain the correct layers for activation analysis for any given model, a sweep is done through the individual layers to identify which should be set as output layers to capture activations. These layers include all dense, convolution, max pool and 'model' (the VGG-16 layer flattened) layers.

The resultant *numpy* array contains the layer activations which can then be taken and analysed for deactivated neurons. A deactivated neuron is defined as a neuron (or for convolutional layers, a feature map) with activations of 0 for the entire test dataset. Therefore, using nested for loops of time complexity $O(a)$, where a is the number of activations, the total number of deactivated neurons, ϑ, can be found. This is computationally expensive as some activations exceed 9 x 10^6, therefore code optimization is a potential area for improvement. The code can be found in Appendix 3.

Once the deactivated neurons were identified, finding NNU for the given model can be found using equation 3.1:

$$NNU = \frac{\vartheta}{Total\ Activations},\ where\ \vartheta\ is\ deactivated\ neurons \tag{3.1}$$

Activation Spectrum Test

A similar process as the NNU test can be done to identify the Activation Spectrum (AS). Once the layer activations have been predicted, the *numpy* array results can be printed sequentially, layer by layer, as frequency histograms to create a model's AS using the popular Python graphing library, *matplotlib* (see figure 3.9). The AS of a model is scaled by the maximum activation of the model, hence code is reused for the Activation Range. The code can be found in Appendix 3.

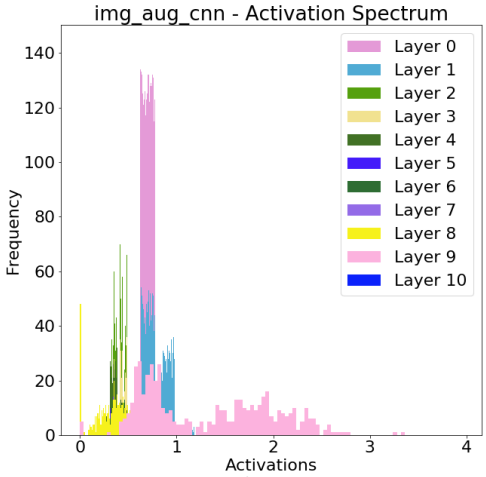


Figure 3.9: Example activation spectrum.

Activation Range Test

A similar process as the NNU test can be done to identify the Activation Range (AR). As the activations are looped through, the maximum activation per layer is tracked instead of deactivated neurons. The maximum activation of each layer is then plotted as a set of bar graphs on a single plane using *matplotlib* (see figure 3.10). The code can be found in Appendix 3.

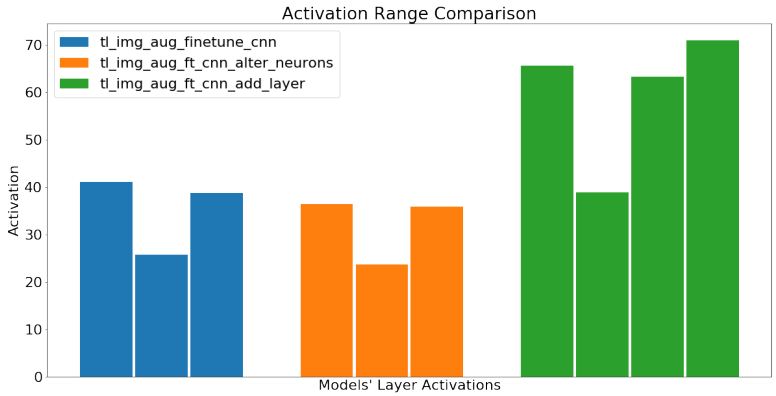


Figure 3.10: Example activation range.

Chapter 4

Evaluation

This chapter evaluates the results from both the investigation phase and the exploration phase.

4.1 Investigation Phase Results

4.1.1 Confusion Matrix

As mentioned in the DNN background section, the standard convention for assessing DNN, and TL applications by extension, are the high-level metrics composed from elements of the confusion matrix (CM). Hence, it is important to develop these high-level, standard metrics for the system to both follow best practices, and also use the qualitative metrics to provide additional perspective to the values of the standard metrics.

Metrics	Value
Accuracy	0.8
Misclassification Rate	0.2
True Positive Rate	0.848
False Positive Rate	0.294
True Negative Rate	0.706
Precision	0.848
Prevalence	0.66
Null Error Rate	0.34
Cohen's Kappa	0.554
F1-Score	0.848

Table 4.1: Confusion matrix (CM) metrics for the baseline cat classifier model.

The definitions of these metrics can be found in chapter 2.2.1. The main high-level metric that will be focused on is accuracy.

4.1.2 Baseline Hyper-Parameter Sweep Test Results

The hyper-parameter sweep tests were initially run on the baseline model for the cat dataset. These sweep tests created 400 different sets of results for predictions for the training and test sets (200 results each). As a starting point, the confusion matrix metrics for each sweep test were graphed to visualise patterns or trends. An example of this visualisation is shown in figure 4.1 for accuracy across the test dataset. A visualisation set for each of the metrics in table 4.1 was created. Therefore, for the ten CM metrics, four sweep tests, and both training and test datasets, 80 graphs were created to visualise potential trends for qualitative metrics. This, in turn, creates 80 visualisations of trends for *any model* the sweep test is run on.

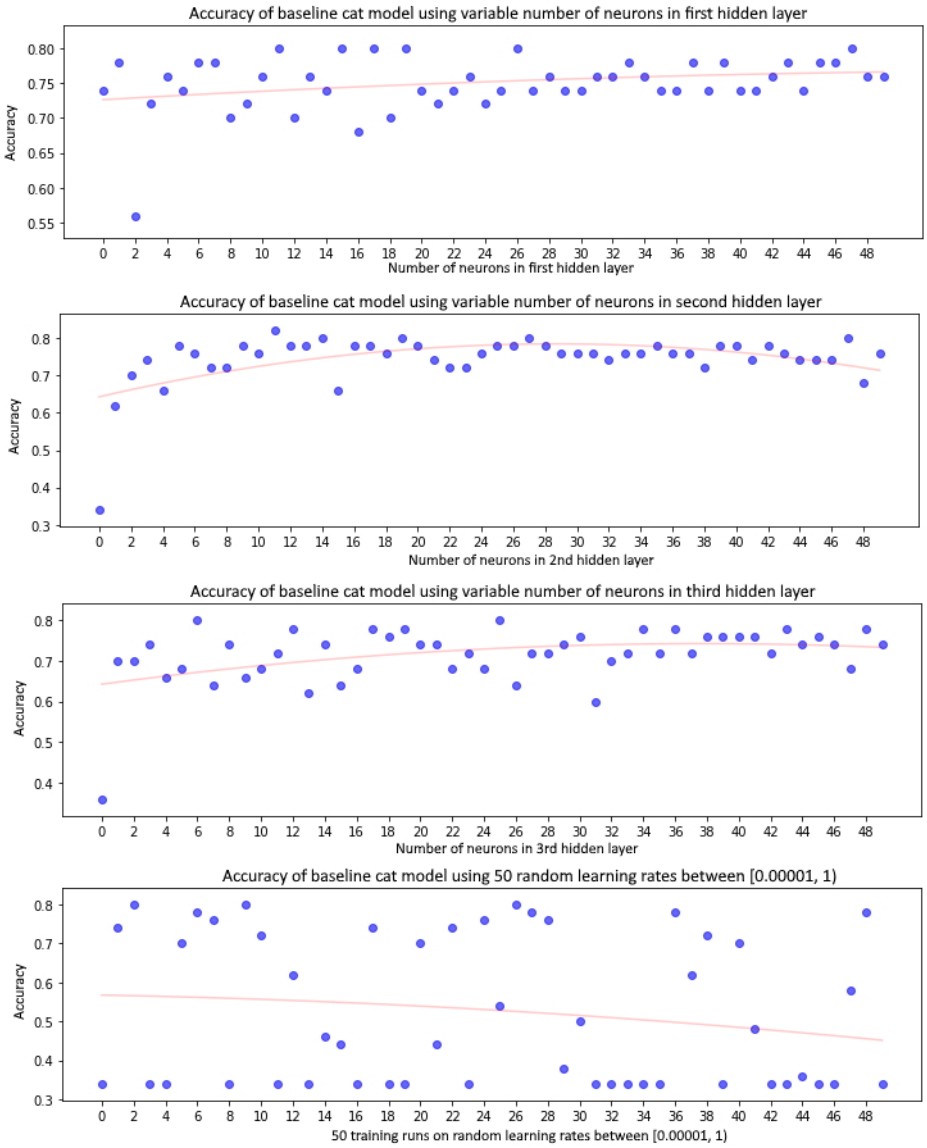


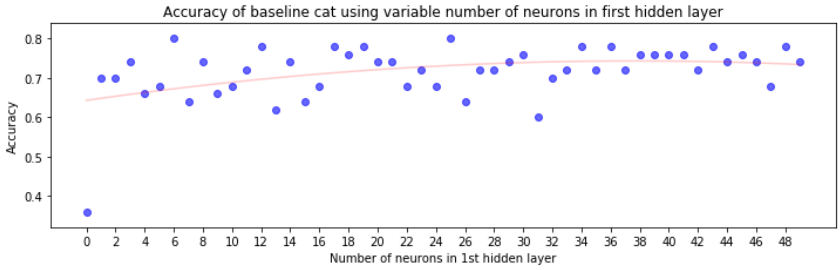
Figure 4.1: The four sweep tests accuracy results on the baseline cat model using the test dataset. The red line marks the line of best fit.

4.1.3 Transfer Learning Hyper-Parameter Sweep Test Results

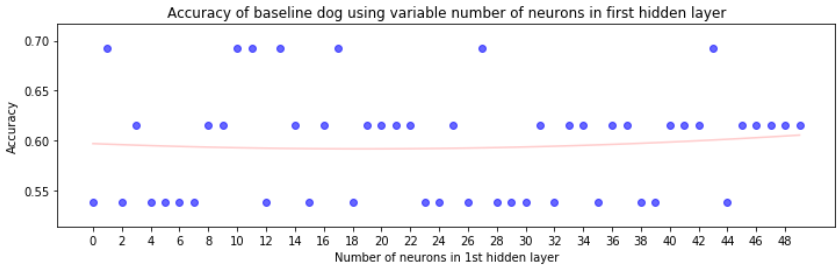
After applying the transfer learning technique of model adaptation, another 80 graphs are generated displaying the trends for the TL model. In addition to this, the baseline dog model is also run in isolation through the sweep tests. There are now visualisations of the sweep tests for the baseline cat model, the baseline dog model, and the TL model.

Consolidating Results To Remain In Scope

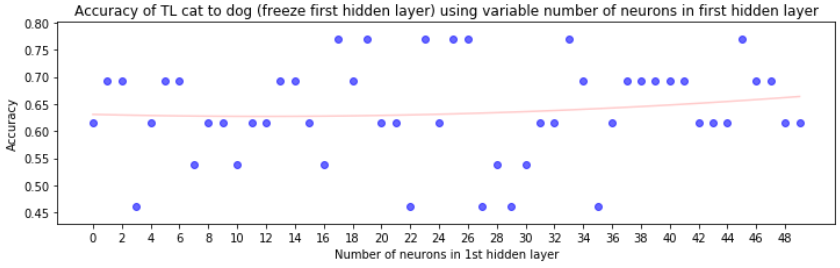
All of these graphs are generated to qualitatively compare the baseline cat model, the baseline dog model, and the TL model, to detect if there are trends. However, there is simply just too much data being generated to closely qualitatively compare every single metric. For the sake of remaining in scope, the training set results were ignored, focusing on the test set as this produces more valuable performance assessment results, and accuracy was the CM metric that was closely analysed, as this was the only predominant high-level metric that was consistently employed for performance assessment in the literature. This helps reduce the number of graphs for comparison from 80 graphs to 4 per model (one for each sweep test).



(a) Baseline cat model



(b) Baseline dog model



(c) TL cat to dog model

Figure 4.2: Comparing baseline cat, baseline dog and TL cat to dog models for accuracy sweep test on the first hidden layer. Full set of sweep visualisations can be found in Appendix 2

When comparing the three models in figure 4.2, it is trivial when looking at the lines of best fit (red lines) to see that the TL model (figure 4.2c) improves the overall accuracy of the dog recognizer (figure 4.2b) by utilizing the knowledge gained from the cat recognizer (4.2a). However, these graphs in isolation do not provide much insight into particular identifiable metrics, as there are no obvious qualitative conclusions as to why the TL model improves in performance

4.1.4 Maximum Activation Results

Whilst analysing the sweep test accuracy results for all models under the test set, other specific investigative evaluations were happening in parallel. The first of which was understanding the input that produced the maximum activation for each neuron, similar to Kensert et al. [15]. The full output of this test can be found in Appendix 2.

The idea behind this test was to reveal specific neurons being 'cat-neurons' or 'non-cat-neurons', similar to Rodriquez work [25]. However, the potential take away from this test revealed that there was a consistency of maximum activation values between layers for the most successful model; the baseline cat-classifier. This is shown in figure 4.3.

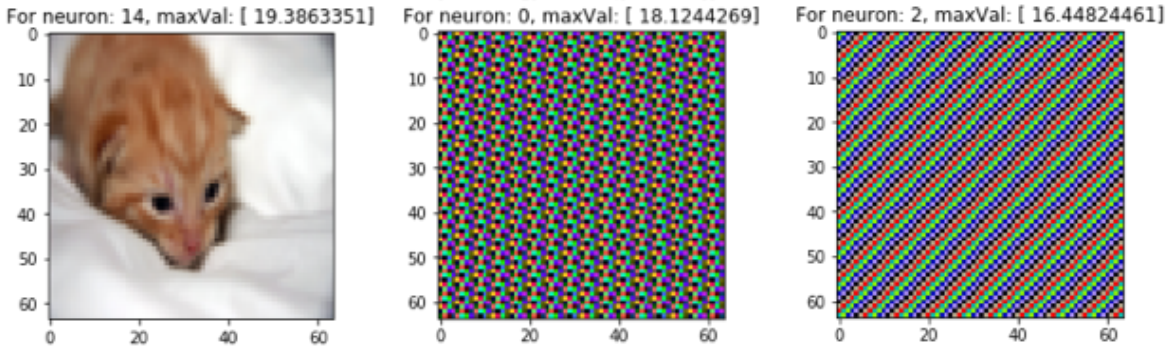


Figure 4.3: The maximally activated neuron from each particular layer of baseline cat model share somewhat consistent values.

The last two images of figure 4.3 were a by-product created to detect trends in intermediate activations. However, no useful conclusions can be made from these visualisations.

4.1.5 'Reverse Engineering' Results

The 'reverse-engineering' test was used to reveal qualitative metrics from activation trends. This test revealed a breakthrough for the thesis by revealing the existence of 'deactivated neurons', summarised below in figure 4.4. Neurons 4, 7, 12, 14 and 19 all 'flat-line' during the activation for all three inputs (the full set of graphs showing all deactivated neurons is found in Appendix 2). This indicates that regardless of input, these particular neurons are 'deactivated'. This hypothesis was confirmed for the entire dataset, revealing they were deactivated for all input.

