UNIVERSITY OF BATH

The University of Bath

Faculty of Engineering and Design

MEng. Integrated Mechanical and Electrical Engineering

CLASSIFICATION USING DEEP CONVOLUTIONAL NEURAL NETWORKS FOR THE IDENTIFICATION OF GENETIC DISORDERS IN DOGS

Date: May 10, 2019

Word Count: 11,890

Author: William Norman Supervisor: Dr Adrian Evans

Assessor: Dr Alan Hunter

Summary

Dogs have been selectively bred for thousands of years. This breeding has led to hundreds of different dog breeds each with their own traits and characteristics. One of the problems of selective breeding is it narrows the gene pool and can lead to dangerous genetic diseases within the breeds. Due to the vast number of breeds and genetic disorders, it can be hard for a owner or vet or breeder to recognise a dog they might not have seen before and identify if it is carrying or experiencing one of these disorders. This report disuses a potential solution of an application that can be used to identify the breed of dog based on a picture and inform the user of which congenital disorders it may be at risk of, along with various information such as how it is treated and symptoms of the disorder.

The main technical challenge behind this application is designing a classifier which can identify the breed or breed mix of the dog. Therefore, this report goes into most detail on this classification operation. There are several different classification techniques that could be used; however, the promising method appears to be the Convolutional Neural Network (CNN). The CNN is deep learning classification technique that is based on the visual cortex in animals. The CNN is implemented using a well labelled large dog image database, and it is found that based more simple architecture it performs reasonably poorly. The performance CNN is then improved using transfer learning, where bottleneck features from deep CNN ImagNet classifiers are used to extract features from the dog images, then are fed to our classifier.

It is found that after the network is trained for a series of epochs, that the classifier has a high performance of 74.8% top 1 accuracy and a top 5 accuracy of 94%. The dog breed classes have different performances and are analysed using a confusion matrix. The recall and precision values for each of the classes are hence found. Future work on this project includes increasing the dataset size as not all dog breeds and cross breeds are included in this dataset. Further work includes full app development on Android and iOS.

CONTENTS

1 Introduction

The aim of this section is to outline the purpose of the project along with the reasoning behind the design of the application. The project goals and milestones are to be described along with the required resources to accomplish the task of producing the application.

1.1 Project Outline

1.1.1 Project Overview

A 2019 study into 100,000 dogs genotyped for 152 genetic variants underlying canine inherited disorders [1] showed that approximately 2 in 5 dogs carried at least 1 copy of a tested disease variant. Also 2% of mixed breed dogs and 5% of pure-bred dogs are at risk of becoming affected by at least 1 of the diseases genotyped in this study. Knowledge of these disorders is of great importance for veterinary care and organisations like kennel clubs, breeding clubs and dog registries. In recent years there have been large advancements in genetic screening technologies which now enable more in-depth understanding and identification of rare canine congenital disorders. More health-related big data is being produced which allows technology to have more of a role in healthcare related environments.

The Fédération Cynologique Internationale (FCI) recognises more than 360 different dog breeds worldwide and there are over 700 [2] inherited disorders and traits that have been described in the domestic dog. This is a large quantity of information for a veterinarian, breeder or owner to keep track of. Therefore, this project aims to tackle the problem by introducing an application, that can be used to identify a particular dog breed or breed mix and inform the user of the various disorders that may affect that breed or breed mix. The incidence risk, disorder symptoms, how the disorder is diagnosed, how it is treated and further information for breeders and veterinarians will also be available through the application.

Dog breed classification is a very complicated task that requires some advanced machine learning techniques. Deep Learning is a sub-field of machine learning inspired by the processes and structure of the human brain. Deep Learning is a state of the art-technique used to extract information directly from images, audio or text. Deep Learning is ideally suited for this task, as classifying dog breeds is a fine-grained classification problem, which means that inter-class variation is often small, as shown in the figures below. In recent years very powerful toolsets and large datasets have become publicly available making classification problems like these now possible.

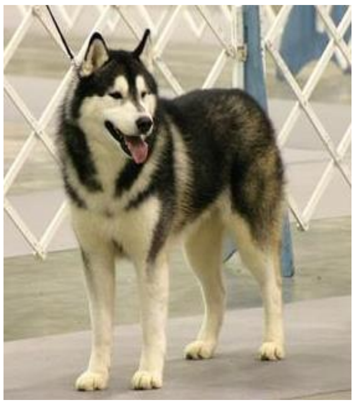


Figure 2 - Siberian Huskey [17]

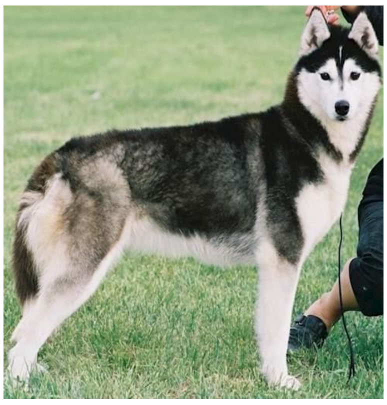


Figure 1 - Eskimo Dog [17]

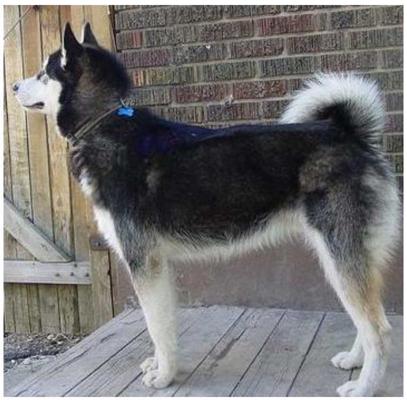


Figure 3 – Malamute [17]

The current method vets use to identify a dog's breed is by swabbing the dog's DNA and sending it off to a laboratory to be analysed. This process can take between two and three weeks to get results. If a deep learner can perform the same task with a comparable accuracy then it has potential to save a lot of time and money. It can also be used as a way of getting a good estimate of the dog breed before confirming it using the DNA sample. For breeders, knowledge of inherited disorders is very important, and having it readily available for the applicable breed would be very useful.

Deep learning techniques have been used for a wide range of tasks such as self-driving cars and for the ImageNet classification competition to classify a wide range of objects. These problems all have large inter-class variation which is less challenging. Classifying dog breeds is a more problematic task due to the inter-class variation and therefore state of the art techniques need to be used to achieve the best performance in this project.

1.1.2 Project Aims

The main goal of this project is to design and produce a classifier that can achieve a high level of accuracy at correctly classifying dog breeds. This would allow the production of an application that can then be used to identify appropriate congenital disorders in dogs. The secondary aim of the project is to produce the application itself along with assembling the congenital disease database. The project goals can be found in the table below. The schedule for the project is outlined using a Gantt chart found in the Appendices.

Milestone Summary	Importance (3-High, 1-Low)	Description
Classifier Design & Investigation	3	Investigation into current classification techniques and high performing fine-grained image classifiers. Design of optimal classifier for this project.
Classifier Implementation	3	Implementation of selected classifier in code using external machine learning libraries and frameworks.
Classifier Evaluation	3	Evaluation of implemented classifier using an appropriate metric, to find classifier performance and under-fitting or over-fitting.
Classifier Optimisation	2	Improvement of classifier performance using new deep learning techniques and hyper-parameter tuning.
Application Design	2	Design of an application that can be used by veterinarians, breeders or owner to classify and identify congenital disorders.
Application Implementation	1	Implementation of the application designed, attention being made into user experience and low latency
Application Evaluation	1	Focus group testing of application to evaluate application performance and potential improvements

Table 1 - Project Milestones

1.2 Resources

1.2.1 Python 3.6

There are a number of different languages with which the classifier can be implemented, and these include Python, Matlab, R and Java. Python is typically considered the best for machine learning programming due to the simplicity of the syntax and the availability of Python libraries which can make tasks easier, one such example is Numpy a library for mathematical computations. Many machine learning frameworks and libraries use Python and therefore it is ideal for this project. The version of Python to be used for implementation is 3.6 to allow for compatibility with the libraries and software currently used. The Integrated Development Environment (IDE) used to edit the code is Spyder which launched through Anaconda. Anaconda is a desktop GUI which allows custom library repositories to be launched with ease, simplifies the package management and makes it ideal for this project.

1.2.2 Hardware and Framework

TensorFlow is a [3] machine learning framework that offers the use of running computations on the systems GPU rather than the CPU. As shown by the figure below running computations on a GPU is considerably quicker and therefore for this project the network is trained and tested using the systems NVIDIA GTX 1080 graphics card, running the CUDA parallel computing platform in conjunction with the NVIDIA CUDA Deep Neural Network Library (cuDNN).

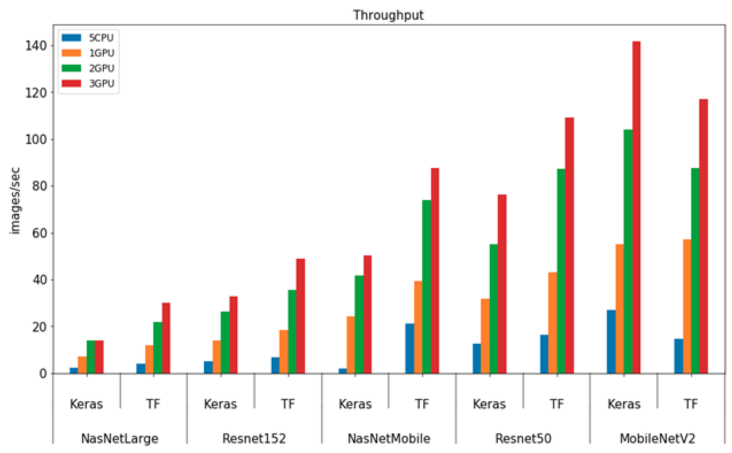


Figure 4 – CPU vs GPU performance Comparison https://azure.microsoft.com/en-us/blog/gpus-vs-cpus-for-deployment-of-deep-learning-models/

1.2.3 Application Implementation

The final application is to be made available on Android and iOS which requires the Android Studio and Xcode platforms for the app development. Training is to be done on a desktop machine, with the mobile hardware used for inference. TensorFlow Lite is used to convert the network into a mobile friendly '.tflite' file which can then be used by the Android and iOS platforms, as shown by the figure below.

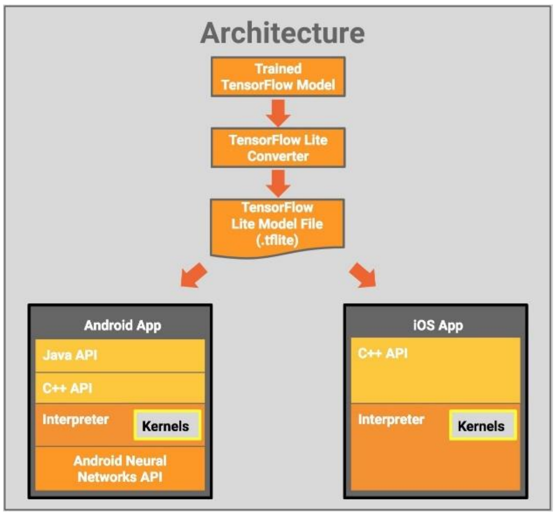


Figure 5 – TensorFlow Application Implementation https://www.tensorflow.org/lite/guide

2 Literature Review

This chapter aims to investigate and summarise the surrounding literature for image classification and the current state of the techniques used to classify images. The different methods are to be analysed and their advantages and limitations compared. The concept of Image classification is introduced in this section and the machine learning techniques such as Support Vector Machines (SVMs), K-Nearest Neighbours (KNNs), Convolutional Neural Networks (CNNs) and Decision Trees (DTs) are investigated to see if they could be applicable for this classification project. The literature around the CNN is then investigated in further detail as it appears the most applicable, this literature includes the architecture of the CNN and the methods that can be used to improve its performance such as deep transfer learning and alternative optimisation methods.

2.1 Image Classification

2.1.1 Context

One of the most important ways for a computer to interact with its environment is machine vision which involves a computer identifying the contents of images or videos then acting on this information. Examples of this include facial recognition [4] in phones used to unlock devices, object tracking for applications such as Hawk Eye officiating aid for sports [5] and object recognition for tasks such as self-driving cars [6]. Image Classification is a process which all these applications use, for sorting the contents of the images into clusters or classes. These classes are then used to inform decisions made by the computer.

Image Classification works by extracting high level information from low level data such as pixel values which can be either grayscale or colour values. Colour images can be represented in several colour space models including RGB (Red, Green, Blue), YUV (Y – Luma Component, UV – Chrominance Component) and HSI (Hue, Saturation, Intensity). RGB is the most typically used as it is the model that monitors use to display colour and so this model will be used to process the images.

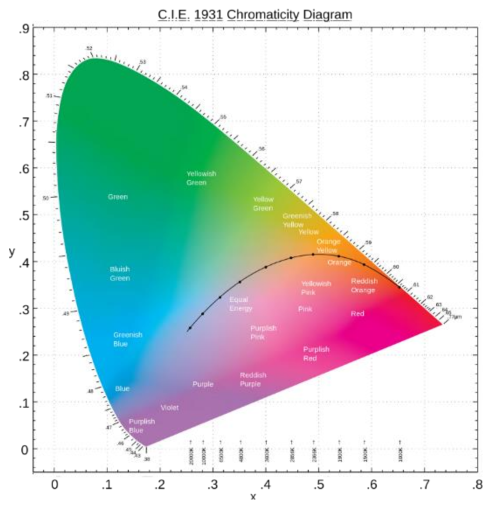


Figure 6 – RGB Chromaticity Diagram https://medium.com/hipster-color-science/a-beginners-guide-to-colorimetry-401f1830b65a

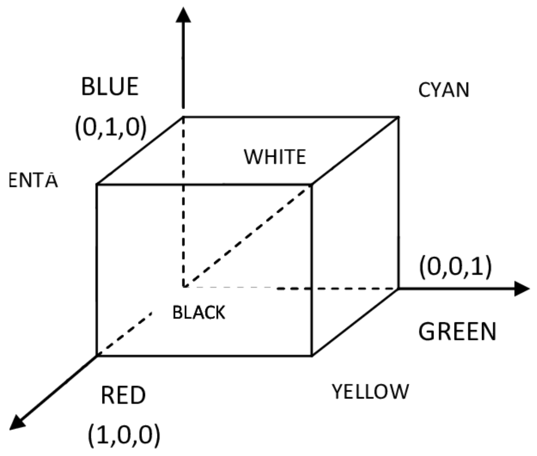


Figure 7 - 3D representation of RGB colour space model [46]

2.1.2 Support Vector Machine

Support Vector Machines are one potential classification model that could be used for this task as, they have had strong empirical successes for tasks such as text recognition [7], object recognition and facial detection [8]. Support vector machines work by creating a hyperplane between a series of data points in a feature space. The maximum possible margin between the data points is then calculated and when a new data point is fed into the machine which-ever side of hyperplane the data point is on it will be classed as such. A more detailed description can be found described by V.Vapnik [9].

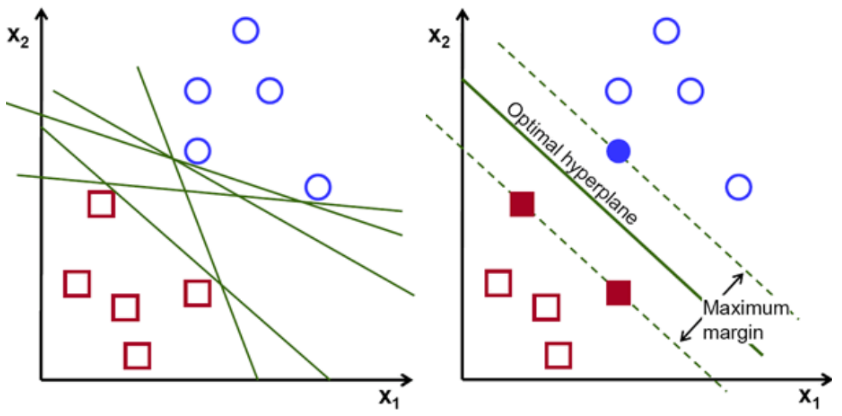


Figure 8 – SVM Diagram https://towardsdatascience.com/support-vector-machine-vs-logistic-regression-94cc2975433f

Support Vector Machines were originally designed for binary classification [10] however further research has been done into multiclass classification applications [11], and results show that the computation time for non-binary classifiers is considerably higher than that of binary classifiers. Therefore, for this non-binary classification task an alternative method would be preferable. Also, SVMs require more image pre-processing than alternative methods such as CNNs, these include feature extraction and Principal Component Analysis (PCA) to reduce the dimensionality on the input data.

2.1.3 K-Nearest Neighbour

K-Nearest Neighbour is another method that could be used. It is similar to the Support Vector Machine, as features are extracted from the training data and distributed into a feature space. The test data then has its features extracted using a similar method as the SVM. They are then compared to the "k" nearest points to identify which class the point belongs to. As shown by the figure below, the number of 'K' neighbours can influence the decision the machine makes [12].

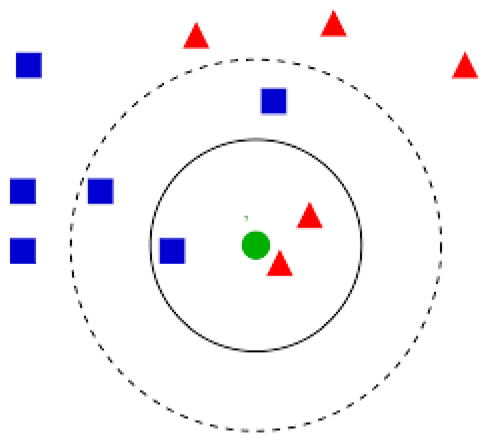


Figure 9 – KNN Diagram https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/

KNNs are an example of a non-parametric classifier [13] which means they require no prior training or parameter tuning before they can classify images. Computationally this is a vast advantage over some of the other methods, like the SVM where fitting the hyperplane can be computationally very complicated.

KNNs as with the SVMs, require feature extraction before any information can be fed to the classifier. This is not ideal for this project as a low latency app is preferred and image pre-processing could cause an increase latency. A limitation to the KNN is that each of the input vectors (features) are considered equally in the assignment for a given class. Features that do not give as much representation to the class are given equal weight to features that are more representative of the class, which can lead to miss classification.

2.1.4 Feature Extraction

Of a selection of the classification methods that could be used, feature extraction is a step that is required before the image data can be inputted into the classifier. Feature extraction is a dimensionality reduction tool that can be used to convert a large amount of raw data into a more manageable set of features, which still accurately describe the original dataset. Well regarded feature extraction techniques include Histogram Orientated Gradients (HOG) [14], Speeded-Up Robust Features (SURF) [15] and Scale-Invariant Feature Transform (SIFT) [16].

Some of these approaches have previously been applied to the Stanford Dog Breed Dataset with the SIFT with a Gaussian Kernel achieving an accuracy of 22% using a linear classifier [17]. The linear classifier using the SIFT feature extraction produces the results as found in the appendices on the Stanford dog's dataset.

2.1.5 Gnostic Fields

Gnostic Field [18] approaches have achieved high classification accuracy on the Stanford dog breed data set (47%) in the past. Gnostic Fields excel at fine-grained differences between classes, which is why they are good for tasks such as dog breed classification. A Gnostic field is inspired by a biological process in the brain, where each field contains one Gnostic set per category. Each set contains units used to match category specific patterns which makes the Gnostic field approach good at handling various size and shape images.