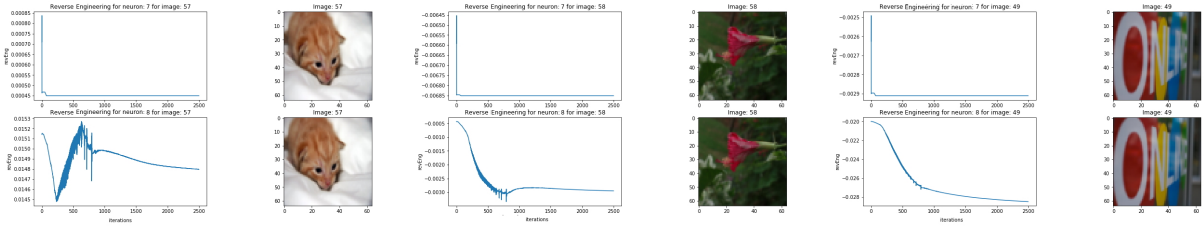
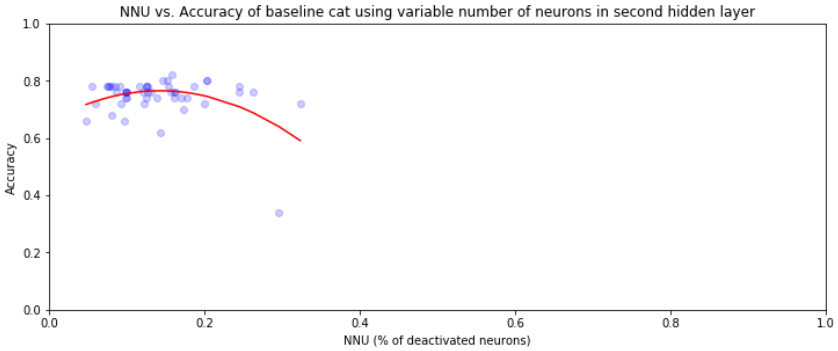


Figure 4.4: 'Reverse engineering' tests for three varied inputs showcasing the difference between a deactivated neuron and an activated neuron. Full set of graphs found in Appendix 2.

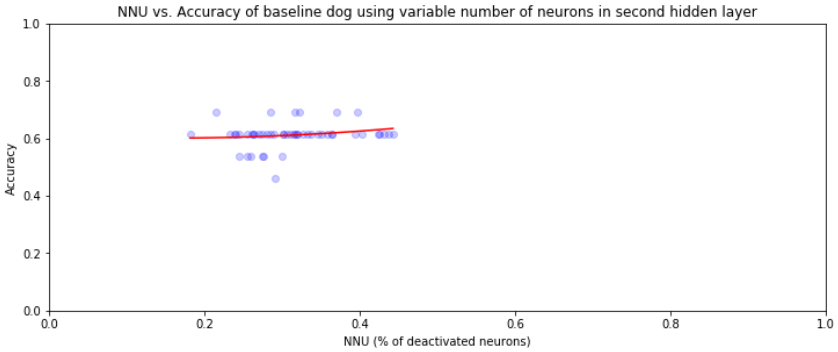
4.1.6 Potential Metrics Gleaned From Tests

Neural Network Utilization

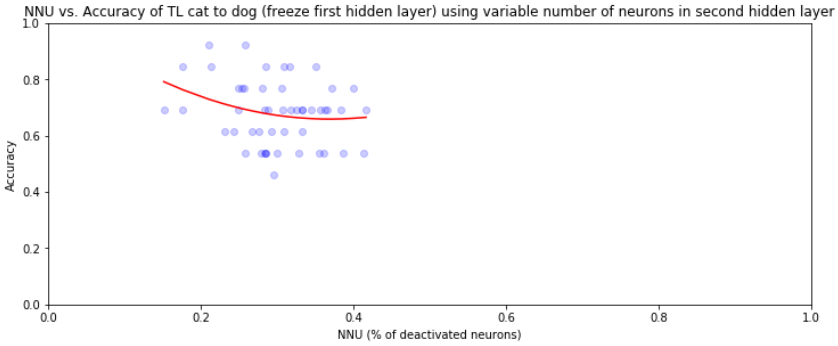
Combining the results from the hyper-parameter sweeps for accuracy and the breakthrough of deactivated neurons in 'reverse engineering', the first potential qualitative metric was identified. **Neural Network Utilization (NNU)** is defined as the percentage of deactivated neurons within the network. A percentage is vital as the total number of neurons varies from model to model. Below in figure 4.5, the NNU against accuracy reveals potential results, as trends differ between baseline cat, dog and TL models.



(a) Baseline cat model



(b) Baseline dog model



(c) TL cat to dog model

Figure 4.5: Comparing baseline cat, baseline dog and TL cat to dog models for NNU vs. accuracy using sweep test on the second hidden layer. Full set of sweep visualisations can be found in Appendix 2.

However, while there is variance in NNU across the models which indicates potential value as a new metric, the results of this test are inconclusive, most likely due to the small-scale nature of this investigation phase. Hence, this potential metric's significance will be fully extrapolated in the *exploration phase*.

Activation Spectrum and Range

The results of maximum activation and the reveal of deactivated neurons imply that additional qualitative metrics can be gleaned from analysing the activations across the entirety of models.

Hence, two additional qualitative metrics are proposed. Firstly, **Activation Spectrum**, which will analyse the spectrum of activations of neurons across the entire network. This was explored, but results are irrelevant due to the inconclusive nature of such a small number of neurons and small datasets. The full implication of this metric is revealed in the *exploration phase*.

Secondly, **Activation Range**, which reveals the maximum activation range of individual layers. Early tests in this phase from 4.1.4 revealed that successful models had a consistency of maximum activation in each layer. This implication will be solidified in the *exploration phase*.

4.2 Exploration Phase Results

4.2.1 Accuracy Results

Firstly, the accuracy across the test set of each model is required to use the potential metrics to qualitatively assess performance.

Model	Accuracy
1 - basic cnn	0.759
2 - img aug cnn	0.835
3 - transfer cnn	0.891
4 - tl img aug cnn	0.896
5 - tl img aug ft cnn	0.949

Table 4.2: Accuracy across test set for macro-models.

The pre-condition of each macro-model improving upon the accuracy of each predecessor is fulfilled. Following this pre-condition, the qualitative metrics can now attempt to provide insight into these values.

Model	Accuracy
5 - tl img aug ft cnn	0.949
5a - tl img aug ft cnn alter neurons	0.944
5b - tl img aug ft cnn add layer	0.956
5c - tl img aug ft cnn alter lr	0.933

Table 4.3: Accuracy across test set for micro-models.

These values indicate that making micro-structural changes to models does not affect performance as significantly as the macro-structural changes. However, the qualitative metrics will still be investigated to potentially provide insight into these high-level values as well.

4.2.2 Neural Network Utilization (NNU) Results

Macro-models results

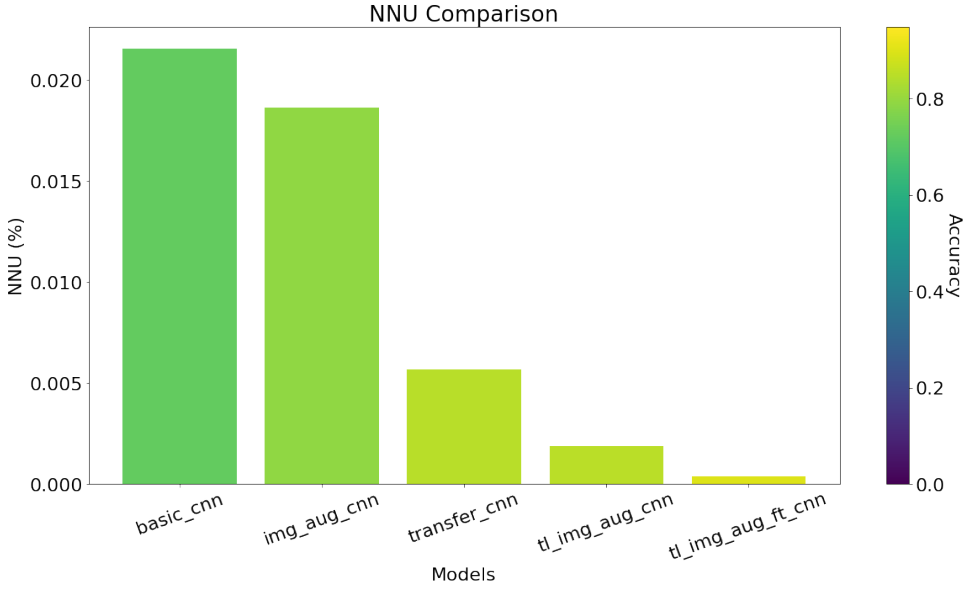


Figure 4.6: NNU results for macro-models 1 to 5.

Model	Deactivated Neurons	Total Activations	NNU	Accuracy
1	366	1697000	0.02157%	0.759
2	316	1697000	0.01862%	0.835
3	58	1025000	0.00566%	0.891
4	172	9217000	0.00187%	0.896
5	35	9217000	0.00038%	0.949

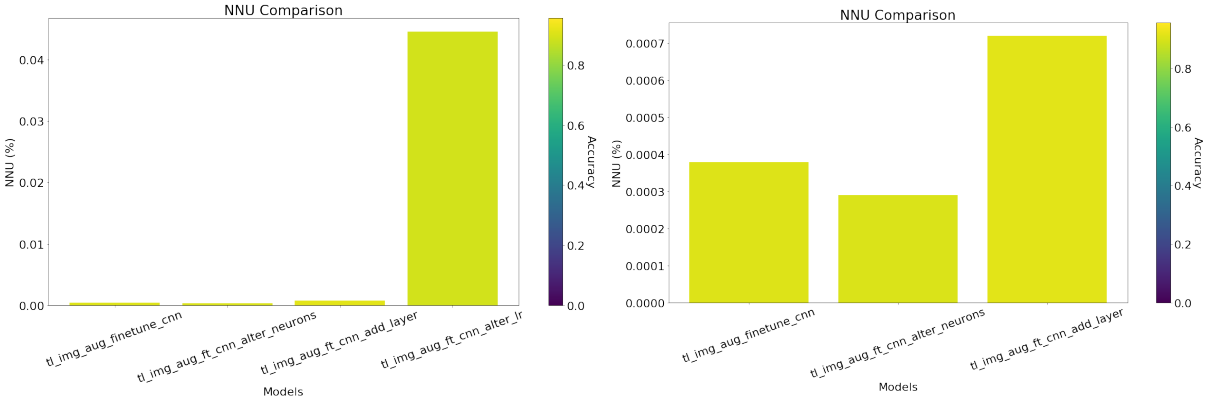
Table 4.4: Accuracy across test set for macro-models.

NNU is the most promising of the three qualitative assessment metrics, evident from the substantial trend in figure 4.6 and table 4.4. As shown in the graph, NNU decreases as accuracy increases (accuracy depicted by colour-mapping), showing an inversely proportional relationship between NNU and accuracy.

The major implications, inferring from the macro-structural differences, is that applications can improve their performance by reassessing regularization techniques, such as L2 or dropout, as well as employ data augmentation techniques such as image augmentation. These techniques decrease NNU and thus increase performance (i.e. accuracy).

In addition, like all DL applications, data quality and quantity will be a bottleneck for performance. This is indicative by the results of TL models outperforming the other models as the TL models exploit the VGG-16 model which is trained on the extremely substantial ImageNet dataset. Therefore, increasing data quality and quantity affects the overall NNU which affects performance.

Micro-model results



(a) All micro-models

(b) Micro-models excluding outlier

Figure 4.7: NNU results for micro-models.

Model	Deactivated Neurons	Total Activations	NNU	Accuracy
5	35	9217000	0.00038%	0.949
5a	27	9193000	0.00029%	0.944
5b	70	9729000	0.00072%	0.956
5c	4106	9217000	0.04455%	0.933

Table 4.5: Accuracy across test set for micro-models.

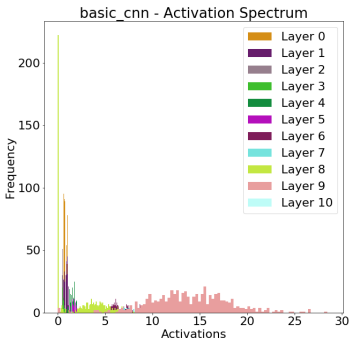
From figure 4.7a and table 4.5, micro-models provide less conclusive results as the macro-models. Particularly as the altered learning rate is a substantial outlier. Various attempts were run for the altered learning rate model and all presented the same outlier behaviour. The other three models (5, 5a, 5b) shown in figure 4.7b confirm the results from the macro-models, as their high-accuracy is reflected by low NNU, with each micro-model having smaller NNU than the other macro-models.

However, when comparing these three models, NNU seems to be *proportional* to accuracy not inversely proportional. The difference of NNU of these models is relatively small when compared to the macro-models, however, it does imply that while minimising NNU can be indicative of performance, it is not the only contributing factor. Hence, macro-structural differences of models substantially affect NNU which affects performance, whereas micro-structural differences do not provide a strong correlation between NNU and performance.

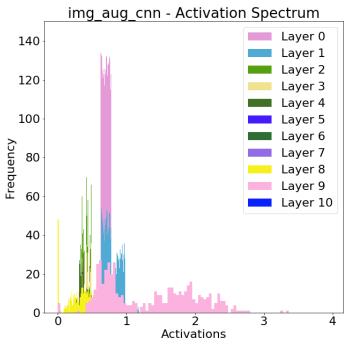
An additional implication of minimising NNU by deleting deactivated neurons is explored in another test in section 2.

4.2.3 Activation Spectrum (AS) Results

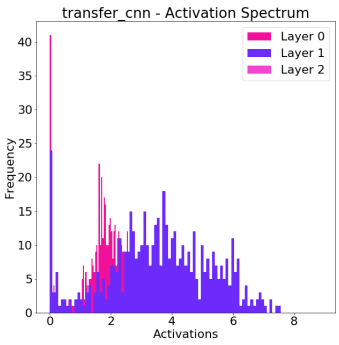
Macro-models results



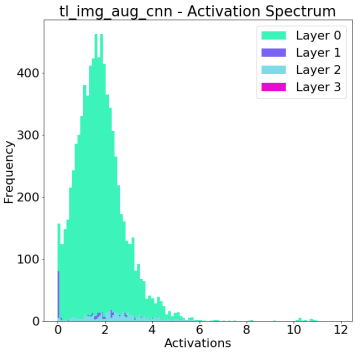
(a) Model 1 - Basic CNN without regularization



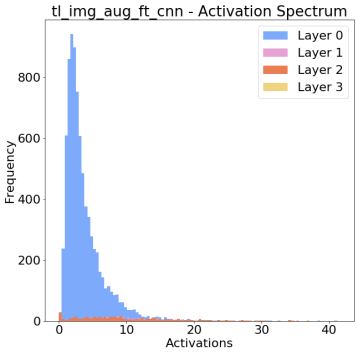
(b) Model 2 - CNN with image augmentation and regularization



(c) Model 3 - TL with frozen layers



(d) Model 4 - TL with image augmentation and frozen layers



(e) Model 5 - TL with image augmentation and fine-tuned layers

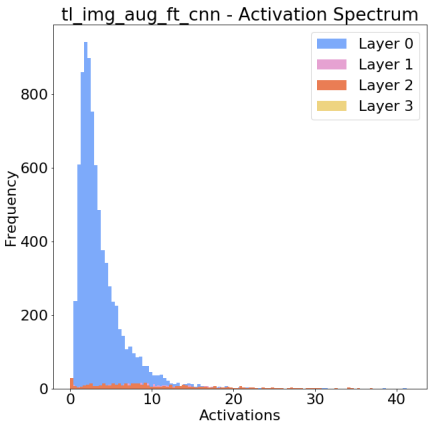
Figure 4.8: Activation spectra for macro-models.

The second qualitative assessment metric, **Activation Spectrum**, provides a visualisation of the entire network's maximum activations per neuron. As postulated in the investigation evaluation, depicting the spectrum of activation provides some qualitative understanding of why different systems perform better than others. The spectra use different colours to delineate activations in different layers.

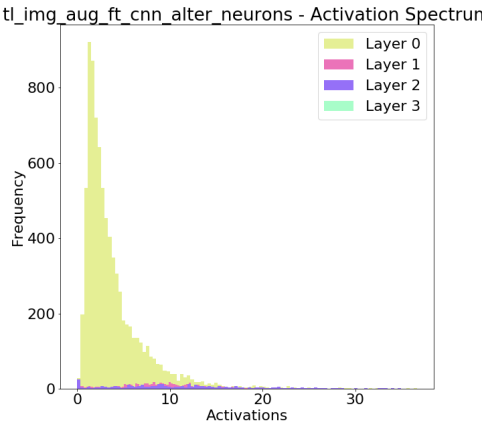
Comparing models 1 and 2 to models 3, 4 and 5 in figure 4.8, it is evident that introducing TL begins to shift activations to have a right-skewed distribution. This implies right-skewed distributions of maximum activations of each neuron correspond to better-performing systems. In addition, re-emphasising the results of NNU, minimising NNU in producing better performing systems is visually evident in the spectrum.

This qualitative metric could similarly be used to illustrate improvements to a system. To improve AS, minimising NNU, re-evaluating the activation function and initial weight distribution could all be strategies to develop right-skewed distributions while limiting deactivated neurons. Furthermore, using L2 regularization, which penalising the square values of weights, will drive weights to be smaller and hence should promote more right-skewed spectra (see 2.2.2 for more on L2).

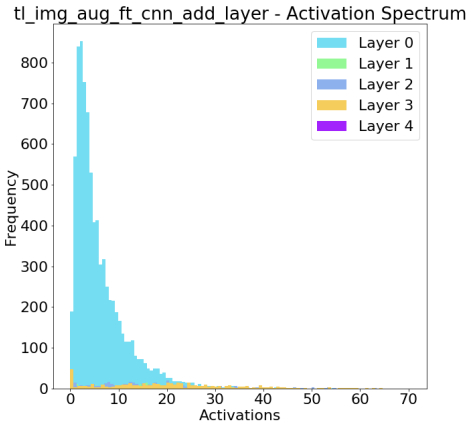
Micro-models results



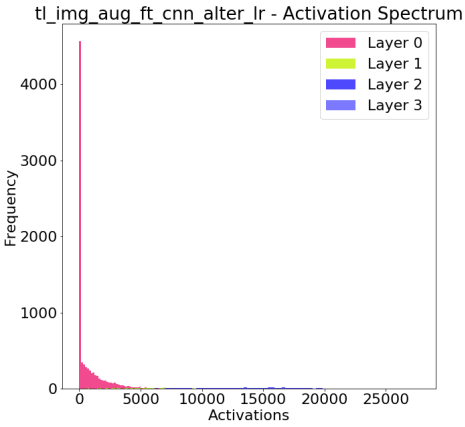
(a) Model 5 - TL with image augmentation and fine-tuned layers



(b) Model 5a - TL with img aug and ft layers with altered neurons



(c) Model 5b - TL with img aug and ft layers with an added layer



(d) Model 5c - TL with img aug and ft layers with altered learning rate

Figure 4.9: Activation spectra for micro-models

Since all of these micro-models share similar performances, it is unsurprising that all contain right-skewed distributions with minimal NNU (excluding figure 4.9d which is an outlier). This further emphasises the need to develop right-skewed spectra when seeking to improve systems.

4.2.4 Activation Range (AR) Results

Macro-models results

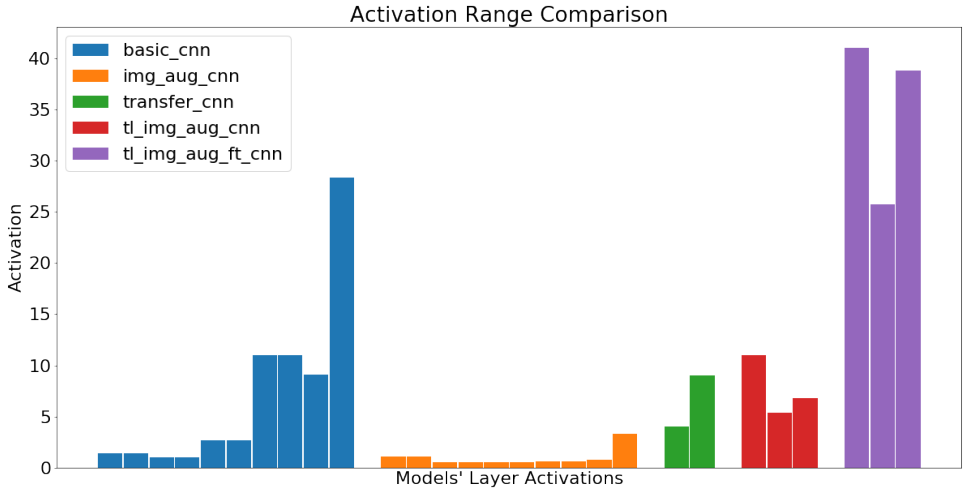


Figure 4.10: Activation range results for macro-models.

The final qualitative assessment metric developed, **Activation Range**, measures the maximum activation within each individual layer of a model. This metric provides additional insight into the nature of activation values affecting performance, as the best performing model - model 5, coloured purple in figure 4.10 - has the largest activation of any of the macro-models. However, from other experiments, such as that depicted in figure 4.11 with a retrained model 1, larger magnitudes of AR do not necessarily imply better performance, as model 1 in this experiment still had an accuracy of 75.6%.

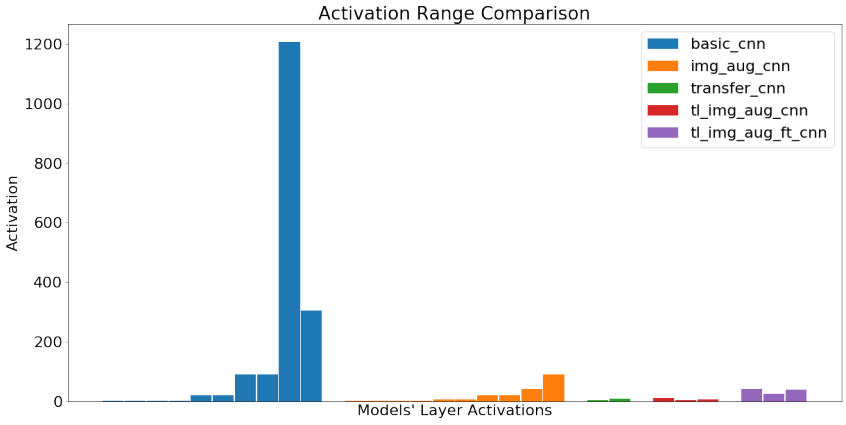
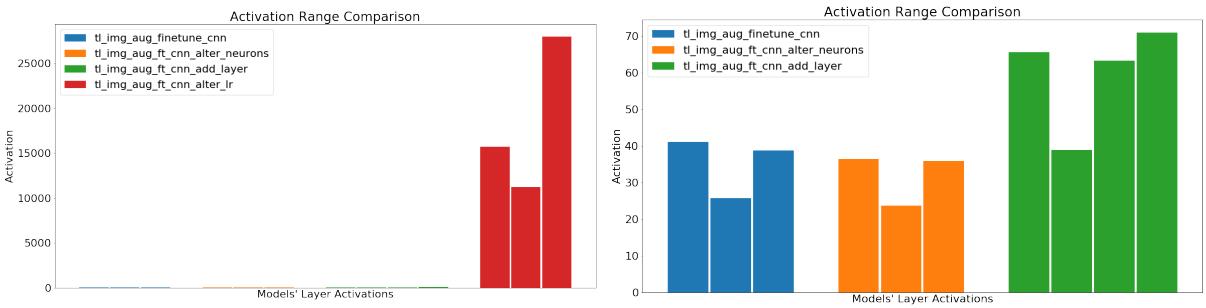


Figure 4.11: Another iteration of macro-model activation range results showing inconsistencies in magnitude.

However, from figure 4.10 and 4.11, it is clear that consistency in each layer's activation within a model reflects better performance, as shown in models 3, 4 and 5.

Micro-models results



(a) All micro-models

(b) Micro-models excluding outlier.

Figure 4.12: Activation range results for micro-models.

Consistency in each layer's activations implying better performance is also highlighted by the micro-models results in figure 4.12, as all of the high-performing models all maintain consistency of each layer's maximum activation. Hence, the implications of using AR to improve a model's performance are to seek consistencies of layer activations. This can be accomplished by applying similar strategies to improving the NNU and AS, by employing regularization within the network (particularly considering L2 to drive weights to be smaller) as well as reassessing activation function choices.

4.2.5 Unsuccessful Tests

Several other tests for qualitative metrics were explored, however, were ultimately unsuccessful for various reasons. These are dictated and explored below.

Maximally Activating Features

Using *Keras*, it is possible to analyse the intermediate activations within a network, as mentioned previously in the NNU test section of chapter two. This is particularly useful when exploring image classification, as specific features of images are more reactive for specific neurons (or feature maps for convolutional layers). By plotting these activations as images, the results reveal the more reactive features of each neuron, as specific sections of these images are significantly brighter than others, as seen in figure 4.13.

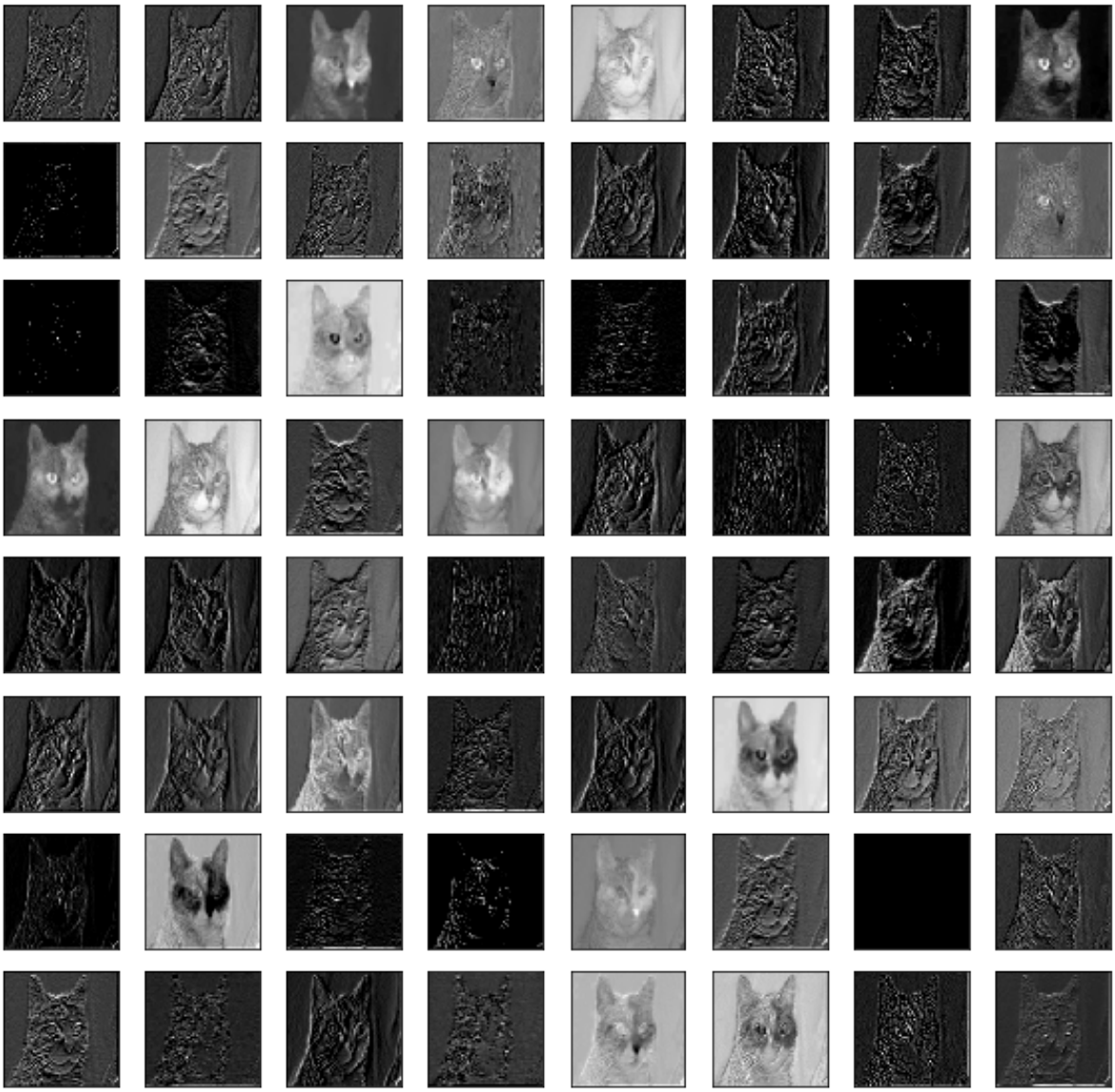


Figure 4.13: Visualisation created in an attempt to expose specific features which triggered significant activation within networks.

This visualisation was created to potentially reveal a metric of exposing and isolating the specific features which trigger significant activations. However, nothing conclusive amounted from this exploration.

Most prevalent pixel inversions

Stemming from the work of Samek et al. [30], an attempt was made to measure which particular pixels of an input image triggered the highest activation, using the intermediate activations of the previous unsuccessful test. Upon finding these pixels, a test would have been created to randomize the colours of those pixels and see how the network reacted to the altered input. This, in turn, could have extended to a potential metric similar to Maximally Activating Features, however, remained inconclusive.

Deactivation Deletion Results

The final unsuccessful test stemmed from the implications found in exploring NNU, as it was identified that minimising NNU produced superior results. With this implication, an intuitive step could be to delete these particular neurons from the network and see if performance increased, decreased, or remained the same. After various tests on the various models available, deleting the deactivated neurons had little to no impact on performance, as accuracy remained stagnant. Therefore, this test did not produce any additional metrics but did solidify that removing deactivated neurons has no impact on improving performance.

Chapter 5

Conclusion

5.1 Summary Of Solution

The search for qualitative assessment metrics was successful as three metrics were identified and explored for developing additional insight into the performance of transfer learning applications.

In the *investigation phase*, hyperparameter sweep tests were developed to analyse and compare trends across the high-level confusion matrix metrics. In addition, while analysing the networks at the activation level using the 'reverse-engineering' tests, a pivotal result was discovered in the identification of *deactivated neurons*; neurons with no activation across entire datasets. Combining the sweep tests and this pivotal result, Neural Network Utilization (NNU), Activation Spectrum (AS), and Activation Range (AR) were all identified as potential metrics to explore.