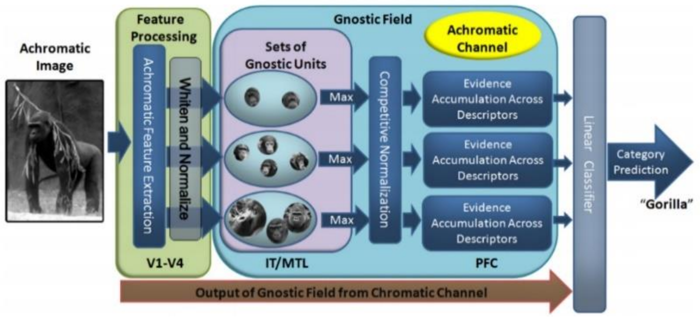

Achromatic Image
Feature Processing
Achromatic Feature Extraction
Whiten and Normalize
V1-V4
Gnostic Field
Sets of Gnostic Units
Max
Max
Max
IT/MTL
Competitive Normalization
Achromatic Channel
Evidence Accumulation Across Descriptors
Evidence Accumulation Across Descriptors
Evidence Accumulation Across Descriptors
PFC
Output of Gnostic Field from Chromatic Channel
Linear Classifier
Category Prediction
"Gorilla"

Figure 10 – Gnostic Field Diagram http://www.chriskanan.com/wp-content/uploads/Kanan_WACV_2014.pdf

2.1.6 Selective Pooling Vector

The Selective Pooling vector method has achieved the highest accuracy on the Stanford dog breed dataset achieving a top one categorical accuracy of 52% [19]. The Selective Pooling Vector method involves selecting the most confident local descriptors for a non-linear learning function. The method Local descriptors are encoded into vector format then compared to a codebook 'D'. Vectors below a given quantisation error threshold are selected for classification. The result of 52% was achieved using a conventional SVM.

The Selective Pooling Vector method has shown high performance on fine-grained tasks which is why it would be suitable for this project. However, it is somewhat limited as it uses an SVM to class the features and so suffers from the drawbacks of an SVM classifier. It is designed to be a binary classifier and so extending the classifier into a multi-class classifier is a lot more complex. The SVM also cannot predict probability it can only predict discrete classes.

2.2 Convolutional Neural Networks

2.2.1 Applications and Background

The CNN was first put forward by Kunihiko Fukushima [20]. This method involves a hierarchical multi-layered network capable of pattern recognition thorough learning and was based on the discoveries by Hubel and Wiesel on the visual cortex in animals. The CNN has gained tremendous popularity over recent years due to the wide range of application domains for this technique. CNNs have had strong performances on large datasets in recent years including in the ImageNet [21] competition dataset. In 2012, AlexNet by Alex Krizhevsky was able to out-perform the competition by achieving an error rate of 16% [22] which was a large break-through for machine vision and object recognition. The CNN proves the most promising classifier for this type of project.

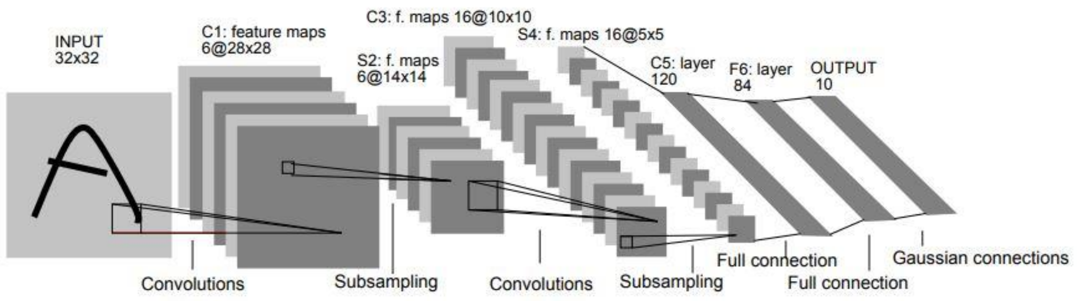


Figure 11 – LeNet-5 Diagram https://engmrk.com/lenet-5-a-classic-cnn-architecture/

The CNN is a class of Multilayer Perceptron (MLP) and, as shown in the figure above, it consists of an input layer an output layer and a series of hidden layers. The hidden layers include convolutional layers, pooling layers, fully connected layers, dropout layers and normalisation layers [23]. The main difference between CNNs and ANNs is the convolutional layers of the CNN [23]. The convolutional layers pass filers over the image pixels. Some of the filters may detect edges, corners or circles and these simple shape filters would be higher in the network. The deeper you go into the network the more complex and sophisticated these filters become [24], The filters produce feature maps which are then used in classification.

2.3 Training and Optimisation

2.3.1 Loss Calculation

Before the CNN can be trained the metric for determining the loss needs to be selected. The loss function is a method of calculating the negative consequences of an error. Thus, the loss function is used during back-propagation to recalculate the weights and biases of the network. Several classification loss functions could be used, which include Hinge [25], Categorical Cross Entropy [26] or Square Loss [25]. For back-propagation the Categorical Cross Entropy loss function is preferred over the other loss functions as it has better accuracy than the others. Hinge has better speed characteristics which are better for real-time loss calculation, however accuracy is preferred for this task.

2.3.2 Optimisation

Optimisation algorithms are used during back-propagation to minimise the loss calculated. There are a number of different optimisation algorithms that are applicable. Stochastic Gradient Descent (SGD) [27] is one of the simpler optimisation algorithms that uses the gradient of the loss function to find the global minima, where parameters are updated each training sample. The problem with SGD is that due to the frequent parameter updates the convergence on the global minima will keep overshooting. SGD with Momentum softens the oscillations, allowing it to reach the global minima faster. Adagrad [28] allows the learning rate (step size) to be adapted, so that infrequent weight parameters have big updates and more frequent weight parameters have smaller updates. A drawback to this algorithm is that the learning rate is always decaying which causes the model to stop learning and gaining new knowledge. The Adam Optimiser [29] uses the squared sum of its historical gradients to scale the learning rate as well gradient descent with momentum to reduce overshoot (similar to SGD with momentum). This optimiser has had some of the best results in practise and therefore is the choice for this application.

2.4 Deep Transfer Learning

Deep Transfer Learning [30] is a technique used to repurpose a model trained on one set of data and use its insights to apply to a new model for a new or similar task trained on a new dataset. The bottleneck features which are the output of the pre-trained model before the final layer are used as inputs to the new model. There are a large number of models that could be used to extract bottleneck features however the ImageNet [21] competition classifiers are a clear choice, has they have been trained on very large datasets of >50 million images which contain a dog breed subset. Some of the models are open source on the Keras 'Applications' Library, including the Xception [31], VGG16 [32], VGG19, ResNet [33], Inception V3 [34] and MobileNet [35] Models.

3 Dataset

This section will investigate the process for selecting the datasets used for training the dog breed classifier and identifying breed specific hereditary disorders. It will examine the characteristics of the datasets such as image properties, class sizes and classification tools like bounding boxes. A classification technique will then be selected based on the database size and properties.

3.1 Dog Breeds and Hereditary Disorders

Before selecting the datasets, it important to identify contents that the databases are required to contain for the project. The definition of a breed can be interpreted in several ways. This report will interpret it as a group of animals of the same species that have a distinctive appearance and have been developed by selective breeding. Dogs have been selectively bred for thousands of years and over this time many different breeds have been produced. The American Kennel Club recognises 192 different dog breeds (https://www.akc.org/dog-breeds/) while the Fédération Cynologique Internationale (FCI) officially recognises 360 different dog breeds world-wide.

The Inherited Diseases in Dogs (IDID) [36] database lists more than 479 disease fields, however many of these diseases are associated with multiple breeds. The diseases incidence varies along with the severity of the diseases. The more popular breeds tend to have a higher occurrence of inherited disorders and less common or newer breeds tend to have fewer listed diseases due to the time taken before enough dogs are affected to recognise it as an inherited condition. The IDID lists how the disorders are diagnosed along with how the disease is to be treated.

3.2 Database Selection

When selecting an image dataset for a classification task, the set of criteria that need to be considered are as follows: the images should be of good resolution clearly showing the classification object and the dataset should be sufficiently large enough to perform the task of training network to classify new images. There is no set rule for how large the dataset needs to be for high performance classification, however the larger the dataset typically the better the classifier will perform as shown by the figure below. Images of the objects should be from different angles, light conditions, occlusions and locations. The objects themselves should be of various sizes and as many individuals (different objects) as possible to increase the intra-class variation, to improve the classifiers performance.

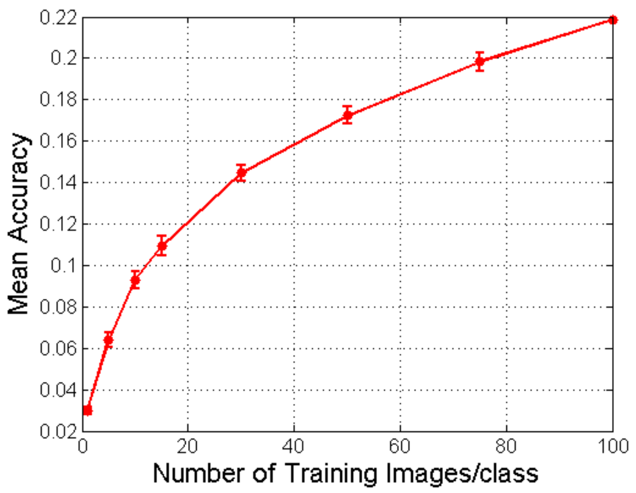


Figure 12 – Number of training images vs mean accuracy http://vision.stanford.edu/aditya86/ImageNetDogs/

The dataset for classifying the dog breeds was a clear choice. The Stanford dog breed dataset [17] has the largest volume of classified images of 120 different breeds of dogs, with a 20,580 images in total. This is the largest dog dataset publicly available. The images are of varying sizes, illumination levels, orientations and contain occlusions and large amount of intra-class variation. Each image is annotated using object bounding boxes and class labels which is useful for implementation. A large proportion of the images contain humans and are taken in different environments, which provides a good level of challenge for our classifier.

Finding a dataset for identifying inherited disorders is slightly more challenging. There are three main databases that contain information on inherited disorders in dogs: The Canine Inherited Disorders Database (CIDD), the Inherited Diseases in Dogs (IDID) Database and the Online Mendelian Inheritance in Animals (OMIA) Database. The IDID produced by the University of Cambridge Veterinary school contains peer-reviewed literature on the various disorders so is deemed to be the most credible.

3.3 Training, Testing and Validation Split

3.3.1 Training

The first step after deciding upon the image dataset for a classifier is to split the data into training, testing and validation subsets. The training subset is used to adjust the weights of the network during back propagation. The weights are typically updated every batch for a given number of epochs. The size of the training set can affect the performance of the classifier. [37] recommends a non-parametric resampling method for selecting an optimum training size, the dog breed data set [17] comes with a pre-split training index of 60% however. This will be used as a starting point before evaluating the effect the training split has on the classifier.

3.3.2 Testing
The testing subset is used to calculate the classifier loss and accuracy using a give loss formulation. The training set is important to evaluate the performance of the classifier. Therefore, the training samples must be different to that of the testing samples. The dataset [17] is also indexed with a 40% testing subset, however as a validation subset is being used the subset is split into a 20% of the original set testing subset.

3.3.3 Validation
The validation subset is used to reduce overfitting, which is where the model becomes too well trained on the training set so when given new data it struggles to classify. This could be due to learning noise or random fluctuations. This data subset is not used to adjust weights, it is used to verify that any increase in the overall accuracy of the classifier using the training data actually yields an increase in accuracy on a dataset that has not been seen by the classifier. If the training accuracy rises but the validation accuracy remains the same or decreases than the classifier is becoming overfit. When selecting the final model validation loss is to be considered.

3.4 Model and Environment Selection

3.4.1 Model Selection
The classification techniques outlined in the Literature Review were investigated in further detail, to find the approach that would give the best accuracy for a data set of this size. Approaches using SVMs and KNNs require more data pre-processing such as feature extraction. These feature sets would also be very large so processes like PCA would also need to be applied to reduce the dimensionality, to reduced computation time.

For a consumable application low latency is preferred so less pre-processing is preferable. Historically KNNs and SVMS have been outperformed by CNNs for classification tasks, therefore the approach selected to apply to this project was the Convolutional Neural Network. There are many variations of the CNN architecture these are to be further investigated to optimise our performance and computation time.

3.4.2 Environment Selection
When choosing the environment to implement the network in a number of things needed to be considered. The complexity of the implementation was key as this project was needed to be completed in a short period of time. Therefore, open source libraries are used as they provide high performance tools for implementing classifiers. TensorFlow [3] is an open-source machine learning framework created by google, its main focus is deep learning. TensorFlow is capable of running on CPUs and GPUs and has had wide ranging very well-regarded successes. Keras a high-level neural network API that is run on top of TensorFlow, it also runs for CPUs and GPUs and has exceptional user modularity and accessibility which makes it good for fast network prototyping. Therefore, the Keras TensorFlow commination is used for implementing our classifier.

4 Classification Methodology
This section aims to cover the methodology behind producing the classifier and the reasoning behind the decisions that have been made. The steps for preparing the data for use by the classifier have been outlined. The architecture of the CNN is explained in this section along with the function of each of the layers and the final selected architecture. The classifier performance is evaluated using a suitable metric and assessed for over or under fitting. The performance is improved using Deep Transfer Learning and the implementation of this is explained. The transfer networks are evaluated, and the

best is selected for extracting bottleneck features. The final network is tuned, and the results are found and displayed in a confusion matrix.

4.1 Image Pre-processing

4.1.1 Bounding Boxes

Bounding boxes are digital co-ordinates of a rectangular border that fully encloses an object in a digital image. Bounding boxes can be created using object detection programs like the You Only Look Once (YOLO) real time object detection system [38]. Bounding boxes allow the program to know the location in the image of the object that needs classifying, as shown by the figure below. This allows for a large reduction in the computation that is required as only what is inside the bounding box needs to be processed by the network. The YOLO: object detection system works in real-time which makes it

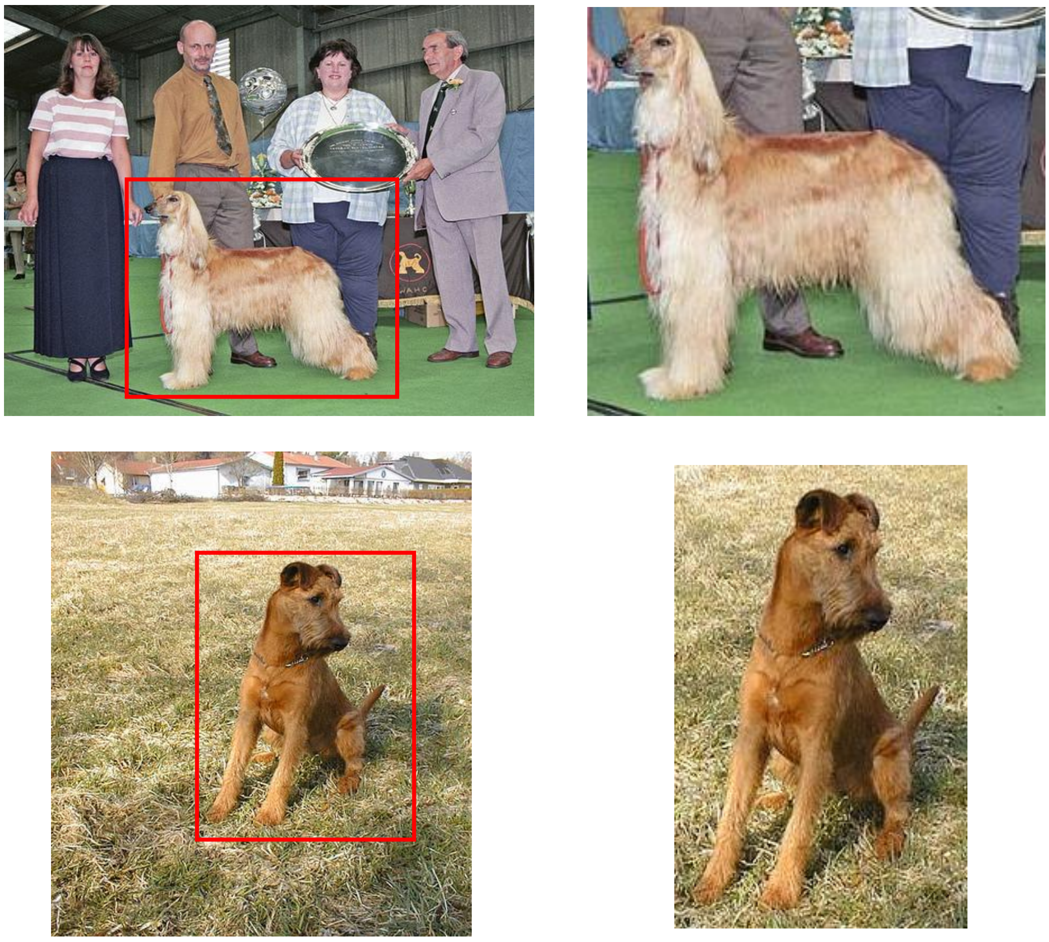


Figure 13 – Example Bounding Box Cropping

The YOLO system works by setting an overlaying grid over the image. Each cell of the grid predicts a series of bounding boxes with corresponding confidences that the bounding box contains an object. Each cell of the grid then predicts a class probability as shown by the coloured diagram below. Box and Class predictions are then combined. Non-maximal suppression and thresholding is applied to the detections to remove low confidence and duplicate detections. The final detections are then found as shown by the diagram below.

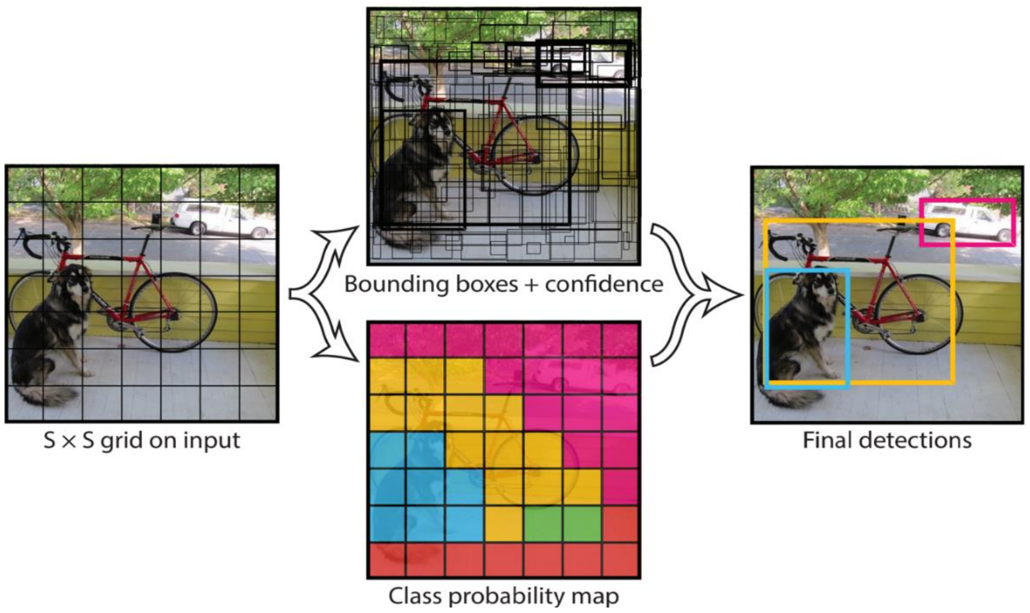


Figure 14 – YOLO Operation Diagram https://arxiv.org/pdf/1506.02640.pdf

The Stanford Dog Breed dataset [17] contains an XML file containing the bounding box locations of the dogs for each image file along with the class label for the dog. The annotations XML fine gives the xmax, xmin, ymax and ymin pixel locations for each of the bounding boxes, which is used to crop the content.