NNU assesses the percentage of deactivated neurons to total neurons across an entire network. AS visualises the maximum activation distribution of each individual neuron across the entire network. AR visualises the range of each layer of the network, plotting the maximum activation per layer. The full extent of these metrics could not be achieved using the smaller-scale experiment, as results were promising but inconclusive.

In the *exploration phase*, a larger scale experiment was developed, such that the qualitative insights and implications of the identified metrics from the *investigation phase* could be recognised and examined. Utilising the powerful high-level, robust, DL Python library, *Keras*, a series of models were developed. Five models with macro-structural differences were created, each increasing with complexity than the previous model. The increase in complexity was intended to provide an increase in performance. Taking the best model from the macro-models, another three models were created with micro-structural changes between each model.

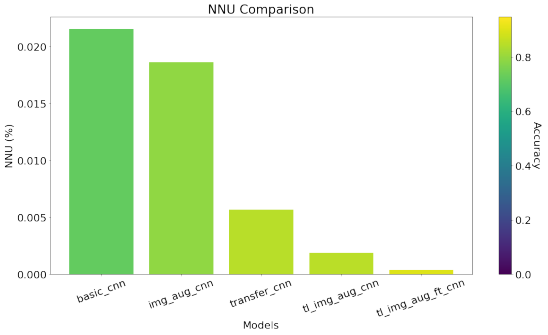
Using both the macro-models and micro-models, the three metrics were explored. *Keras* has powerful modelling capabilities, granting the ability to explore all intermediate activations within the network. These intermediate activations formed the basis of identifying deactivated neurons, extracting maximum activation values, and hence, developing the three metrics. Using these results, visualisations of all three metrics were developed to qualitatively assess the performance of transfer learning.

5.2 Implications Of Qualitative Metrics

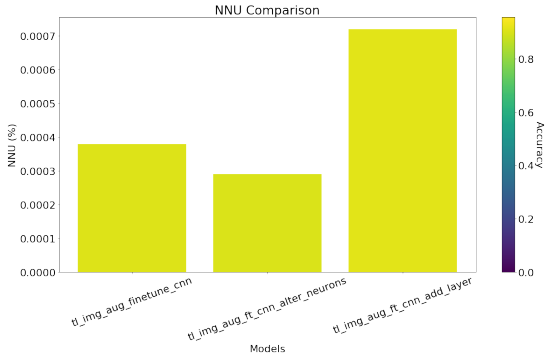
Neural Network Utilization (NNU)

Each of the three qualitative metrics can provide additional insight into understanding the performance of a deep learning system; and by extension, a transfer learning system.

The results from the NNU tests were the most insightful in providing a qualitative perspective to TL systems. It was found that across the macro-models, NNU was inversely proportional to accuracy (see figure 5.1a), implying that a reason for superior performing systems is the minimal percentage of deactivated neurons in the network. It also gives credence to how to improve NNU, and therefore improve the system's performance, as seen by the macro-structural changes being contingent factors to reducing NNU; macro-structural changes being regularization techniques, data augmentation techniques, and improving data quantity and quality. The analysis of micro-models (see figure 5.1b) revealed that minor-structural changes within well-performing systems affect NNU much less significantly compared to macro-structural changes.



(a) NNU results for macro-models 1 to 5.

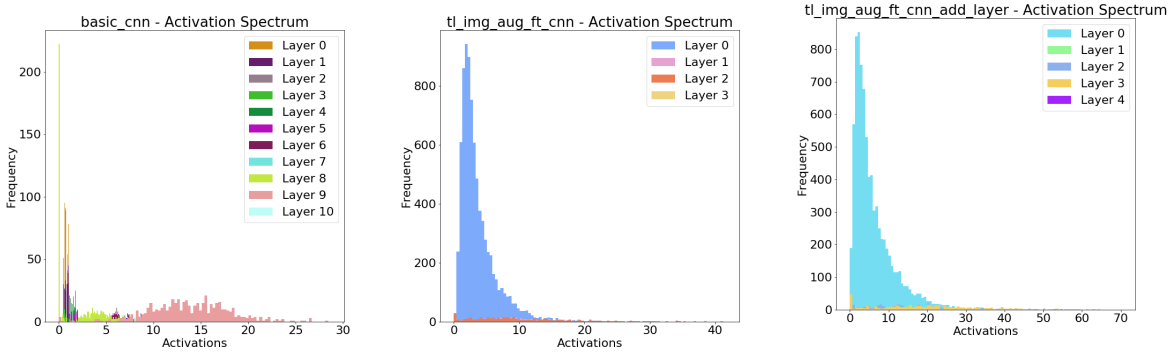


(b) NNU results for micro-models 5, 5a, 5b.

Figure 5.1: NNU results of all models.

Activation Spectrum (AS)

The results from AS tests revealed that the distribution of activation across an entire network can also be used to qualitatively assess the performance of applications. Particularly when comparing the macro-models, a high amount of deactivated neurons (which is the first green spike in figure 5.2a) and an otherwise uniform distribution produced poorer results, whereas having a reduced amount of deactivated neurons and a right-skewed distribution (seen in figures 5.2b and 5.2c) produced superior results. The implication of this test reveals other potential methods of improving systems such as reassessing activation functions, initial weight distributions, and utilizing regularization techniques such as L2 to drive weights to be smaller.



(a) Model 1 - Basic CNN without regularization.

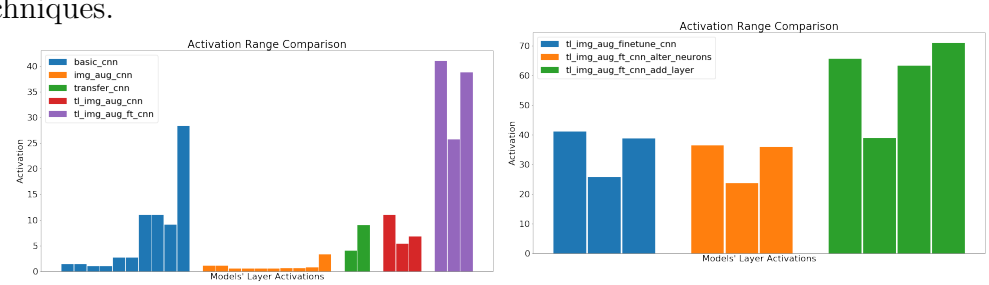
(b) Model 5 - TL with image augmentation and fine-tuned layers.

(c) Model 5b - TL with img aug and ft layers with an added layer.

Figure 5.2: Activation Spectra for models 1, 5 and 5b.

Activation Range (AR)

Finally, AR tests revealed additional qualitative understanding by analysing the maximum activations of each layer rather than individual neurons. These tests revealed that superior performing models maintained more consistent ranges per layer in comparison to poorer performing models, which produced more erratic distributions. This is evident when comparing the earlier, poorer performing macro-models in figure 5.3a to the later macro-models in the same figure and micro-models in figure 5.3b. Similarly, this metric could be improved by assessing activation functions, weight distributions and regularization techniques.



(a) AR macro-models.

(b) AR micro-models excluding outlier.

Figure 5.3: Activation range results for micro-models.

5.3 Future Work

The premise of the thesis was to develop potential metrics to qualitatively assess the performance of TL systems, which was intended to be a foundation into unpacking the 'black-box' nature of deep learning. Therefore, future work would entail taking these metrics and exploring the 'black-box' further. A starting point could be developing additional relationships between the qualitative metrics and other high-level metrics from the confusion matrix.

In addition, several other concepts were developed to identify additional qualitative metrics, such as exposing the specific features of data that triggered significant activation (seen in figure 5.4), which was discussed in section 4.2.5.

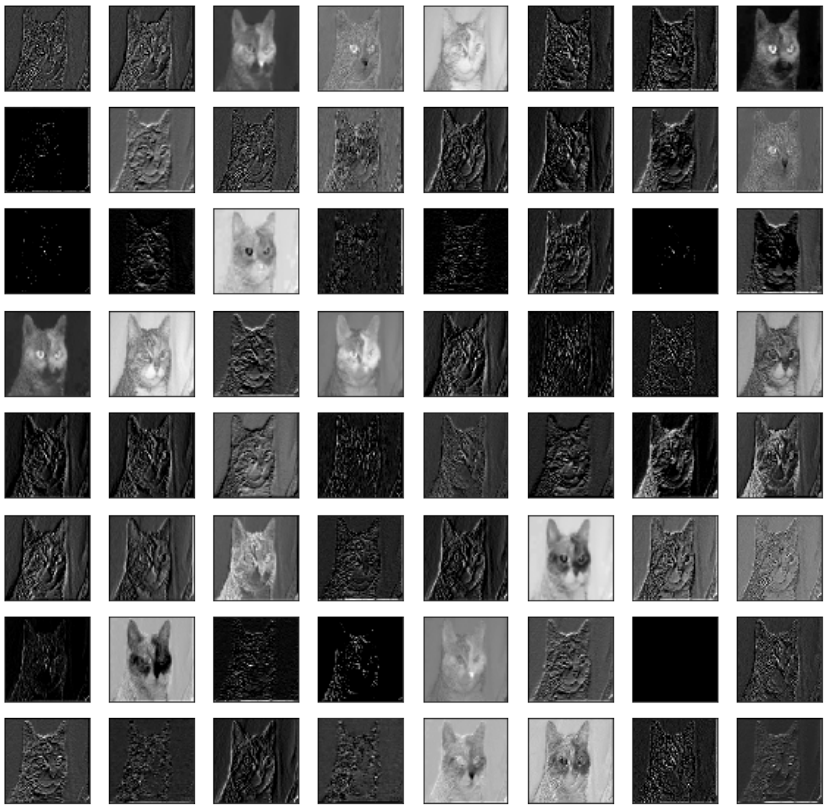


Figure 5.4: Visualisation created in an attempt to expose specific features which triggered significant activation within networks.

Another area that was not explored was analysing how weights changed before and after the transfer of knowledge occurred, which could also develop additional qualitative metrics. Most of the qualitative metrics from the thesis were gleaned from analysis at the activation level, and analysing the weights themselves would be the next layer of abstraction.

Bibliography

[1] O. Abdel-Hamid, L. Deng, and D. Yu. Exploring convolutional neural network structures and optimization techniques for speech recognition. In *Interspeech*, volume 11, pages 73–5, 2013.

[2] B. Ahmed. Summary of transfer learning, 2019. School of El. Eng. and Telecom., UNSW.

[3] A. Bahnsen. Building ai applications using deep learning. https://albahnsen.com/2017/06/06/building-ai-applications-using-deep-learning/, May 2018.

[4] R. Caruana, D. L. Silver, J. Baxter, T. M. Mitchell, L. Y. Pratt, and S. Thrun. Learning to learn: knowledge consolidation and transfer in inductive systems, 1995.

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[6] L. Deng and D. Yu. Deep learning: Methods and applications. *Foundations and Trends® in Signal Processing*, 7(3–4):197–387, 2014.

[7] M. M. Ghazi, B. Yanikoglu, and E. Aptoula. Plant identification using deep neural networks via optimization of transfer learning parameters. *Neurocomputing*, 235:228–235, 2017.

[8] A. Ghoshal, P. Swietojanski, and S. Renals. Multilingual training of deep neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 7319–7323, May 2013.

[9] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, J. Cai, et al. Recent advances in convolutional neural networks. *Pattern Recognition*, 77:354–377, 2018.

[10] Y. He and G. Ding. Deep transfer learning for image emotion analysis: Reducing marginal and joint distribution discrepancies together. *Neural Processing Letters*, Apr 2019.

[11] G. Heigold, V. Vanhoucke, A. Senior, P. Nguyen, M. Ranzato, M. Devin, and J. Dean. Multilingual acoustic models using distributed deep neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 8619–8623, May 2013.

[12] G. Hinton, L. Deng, D. Yu, G. Dahl, A. rahman Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. Sainath, and B. Kingsbury. Deep neural networks for acoustic modeling in speech recognition. *Signal Processing Magazine*, 2012.

[13] J. Huang, J. Li, D. Yu, L. Deng, and Y. Gong. Cross-language knowledge transfer using multilingual deep neural network with shared hidden layers. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 7304–7308, May 2013.

[14] Kaggle. Dogs vs. cats. https://www.kaggle.com/c/dogs-vs-cats/data.

[15] A. Kensert, P. J. Harrison, and O. Spjuth. Transfer learning with deep convolutional neural networks for classifying cellular morphological changes. *SLAS DISCOVERY: Advancing Life Sciences R&D*, 24(4):466–475, 2019. PMID: 30641024.

[16] H. Li, N. A. Parikh, and L. He. A novel transfer learning approach to enhance deep neural network classification of brain functional connectomes. *Frontiers in neuroscience*, 12:491, 2018.

[17] M. M. Najafabadi, F. Villanustre, T. M. Khoshgoftaar, N. Seliya, R. Wald, and E. Muharemagic. Deep learning applications and challenges in big data analytics. *Journal of Big Data*, 2(1):1, 2015.

[18] A. Ng. Nips 2016 tutorial: "nuts and bolts of building ai applications using deep learning" by andrew ng, May 2018.

[19] A. Ng. Improving deep neural networks: Hyperparameter tuning, regularization and optimization. https://www.coursera.org/learn/deep-neural-network/home/welcome, March 2019.

[20] Nielsen and M. A. Neural networks and deep learning, Jan 1970.

[21] S. Nitish. Dropout: A simple way to prevent neural networks from overfitting. *J. Machine Learning Reasearch*, 2014:1929.

[22] C. Olah. Understanding lstm networks. https://colah.github.io/posts/2015-08-Understanding-LSTMs/, Aug 2015.

[23] L. Perez and J. Wang. The effectiveness of data augmentation in image classification using deep learning. *arXiv preprint arXiv:1712.04621*, 2017.

[24] R. Ranjan, S. Sankar, C. Castillo, and R. Chellappa. An all-in-one convolutional neural network for face analysis. 11 2016.

[25] J. Rodriguez. What's new in deep learning research: Jennifer aniston and the process of understanding learning by. https://medium.com/jrodthoughts/whats-new-in-deep-learning-research-jennifer-aniston-and-the-process-of-understanding-learning-by-d961ae2df0e3, Mar 2018.

[26] S. Romdhani. Implementation of dnn-hmm acoustic models for phoneme recognition, 2015.

[27] P. Roy. Natural images. https://www.kaggle.com/prasunroy/natural-images, Aug 2018.

[28] S. Ruder. Transfer learning - machine learning's next frontier, Apr 2019.

[29] P. S. A survey on transfer learning. *J. Knowledge and Data Engineering, IEEE Transactions on*, 22:1345, 2010.

[30] W. Samek, A. Binder, G. Montavon, S. Lapuschkin, and K.-R. Müller. Evaluating the visualization of what a deep neural network has learned. *IEEE transactions on neural networks and learning systems*, 28(11):2660–2673, 2016.

[31] D. Sarkar. A comprehensive hands-on guide to transfer learning with real-world applications in deep learning, Nov 2018.

[32] L. Shao, F. Zhu, and X. Li. Transfer learning for visual categorization: A survey. May 2015.

[33] C. Shorten and T. M. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):60, 2019.

[34] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[35] S. Tejaswi and S. Umesh. Dnn acoustic models for dysarthric speech. In *2017 Twenty-third National Conference on Communications (NCC)*, pages 1–4, March 2017.

[36] Udacity. Aind term 2 - vui capstone project. https://github.com/udacity/AIND-VUI-Capstone, Feb 2018. [Online; accessed 1-May-2019].

[37] Ujjwalkarn. An intuitive explanation of convolutional neural networks. https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/, May 2017.

[38] D. Wang and T. F. Zheng. Transfer learning for speech and language processing. *CoRR*, abs/1511.06066, 2015.

[39] D. Yan and S. Guo. Leveraging contextual sentences for text classification by using a neural attention model. *Computational intelligence and neuroscience*, 2019, 2019.

[40] W. Yue. Constrained deep transfer feature learning and its applications. *C. In CVPR*, 2016.

[41] W. Zhao. Research on the deep learning of the small sample data based on transfer learning. *AIP Conference Proceedings*, 1864(1):020018, 2017.

[42] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017.

Appendix

Appendix 1 - Risk assessment

HS017

HS Risk management form

For additional information refer to HS329 Risk Management Procedure

Faculty/Division: Engineering			School/Unit: EET	
Document number: HS_ENG_001	Initial Issue date: 21.03.19	Current version: V1	Current Version: V1 Issue date: 21.03.19	Next review date: 21.09.19

Risk management name **Electrical Engineering Thesis**

Form completed by	J. Smith	*Signature*	*Date 21.03.19*
Responsible supervisor/ authorising officer	B. Ahmed	*Signature* Beenakmed	*Date* 17/04/19

Identify the activity and the location of the activity

Description of activity Electrical engineering thesis in the machine learning space. Specifically, acoustic model adaptation in low resource domains.

Description of location UNSW Campus

Identify who may be at risk from the activity:

This may include fellow workers, visitors, contractors and the public. The types of people may affect the risk controls needed and the location may affect the number of people at risk

Persons at risk Self

How they were consulted on the risk N/A

List legislation, standards, codes of practice, manufacturer's guidance etc used to determine control measures necessary

Work Health and Safety Act 2011
Work Health and Safety Regulation 2011

Identify hazards and control the risks.
1. An activity may be divided into tasks. For each task identify the hazards and associated risks. Also list the possible scenarios which could sooner or later cause harm.
2. Determine controls necessary based on legislation, codes of practice, Australian standards, manufacturer's instructions etc.
3. List existing risk controls and any additional controls that need to be implemented
4. Rate the risk once all controls are in place using the matrix in HS329 Risk Management Procedure

SHADED GREY AREAS
If you need to determine whether it's reasonably practicable to implement a control, based on the risk complete the shaded grey columns

Feel free to resize the boxes to suit your situation/the amount of text you need to use

Task/ Scenario	*Hazard*	*Associated harm*	*Existing controls*	*Any additional controls required?*	*Risk Rating*			*Cost of controls (in terms of time, effort, money)*	*Is this reasonably practicable* Y/N
					C	*L*	*R*		
Using on campus computer facility	Tripping on obstacles (chairs, bags, wires)	Slip/trip/fall injury	• University WH & S issues are enforced across all campus labs (loose wires are secured, chairs have designated spaces, sharp hazards are minimised or identified etc.) • Laboratory regulations are publicly displayed in all labs	• Bags/external obstacles are placed in safe secure spaces (under desks) • Following standard lab safety protocol (no running etc.)	*2*	*D*	*L*	*No cost other than being alert*	*Y*
Using on electrical equipment (computers)	Electrical incidents (electrocution)	Physical injury	• UNSW have tested and tagged electrical equipment based on the Australian standard AS3760 • Existing system to report any damaged electrical equipment (power cords or power points) to supervising staff for students using labs	• Use residual current devices with electrical equipment (specific extension cords) that reduce risk of electrocution if fault arises	*3*	*E*	*M*	*No additional cost if residual current devices already being employed*	*Y*
Extended use of computers	Straining neck/back/ eyes	Physical injury	• Well-supported chairs supplied in UNSW labs • Follow ergonomic safety at computers information sheet in labs	• Take regular breaks (every 30 minutes) • Sit with good posture	*1*	*C*	*L*	*No cost other than being alert*	*Y*
Running code on NCI high performance cluster	Crashing/ faulting system	Affects other queued jobs and health of cluster	• Limits exist and must be specified when submitting job to run • Existing support exists via ticketing system or by emailing help@nci.org.au if CPU is unable to handle supplied job limit	• Ensure job parameters are accurate for what is being submitted	*1*	*E*	*L*	*Costs some effort to ensure accurate parameter calculation*	*Y*

RISK RATING METHODOLOGY AND MATRIX

Consider the Consequences	Consider the Likelihood	Calculate the Risk
Consider: What type of harm could occur (minor, serious, death)? Is there anything that will influence the severity (e.g. proximity to hazard, person involved in task etc.). How many people are exposed to the hazard? Could one failure lead to other failures? Could a small event escalate?	Consider: How often is the task done? Has an accident happened before (here or at another workplace)? How long are people exposed? How effective are the control measures? Does the environment effect it (e.g. lighting/temperature/pace)? What are people's behaviours (e.g. stress, panic, deadlines) What people are exposed (e.g. disabled, young workers etc.)?	1.Take the consequences rating and select the correct column 2.Take the likelihood rating and select the correct row 3. Select the risk rating where the two ratings cross on the matrix below. **VH = Very high, H = High, M = Medium, L = Low**
5. Severe: death or permanent disability to one or more persons **4. Major:** hospital admission required **3. Moderate**: medical treatment required **2. Minor**: first aid required **1. Insignificant**: injuries not requiring first aid	**A. Almost certain**: expected to occur in most circumstances **B. Likely**: will probably occur in most circumstances **C. Possible**: might occur occasionally **D. Unlikely:** could happen at some time **E. Rare:** may happen only in exceptional circumstances	

		CONSEQUENCES				
		1	2	3	4	5
LIKELIHOOD	A	M	H	H	VH	VH
	B	M	M	H	H	VH
	C	L	M	H	H	VH
	D	L	L	M	M	H
	E	L	L	M	M	M

Risk level	Required action
Very high	**Act immediately**: The proposed task or process activity must not proceed. Steps must be taken to lower the risk level to as low as reasonably practicable using the hierarchy of risk controls
High	**Act today:** The proposed activity can only proceed, provided that: (i) the risk level has been reduced to as low as reasonably practicable using the hierarchy of risk controls and (ii) the risk controls must include those identified in legislation, Australian Standards, Codes of Practice etc. and (iii) the document has been reviewed and approved by the Supervisor and (iv) a Safe Working Procedure or Safe Work Method has been prepared and (v) the supervisor must review and document the effectiveness of the implemented risk controls
Medium	**Act this week**: The proposed task or process can proceed, provided that: (i) the risk level has been reduced to as low as reasonably practicable using the hierarchy of controls and (ii) the document has been reviewed and approved by the Supervisor and (iii) a Safe Working Procedure or Safe Work Method has been prepared.
Low	**Act this month:** Managed by local documented routine procedures which must include application of the hierarchy of controls.

List emergency procedures and controls

List emergency controls for how to deal with fires, spills or exposure to hazardous substances and/or emergency shutdown procedures

- Go to emergency evacuation point on campus (refer to campus map)
- Use fire exits instead of lifts in case of emergency
- Leave in an orderly manner and ensure safety first rather than retrieving
- Follow “Emergency on Campus” instructions publicly displayed in labs
- For hazardous spills, if a major spill, evacuate. Else, work in pairs to clean up spill (as per “Emergency on Campus”)

Implementation			
Additional control measures needed:	**Resources required**	**Responsible person**	**Date of implementation**
N/A			

REVIEW			
Scheduled review date:			
Are all control measures in place?			
Are controls eliminating or minimising the risk?			
Are there any new problems with the risk?			
Review by: (name)			
Review date:			

Acknowledgement of Understanding

All persons performing these tasks must sign that they have read and understood the risk management (as described in HS329 Risk Management Procedure).

Note: for activities which are low risk or include a large group of people (e.g. open days, BBQ’s, student classes etc), only the persons undertaking the key activities need to sign below. For all others involved in such activities, the information can be covered by other methods including for example a safety briefing, induction, and/or safety information sheet (ensure the method of communicating this information is specified here)

Risk management name and version number:	**I have read and understand this risk management form**

Name	Signature	Date
J. Smith	J Smith	21.03.19
B. Ahmed	B Ahmed	17/04/19

Appendix 2 - Investigation Phase

Maximum Activation Test

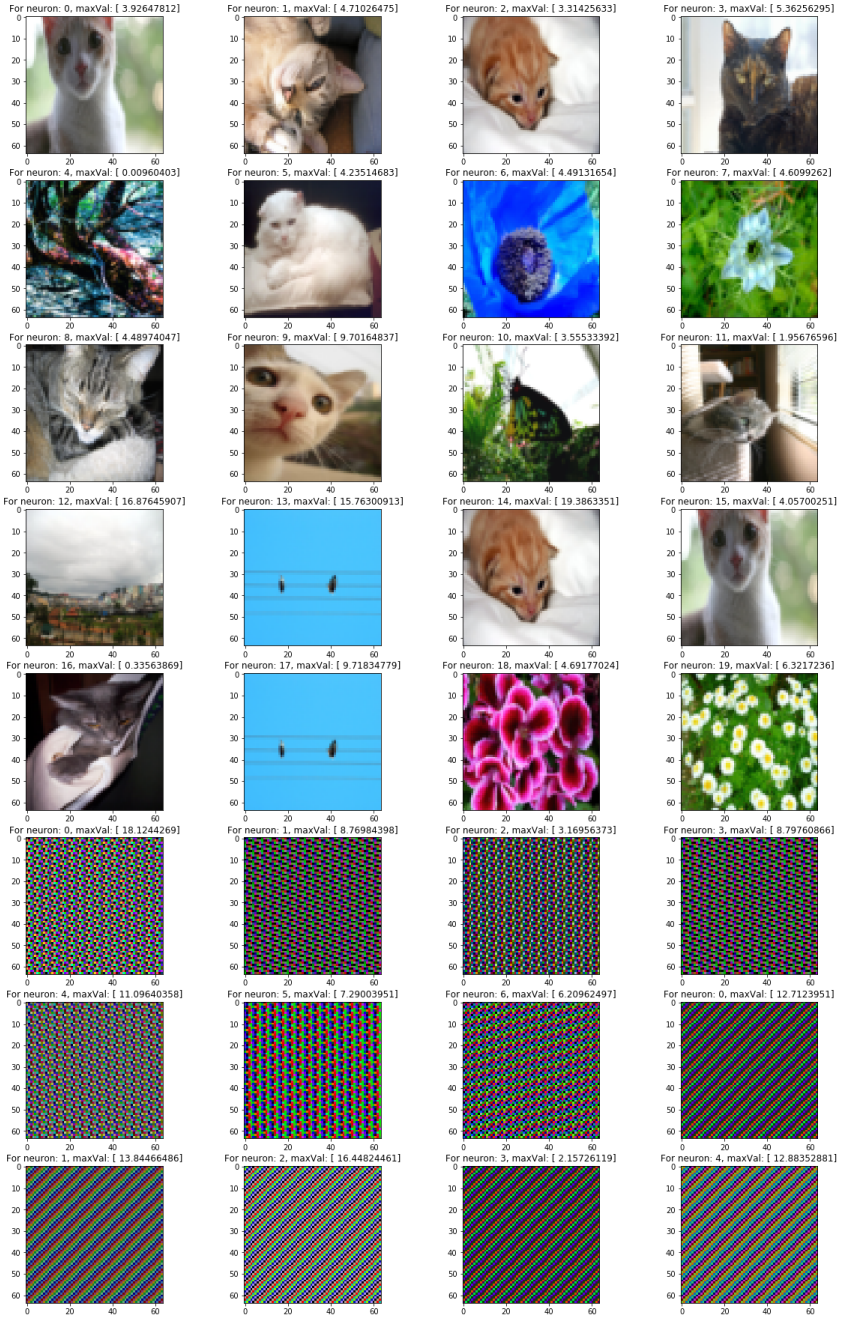


Figure 5: Input image for each individual neuron that produced the maximum activation.