4.1.2 Re-sizing Images

After the bounding boxes have been found a function was written to extract the contents of each of the bounding boxes. The contents were all resized so that the inputs into the network would be of consistent dimensions. The scale selected for the resized images was 250x250 as most well-regarded classifiers rescale their images in a range of 224 to 256. The re-sized files were saved in a new file directory containing three main folders; training, testing and validation. In each of these folders there is a subfolder for each of the classes.

4.1.3 Input Tensors

As the images are RGB there are three dimensions to each of the image pixel values. The three dimensions comprise of the red, green and blue intensities respectively. The input to the CNN will therefore have dimensions of the image size by the three colour dimensions (250x250x3). When creating the training, testing and validation tensors, the input matrices for each of the images in the subset are combined into an overall 4D tensor which the network runs through.

The Keras ImageData Generator is used to prepare the image data for input into the network. Using the image data generator allows the size of the images input images to be controlled. Images can also be augmented which involves randomly inverting, rotating and scaling the images as shown by the figure below. Using augmented images essentially creates more synthetic data to train the network, which improves the model’s generalisation performance. In practice images that the network may be exposed to will have a range of orientations scales and intensities, using augmentation better prepares the network for these images.

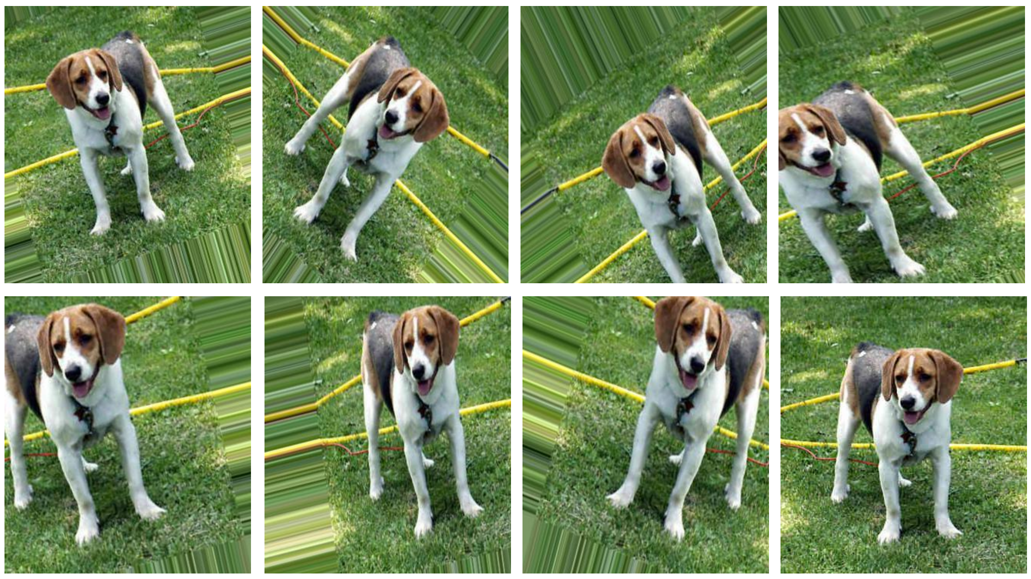


Figure 15 - Augmented Image using ImageDataGenerator [17]

4.2 CNN Architecture

This section will aim to give an overview of the CNN architecture, the functions of each of layers and the decisions behind the hyper-parameters regarding the layers.

4.2.1 Convolutional Layers

The convolutional layers derive their name from the convolutional operator, where a given kernel is convolved with pixels in the image to produce an output. The purpose of the convolutional layer is to extract the features from the image. Each filter on the first layer is typically a smaller filter i.e 5x5x3 (5 pixels wide and high and 3 deep for the RGB channels). The filter is convolved with the image and a 2D activation map is produced, that gives the responses to the filter at every special position in the image. The network will learn filters that see edges of various orientations, blotches, circles and so on until there is an entire filter set on each layer [22]. As you move deeper into the network the convolutional layers search for more higher-level features, such as eyes and ears. A number of parameters can be controlled when designing the convolutional layers. These include; Depth, which is the size of the filter set that the layer has, Stride which is the size of steps the kernel takes when moving across the image pixels (typically this value is 1) and Padding which allows us to control the special size of the output (typically the image is padded with 0s).

After a feature map is produced from the convolution, it is then passed on to the following layers. The number of feature maps corresponds to the number of filters that have been applied to the image. An example of the basic shapes filtered for by the high-level convolutional layers are shown by the figure below. These high-level features include edges and blobs, then as you move deeper become more complex such as textures.

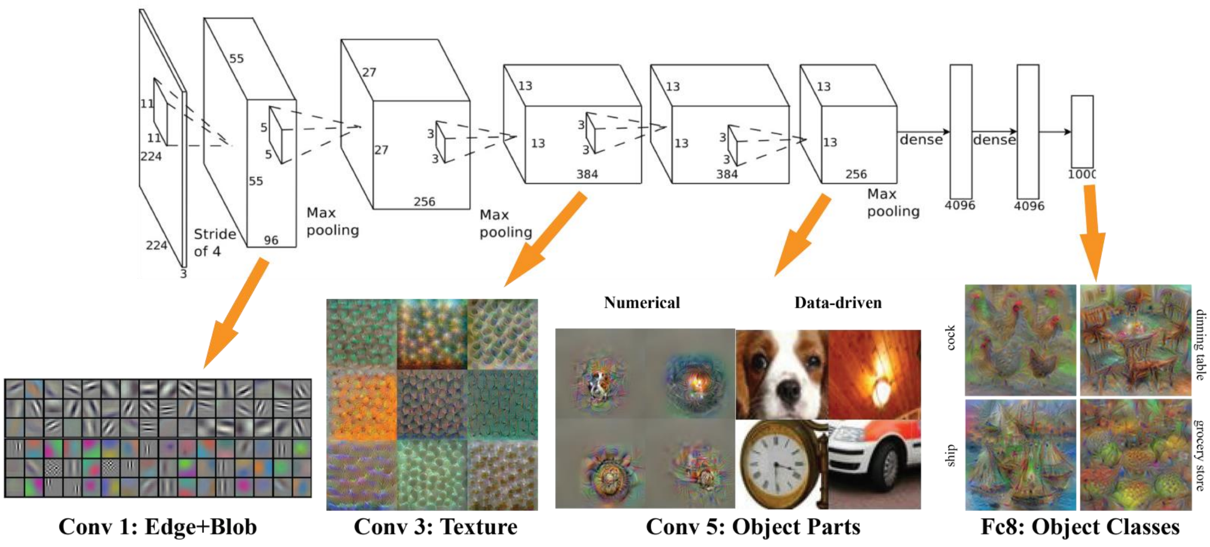


Figure 17 – VGG 16 Convolutional Network Filter Diagrams https://towardsdatascience.com/understanding-ssd-multibox-real-time-object-detection-in-deep-learning-495ef744fab

As you move deeper into the network the other convolutional layers are doing dot products of the input to the previous convolutional layers. By applying filters to each of the feature maps found by the previous convolutional layers these filters typically search for larger objects made out of the shapes produced by the lower filters. The filters of the deep levels from the VGG16 architecture are visualised below with their corresponding classes.

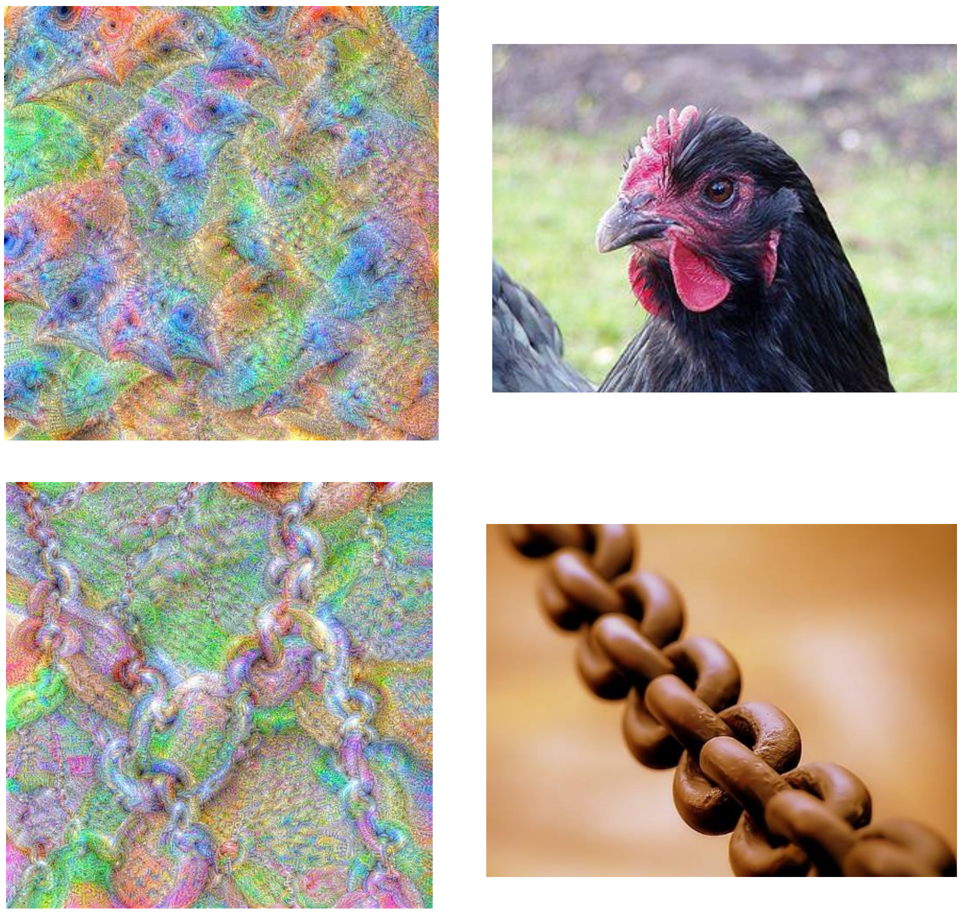


Figure 16 – VGG16 Deep layer Filter Visulisations https://towardsdatascience.com/how-to-visualize-convolutional-features-in-40-lines-of-code-70b7d87b0030

Typically networks like AlexNet, LeNet and VGG-16 have an increasing number of channels or filters as you move deeper into the network. This is so they can be combined to form many complex shapes of the deeper filter. Typically, in machine learning the practice is to base the model architectures on previous models that have been used for similar tasks. The structure of LeNet-5 [24] is used to base the initial CNN architecture, as LeNet-5 uses sequential convolutions with periodic special reduction using pooling, which has proven results. The number of convolutional layers chosen for this initial build was 3 as this is similar to LeNet and most literature states that you should start with around 3 layers and increase until the improvement in accuracy is minimal.

As this is a fine-grained classification task the inter-class variation is very small which means small and local features will most likely be key to differentiate the different classes. This indicates smaller size kernels should be used like 3x3 or 5x5 rather than larger kernels such as 7x7. The Keras library contains a function which allows a 2D convolutional layer to be produced. The parameters such as kernel size, stride, padding, data format and activation can all be specified, as shown below. This makes building the network much easier.

```
model.add(Conv2D(filters = 16,kernel_size = (5,5),strides = (2,2),padding = 'valid',
                 activation ='relu',input_shape = (img_width, img_height, 3)))
```

Figure 18 - Convolutional Layer Example

4.2.2 Pooling Layers

Pooling Layers are used to reduce the dimensionality of their input. These layers are used to reduce the number of parameters (variance) leading to reduced computation. Pooling also controls overfitting. There are two types of pooling: local and global. Local pooling takes neighbouring units in a feature map and applies a pooling kernel. Typically, max pooling is used where the maximum value of the neighbourhood is taken as the output value. Average pooling could be used but sometimes it cannot extract good features as it takes into account all the results which may or may not be important for the classification. The average pooling uses all the values from the feature map but if they are not all needed it will lower the accuracy.

Global average pooling also reduces overfitting by reducing the number of parameters [39]. Global average pooling can be used to reduce the 3-dimensional output tensor from the convolutional layers into a single vector containing a single object for each object/class in the classification task. This is done by taking the mean of each feature map and supplying the result to the fully connected layer. This removes the need of having the same number of feature maps as categories in the network. There, global average pooling is performed before features are passed to the fully connected layers.

The Keras library contains functions which are used for creating 2D pooling layers for both global and local pooling. As with the convolutional layers the kernel or poo size can be controlled along with the strides and padding. As 'None' stride is selected it defaults to the pool size.

```
model.add(MaxPooling2D(pool_size=(3, 3), strides=None, padding='valid'))
```

```
model.add(GlobalAveragePooling2D())
```

Figure 19 - Max and Global Pooling Examples

4.2.3 Dropout Layers

Dropout is a regularisation technique proposed by Srivastava, et al. [40]. In dropout randomly selected neurones are ignored during the training process. This means their contribution to downstream layers is temporarily removed, and during back-propagation their weights are not updated. As the CNN trains the neuron weights settle into their context within the network. Weights of neurones become trained for specific features which is a form of specialisation. Neighbouring neurones start to depend on this specialisation which can start to form a fragile model. When some neurones are dropped other neurones step in to handle the feature representation from the dropped neurones. This causes multiple internal representations to be learnt by the network.

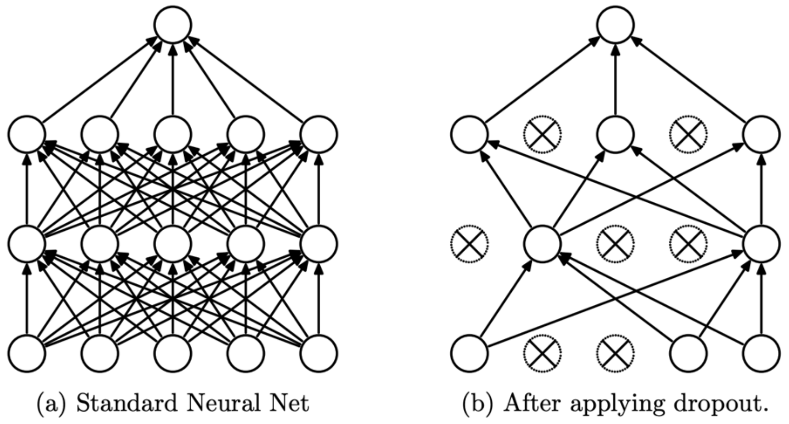


Figure 20 – Dropout Diagram http://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf

Dropout performs best when it is used on fully connected layers. Batch normalisation should be used between convolutional layers, as this produces the highest accuracy. Srivastava [40] states that 0.5 dropout, seems to be close to the optimal value for most networks and this value should be tuned down to improve performance. An optimal value of 0.4 dropout is found for our network after tuning.

4.2.4 Normalisation Layers

Normalisation is a technique that mimics a neuro-biological process called 'lateral inhibition'. This is where a contrast is created between neighbouring neurones, causing a more significant local maximum [41]. This leads to an increase in sensory perception. In order to input data into the network it is important to normalise it, zero the mean to remove the chance of early saturation of non-linear activation functions such as the Hyperbolic Tangent. This problem appears in the intermediate layers because the distribution of activations is constantly changing during training. This slows down the training process because the neurones have to adapt to the new activations known as the Covariate Shift [41]. To reduce the effect of the Covariate Shift batch normalization is used which normalises the input to the layer it is applied over.

Normalisation has a similar function to dropout by reducing overfitting. Dropout uses a weight sharing method to reduce overfitting. Research on batch normalisation shows much faster learning without a loss in generalisation for inception-based architectures.

4.2.5 Fully Connected Layers

Fully connected layers are layers common to ANNs and are present in CNNs, each neurone is connected to every other neurone [42] in the next and previous layers, as shown by the figure below. Each of the neurones take a weight and produce an output based on a given activation function. The fully connected layers allow for the learning of non-linear combinations of high-level features produced from the convolutional layers. The purpose of the fully connected layer is the same as a multi-layered perceptron. The weights and biases of the network are trained over a series of epochs using back-propagation [43]. The output of the fully connected layers is typically the outputted from the CNN after some normalization or dropout.

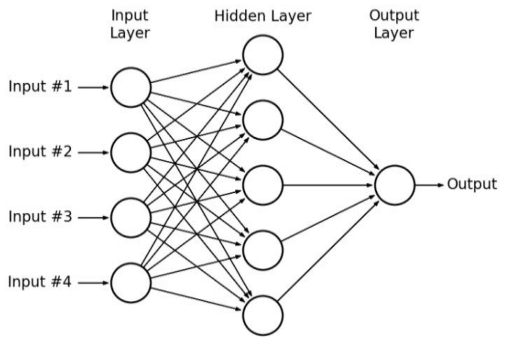


Figure 21 – MLP Diagram https://www.researchgate.net/figure/A-hypothetical-example-of-Multilayer-Perceptron-

The main difference between convolutional layers and fully connected layers, are that convolutional layers are only connected to a local region in the input and many neurones in the convolutional volume share parameters. However, neurones in both layers still compute dot products of their input so their functional form is identical. The fully connected layers can distinguish data that is not linearly separable, which allows it to classify the features extracted from the convolutional layers.

When deciding on the number of neurones in the fully connected layers, the number of parameters is the main consideration, as fully connected layers typically contain the most parameters. The more parameters the network has the increased the risk to overfitting. Therefore, a rule of thumb is to try multiple configurations of the fully connected layers and evaluate using cross validation and select the best configuration. Ideally this size is kept as small as possible without sacrificing accuracy.

4.2.6 Output Layer

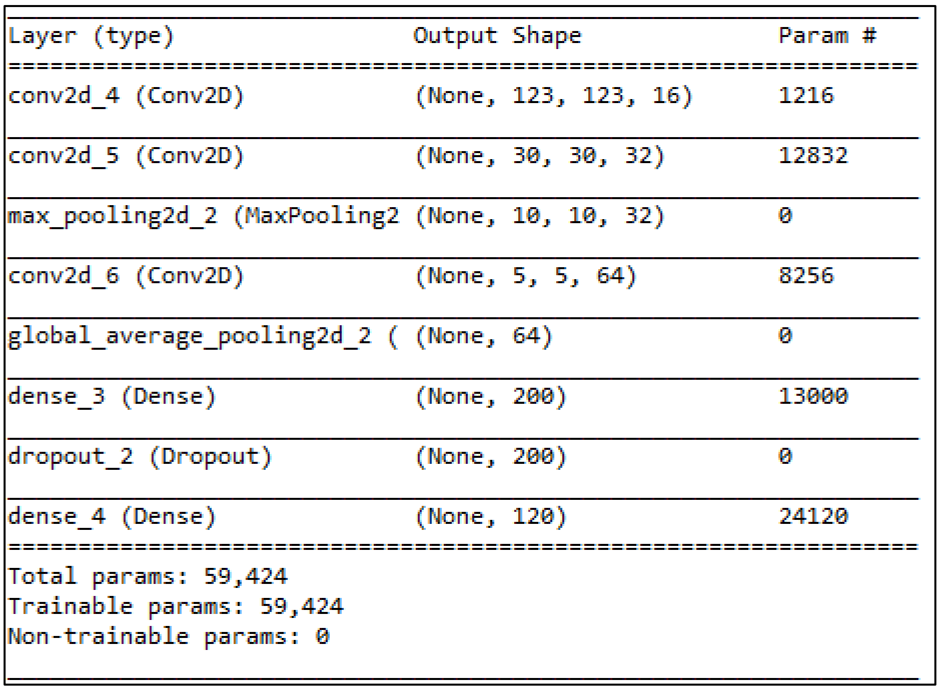
The output layer is the final layer of the network, and thus it is used to make class predictions. The layer consists of a set of neurones the same size as the number of classes. The Dog Breed set [17] contains 120 different classes of dog, therefore the number of neurones in the final layer is 120. The output layer typically utilises the Softmax activation function for multiclass tasks. Softmax shown below, allows for a class probability distribution between 0 and 1. The sum of all the probabilities in this layer is equal to 1.

$$\sigma(x_j) = \frac{e^{x_j}}{\sum_i e^{x_i}} \qquad (1.1)$$

The output of the network is therefore a vector of length 120 where each value along the vector is the classification probability of the corresponding class. One-hot encoding is when comparing the output vector to the actual class, as this is the simplest way to compare the results. The advantage of Softmax is it gives the high value the higher probability, which is useful for classification tasks.