'Reverse Engineering' Test

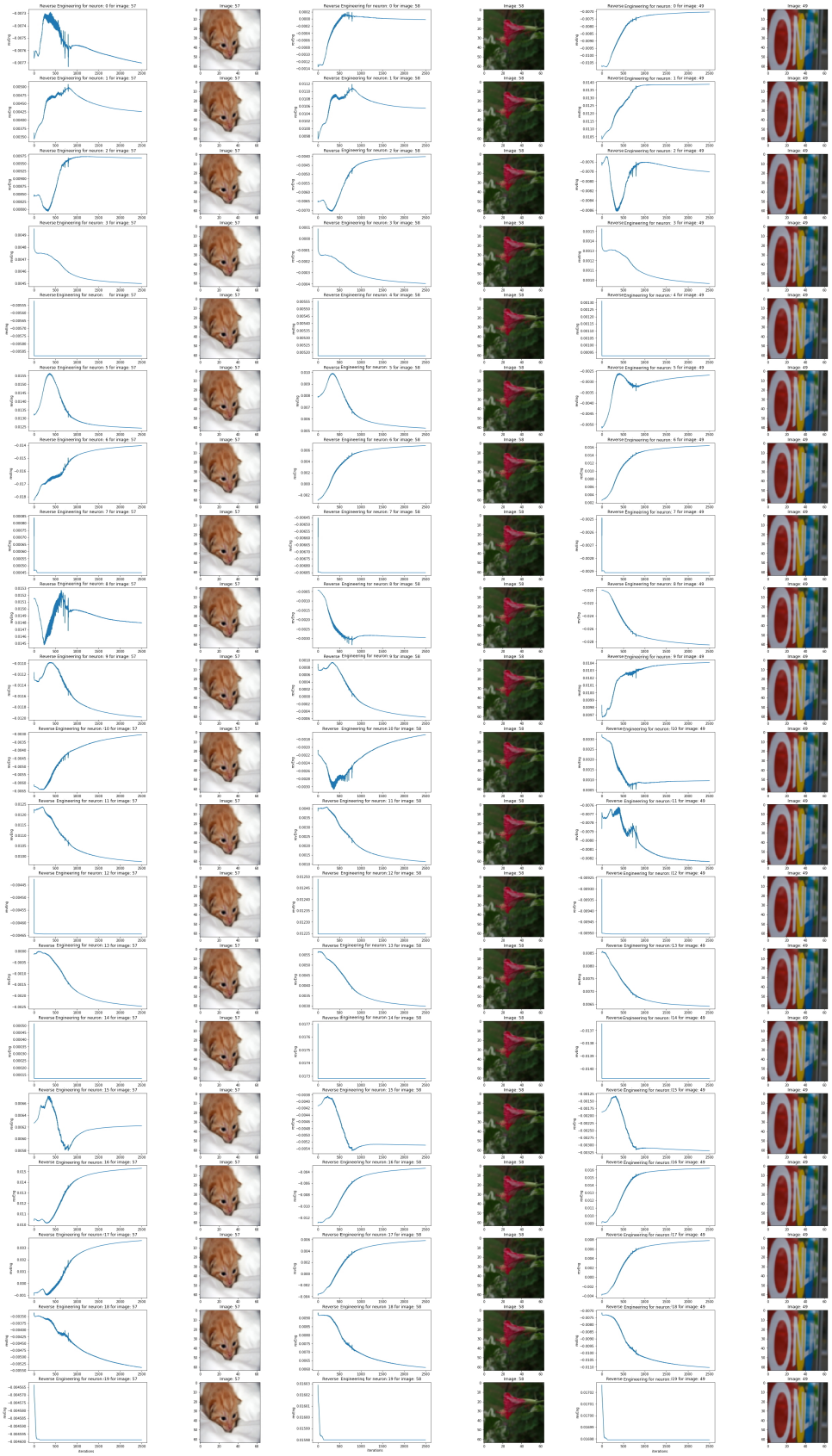
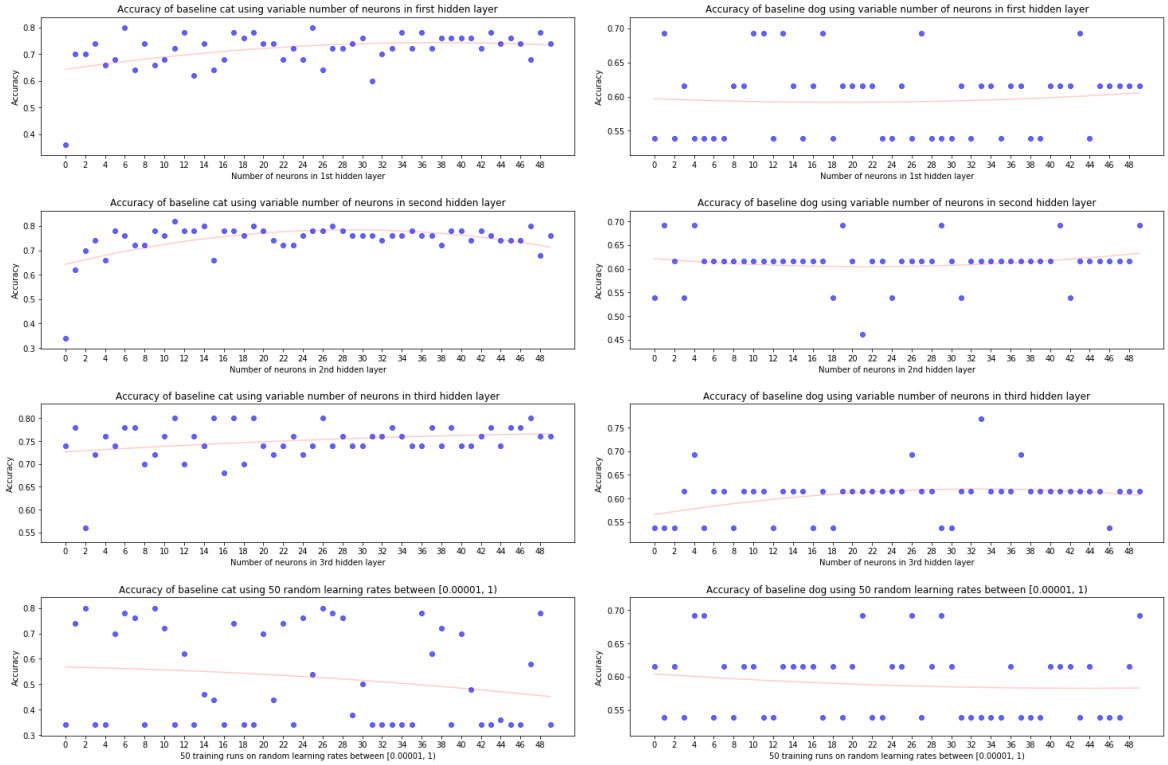
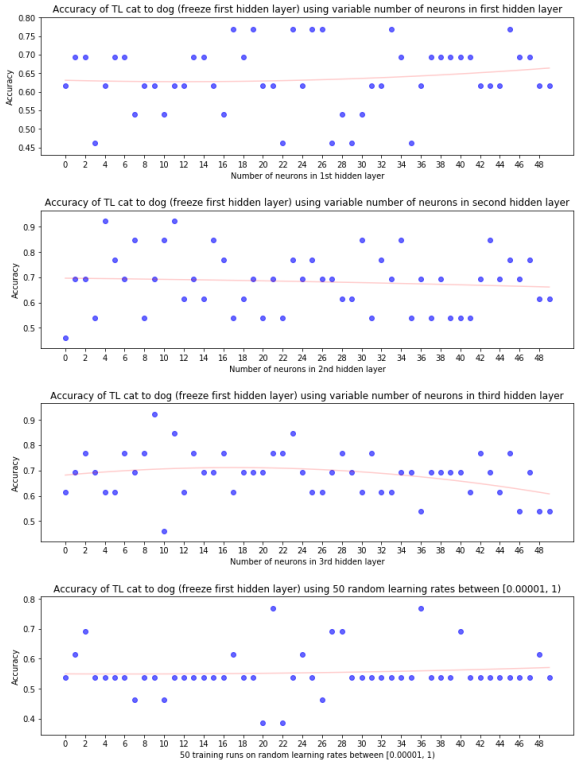


Figure 6: 'Reverse engineering' test results. Neurons 4, 7, 12, 14 and 19 are deactivated.

Full sweep test results for baseline cat, dog and TL cat to dog

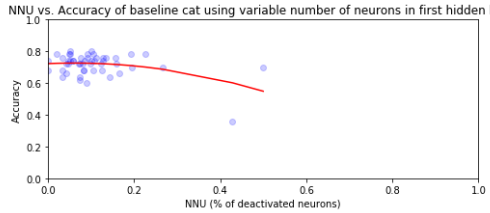


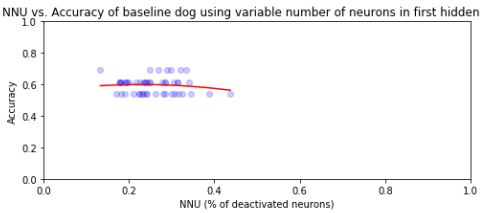
(a) Baseline cat model (b) Baseline dog model

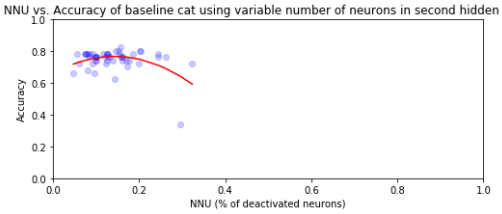


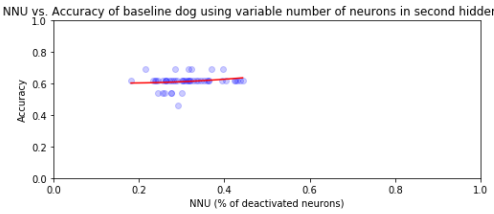
(c) TL cat to dog model

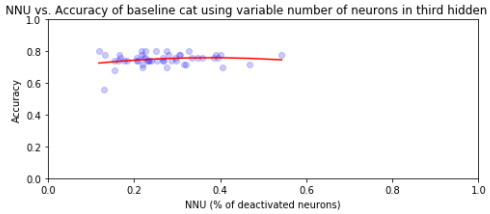
Figure 7: Comparing baseline cat, baseline dog and TL cat to dog models for accuracy using all four sweep tests.

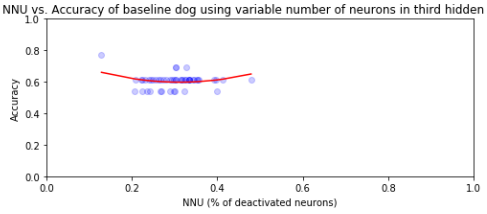


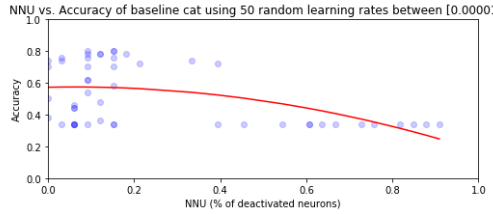


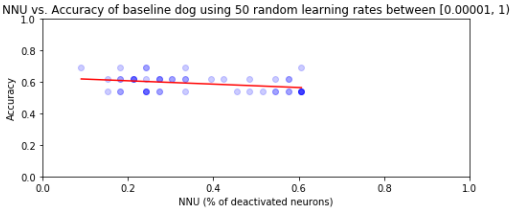






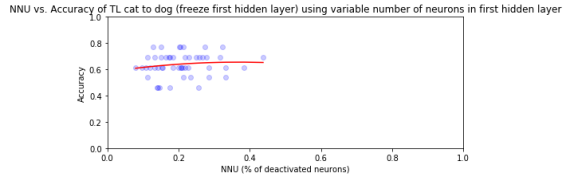


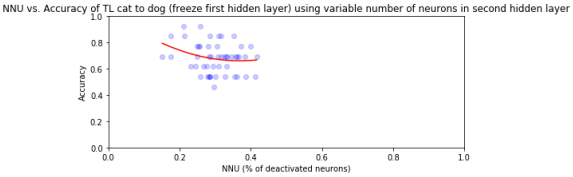


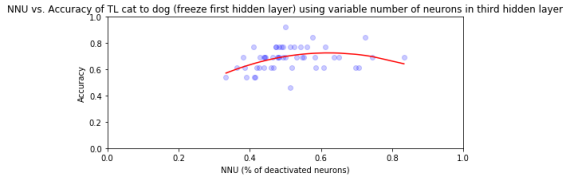


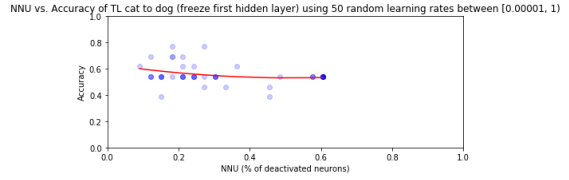
(a) Baseline cat model

(b) Baseline dog model









(c) TL cat to dog model

Figure 8: Comparing baseline cat, baseline dog and TL cat to dog models for accuracy using all four sweep tests.

Appendix 3 - Python Code For Exploration Phase

Detailed Model Structures

Macro-Models Model Summaries

Model 1 Code

```
# Model 1 - Basic CNN
model = Sequential()

model.add(Conv2D(16, kernel_size=(3, 3), activation='relu',
                 input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dense(1, activation='sigmoid'))


model.compile(loss='binary_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])

model.summary()
```

Model 1 Summary

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 148, 148, 16)	448
max_pooling2d_1 (MaxPooling2	(None, 74, 74, 16)	0
conv2d_2 (Conv2D)	(None, 72, 72, 64)	9280
max_pooling2d_2 (MaxPooling2	(None, 36, 36, 64)	0
conv2d_3 (Conv2D)	(None, 34, 34, 128)	73856
max_pooling2d_3 (MaxPooling2	(None, 17, 17, 128)	0
flatten_1 (Flatten)	(None, 36992)	0
dense_1 (Dense)	(None, 512)	18940416
dense_2 (Dense)	(None, 1)	513

Total params: 19,024,513
Trainable params: 19,024,513
Non-trainable params: 0

Model 2 Code

```
# Model 2 - CNN with regularization and image augmentation

# Develop augmented image data to add additional data from existing ddata
train_datagen = ImageDataGenerator(rescale=1./255, zoom_range=0.3, rotation_range=50,
                                   width_shift_range=0.2, height_shift_range=0.2, shear_range=0.2,
                                   horizontal_flip=True, fill_mode='nearest')

val_datagen = ImageDataGenerator(rescale=1./255)


img_id = 1
cat_generator = train_datagen.flow(train_imgs[img_id:img_id+1], train_labels[img_id:img_id+1],
                                   batch_size=1)
cat = [next(cat_generator) for i in range(0,5)]
fig, ax = plt.subplots(1,5, figsize=(16, 6))
print('Labels:', [item[1][0] for item in cat])
l = [ax[i].imshow(cat[i][0][0]) for i in range(0,5)]

# Load data generators
train_generator = train_datagen.flow(train_imgs, train_labels_enc, batch_size=30)
val_generator = val_datagen.flow(validation_imgs, validation_labels_enc, batch_size=20)
input_shape = (150, 150, 3)

# Define structure
model = Sequential()

model.add(Conv2D(16, kernel_size=(3, 3), activation='relu',
                 input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-4),
              metrics=['accuracy'])

model.summary()
```

Model 2 Summary

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 148, 148, 16)	448
max_pooling2d_1 (MaxPooling2	(None, 74, 74, 16)	0
conv2d_2 (Conv2D)	(None, 72, 72, 64)	9280
max_pooling2d_2 (MaxPooling2	(None, 36, 36, 64)	0
conv2d_3 (Conv2D)	(None, 34, 34, 128)	73856
max_pooling2d_3 (MaxPooling2	(None, 17, 17, 128)	0
conv2d_4 (Conv2D)	(None, 15, 15, 128)	147584
max_pooling2d_4 (MaxPooling2	(None, 7, 7, 128)	0
flatten_1 (Flatten)	(None, 6272)	0
dense_1 (Dense)	(None, 512)	3211776
dropout_1 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 512)	262656
dropout_2 (Dropout)	(None, 512)	0
dense_3 (Dense)	(None, 1)	513

Total params: 3,706,113
Trainable params: 3,706,113
Non−trainable params: 0

VGG-16 Model Summary

	Layer Type	Layer Name	Layer Trainable
0	<keras.engine.input_layer.InputLayer object at 0x0000017A55CD9D30>	input_1	False
1	<keras.layers.convolutional.Conv2D object at 0x0000017A6133EBA8>	block1_conv1	False
2	<keras.layers.convolutional.Conv2D object at 0x0000017A6133E668>	block1_conv2	False
3	<keras.layers.pooling.MaxPooling2D object at 0x0000017AF0CA3128>	block1_pool	False
4	<keras.layers.convolutional.Conv2D object at 0x0000017AB01FC390>	block2_conv1	False
5	<keras.layers.convolutional.Conv2D object at 0x0000017AF0DAB0F0>	block2_conv2	False
6	<keras.layers.pooling.MaxPooling2D object at 0x0000017AF0DAB9B0>	block2_pool	False
7	<keras.layers.convolutional.Conv2D object at 0x0000017AF0DAB940>	block3_conv1	False
8	<keras.layers.convolutional.Conv2D object at 0x0000017AF0DF5278>	block3_conv2	False
9	<keras.layers.convolutional.Conv2D object at 0x0000017AF0DF5F98>	block3_conv3	False
10	<keras.layers.pooling.MaxPooling2D object at 0x0000017AF0E29F98>	block3_pool	False
11	<keras.layers.convolutional.Conv2D object at 0x0000017AF0E29EF0>	block4_conv1	False
12	<keras.layers.convolutional.Conv2D object at 0x0000017AF0E5F2E8>	block4_conv2	False
13	<keras.layers.convolutional.Conv2D object at 0x0000017AF0E77BA8>	block4_conv3	False
14	<keras.layers.pooling.MaxPooling2D object at 0x0000017AF0E8E9E8>	block4_pool	False
15	<keras.layers.convolutional.Conv2D object at 0x0000017AF0E8E8D0>	block5_conv1	False
16	<keras.layers.convolutional.Conv2D object at 0x0000017AF0EC3710>	block5_conv2	False
17	<keras.layers.convolutional.Conv2D object at 0x0000017AF0EDF518>	block5_conv3	False
18	<keras.layers.pooling.MaxPooling2D object at 0x0000017AF0EF89B0>	block5_pool	False
19	<keras.layers.core.Flatten object at 0x0000017A6133EB70>	flatten_2	False