4.2.7 Initial CNN Overall Architecture

After the layers for the initial model have been selected the next step is to assemble them into an overall architecture. The initial model designed has 3 convolutional layers, one max-pooling layer within the convolutions, one global pooling layer after the convolutions, one fully connected layer, a dropout layer and a fully connected output layer. The total number of trainable parameters in this model is 59,424.



```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 123, 123, 16)      1216
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 30, 30, 32)        12832
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 10, 10, 32)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 5, 5, 64)          8256
_________________________________________________________________
global_average_pooling2d_2 ( (None, 64)                0
_________________________________________________________________
dense_3 (Dense)              (None, 200)               13000
_________________________________________________________________
dropout_2 (Dropout)          (None, 200)               0
_________________________________________________________________
dense_4 (Dense)              (None, 120)               24120
=================================================================
Total params: 59,424
Trainable params: 59,424
Non-trainable params: 0
_________________________________________________________________
```

Figure 22 - Initial Model Summary

The initial model's architecture is based on the LeNet-5 [43] design (shown below), due to the simplicity and the similar multi-class classification function. The next step is to train, test and evaluate the classifiers performance, then improve if necessary.

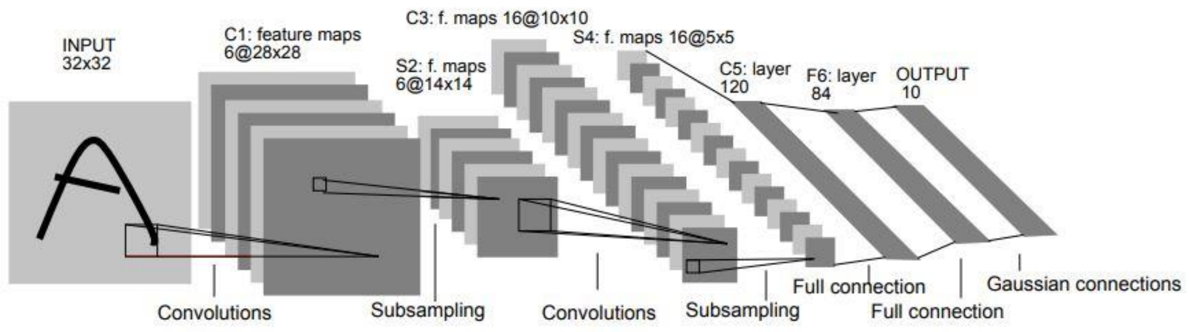


Figure 23 - LeNet-5 [43] Architecture Diagram

4.3 Classifier Optimisation and Evaluation

4.3.1 Activation Function Selection

Activation functions define the outputs of the neurones on all the different layers of the network. The activation function maps the input on a distribution typically between 0 & 1 or -1 & 1. There are a number of different activation functions that could be used and one of the most typical is the Hyperbolic Tangent.

$$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \qquad (1.2)$$

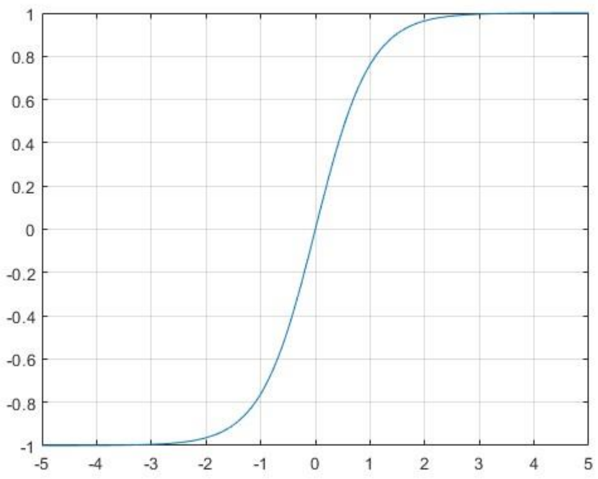


Figure 24 – Hyperbolic Tangent Activation Function https://uk.mathworks.com/help/matlab/ref/tanh.html

The Hyperbolic Tangent as shown above is non-linear which means it is able to generalize or adapt to a variety of data points. The Hyperbolic Tangent is in the range -1 to 1, which means negative values will be mapped strongly negative, and values close to 0 will be mapped close to 0. However, when using gradient decent for training, calculating the derivative of the Hyperbolic Tangent is not computationally efficient [22].

One of the newer activation functions used is the Rectified Linear Unit (ReLU) activation function. The ReLU function is half rectified and takes the maximum of the input from 0 to infinity. The ReLU function is described by the equation below.

$$f(x) = \max(0, x) \qquad (1.3)$$

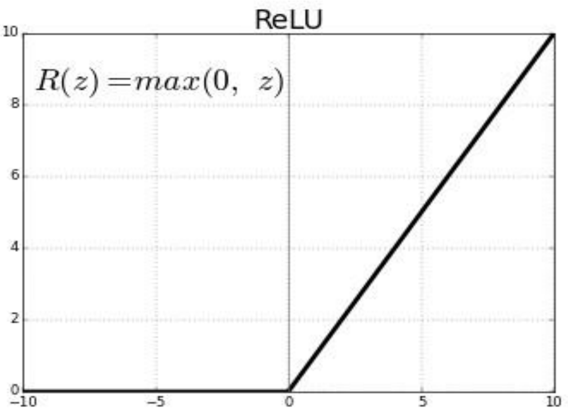


Figure 25 – ReLU Activation Function https://medium.com/@kanchansarkar/relu-not-a-differentiable-function-why-used-in-gradient-based-

The main advantage of the ReLU function is that both its function and its derivative are monotonic, which means computationally it is far more efficient than the Hyperbolic Tangent but without a significant effect on the generalisation accuracy. ReLU is the fastest activation function and therefore it will reduce the training time. However, for negative inputs the ReLU function approximates the value to 0 [44]. As the pixel values will fall below zero, the ReLU function is well suited for this task.

4.3.2 Training and Optimisation

Gradient Descent is a process used during back-propagation to minimise the loss of the network. At the start of training the parameters are initialised with random values and the loss is calculated. The partial derivative or the loss is found with respect to the old and new variable values, causing the parameter to increment towards the local minimum. The equation below shows the iterative updating of the parameter value θ. The gradient term is subtracted as we are moving towards a local minima. Every time the training is initialised new random parameter values are used, therefore training the network multiple times to avoid local minima is recommended, as shown by the figure below where different starting points result in different minima.

$$\theta_{new} = \theta_{current} - \eta \frac{\partial L}{\partial \theta_{current}} \qquad (1.4)$$

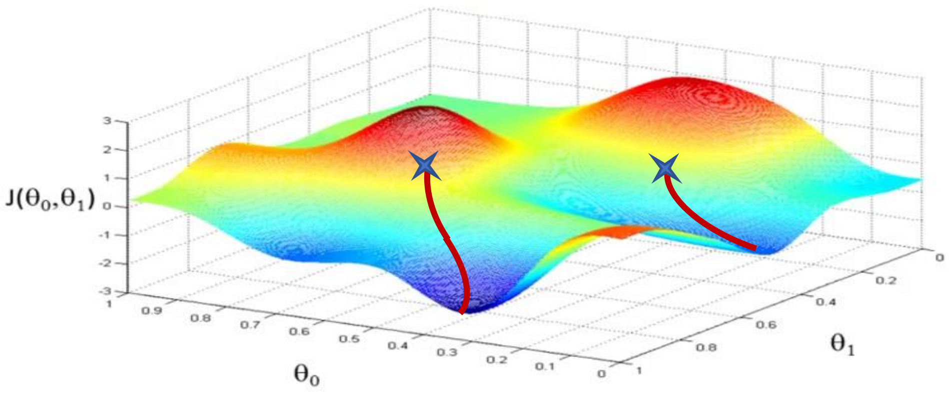


Figure 26 – Gradient Descent Diagram https://medium.com/artificially-intelligent-blog/machine-learning-6-enter-gradient-descent-ef04726c84bb

As described in the Literature Review the optimisation technique being used to train the network is the Adam Optimiser [29]. The Adam optimiser is different from the SGD, in that SGD maintains a single learning rate for all the weight updates and doesn't change the learning rate throughout the whole training process. Adam on the other hand combines the advantages the optimisers AdaGrad and Root Mean Square Propagation (RMSProp). AdaGrad uses a per-parameter learning rate which improves the performance on problems with sparse gradients. The RMSProp optimiser also maintains a per-parameter learning rate, that adapts the learning rate based on how quickly the gradient is changing, which means the optimiser performs well on noisy problems.

Adam unlike RMSProp which adapts the learning rates based on the mean of the gradients, uses the uncentred variance to adapt the learning rate. An exponential moving average of the gradient and squared gradient is found, and the decay of these are controlled using beta 1 & beta 2. Adam has had empirical results which is why it is preferred over other techniques.

One parameter that can influence the training rate and over or underfitting is the batch size and epoch number. The batch size dictates the number of training samples the network is exposed to before the model is updated using back-propagation. When using a batch size of 1 there is a large amount of element noise as only one data point is used to find the new step direction. On the other hand, a very large batch set is slow but more accurate since the variance drops. For a model of this depth a starting batch size of 128 is appropriate, then increasing interactively until the performance levels off. The epoch number is the number of times the training data is iterated through. This value is typically increased until the validation accuracy levels off, to prevent the model overfitting.

4.3.3 Performance Evaluation

The initial model is run for 150 epochs on a batch size of 128. As we can see from the following graphs the model increases in accuracy over this period, but the validation accuracy begins to plateau. This divergence is mirrored by the model's loss. The validation loss decreases and reaches a minimum around 40 epochs and begins to diverge with the training loss. This divergence indicates the model is becoming over-fit on the training data and beginning to learn noise on the training images which reduces its performance on the validation images.

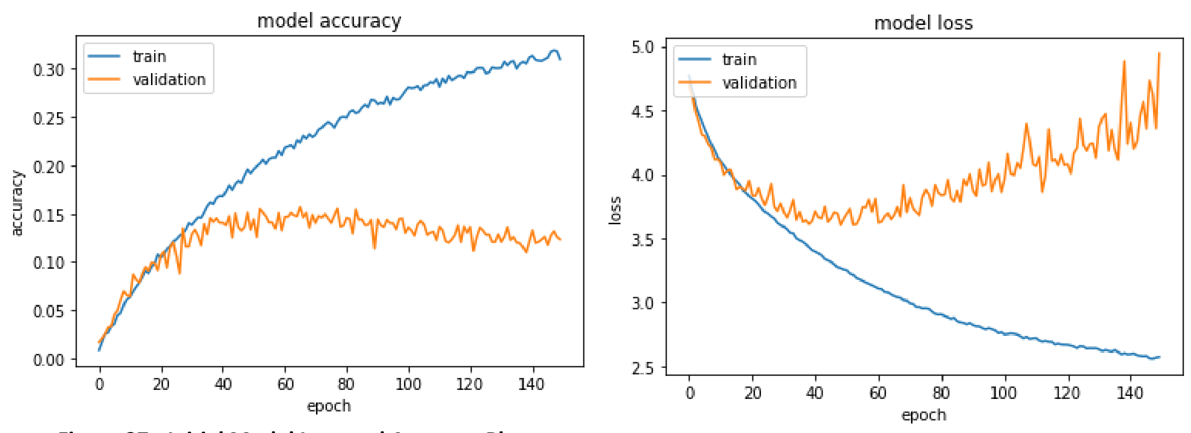


Figure 27 - Initial Model Loss and Accuracy Plots

The reason the model is overfitting is most likely the architecture of the model is not well designed for generalisation. The model may have too many parameters which will be causing it the extract features such as collars which the dogs are wearing. To avoid this issue of filters learning irrelevant features, transfer learning can be used, to implement pre-trained filters from large network trained to broad datasets.

4.4 Deep Transfer Learning

Deep transfer learning is a technique as described in the Literature Review, for repurposing deep networks trained on very large datasets for other for other tasks. Deep neural networks that are trained on natural images, extract features throughout the layers these features are general until the last layer transitions them from general to specific. Transfer learning takes these general features and repurposes them.

4.4.1 Bottleneck Features

Models trained on very large datasets just like any other CNN need to identify features like curves and edges in the first few layers. Depending on the task these similarities are found deeper into the network, which is why they may be applicable to our model. As shown in Appendix A, the bottleneck features are extracted after the last convolutions of the network, before the features are fed into the fully connected layers. The training, testing and validation sets are all fed to the transfer network and the bottleneck features extracted. The bottleneck features are then fed into our trainable network with an input size the same as the output size of the transfer network. Our network is then trained over a series of epochs using the bottleneck features.

4.4.2 Network Selection

A key part of the transfer learning process is selecting a network that will perform well on the dataset being used. The ImageNet [21] classifiers are trained on over 14 million images with over different classes 1000. These 1000 classes include 90 of the 120 Stanford dog breed classes [17], which make the ImageNet classifier well suited for feature transfer.

There are several different ImageNet classifiers that could be selected for our task. The list is narrowed by which networks are available using the Keras 'Applications' library. The networks available include the following:

Model Name	Date Produced	ImageNet Performance (Top-1 accuracy)	Transfer Performance (20 epochs)
VGG-16 [32]	2014	0.715	0.3489
VGG-19 [32]	2014	0.752	0.2930
ResNet-50 [33]	2015	0.770	0.0202
Inception-V3 [34]	2015	0.782	0.6033
Xception [31]	2016	0.790	0.6944
MobileNet [35]	2017	0.706	0.5231

Table 2 - ImageNet Classifier Comparison

From the results above the Xception model, has the best empirical success on the ImageNet set and performs best at initially classifying the dog breeds. It is therefore selected for transfer learning. Xception stands for 'extreme inception' as it is based on the inception architecture. The inception architecture computes multiple different transformations for the same input in parallel, concatenating their results into one single output, which allows multiple kernel sizes to be used. To reduce computational complexity 1x1 convolutions are performed to reduce dimensionality.

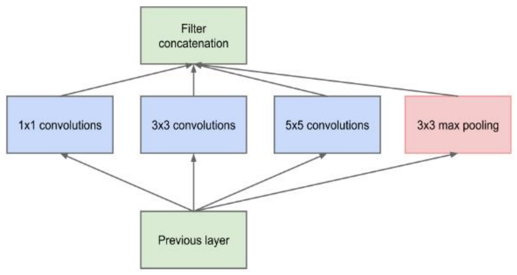


Figure 28 – Inception Diagram https://nicolovaligi.com/history-inception-deep-

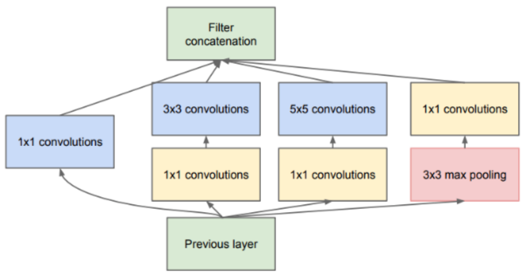


Figure 29 - Inception Diagram https://nicolovaligi.com/history-inception-deep-

The Xception model builds on this by coupling the cross-channel and special correlations as shown by the figure below. Francois Chollet [31] refers to this as a “depth-wise separable convolution”, this consists of a NxN special convolution performed independently for each channel followed by a 1x1 pointwise convolution across the channels. Appendix – B shows the overall architecture of the Xception model, SeparableConv indicates where the depth-wise separable convolutions are occurring.