Model 3 Code

```
# Model 3 - CNN with pre-trained feature extractor
input_shape = vgg_model.output_shape[1]

model = Sequential()
model.add(InputLayer(input_shape=(input_shape,)))
model.add(Dense(512, activation='relu', input_dim=input_shape))
model.add(Dropout(0.3))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-5),
              metrics=['accuracy'])

model.summary()
```

Model 3 Summary

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 512)	4194816
dropout_1 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 512)	262656
dropout_2 (Dropout)	(None, 512)	0
dense_3 (Dense)	(None, 1)	513

Total params: 4,457,985
Trainable params: 4,457,985
Non-trainable params: 0

Model 4 Code

```
# Model 4 - CNN with pre-trained feature extractor (with image augmentation)
# Augment image data
train_datagen = ImageDataGenerator(rescale=1./255, zoom_range=0.3, rotation_range=50,
                                   width_shift_range=0.2, height_shift_range=0.2, shear_range=0.2,
                                   horizontal_flip=True, fill_mode='nearest')

val_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow(train_imgs, train_labels_enc, batch_size=30)
val_generator = val_datagen.flow(validation_imgs, validation_labels_enc, batch_size=20)

model = Sequential()
model.add(vgg_model)
model.add(Dense(512, activation='relu', input_dim=input_shape))
model.add(Dropout(0.3))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=2e-5),
              metrics=['accuracy'])

model.summary()
```

Model 4 Summary

Layer (type)	Output Shape	Param #
model_1 (Model)	(None, 8192)	14714688
dense_7 (Dense)	(None, 512)	4194816
dropout_5 (Dropout)	(None, 512)	0
dense_8 (Dense)	(None, 512)	262656
dropout_6 (Dropout)	(None, 512)	0
dense_9 (Dense)	(None, 1)	513

Total params: 19,172,673
Trainable params: 4,457,985
Non-trainable params: 14,714,688

Model 5 Code

```
# Model 5 - CNN with pre-trained feature extractor (with image augmentation and fine-tuning)
vgg_model.trainable = True

set_trainable = False
for layer in vgg_model.layers:
    if layer.name in ['block5_conv1', 'block4_conv1']:
        set_trainable = True
    if set_trainable:
        layer.trainable = True
    else:
        layer.trainable = False

layers = [(layer, layer.name, layer.trainable) for layer in vgg_model.layers]
pd.DataFrame(layers, columns=['Layer Type', 'Layer Name', 'Layer Trainable'])

train_datagen = ImageDataGenerator(rescale=1./255, zoom_range=0.3, rotation_range=50,
                                   width_shift_range=0.2, height_shift_range=0.2, shear_range=0.2,
                                   horizontal_flip=True, fill_mode='nearest')
val_datagen = ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow(train_imgs, train_labels_enc, batch_size=30)
val_generator = val_datagen.flow(validation_imgs, validation_labels_enc, batch_size=20)

model = Sequential()
model.add(vgg_model)
model.add(Dense(512, activation='relu', input_dim=input_shape))
model.add(Dropout(0.3))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-5),
              metrics=['accuracy'])
```

Model 5 Summary

Layer (type)	Output Shape	Param #
model_1 (Model)	(None, 8192)	14714688
dense_10 (Dense)	(None, 512)	4194816
dropout_7 (Dropout)	(None, 512)	0
dense_11 (Dense)	(None, 512)	262656
dropout_8 (Dropout)	(None, 512)	0
dense_12 (Dense)	(None, 1)	513

Total params: 19,172,673
Trainable params: 17,437,185
Non-trainable params: 1,735,488

Micro-Models Model Summaries

Model 5a Code

```
# Model 5a - CNN with pre-trained feature extractor (with fine-tuning) -> altered neurons in dense layers
vgg_model.trainable = True

set_trainable = False
for layer in vgg_model.layers:
    if layer.name in ['block5_conv1', 'block4_conv1']:
        set_trainable = True
    if set_trainable:
        layer.trainable = True
    else:
        layer.trainable = False

layers = [(layer, layer.name, layer.trainable) for layer in vgg_model.layers]
pd.DataFrame(layers, columns=['Layer Type', 'Layer Name', 'Layer Trainable'])

train_datagen = ImageDataGenerator(rescale=1./255, zoom_range=0.3, rotation_range=50,
                                   width_shift_range=0.2, height_shift_range=0.2, shear_range=0.2,
                                   horizontal_flip=True, fill_mode='nearest')
val_datagen = ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow(train_imgs, train_labels_enc, batch_size=30)
val_generator = val_datagen.flow(validation_imgs, validation_labels_enc, batch_size=20)

model = Sequential()
model.add(vgg_model)
model.add(Dense(600, activation='relu', input_dim=input_shape))
model.add(Dropout(0.3))
model.add(Dense(400, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-5),
              metrics=['accuracy'])
model.summary()
```

Model 5a Summary

Layer (type)	Output Shape	Param #
model_1 (Model)	(None, 8192)	14714688
dense_4 (Dense)	(None, 600)	4915800
dropout_3 (Dropout)	(None, 600)	0
dense_5 (Dense)	(None, 400)	240400
dropout_4 (Dropout)	(None, 400)	0
dense_6 (Dense)	(None, 1)	401

Total params: 19,871,289
Trainable params: 18,135,801
Non-trainable params: 1,735,488

Model 5b Code

```
# Model 5b - CNN with pre-trained feature extractor (with fine-tuning) => additional layer
vgg_model.trainable = True

set_trainable = False
for layer in vgg_model.layers:
    if layer.name in ['block5_conv1', 'block4_conv1']:
        set_trainable = True
    if set_trainable:
        layer.trainable = True
    else:
        layer.trainable = False

layers = [(layer, layer.name, layer.trainable) for layer in vgg_model.layers]
pd.DataFrame(layers, columns=['Layer Type', 'Layer Name', 'Layer Trainable'])

train_datagen = ImageDataGenerator(rescale=1./255, zoom_range=0.3, rotation_range=50,
                                   width_shift_range=0.2, height_shift_range=0.2, shear_range=0.2,
                                   horizontal_flip=True, fill_mode='nearest')
val_datagen = ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow(train_imgs, train_labels_enc, batch_size=30)
val_generator = val_datagen.flow(validation_imgs, validation_labels_enc, batch_size=20)

model = Sequential()
model.add(vgg_model)
model.add(Dense(512, activation='relu', input_dim=input_shape))
model.add(Dropout(0.3))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-5),
              metrics=['accuracy'])
model.summary()
```

Model 5b Summary

Layer (type)	Output Shape	Param #
model_1 (Model)	(None, 8192)	14714688
dense_7 (Dense)	(None, 512)	4194816
dropout_5 (Dropout)	(None, 512)	0
dense_8 (Dense)	(None, 512)	262656
dropout_6 (Dropout)	(None, 512)	0
dense_9 (Dense)	(None, 512)	262656
dropout_7 (Dropout)	(None, 512)	0
dense_10 (Dense)	(None, 1)	513

Total params: 19,435,329
Trainable params: 17,699,841
Non-trainable params: 1,735,488

Model 5c Code

```
# Model 5c - CNN with pre-trained feature extractor (with fine-tuning) => changed LR
vgg_model.trainable = True

set_trainable = False
for layer in vgg_model.layers:
    if layer.name in ['block5_conv1', 'block4_conv1']:
        set_trainable = True
    if set_trainable:
        layer.trainable = True
    else:
        layer.trainable = False

layers = [(layer, layer.name, layer.trainable) for layer in vgg_model.layers]
pd.DataFrame(layers, columns=['Layer Type', 'Layer Name', 'Layer Trainable'])

train_datagen = ImageDataGenerator(rescale=1./255, zoom_range=0.3, rotation_range=50,
                                   width_shift_range=0.2, height_shift_range=0.2, shear_range=0.2,
                                   horizontal_flip=True, fill_mode='nearest')
val_datagen = ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow(train_imgs, train_labels_enc, batch_size=30)
val_generator = val_datagen.flow(validation_imgs, validation_labels_enc, batch_size=20)

model = Sequential()
model.add(vgg_model)
model.add(Dense(512, activation='relu', input_dim=input_shape))
model.add(Dropout(0.3))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=2e-5),
              metrics=['accuracy'])
```

Model 5c Summary

Layer (type)	Output Shape	Param #
model_1 (Model)	(None, 8192)	14714688
dense_10 (Dense)	(None, 512)	4194816
dropout_7 (Dropout)	(None, 512)	0
dense_11 (Dense)	(None, 512)	262656
dropout_8 (Dropout)	(None, 512)	0
dense_12 (Dense)	(None, 1)	513

Total params: 19,172,673
Trainable params: 17,437,185
Non-trainable params: 1,735,488

Exploration Metric Tests

Imports

```
#!/usr/bin/env python
# coding: utf-8

"""
# Final Large Scale TL Experiment For Thesis C #
# Author: Joel Smith z5076397

After completing several small scale investigations into TL for image classification, building a DNN from scratch using numpy, a larger scale investigation will act as the final experiment for the thesis project.

This larger scale investigation is based off
https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in
-deep-learning-212bf3b2f27a where the famous 'Dog vs. Cat' dataset will be used. The dataset contains 25,000 images of dogs and cats
    to be applied on the pretrained VGG-16 model trained on the ImageNet database.
"""

# Import methods created by author
import tl_metric_lib as tml

# # Basic imports
import glob
import numpy as np
import os
import math
import re

# # Visualisations
import matplotlib.pyplot as plt
from keras.preprocessing.image import ImageDataGenerator, load_img, img_to_array, array_to_img
from IPython.display import display
from PIL import Image

# # Pre-trained models
from keras.applications import vgg16
from keras.models import Model
import keras
import pandas as pd

# CNN methods
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, InputLayer, Lambda
from keras.models import Sequential
from keras import optimizers

# # Evaluating models
from keras.models import load_model
import model_evaluation_utils as meu
from sklearn import metrics
import keras.backend as K
```

Global definitions

```
## Load all models and parameters ##
input_shape = (150, 150, 3)
IMG_DIM = (150, 150)

# load saved models
basic_cnn = load_model('cats_dogs_basic_cnn.h5')
img_aug_cnn = load_model('cats_dogs_cnn_img_aug.h5')
tl_cnn = load_model('cats_dogs_tlearn_basic_cnn.h5')
tl_img_aug_cnn = load_model('cats_dogs_tlearn_img_aug_cnn.h5')
tl_img_aug_finetune_cnn = load_model('cats_dogs_tlearn_finetune_img_aug_cnn.h5')

# load other configurations
IMG_DIM = (150, 150)
input_shape = (150, 150, 3)
num2class_label_transformer = lambda l: ['cat' if x == 0 else 'dog' for x in l]
class2num_label_transformer = lambda l: [0 if x == 'cat' else 1 for x in l]

# Use a pre-trained model as a feature extractor and fine tune
vgg = vgg16.VGG16(include_top=False, weights='imagenet',
                                        input_shape=input_shape)

output = vgg.layers[-1].output
output = keras.layers.Flatten()(output)
vgg_model = Model(vgg.input, output)

vgg_model.trainable = False
for layer in vgg_model.layers:
    layer.trainable = False

pd.set_option('max_colwidth', -1)
layers = [(layer, layer.name, layer.trainable) for layer in vgg_model.layers]
pd.DataFrame(layers, columns=['Layer Type', 'Layer Name', 'Layer Trainable'])

# Load and prepare dataset
test_files = glob.glob('test_data/*')
test_imgs = [img_to_array(load_img(img, target_size=IMG_DIM)) for img in test_files]
test_imgs = np.array(test_imgs)
print(test_files[0])
test_labels = [fn.split('\\')[1].split('.')[0].strip() for fn in test_files]

test_imgs_scaled = test_imgs.astype('float32')
test_imgs_scaled /= 255
test_labels_enc = class2num_label_transformer(test_labels)
```

NNU Methods

```
"""
Metric Methods
"""
def NNU_comparison(models, model_names=[], accuracy_results=[]):
    """
    Calculates NNU for all provided models

    Arguments:
    models -- list of pre-trained keras models
    model_names -- list of strings of identifying names for models
    accuracy_results -- list of floats of high-level accuracy of models

    Returns:
    NNU_results -- NNU metric of all models in form [(deactivated_neurons, total_neurons), ...]
    """
    test_bottleneck_features = tml.get_bottleneck_features(vgg_model, test_imgs_scaled)

    results = []
    for i, preloaded_model in enumerate(models):
        print("Testing model {}".format(i))

        # Get output at each layer
        indexes = tml.layer_index_for_activations(preloaded_model)
        model = tml.create_activations_model(preloaded_model, indexes)

        # get activations
        layer_activations = tml.compute_activations(model,i, vgg_model, test_imgs_scaled)

        # Analyse per neuron if activation is 'deactivated'
        deactivated, total_activations = tml.analyse_for_deactivation(layer_activations)
        input_size = layer_activations[0].shape[0]

        # Neuron is deactivated if the neuron is zeroed for all provided input. Therefore should equal len(input) in dict
        NNU_results = tml.find_true_deactivation_results(deactivated, total_activations, input_size, results)

    if not (model_names):
        model_names = [str(i) for i in range(len(models))]

    if not (model_names):
        model_names = ["unknown" for i in range(len(models))]

    print_NNU_results(NNU_results, model_names, accuracy_results)

    return NNU_results
```

AS Methods

```
def activation_spectrum(models, model_names=[]):
    """
    Visualises AS and stores graphs for all provided models

    Arguments:
    models -- list of pre-trained keras models
    model_names -- list of strings of identifying names for models
    """

    results = []
    if not (model_names):
        model_names = [str(i) for i in range(len(models))]

    for i, preloaded_model in enumerate(models):
        print("Testing model {} ...".format(i))
        # Get output at each layer
        layer_indexes = tml.layer_index_for_activations(preloaded_model)
        model = tml.create_activations_model(preloaded_model,layer_indexes)

        # get activations
        layer_activations = tml.compute_activations(model,i,vgg_model, test_imgs_scaled)

        input_size = layer_activations[0].shape[0]

        # find maxActivations
        max_activations = tml.analyse_for_activation_range(layer_activations)

        # Plot spectrums
        tml.plot_activation_spectrum(layer_activations, max_activations, separate=False, name=model_names[i])
```