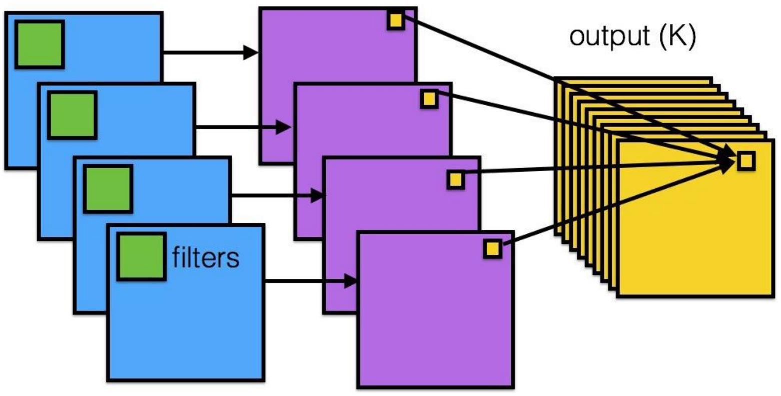
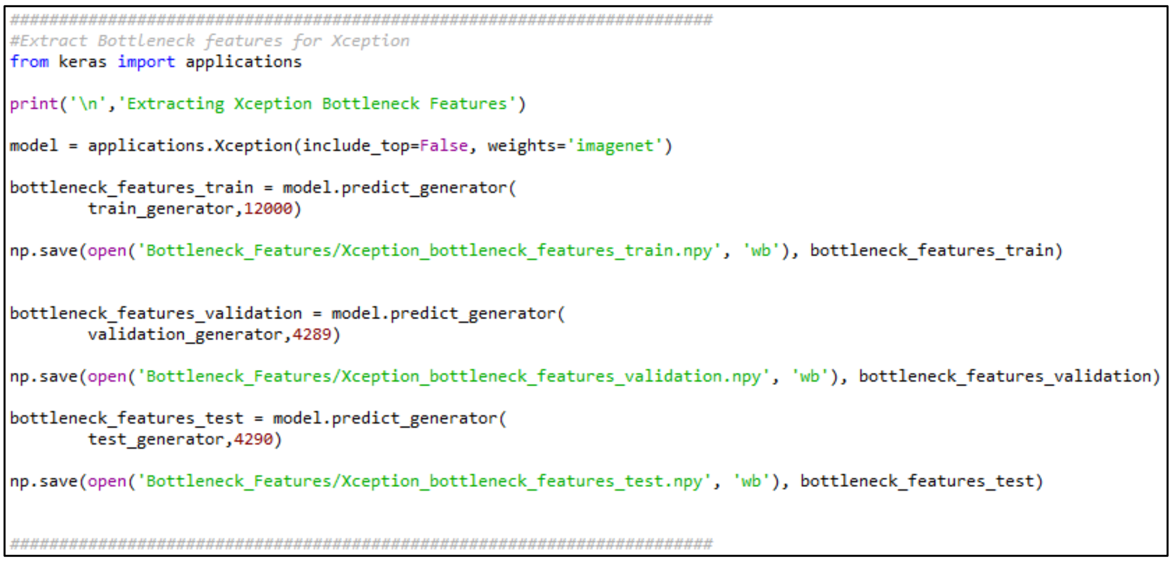


Figure 30 – Xception Diagram https://www.kdnuggets.com/2017/08/intuitive-guide-deep-network-architectures.html/2

4.4.3 Transfer Learning Implementation

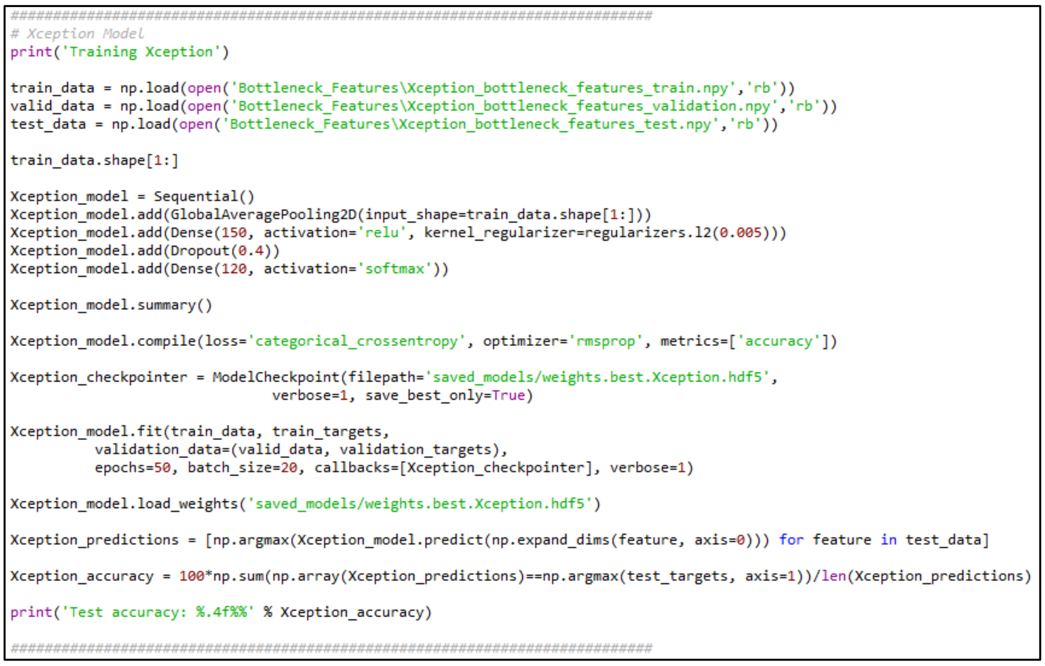
The ImageNet classifiers are implemented using the Keras ‘Applications’ library. The bottleneck features are extracted by using model prediction and setting the network not to include the top fully connected layers. The weights are set to the ImageNet weights and the train, test and validation predictions are made. The resultant feature arrays are then saved as ‘.npy’ arrays, as shown by the figure below.



```
###############################################################################
#Extract Bottleneck features for Xception
from keras import applications

print('\n','Extracting Xception Bottleneck Features')

model = applications.Xception(include_top=False, weights='imagenet')

bottleneck_features_train = model.predict_generator(
        train_generator,12000)

np.save(open('Bottleneck_Features/Xception_bottleneck_features_train.npy', 'wb'), bottleneck_features_train)


bottleneck_features_validation = model.predict_generator(
        validation_generator,4289)

np.save(open('Bottleneck_Features/Xception_bottleneck_features_validation.npy', 'wb'), bottleneck_features_validation)

bottleneck_features_test = model.predict_generator(
        test_generator,4290)

np.save(open('Bottleneck_Features/Xception_bottleneck_features_test.npy', 'wb'), bottleneck_features_test)


###############################################################################
```

Figure 31 - Bottleneck Feature Extraction Implementation

Once the bottleneck features have been extracted, they are loaded and inputted into a global average pooling layer to reduce the dimensionality. The features are then passed to the fully connected layers using the ReLU activation function. Kernel L2 regularization [45] is used to reduce overfitting and is incorporated into the optimisation of the network. The model with the lowest validation loss is saved and tested using the bottleneck features from the testing set.



```
################################################################################
# Xception Model
print('Training Xception')

train_data = np.load(open('Bottleneck_Features\Xception_bottleneck_features_train.npy','rb'))
valid_data = np.load(open('Bottleneck_Features\Xception_bottleneck_features_validation.npy','rb'))
test_data = np.load(open('Bottleneck_Features\Xception_bottleneck_features_test.npy','rb'))

train_data.shape[1:]

Xception_model = Sequential()
Xception_model.add(GlobalAveragePooling2D(input_shape=train_data.shape[1:]))
Xception_model.add(Dense(150, activation='relu', kernel_regularizer=regularizers.l2(0.005)))
Xception_model.add(Dropout(0.4))
Xception_model.add(Dense(120, activation='softmax'))

Xception_model.summary()

Xception_model.compile(loss='categorical_crossentropy', optimizer='rmsprop', metrics=['accuracy'])

Xception_checkpointer = ModelCheckpoint(filepath='saved_models/weights.best.Xception.hdf5',
                               verbose=1, save_best_only=True)

Xception_model.fit(train_data, train_targets,
          validation_data=(valid_data, validation_targets),
          epochs=50, batch_size=20, callbacks=[Xception_checkpointer], verbose=1)

Xception_model.load_weights('saved_models/weights.best.Xception.hdf5')

Xception_predictions = [np.argmax(Xception_model.predict(np.expand_dims(feature, axis=0))) for feature in test_data]

Xception_accuracy = 100*np.sum(np.array(Xception_predictions)==np.argmax(test_targets, axis=1))/len(Xception_predictions)

print('Test accuracy: %.4f%%' % Xception_accuracy)

################################################################################
```

Figure 32 - Transfer Model Implementation

4.4.4 Final Architecture

The final architecture of the network includes the 36 convolutional layers from the Xception model, structured into 14 modules, as shown in Appendix – B. The Xception convolutional layers are connected to two fully connected layers with 150 and 120 neurones respectively. The last two layers have trainable weights, whereas the Xception model uses the fixed ImageNet weights. Two layers are selected for the fully connected layers as more than two will increase the number of trainable parameters and could lead to overfitting. The number of trainable parameters in the final network is 325,325, this is considerably more than the previous initial model.

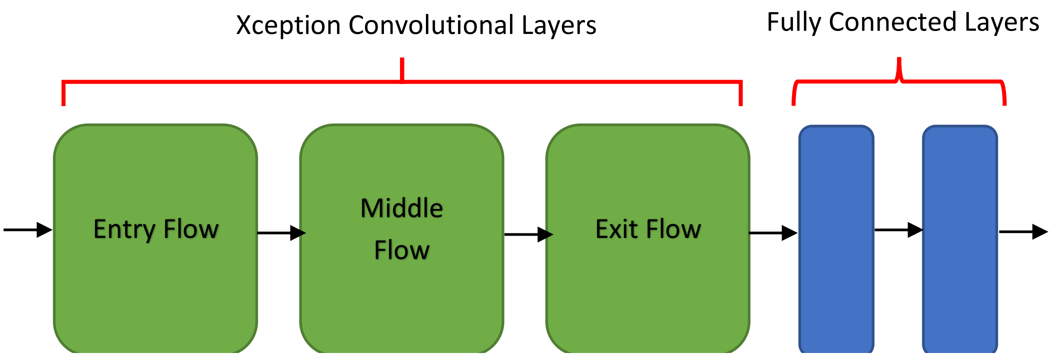


Figure 33 - Final Classifier Architecture Diagram

4.5 Final Performance

4.5.1 Classification Results

The loss and accuracy plots for the transfer classifier are shown in the figure below. The CNN produced by using the transfer learning approach achieved a considerably higher accuracy than the initial model based on LeNet-5. A peak validation top 1 accuracy of 0.748 was achieved at the lowest validation loss. As you can see from the figure below the accuracy plateaus after approximately 25 epochs, whilst the training accuracy continues to rise. As with the initial classifier this shows the classifier is becoming better trained at handling the training data but remains with the same performance when exposed to new validation data. One way to increase the accuracy further is to further augment the data or use a larger dataset.

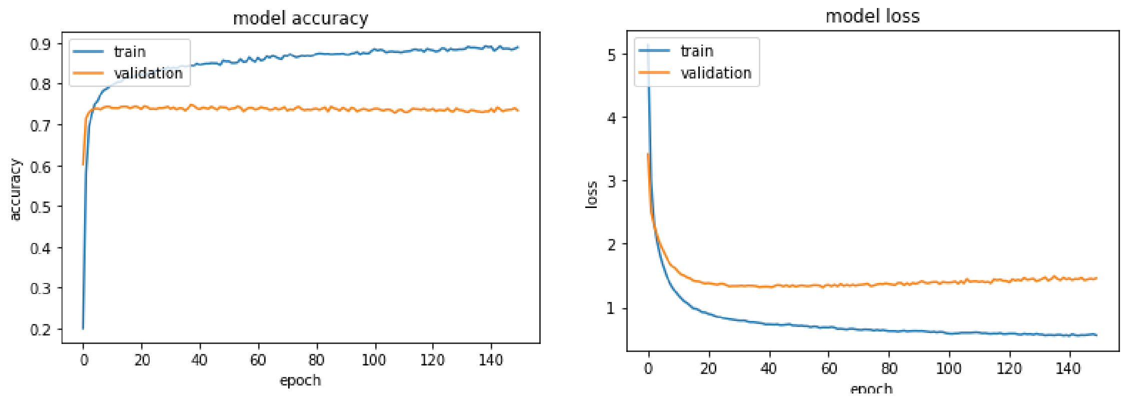


Figure 34 - Transfer Model Loss and Accuracy Plots

The table below show the results of using the CNN with the deep feature transfer learning. When compared to the other classification approaches on the Stanford dog dataset final model performed significantly better.

Method	Top-1-Accuracy (%)
SIFT + Gaussian Kernel + Linear Classifier [17]	22
Gnostic Fields + Linear Classifier [18]	47
Selective Pooling Vector + SVM [19]	52
Initial Convolutional Neural Network	15
Transfer Neural Network	74

Table 3 - Stanford Dogs Accuracy Comparison

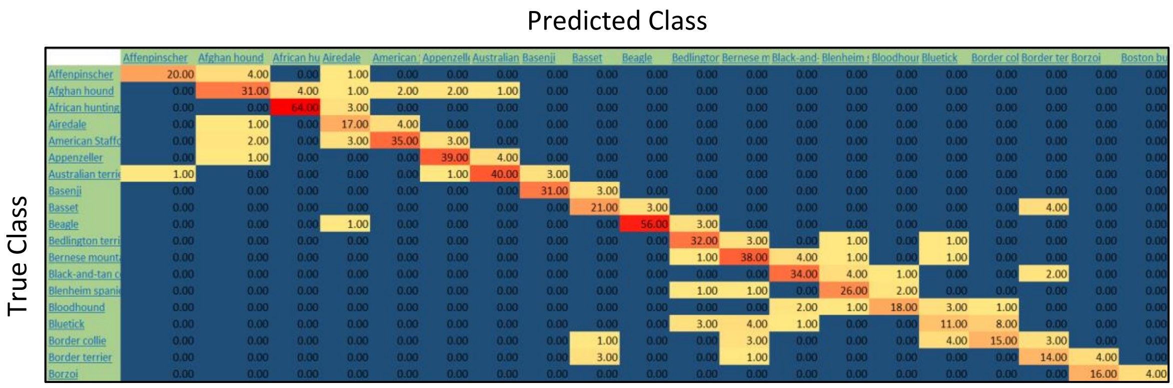
5 Result Analysis

This section discusses the classifier performance results and the effect of the different techniques on the given results. The advantages and drawbacks of each of the techniques are investigated along with the specific class performances. The confusion matrices from the initial model and the transfer model are compared to evaluate the performance improvement. The top k accuracy is evaluated to see how applicable the classifier is for app implementation.

5.1 Confusion Matrix

When estimating the performance of a classifier the classification accuracy is often used. This is the ratio between the number of correct predictions made and the number of total predictions. However, when dealing with a multi-class classification problem, the reasoning for the error can become unclear when only using classification accuracy. If the classifier achieves 90% this could be due to it correctly classing 9 out of 10 classes and completely ignoring the 10th class. Also, if you have biased data which favours one class, a high classification accuracy could be achieved which wouldn't represent the overall performance on all the classes. To avoid these issues a confusion matrix/error matrix is used to evaluate the classifiers performance. The confusion matrix displays the expected classes along one axis and the predicted classes along the other. The counts of the correct and incorrect classes are then filled into the table. This representation allows us to view how the incongruities occur, for example if one class is misclassified as another.

The confusion matrix for the final classifier for a top 1 classification accuracy can be found in the appendices. An example confusion matrix for a subset of 30 of the dog breeds is shown in the figure below. An ideal classifier will produce an identity matrix as its output. As you can see from the matrix heatmap the African hunting dog and the beagle have the strongest results from this subset, this may be due to their unique appearances.


Predicted Class
True Class

	Affenpinscher	Afghan hound	African hu	Airedale	American :	Appenzell	Australian	Basenji	Basset	Beagle	Bedlingtor	Bernese m	Black-and-	Blenheim :	Bloodhour	Bluetick	Border col	Border ter	Borzoi	Boston bu
Affenpinscher	20.00	4.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Afghan hound	0.00	31.00	4.00	1.00	2.00	2.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
African hunting	0.00	0.00	64.00	3.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Airedale	0.00	1.00	0.00	17.00	4.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
American Staffo	0.00	2.00	0.00	3.00	35.00	3.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Appenzeller	0.00	1.00	0.00	0.00	0.00	39.00	4.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Australian terri	1.00	0.00	0.00	0.00	0.00	1.00	40.00	3.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Basenji	0.00	0.00	0.00	0.00	0.00	0.00	0.00	31.00	3.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Basset	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	21.00	3.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	4.00	0.00	0.00
Beagle	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	56.00	3.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Bedlington terri	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	32.00	3.00	0.00	1.00	0.00	1.00	0.00	0.00	0.00	0.00
Bernese mount	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	38.00	4.00	1.00	0.00	1.00	0.00	0.00	0.00	0.00
Black-and-tan c	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	34.00	4.00	1.00	0.00	0.00	2.00	0.00	0.00
Blenheim spani	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	1.00	0.00	26.00	2.00	0.00	0.00	0.00	0.00	0.00
Bloodhound	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	2.00	1.00	18.00	3.00	1.00	0.00	0.00	0.00
Bluetick	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	3.00	4.00	1.00	0.00	0.00	11.00	8.00	0.00	0.00	0.00
Border collie	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	3.00	0.00	0.00	0.00	4.00	15.00	3.00	0.00	0.00
Border terrier	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	3.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	14.00	4.00	0.00
Borzoi	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	16.00	4.00

Figure 35 - Confusion Matrix Section

The confusion matrix can then used to calculate the recall and the precision of the classifier and to analyse the individual class performance. Recall is the proportion of correctly predicted labels of X from all instances that should have label X. Precision is given the predicted labels of X how many were correctly predicted, as shown by the equations below.

$$Recall = \frac{Sum(True\ Positives)}{Sum(True\ Labels)} \qquad Precision = \frac{Sum(True\ Positives)}{Sum(Label\ Predictions)}$$

Recall, also known as sensitivity, shows how sensitive the network is to the given class. The Recall for all the different classes can be found in the appendices. As you can see Collie and Walker Hound have the lowest recall values of 0.370 & 0.407 respectively. The Burmese Mountain dog and Bedlington Terrier achieved the highest recall of 0.915 & 0.902.

The precision results can also be found in the appendices. The results show that the two classes achieving the lowest precision are the Eskimo dog and the Siberian Huskey, with values of 0.333 & 0.500 respectively. The classes with the highest precision are the Chow and Airedale, with precision values of 0.913 and 0.907 respectively.

5.2 Class Evaluation

The recall results show the Collie has the lowest value of 0.333 which may be due to the Collie class having a very large amount of intra-class variation as shown by the figures below. The large amounts of variation may cause this class to be misclassified. The Collie has a high precision of 0.714 which indicates the images the classifier decides are Collies are likely to be Collies. However due to the low recall a lot of Collies are missed. There is a large amount colour variation within the class which may contribute to the low recall.

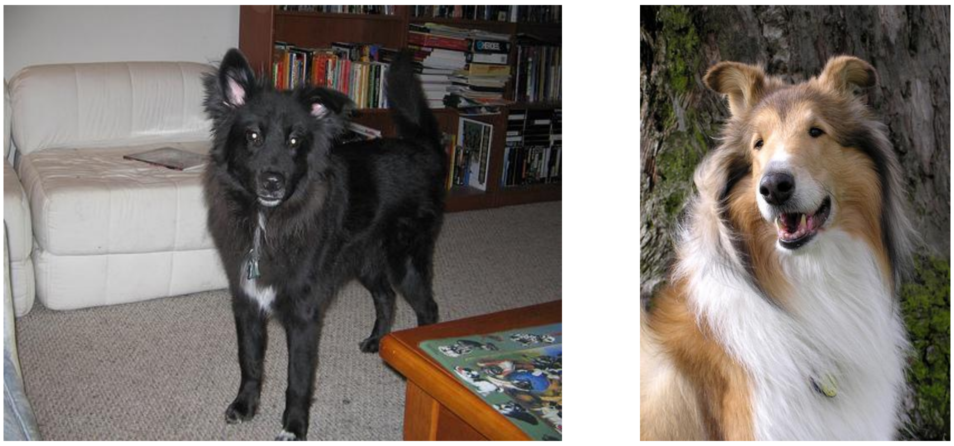


Figure 36 - Collie Training Images [17]

The Burmese Mountain dog has the highest recall value of 0.915. After evaluating the images of the breed it is clear it has very distinctive and unique features. There is only a small amount of intra-class variation and individuals have similar size and shape features. The breed has a distinctive white stipe between the eyes and a distinctive fur pattern which may be used by the classifier to identify the breed.

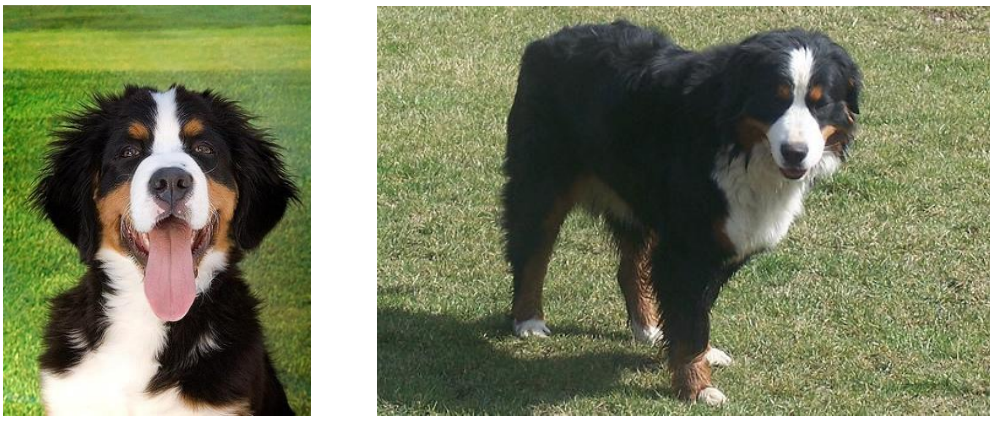


Figure 37 - Burmese Mountain Dog Training Images [17]

The high recall for the Burmese Mountain dog class indicates that fur colour/patterns play a key role in the classifier's identification process. As classes in this database have fine-grained differences an ideal classifier will use a number of different features such as size, shape and colour to identify the different classes.

The classes with the lowest precision are the Eskimo Dog and the Siberian Huskey. These two classes have the most incrorrect predictions for their given class. The Eskimo Dog has 14 out of a possible 33 samples which are predicted as a Siberian Huskey. The Huskey has 7 out of a possible 38 samples that are predicted as the Eskimo Dog and 10 predicted as a Malamute. These breeds are shown below. As you can see these three breeds have many simlar characteristcs and features and therefore are difficult to separate based on apearance alone.

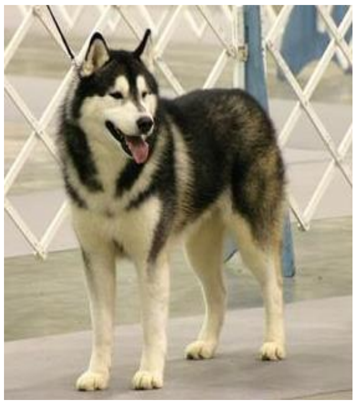


Figure 40 - Siberian Huskey [17]

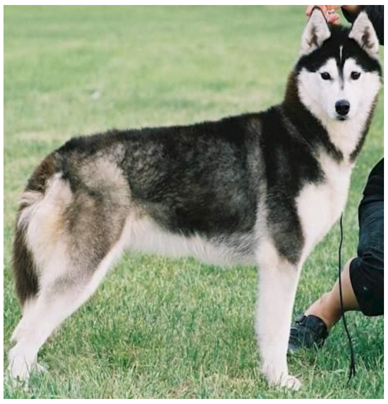


Figure 40 - Eskimo Dog [17]

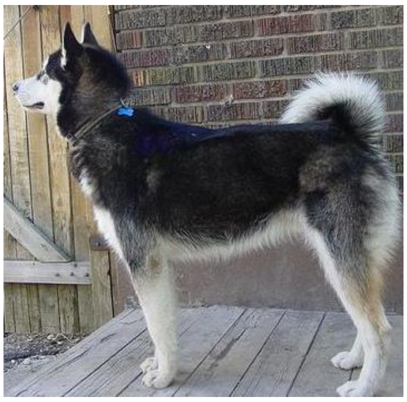


Figure 40 – Malamute [17]

The classes that achieve the highest precision tend to have the most distinctive features, as shown by the figures below.

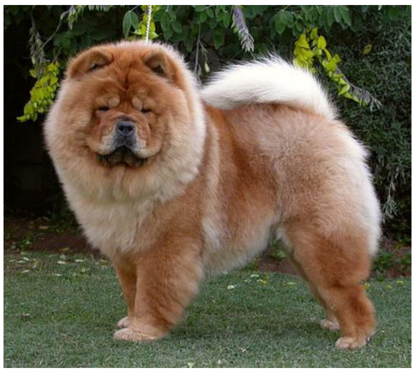


Figure 43 - Chow, Precision 0.913 [17]

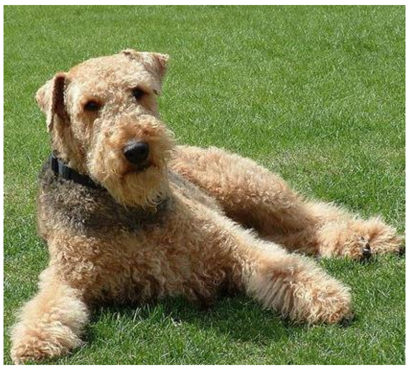


Figure 43 - Airedale, Precision 0.907 [17]

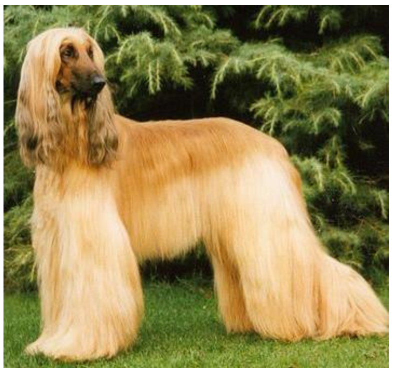


Figure 43 - Afghan Hound, Precision 0.895 [17]

In machine learning Bayes error rate is the lowest achievable prediction error rate for a classifier also known as an irreducible error. The confusion matrix results indicate that the Bayes error rate is being approached as humans miss classify the breeds at a similar rate. Further work would involve testing human performance using vets and owners classifying the breeds and comparing the performance of the classifier to these results.

5.3 Top K accuracy

Top K accuracy is another tool for evaluating a classifiers performance. Top 1 accuracy is the conventional method for evaluating the classifiers performance, where the model compares the true class with the prediction with the highest probability. In top K accuracy the true class is compared to the top K predictions rather than the top 1. The top K accuracy, is a good measure of how close the classifier is to high performance as it makes it clear if there are random guesses being made. The top K accuracies for the final classifier have been found and plotted in the figure below. As you can see the from the figure, the accuracy on the testing data rises from 0.74 up to 0.94 as the K value increases from 1 to 5. As the K value is increased up to 120 the accuracy will slowly rise until it reaches 1.00.

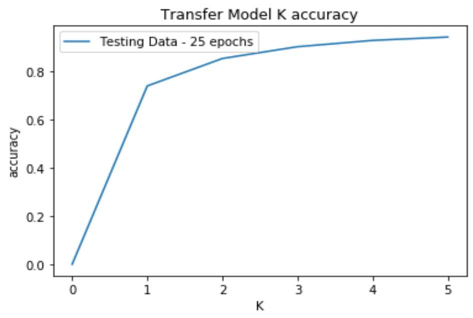


Figure 44 - Top K accuracy after 25 epochs

5.4 Model Comparison

When selecting which network to use for transfer learning all the networks are implemented with the same two fully connected layer output, to evaluate which network performs best on the dataset. The loss and accuracy plots for the different networks over 20 epochs are shown in the figures below. As you can there is a large difference in the performance of some of the architectures which is due to how generalised the networks are. From the graphs it is clear the Xception model performs best and therefore it was selected for the final implementation.

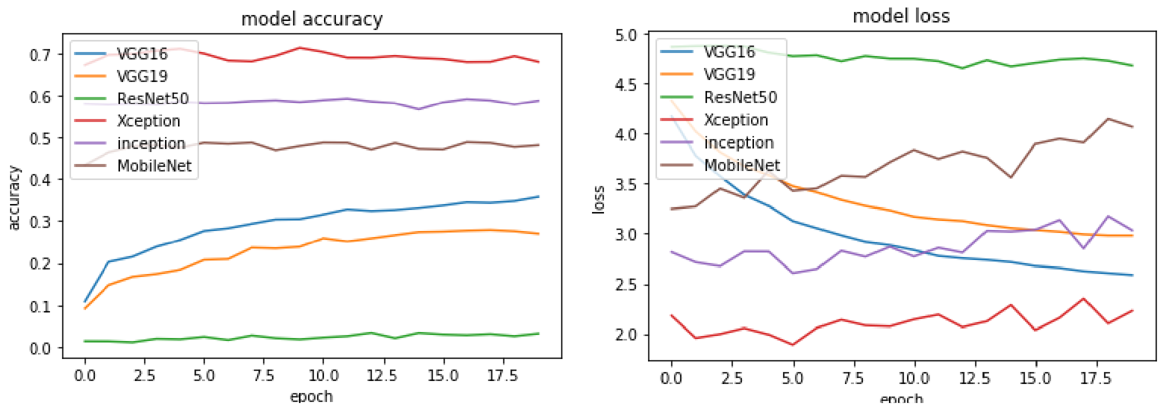


Figure 45 - ImageNet Model Accuracy and Loss Comparisons

After the Xception model was selected and implemented, it is compared to the previous base model performance. The difference in the accuracy plots over 150 epochs is clear the initial model's accuracy rises then begins to decrease as it separates from the training accuracy. The transfer model's accuracy levels off indicating it is being limited from growth due to model depth/size or volume of train data. The transfer model has a considerably higher accuracy than the initial model so is preferred for im

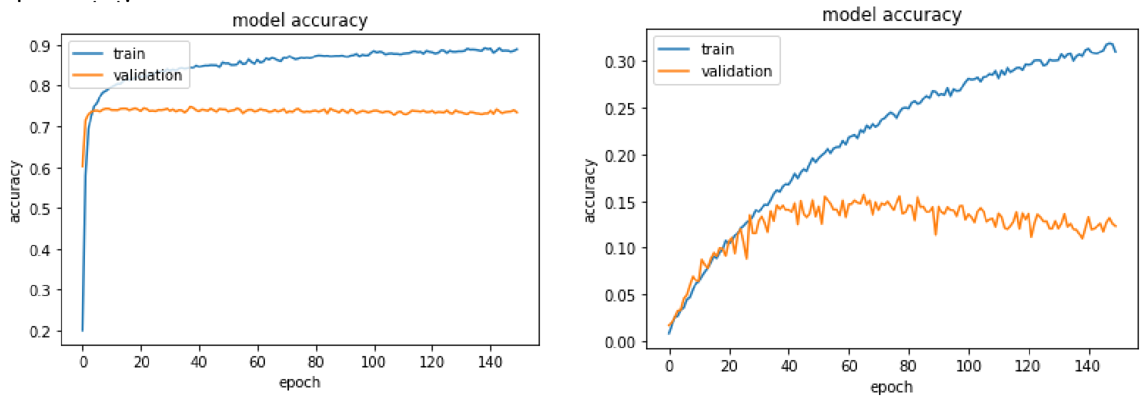


Figure 46 - Final Model Loss and Accuracy Plots

A selection of mixed breed classes was tested with the classifier to evaluate the classifiers performance with handling mixed breed inputs. No mixed breed labelled dataset images were found so a selection of mix-breed images was taken from Google Images. The classifier was then used to predict the top 5 classes of these images. However, the sample size was not large enough to conclusively determine the classifiers performance with handling mix-breed inputs. Future work will involve collating a new database with more labelled mix-breed images to train the network.

6 DISCUSSION AND FUTURE WORK

This section will discuss the project limitations and assumptions made as well as future work that could be made. The implementation of the final application is disused as well as the potential for expanding the class of breeds of dog and other animals.

6.1 Discussion

6.1.1 Project Assumptions and Limitations

Several assumptions are made during this project and the design and implementation of the classifier. These include assumptions made about the dataset used to train the network, that it is correctly labelled, and the bounding boxes correctly encompass the objects in the images. The dataset was created by the Stanford University computer science department and has been build using some of the images from the ImageNet dataset. It is therefore determined to be reliable and correctly labelled. The dataset is assumed to contain enough variation for training a network capable of real-world classification. After inspecting the images, it was decided that there is a good amount of variation and that the dataset would be satisfactory for implementing the classifier. The classifier performance is limited by the size of the dataset as a larger dataset would most likely improve the accuracy.

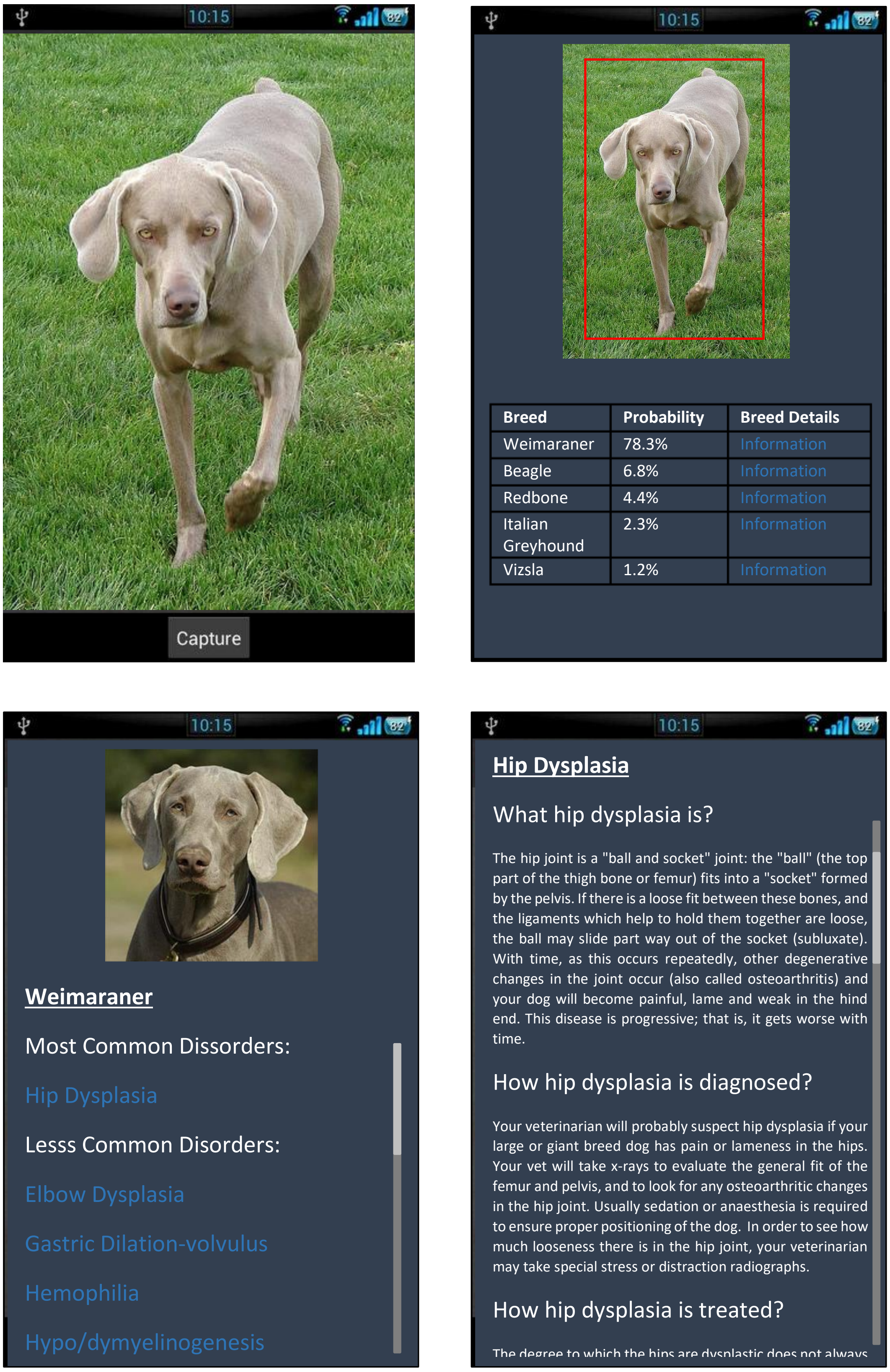
Another assumption made is that, in the final application the bounding boxes that are formed around the dogs will have a high success rate. This is an ambitious assumption as the technology is relatively new. However, YOLO is one of the best object detection algorithms and so is assumed to work for the final app. Further work may involve implementing our own object detector specifically tuned for detecting dogs, which may have a higher success rate than YOLO. When implementing the final application, it is assumed that the genetic disorder databases being used, have a comprehensive list of all current diseases that can affect the breeds in the dataset. Further steps can be taken to consult a veterinary specialist to ensure the correct information is included in the app database.

Crossbreed images taken from Google Images are assumed to be correct based on the image caption. The volume of images collected was not sufficient to make a conclusive determination on the classifier's performance with handling mix-breed classes. Therefore, this assumption can be overlooked.

6.2 Conclusion & Future Work

6.2.1 App Implementation

One of the goals of this project was to design an application that can aid owners and vets in the classification and identification of genetic diseases. Due to the time scale of the project only the backend classifier of the application was implemented. Future work would include implementing the full application on android and iOS. The final application will have the capability of correctly identifying the breeds or cross-breeds, determining which diseases are common in the breed and informing the user. Information on what the disorder is, how it is diagnosed, how it is treated, technical veterinarian information and breeding advise would be provided. A mock up of the application is shown in the figures below. The image is taken, then using the YOLO object detection a bounding box is formed around the dog. Our classifier is then used to identify the breed of dog and give the top 5 predictions of the breed and their probability. The breed can then be selected, and all the corresponding disorders and their prevalence are shown. Each disorder can be selected to give information about the disorder and how it is treated.



Breed	Probability	Breed Details
Weimaraner	78.3%	Information
Beagle	6.8%	Information
Redbone	4.4%	Information
Italian Greyhound	2.3%	Information
Vizsla	1.2%	Information

Weimaraner

Most Common Dissorders:

Hip Dysplasia

Lesss Common Disorders:

Elbow Dysplasia

Gastric Dilation-volvulus

Hemophilia

Hypo/dymyelinogenesis

Hip Dysplasia

What hip dysplasia is?

The hip joint is a "ball and socket" joint: the "ball" (the top part of the thigh bone or femur) fits into a "socket" formed by the pelvis. If there is a loose fit between these bones, and the ligaments which help to hold them together are loose, the ball may slide part way out of the socket (subluxate). With time, as this occurs repeatedly, other degenerative changes in the joint occur (also called osteoarthritis) and your dog will become painful, lame and weak in the hind end. This disease is progressive; that is, it gets worse with time.

How hip dysplasia is diagnosed?

Your veterinarian will probably suspect hip dysplasia if your large or giant breed dog has pain or lameness in the hips. Your vet will take x-rays to evaluate the general fit of the femur and pelvis, and to look for any osteoarthritic changes in the hip joint. Usually sedation or anaesthesia is required to ensure proper positioning of the dog. In order to see how much looseness there is in the hip joint, your veterinarian may take special stress or distraction radiographs.

How hip dysplasia is treated?

The degree to which the hips are dysplastic does not always

Figure 47 - Application Mock-up Design

6.2.2 Dataset Expansion

The scope of the application can be expanded by adding more classes to classify. The Stanford Dogs dataset contains 120 different breeds but the FCI recognise more than 360 different breeds worldwide. Future work will involve adding these other classes to the dataset which will need to be correctly labelled and have bounding box annotations. Sourcing these images in a large task hence it has fallen outside the scope of this project. For instance, the Stanford Dogs dataset contains no Dachshund breed or Dutch Shepherd as shown in the figures below.

Figure 49 – Dutch Shepard https://www.akc.org/dog-breeds/dutch-shepherd/

Figure 48 – Dachshund https://www.akc.org/dog-breeds/dachshund/

The Dachshund especially has a high likelihood of having a genetic disease, it is reported that between 20-25% of Dachshunds will suffer from Intervertebral Disc Disease (IVDD) in their lifetime. Different countries have different proportions of breeds which is why the Stanford Dogs dataset is tailored more towards the American breed pool. As cats are another common pet that suffers from a large range of genetic diseases, cat classes could be added to the dataset. The Fédération Internationale Féline (FIFe) recognises 43 breeds of cat but the largest labelled dataset of cat breeds found was the Oxford Pet Dataset [46] which contained 12 different cat breeds with approximately 200 images per-breed. To improve the performance of the current classifier more training images can be added to the current dataset which, as with the cat images, will have to be labelled and annotated correctly.