Deactivation Deletion Methods

```
def deactivation_deletion(model):
    """
    Given a model, identify the deactivated neurons, 'delete' them from the network and
    retrain model

    Arguments:
    model -- pretrained keras model

    Return:
    deac_model -- trained model with deactivated neurons removed
    """

    preloaded_model = model

    # Get output at each layer
    layer_indexes = tml.layer_index_for_activations(preloaded_model)
    model = tml.create_activations_model(preloaded_model,layer_indexes)

    # get activations
    layer_activations = tml.compute_activations(model,0,vgg_model, test_imgs_scaled)

    input_size = layer_activations[0].shape[0]

    # Analyse per neuron if activation is 'deactivated'
    deactivated, total_activations = tml.analyse_for_deactivation(layer_activations)

    # Neuron is deactivated if the neuron is zeroed for all provided input. Therefore should equal len(input) in dict
```

```
    deac_results = tml.find_true_deactivation_results(deactivated, input_size)

    # create masks from NNU comparison
    mask1_val = np.ones((1,model.layers[1].output_shape[-1]))
    mask2_val = np.ones((1,model.layers[2].output_shape[-1]))
    mask3_val = np.ones((1,model.layers[3].output_shape[-1]))
    masks = [mask1_val, mask2_val, mask3_val]

    for r in deac_results:
        _, layer, neuron = r
        masks[layer][0][neuron] = 0

    mask1 = K.variable(masks[0])
    mask2 = K.variable(masks[1])
    mask3 = K.variable(masks[2])

    # retrain model after 'deleting' neurons from training
    deac_model = tml.train_best_model_with_deleted_deactivations(vgg_model, mask1, mask2, mask3)

    # identify increase/decrease in performance
    return deac_model



# Deactivation Deletion Results
def deactivation_deletion_analysis(deac_model, source_model):
    """
    Given two trained models - a source model and the source model retrained with deactivated neurons removed -
    analyse and compare high-level results

    Arguments:
    deac_model -- pretrained keras model of source_model with deactivated neurons removed
    source_model -- pretrained keras model
    """

    predictions = deac_model.predict_classes(test_imgs_scaled, verbose=0)
    predictions = num2class_label_transformer(predictions)
    meu.display_model_performance_metrics(true_labels=test_labels, predicted_labels=predictions,
                                          classes=list(set(test_labels)))

    predictions = source_model.predict_classes(test_imgs_scaled, verbose=0)
    predictions = num2class_label_transformer(predictions)
    meu.display_model_performance_metrics(true_labels=test_labels, predicted_labels=predictions,
                                          classes=list(set(test_labels)))

    # Deac model
    meu.plot_model_roc_curve(deac_model, test_imgs_scaled,
                             true_labels=test_labels_enc,
                             class_names=[0, 1])

    # previously best model (original) - transfer learning with fine-tuning & image augmentation
    meu.plot_model_roc_curve(source_model, test_imgs_scaled,
                             true_labels=test_labels_enc,
                             class_names=[0, 1])
```

AR Methods

```
def activation_range(models, model_names=[]):
    """
    Visualises AR for all provided models

    Arguments:
    models -- list of pre-trained keras models
    model_names -- list of strings of identifying names for models
    """

    results = []
    test_bottleneck_features = tml.get_bottleneck_features(vgg_model, test_imgs_scaled[0:99])
    for i, preloaded_model in enumerate(models):
        print("Testing model {}".format(i))

        # Get output at each layer
        layer_indexes = tml.layer_index_for_activations(preloaded_model)
        model = tml.create_activations_model(preloaded_model,layer_indexes)

        # get activations
        layer_activations = tml.compute_activations(model,i,vgg_model, test_imgs_scaled)

        input_size = layer_activations[0].shape[0]

        # Find maximum activation
        model_max_activation = tml.analyse_for_activation_range(layer_activations)

        results.append(model_max_activation)

        # print results
        print_AR_results(models, model_names, results, accuracy_results)
    return results
```

Exploration Metrics Method Library

```
def get_bottleneck_features(model, input_imgs):
    """
    Extract bottle neck features of VGG-16 to feed into target CNN
    Arguments:
        model -- vgg-16 model
        input_imgs -- input dataset

    Returns:
        features -- Bottle neck features
    """
    features = model.predict(input_imgs, verbose=0)
    return features

#========================== Activation Analysis Methods ==========================#
def layer_index_for_activations(model):
    """
    Obtain the layer indexes that output activations to be used to create activations model
    This will only be dense, convolution, model (vgg-16 layer) or maxpool layers.
    Arguments:
        model -- keras model

    Returns:
        indexes -- indexes of layers with activations
    """
    indexes = []
    for i, layer in enumerate(model.layers):
        if (re.match(".*dense|.*conv|.*model|.*pool", layer.name)):
            indexes.append(i)
    return indexes

def create_activations_model(model,indexes):
    """
    Creates the 'activations model', which is a keras model with outputs at every layer
    that has activations, allowing visualisation of intermediate activations
    Arguments:
        model -- keras model
        indexes -- indexes of layers with activations

    Returns:
        A keras Model object with outputs defined at given indexes
    """
    outputs = [model.layers[i].get_output_at(-1) for i in indexes]
    return Model(inputs=model.inputs, outputs=outputs)

def compute_activations(model, i, vgg_model, test_imgs_scaled):
    """
    Computes the activations of the activation model

    Arguments:
        model -- keras model
        i -- iteration of models being analysed (useful as Model (3) requires the input to
             bottle-neck features)
        vgg_model -- pretrained VGG-16 model
        test_imgs_scaled -- test-set scaled to be same size

    Returns:
        activations -- model.predict returns a list of numpy arrays for each set of activations
                       across all images of test-set per layer
    """

    if (i == 2): # tl_cnn takes test_bottleneck_features
```

```
        test_bottleneck_features = get_bottleneck_features(vgg_model, test_imgs_scaled)
        return model.predict(test_bottleneck_features)
    else:
        return model.predict(test_imgs_scaled)

def analyse_for_deactivation(layer_activations):
    """
    Analyses the given layer-activations for neurons that are emitting 0 for all test-images (i.e. deactivated)
    Neurons are added as a 'suspect' if for any input they emit 0 activation. True deactivation will occur if
    neurons are found to have 0 for ALL inputs.

    Arguments:
        layer_activations -- a list of numpy arrays for each set of activations
                             across all images of test-set per layer

    Returns:
        deactivated -- A dictionary of all 'suspect' deactivated neurons
        total_activations -- the total number of neurons (or activations) in the network
    """
    deactivated = {}
    total_activations = 0
    for layer_num, activations in enumerate(layer_activations):
        for i, a in enumerate(activations):
            if (a.ndim > 1):
                for c in range(a.shape[-1]):
                    if (np.count_nonzero(a[:,:,c]) == 0):
                        deactivated[("cnn",layer_num,c)] = deactivated.get(("cnn",layer_num,c),0) + 1
            else:
                for x in np.where(a == 0)[0]:
                    deactivated[("dnn",layer_num,x)] = deactivated.get(("dnn",layer_num,x),0) + 1

            # Calculate total activations for this model
            if (activations[0].ndim > 1):
                total_activations += np.product(activations.shape[-1])
            else:
                total_activations += activations.shape[1]
    return deactivated, total_activations


def find_true_deactivation_results(deactivated, total_activations, input_size, NNU_results):
    """
    Takes the 'suspected' deactivated neurons from analyse_for_deactivation and detects which are
    actually deactivated for the entire input size.

    Arguments:
        deactivated -- A dictionary of all 'suspect' deactivated neurons
        total_activations -- the total number of neurons (or activations) in the network
        input_size -- size of input dataset
        NNU_results -- list tracking all of the NNU results of all provided models

    Returns:
        NNU_results -- The NNU results for the given model in form [(deactivated neurons, total neurons), ...]
    """
    completely_deact = []
    for (k,v) in deactivated.items():
        if (v == input_size):
            completely_deact.append(k)

    NNU_results.append((len(completely_deact), total_activations))
    return NNU_results

#================================== AS Methods ==================================#
def plot_activation_spectrum(layer_activations, max_activations, name="model"):
```

```
    """
    Plots the activation spectrum for the given layer-activations of a certain keras model

    Arguments:
        layer_activations -- a list of numpy arrays for each set of activations
                             across all images of test-set per layer
        max_activations -- the maximum activations of each given layer
        name -- name of model
    """

    # Define figure
    plt.figure(figsize=(10,10))
    all_act = []
    all_colors = []

    # Generate color map for differentiating layers
    new_cmap = rand_cmap(len(layer_activations), type='bright', first_color_black=False, last_color_black=False, verbose=True)

    # Simplify data from complex numpy array activation to a flattened activation array for histogram plotting
    for layer_num, activations in enumerate(layer_activations):
        print("Calculating layer {} activation spectrum ...".format(layer_num))
        bins = np.arange(0, math.ceil(maxActivations[layer_num]), math.ceil(maxActivations[layer_num])/10)
        a = np.amax(activations, axis=0)
        if (a.ndim > 1):
            a_flat = np.amax(a, axis=-1)
            color = [list(np.random.random(size=3))] * len(a_flat)
        else:
            a_flat = a
            bins = np.arange(0, math.ceil(maxActivations[layer_num]), math.ceil(maxActivations[layer_num])/100)
            color = list(new_cmap(layer_num))

        # Plot layer on frequency histogram
        _ = plt.hist(a_flat, bins=bins, color=color, label='Layer {}'.format(layer_num))
        plt.title("{} - Activation Spectrum".format(name))
        plt.xlabel("Activations")
        plt.ylabel("Frequency")
    plt.legend()

    # Save plot
    plt.savefig("results/{}-AS-{}.png".format(name, datetime.datetime.now().strftime("%Y%m%d-%H%M%S")))


def train_best_model_with_deleted_deactivations(vgg_model,mask1, mask2, mask3):
    """
    Method for deactivation deletion test. This model takes model (5) and retrains
    given certain masks to 'delete' deactivated neurons

    Arguments:
        vgg_model -- VGG-16 model
        mask1 -- Mask for removing hidden layer 1 of deactivated neurons
        mask2 -- Mask for removing hidden layer 2 of deactivated neurons
        mask3 -- Mask for removing hidden layer 3 of deactivated neurons

    Returns:
        model -- retrained model 5 with deactivated neurons 'deleted'
    """
    vgg_model.trainable = True

    set_trainable = False
    for layer in vgg_model.layers:
        if layer.name in ['block5_conv1', 'block4_conv1']:
            set_trainable = True
        if set_trainable:
```

```
        layer.trainable = True
    else:
        layer.trainable = False

layers = [(layer, layer.name, layer.trainable) for layer in vgg_model.layers]
pd.DataFrame(layers, columns=['Layer Type', 'Layer Name', 'Layer Trainable'])

train_datagen = ImageDataGenerator(rescale=1./255, zoom_range=0.3, rotation_range=50,
                                   width_shift_range=0.2, height_shift_range=0.2, shear_range=0.2,
                                   horizontal_flip=True, fill_mode='nearest')
val_datagen = ImageDataGenerator(rescale=1./255)

# Load datasets
IMG_DIM = (150, 150)

train_files = glob.glob('training_data/*')
train_imgs = [img_to_array(load_img(img, target_size=IMG_DIM)) for img in train_files]
train_imgs = np.array(train_imgs)
train_labels = [fn.split('\\')[1].split('.')[0].strip() for fn in train_files]

validation_files = glob.glob('validation_data/*')
validation_imgs = [img_to_array(load_img(img, target_size=IMG_DIM)) for img in validation_files]
validation_imgs = np.array(validation_imgs)
validation_labels = [fn.split('\\')[1].split('.')[0].strip() for fn in validation_files]

# Scale pixel values from 0-255 to 0-1 for optimizing learning
train_imgs_scaled = train_imgs.astype('float32')
validation_imgs_scaled = validation_imgs.astype('float32')
train_imgs_scaled /= 255
validation_imgs_scaled /= 255

print(train_imgs[0].shape)
array_to_img(train_imgs[200])

# One-hot encode data and configure parameters
batch_size = 30
num_classes = 2
epochs = 30
input_shape = (150, 150, 3)

# encode text category labels
from sklearn.preprocessing import LabelEncoder

# Encode labels
le = LabelEncoder()
le.fit(train_labels)
train_labels_enc = le.transform(train_labels)
validation_labels_enc = le.transform(validation_labels)

# Data generators
train_generator = train_datagen.flow(train_imgs, train_labels_enc, batch_size=30)
val_generator = val_datagen.flow(validation_imgs, validation_labels_enc, batch_size=20)

# Define model
model = Sequential()
model.add(vgg_model)
model.add(Lambda(lambda x: x * mask1))
model.add(Dense(512, activation='relu', input_dim=input_shape))
model.add(Lambda(lambda x: x * mask2))
model.add(Dropout(0.3))
model.add(Dense(512, activation='relu'))
model.add(Lambda(lambda x: x * mask3))
model.add(Dropout(0.3))
```

```
    model.add(Dense(1, activation='sigmoid'))

    model.compile(loss='binary_crossentropy',
                  optimizer=optimizers.RMSprop(lr=1e-5),
                  metrics=['accuracy'])

    history = model.fit_generator(train_generator, steps_per_epoch=100, epochs=100,
                                  validation_data=val_generator, validation_steps=50,
                                  verbose=1)
    return model

#================================== AR Methods ==================================#
def analyse_for_activation_range(layer_activations):
    """
    Analyses each layer's activations to determine the maximum activation per layer
    and hence determine activation range

    Arguments:
        layer_activations -- a list of numpy arrays for each set of activations
                             across all images of test-set per layer

    Returns:
        max_activations -- the maximum activations of each given layer
    """
    max_activation = []
    for layer_num, activations in enumerate(layer_activations):
        currLayerMax = 0
        for i, a in enumerate(activations):
            tempMax = np.amax(a)
            if (currLayerMax < tempMax):
                currLayerMax = tempMax
        max_activation.append(currLayerMax)
    return max_activation

def rand_cmap(nlabels, type='bright', first_color_black=True, last_color_black=False, verbose=True):
    """
    Creates a random colormap to be used together with matplotlib. Useful for segmentation tasks
    Found at: http://stackoverflow.com/questions/14720331/how-to-generate-random-colors-in-matplotlib
    :param nlabels: Number of labels (size of colormap)
    :param type: 'bright' for strong colors, 'soft' for pastel colors
    :param first_color_black: Option to use first color as black, True or False
    :param last_color_black: Option to use last color as black, True or False
    :param verbose: Prints the number of labels and shows the colormap. True or False
    :return: colormap for matplotlib
    """
    from matplotlib.colors import LinearSegmentedColormap
    import colorsys
    import numpy as np


    if type not in ('bright', 'soft'):
        print ('Please choose "bright" or "soft" for type')
        return

    if verbose:
        print('Number of labels: ' + str(nlabels))

    # Generate color map for bright colors, based on hsv
    if type == 'bright':
        randHSVcolors = [(np.random.uniform(low=0.0, high=1),
                          np.random.uniform(low=0.2, high=1),
                          np.random.uniform(low=0.9, high=1)) for i in range(nlabels)]
```

```
        # Convert HSV list to RGB
        randRGBcolors = []
        for HSVcolor in randHSVcolors:
            randRGBcolors.append(colorsys.hsv_to_rgb(HSVcolor[0], HSVcolor[1], HSVcolor[2]))

        if first_color_black:
            randRGBcolors[0] = [0, 0, 0]

        if last_color_black:
            randRGBcolors[-1] = [0, 0, 0]

        random_colormap = LinearSegmentedColormap.from_list('new_map', randRGBcolors, N=nlabels)

    # Generate soft pastel colors, by limiting the RGB spectrum
    if type == 'soft':
        low = 0.6
        high = 0.95
        randRGBcolors = [(np.random.uniform(low=low, high=high),
                          np.random.uniform(low=low, high=high),
                          np.random.uniform(low=low, high=high)) for i in range(nlabels)]

        if first_color_black:
            randRGBcolors[0] = [0, 0, 0]

        if last_color_black:
            randRGBcolors[-1] = [0, 0, 0]
        random_colormap = LinearSegmentedColormap.from_list('new_map', randRGBcolors, N=nlabels)


    return random_colormap
```