6.2.3 CNN improvement

The final CNN was tested on a small set of cross-breed class images. This was not done on a larger set as the time needed to label and annotate each of the images was too great. In future work a new dataset including cross-breed class images is to be created, it will be used to train and test the network so that the CNN will have the capability of classing cross-breed dogs. To further improve the current performance the weights of the Xception model can be optimised during training, so that the Xception model becomes more specialised at classifying dog breeds, rather than being generalised as it is currently. Further work can be undertaken investigating new classification tools that may better suit this fine-grained classification task.

6.2.4 Conclusion

In this report the design of an application to aid vets, owners and breeders in the identification of dog breeds and associated genetic disorders was investigated. A tool for classifying an image of a dog using a convolutional neural network was implemented. This report found that using a basic convolutional neural network based on the LeNet-5 architecture, performed better than other classification techniques such as linear classifiers using gnostic fields and selective pooling vectors. The basic CNN approach was then improved using transfer learning to extract bottleneck features from the Xception ImageNet competition classifier. The training, testing and validation images were all augmented using a datagenerator and fed to the Xception model. The loss calculated using categorical cross entropy was minimised using back-propagation and the ADAM optimiser.

The results from implementing the transfer model showed a very high classification top-1 accuracy of 0.744 and a top-5 accuracy of 0.94. When compared to the initial model and the models using the linear classifier, this technique was considerably higher. The class results using this classifier were investigated and it was found that the classes with similar characteristics were miss-classified most often and the more unique looking dogs had a higher precision and recall. This indicated that the classifier was approaching Bayes error rate. The performance of the classifier shows that it has commercialisation potential with the application. However, for a truly successful app more classes and cross-breed classes should be added to the training dataset in future work.

The final model was implemented using the TensorFlow backend machine learning framework, with the Keras high-level machine learning library. The final application would use the YOLO bounding box real-time object detection algorithm. To improve accuracy further future work may include designing a bespoke object detection algorithm. This application has large commercial potential, as it could prove very useful to breeders, owners and vets alike and no tool or application currently occupies this niche.

Appendices

Appendix A – Bottleneck Features Diagram

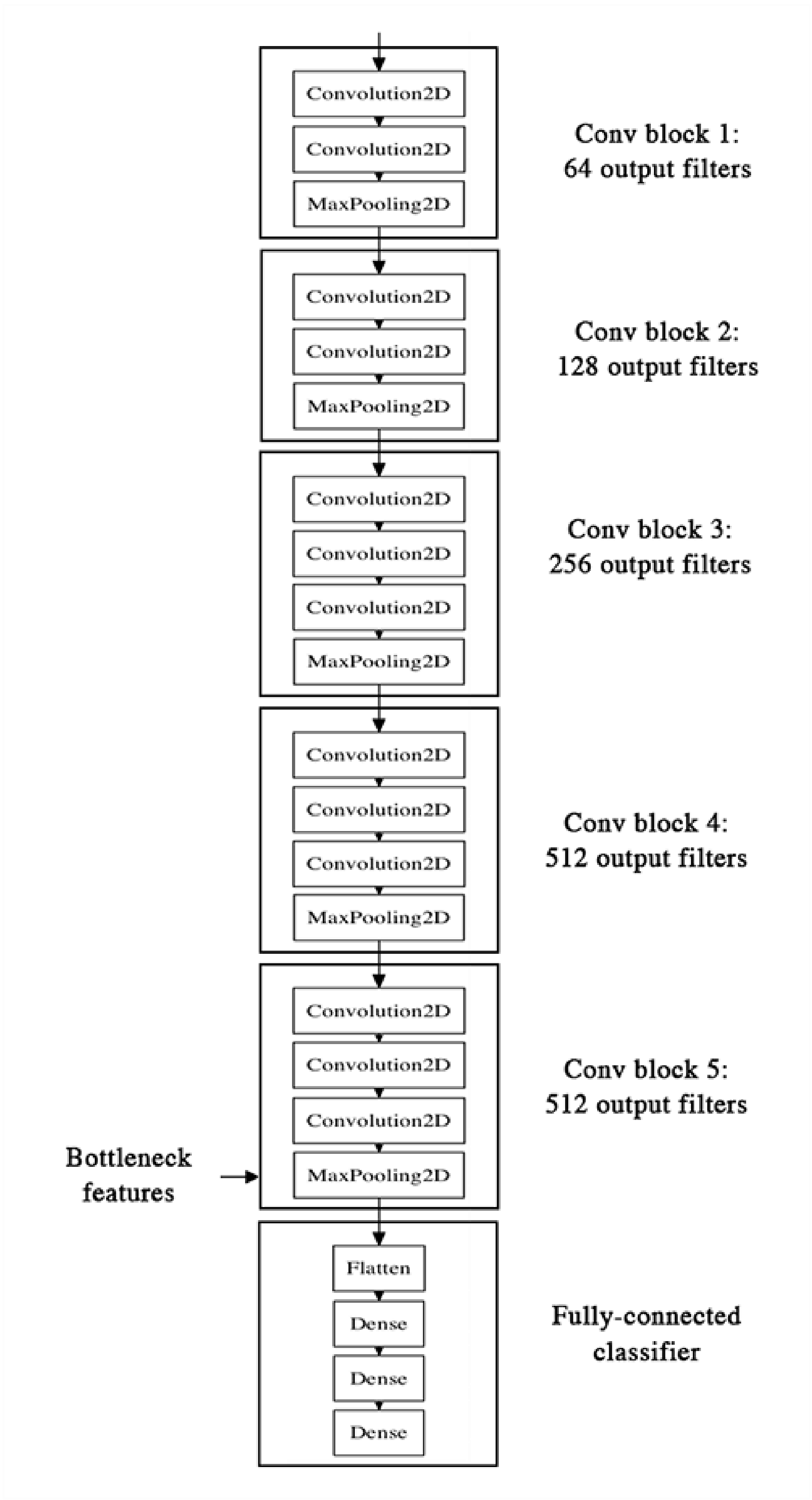
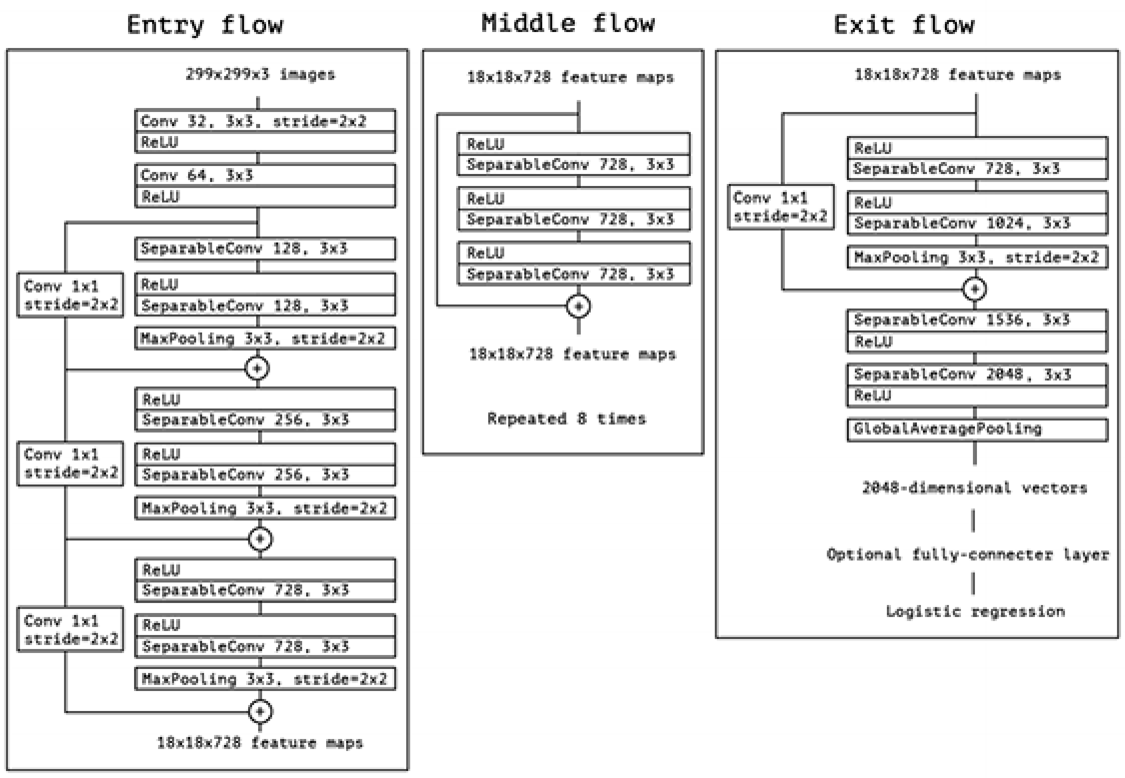
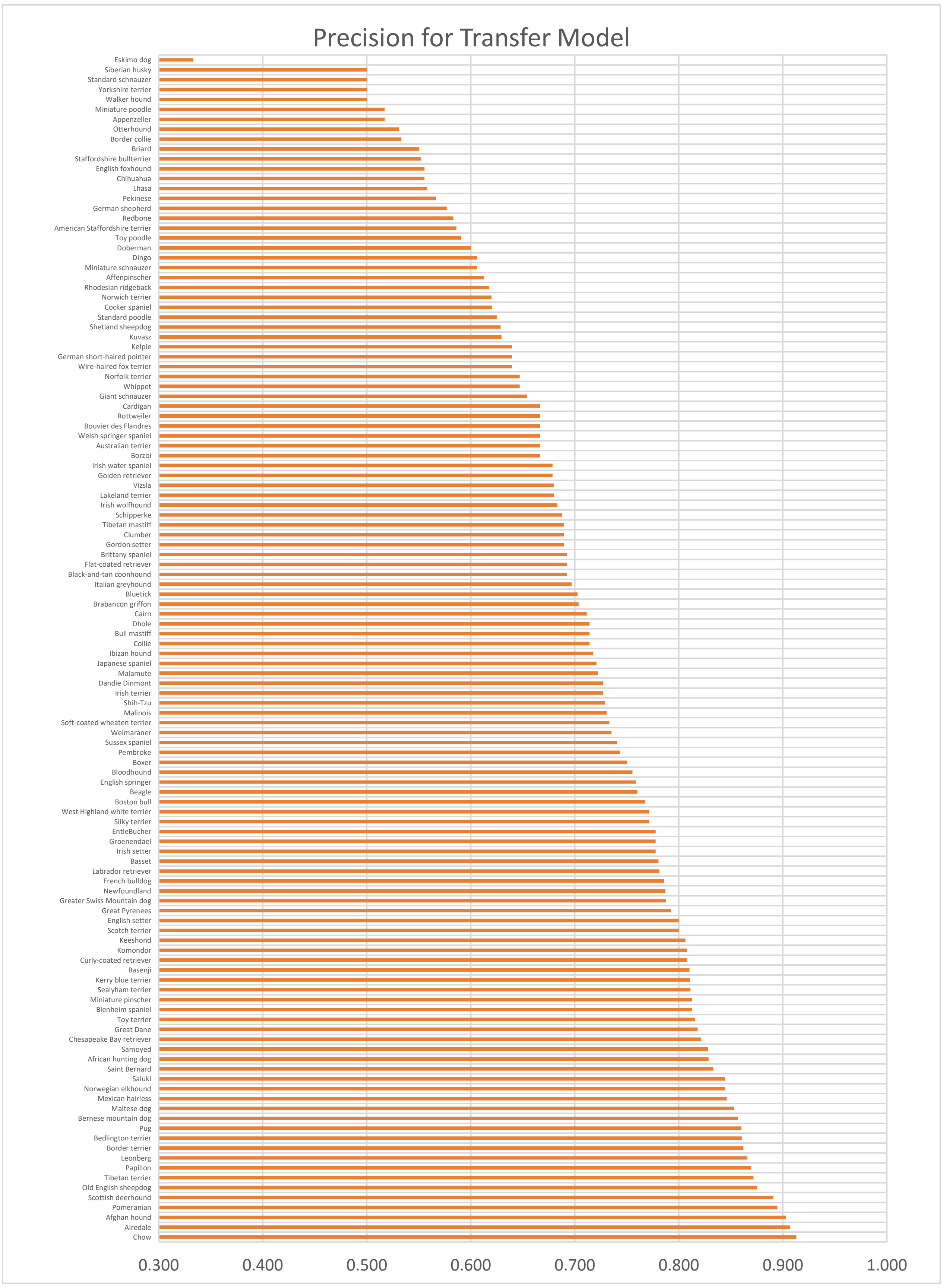


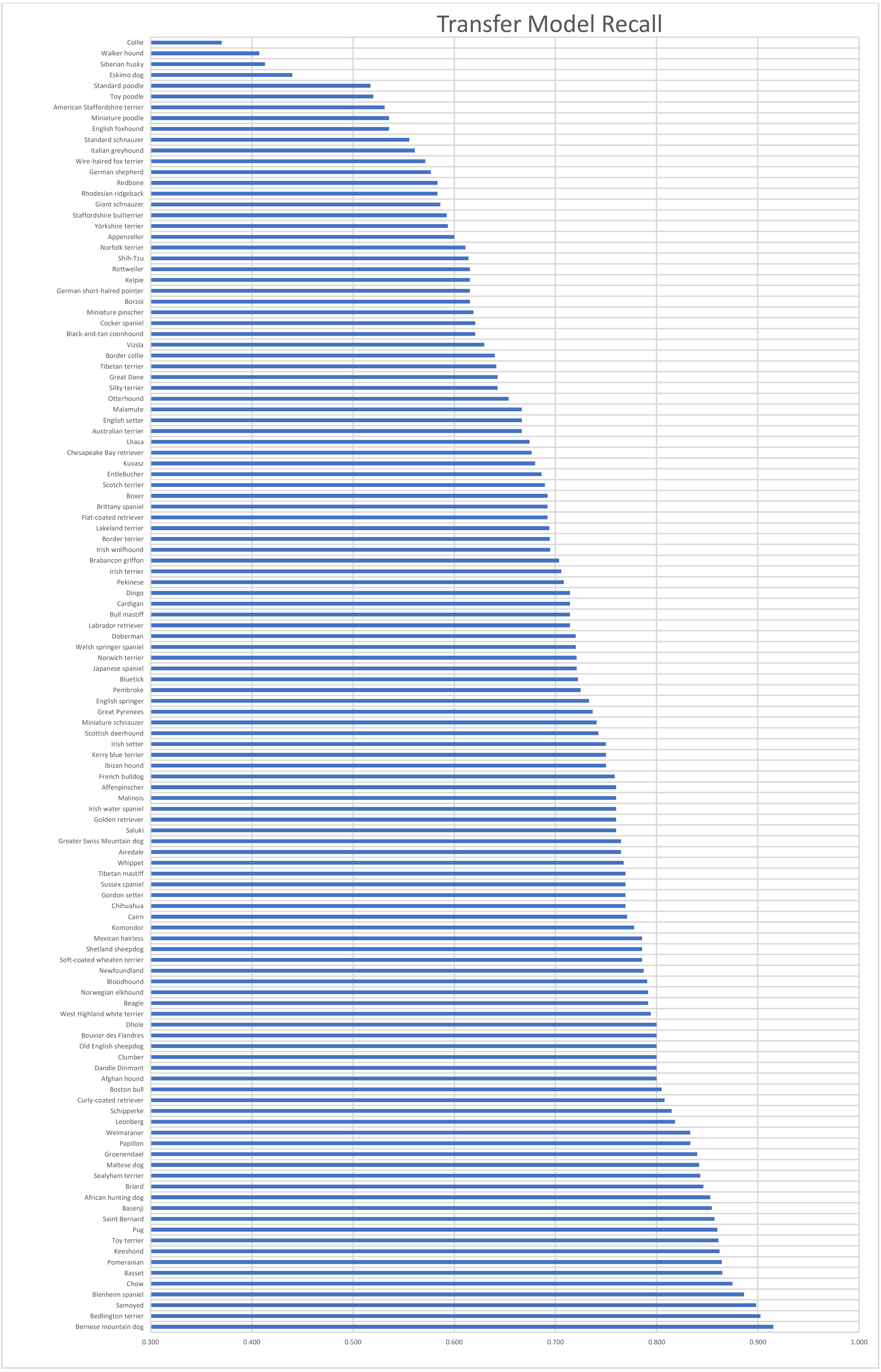
Figure 50 - https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

Appendix B – Xception Architecture



Appendix C – Recall and Precision Charts




Transfer Model Recall
Collie
Walker hound
Siberian husky
Eskimo dog
Standard poodle
Toy poodle
American Staffordshire terrier
Miniature poodle
English foxhound
Standard schnauzer
Italian greyhound
Wire-haired fox terrier
German shepherd
Redbone
Rhodesian ridgeback
Giant schnauzer
Staffordshire bullterrier
Yorkshire terrier
Appenzeller
Norfolk terrier
Shih-Tzu
Rottweiler
Kelpie
German short-haired pointer
Borzoi
Miniature pinscher
Cocker spaniel
Black-and-tan coonhound
Vizsla
Border collie
Tibetan terrier
Great Dane
Silky terrier
Otterhound
Malamute
English setter
Australian terrier
Lhasa
Chesapeake Bay retriever
Kuvasz
EntleBucher
Scotch terrier
Boxer
Brittany spaniel
Flat-coated retriever
Lakeland terrier
Border terrier
Irish wolfhound
Brabancon griffon
Irish terrier
Pekinese
Dingo
Cardigan
Bull mastiff
Labrador retriever
Doberman
Welsh springer spaniel
Norwich terrier
Japanese spaniel
Bluetick
Pembroke
English springer
Great Pyrenees
Miniature schnauzer
Scottish deerhound
Irish setter
Kerry blue terrier
Ibizan hound
French bulldog
Affenpinscher
Malinois
Irish water spaniel
Golden retriever
Saluki
Greater Swiss Mountain dog
Airedale
Whippet
Tibetan mastiff
Sussex spaniel
Gordon setter
Chihuahua
Cairn
Komondor
Mexican hairless
Shetland sheepdog
Soft-coated wheaten terrier
Newfoundland
Bloodhound
Norwegian elkhound
Beagle
West Highland white terrier
Dhole
Bouvier des Flandres
Old English sheepdog
Clumber
Dandie Dinmont
Afghan hound
Boston bull
Curly-coated retriever
Schipperke
Leonberg
Weimaraner
Papillon
Groenendael
Maltese dog
Sealyham terrier
Briard
African hunting dog
Basenji
Saint Bernard
Pug
Toy terrier
Keeshond
Pomeranian
Basset
Chow
Blenheim spaniel
Samoyed
Bedlington terrier
Bernese mountain dog
0.300
0.400
0.500
0.600
0.700
0.800
0.900
1.000

Appendix D – Initial CNN Code

```
# Import Libraries
import keras
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D,GlobalAveragePooling2D
from keras.layers import Activation, Dropout, Flatten, Dense
from keras.callbacks import ModelCheckpoint
from keras import regularizers
from keras.utils import to_categorical
import numpy as np
import scipy.io as sio
import matplotlib.pyplot as plt
import functools

'''
Initialise Targets
'''
# Load the image annotations and lacations indexes
file_mat = sio.loadmat('file_list.mat')
train_mat = sio.loadmat('train_list.mat')
test_mat = sio.loadmat('test_list.mat')
annotations_mat = sio.loadmat('test_list.mat')
Classes = np.load('Classes.npy')


# Initialise the class targets
train_targets = []
test_targets = []
validation_targets = []

# Loop through all labels and add label to target array
for x in range (len(train_mat['file_list'])):
    train_targets = np.append(train_targets, train_mat['labels'][x][0]-1)

for x in range (0,len(test_mat['file_list'])-1,2):
    test_targets = np.append(test_targets, test_mat['labels'][x][0]-1)

for x in range (1,len(test_mat['file_list'])-1,2):
    validation_targets = np.append(validation_targets, test_mat['labels'][x][0]-1)

# One-Hot Encode label targets
train_targets = to_categorical(train_targets)
test_targets = to_categorical(test_targets)
validation_targets = to_categorical( validation_targets)


'''
Initial Scratch Model
'''
# Set image dimension
image_size = 250

# Initialise Datagen
datagen = ImageDataGenerator(rotation_range=40,
                             width_shift_range=0.2,
                             height_shift_range=0.2,
                             rescale=1./255,
                             shear_range=0.2,
                             zoom_range=0.2,
                             horizontal_flip=True,
                             fill_mode='nearest')

# Setup sequential network
model = Sequential()
model.add(Conv2D(filters = 16,kernel_size = (5,5),strides = (2,2),padding = 'valid', activation ='relu',input_shape = (image_size, image_size, 3)))
model.add(Conv2D(filters = 32, kernel_size = (5,5), strides = (4,4), padding = 'valid', activation = 'relu'))
model.add(MaxPooling2D(pool_size=(3, 3), strides=None, padding='valid'))
model.add(Conv2D(filters = 64, kernel_size = (2,2), strides = (2,2), padding = 'valid', activation = 'relu'))
model.add(GlobalAveragePooling2D())
model.add(Dense(200, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(120, activation='softmax'))
model.summary()

# Compile model using Adam optimiser
model.compile(loss='categorical_crossentropy',
              optimizer='Adam',
              metrics=['accuracy'])

# Set batch size
batch_size = 600
```

```
# Initialise Test Datagen
test_datagen = ImageDataGenerator(rescale=1./255)

# Augment training images
train_generator = datagen.flow_from_directory(
        'Annotation/Training',
        target_size=(image_size, image_size),
        batch_size=batch_size,
        class_mode='categorical',
        shuffle = True)

# Augment validation images
validation_generator = test_datagen.flow_from_directory(
        'Annotation/Validation',
        target_size=(image_size, image_size),
        batch_size=batch_size,
        class_mode='categorical',
        shuffle = True)

# Save the model parameters
Initial_checkpointer = ModelCheckpoint(filepath='saved_models/weights.best.Initial.hdf5',
                               verbose=1, save_best_only=True)

# Train the network using the generators
history = model.fit_generator(
                train_generator,
                steps_per_epoch= 12000 // batch_size,
                epochs=20,
                validation_data=validation_generator,
                validation_steps= 4289 // batch_size,
                callbacks=[Initial_checkpointer])

# Save the final weights
model.save_weights('Initial_try.h5')  # always save your weights after training or during training

# Plot results of initial model
print(history.history.keys())
# summarize history for accuracy
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```

Appendix E – Transfer CNN

```
'''
TRANSFER LEARNOMG Using InceptionV3
'''
# Load the Bottleneck Features from the Xception model
train_data = np.load(open('Bottleneck_Features\Xception_bottleneck_features_train.npy','rb'))
valid_data = np.load(open('Bottleneck_Features\Xception_bottleneck_features_validation.npy','rb'))
test_data = np.load(open('Bottleneck_Features\Xception_bottleneck_features_test.npy','rb'))

# Initialise Transfer Model
train_data.shape[1:]
Xception_model = Sequential()
Xception_model.add(GlobalAveragePooling2D(input_shape=train_data.shape[1:]))
Xception_model.add(Dense(150, activation='relu', kernel_regularizer=regularizers.l2(0.005)))
Xception_model.add(Dropout(0.4))
Xception_model.add(Dense(120, activation='softmax'))
# Sumarise the model
Xception_model.summary()
# Set up top 3 accuracy metric
top3_acc = functools.partial(keras.metrics.top_k_categorical_accuracy, k=3)
top3_acc.__name__ = 'top3_acc'
# Compile model
Xception_model.compile(loss='categorical_crossentropy', optimizer='Adam', metrics=['accuracy', top3_acc])
# Save Xception weights to memory
InceptionV3_checkpointer = ModelCheckpoint(filepath='saved_models/weights.best.Xception.hdf5',
                               verbose=1, save_best_only=True)
# Train the Transfer Model Parameters
history = Xception_model.fit(train_data, train_targets,
          validation_data=(valid_data, validation_targets),
          epochs=25, batch_size=300, callbacks=[InceptionV3_checkpointer], verbose=1)
# Load the saved weights
Xception_model.load_weights('saved_models/weights.best.Xception.hdf5')
# Calculate Model Accuracy
Xception_predictions = [np.argmax(Xception_model.predict(np.expand_dims(feature, axis=0))) for feature in test_data]
test_accuracy = 100*np.sum(np.array(Xception_predictions)==np.argmax(test_targets, axis=1))/len(Xception_predictions)
print('Test accuracy: %.4f%%' % test_accuracy)


# PLOT RESULTS
print(history.history.keys())
# summarize history for accuracy
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['top3_acc'])
plt.plot(history.history['val_top3_acc'])
plt.title('Top K Accuracy (K=5)')
plt.ylabel('Accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```

Appendix F – Bottleneck Feature Extraction

```
import tensorflow as tf
import numpy as np
from keras import applications
from keras.preprocessing import image
from keras.models import Model
from keras.models import Sequential
from keras.preprocessing.image import ImageDataGenerator, array_to_img, img_to_array, load_img
from keras.layers import Dense,GlobalAveragePooling2D
from keras.optimizers import Adam
import scipy.io as sio
from keras.callbacks import ModelCheckpoint


datagen = ImageDataGenerator(rescale=1. / 255)

img_width = 200
img_height = 200
Batch_Size  = 1


file_mat = sio.loadmat('file_list.mat')
train_mat = sio.loadmat('train_list.mat')
test_mat = sio.loadmat('test_list.mat')
annotations_mat = sio.loadmat('test_list.mat')

'''
##########################################################################
Initialise Data Generators
'''


train_generator = datagen.flow_from_directory('annotation/Training',
                                                    target_size=(img_height, img_width),
                                                    batch_size=Batch_Size,
                                                    class_mode='categorical',
                                                    shuffle=False)


test_generator = datagen.flow_from_directory('annotation/Testing',
                                                    target_size=(img_height, img_width),
                                                    batch_size=Batch_Size,
                                                    class_mode='categorical',
                                                    shuffle=False)

validation_generator = datagen.flow_from_directory('annotation/Validation',
                                                    target_size=(img_height, img_width),
                                                    batch_size=Batch_Size,
                                                    class_mode='categorical',
                                                    shuffle=False)

'''
##########################################################################
Load VGG16 Model and throw away top layer, then save bottleneck features
'''
print('Extracting VGG-16 Bottleneck Features')

model = applications.VGG16(include_top=False, weights='imagenet')

bottleneck_features_train = model.predict_generator(
        train_generator,12000)

np.save(open('Bottleneck_Features/VGG16_bottleneck_features_train.npy', 'wb'), bottleneck_features_train)


bottleneck_features_validation = model.predict_generator(
        validation_generator,4289)

np.save(open('Bottleneck_Features/VGG16_bottleneck_features_validation.npy', 'wb'), bottleneck_features_validation)

bottleneck_features_test = model.predict_generator(
        test_generator,4290)

np.save(open('Bottleneck_Features/VGG16_bottleneck_features_test.npy', 'wb'), bottleneck_features_test)
```

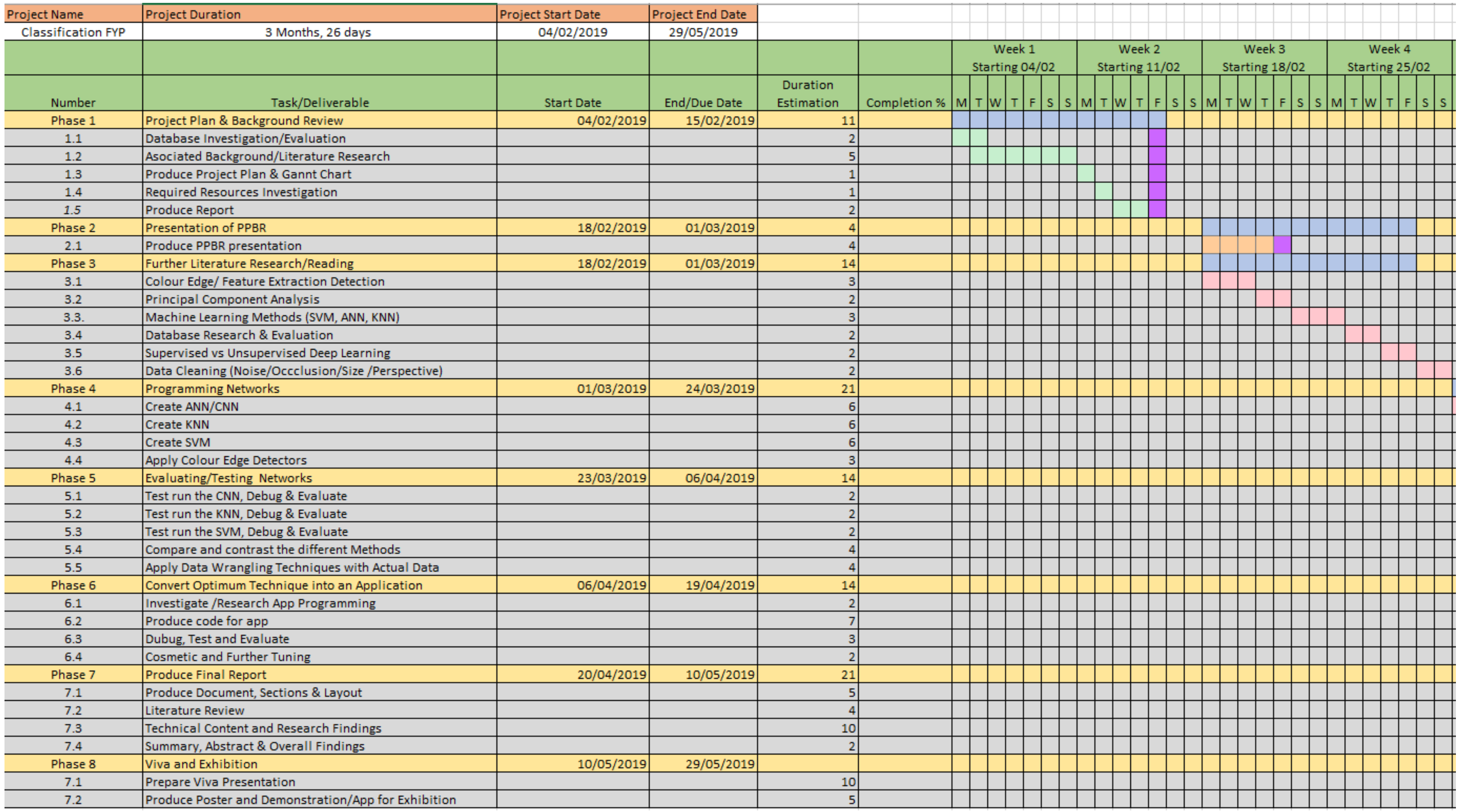
```
##########################################################################
# Extract Bottleneck features for VGG-19
print('Extracting VGG-19 Bottleneck Features','\n',)


model = applications.VGG19(include_top=False, weights='imagenet')

bottleneck_features_train = model.predict_generator(
        train_generator,12000)

np.save(open('Bottleneck_Features/VGG19_bottleneck_features_train.npy', 'wb'), bottleneck_features_train)


bottleneck_features_validation = model.predict_generator(
        validation_generator,4289)

np.save(open('Bottleneck_Features/VGG19_bottleneck_features_validation.npy', 'wb'), bottleneck_features_validation)

bottleneck_features_test = model.predict_generator(
        test_generator,4290)

np.save(open('Bottleneck_Features/VGG19_bottleneck_features_test.npy', 'wb'), bottleneck_features_test)



#########################################################################
#Extract Bottleneck features for ResNet-50
print('\n','Extracting ResNet Bottleneck Features','\n')

model = applications.ResNet50(include_top=False, weights='imagenet')

bottleneck_features_train = model.predict_generator(
        train_generator,12000)

np.save(open('Bottleneck_Features/ResNet50_bottleneck_features_train.npy', 'wb'), bottleneck_features_train)


bottleneck_features_validation = model.predict_generator(
        validation_generator,4289)

np.save(open('Bottleneck_Features/ResNet50_bottleneck_features_validation.npy', 'wb'), bottleneck_features_validation)

bottleneck_features_test = model.predict_generator(
        test_generator,4290)

np.save(open('Bottleneck_Features/ResNet50_bottleneck_features_test.npy', 'wb'), bottleneck_features_test)

#########################################################################
#Extract Bottleneck features for InceptionV3
print('\n','Extracting InceptionV3 Bottleneck Features')

model = applications.InceptionV3(include_top=False, weights='imagenet')

bottleneck_features_train = model.predict_generator(
        train_generator,12000)

np.save(open('Bottleneck_Features/InceptionV3_bottleneck_features_train.npy', 'wb'), bottleneck_features_train)


bottleneck_features_validation = model.predict_generator(
        validation_generator,4289)

np.save(open('Bottleneck_Features/InceptionV3_bottleneck_features_validation.npy', 'wb'), bottleneck_features_validation)

bottleneck_features_test = model.predict_generator(
        test_generator,4290)

np.save(open('Bottleneck_Features/InceptionV3_bottleneck_features_test.npy', 'wb'), bottleneck_features_test)
```

```
##########################################################################
#Extract Bottleneck features for Xception
from keras import applications

print('\n','Extracting Xception Bottleneck Features')

model = applications.Xception(include_top=False, weights='imagenet')

bottleneck_features_train = model.predict_generator(
        train_generator,12000)

np.save(open('Bottleneck_Features/Xception_bottleneck_features_train.npy', 'wb'), bottleneck_features_train)


bottleneck_features_validation = model.predict_generator(
        validation_generator,4289)

np.save(open('Bottleneck_Features/Xception_bottleneck_features_validation.npy', 'wb'), bottleneck_features_validation)

bottleneck_features_test = model.predict_generator(
        test_generator,4290)

np.save(open('Bottleneck_Features/Xception_bottleneck_features_test.npy', 'wb'), bottleneck_features_test)


##########################################################################
#Extract Bottleneck features for MobileNet
print('\n','Extracting MobileNet Bottleneck Features')

model = applications.MobileNet(include_top=False, weights='imagenet')

bottleneck_features_train = model.predict_generator(
        train_generator,12000)

np.save(open('Bottleneck_Features/MobileNet_bottleneck_features_train.npy', 'wb'), bottleneck_features_train)


bottleneck_features_validation = model.predict_generator(
        validation_generator,4289)

np.save(open('Bottleneck_Features/MobileNet_bottleneck_features_validation.npy', 'wb'), bottleneck_features_validation)

bottleneck_features_test = model.predict_generator(
        test_generator,4290)

np.save(open('Bottleneck_Features/MobileNet_bottleneck_features_test.npy', 'wb'), bottleneck_features_test)


##########################################################################
```

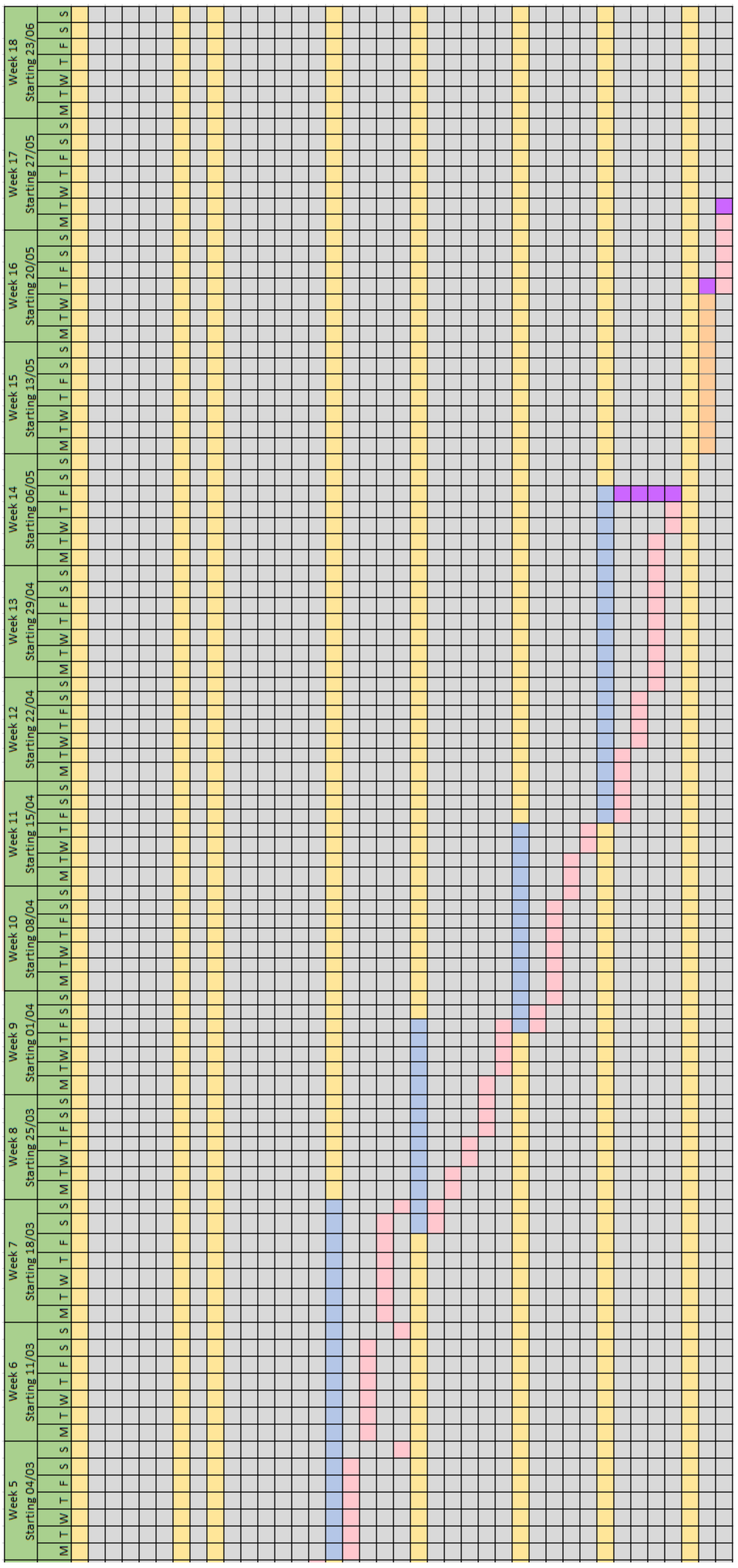
Appendix G – Image Re-Sizing Script

```python
import random
import numpy as np
import scipy.io as sio
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import cv2
import xml.etree.ElementTree as ET
import matplotlib.patches as patches
import os


file_mat = sio.loadmat('file_list.mat')
file_keys = list(file_mat.keys())
print(file_keys,'\n')

# Set new image lacation
path =  'Re-sized_Images/300x300'
val = 'file_list'
loc = file_mat[val]
Class = file_mat['labels']

# Randomly View 20 on the images in the Dataset and their coresponding labels
z = len(file_mat['labels'])

# Loop through all images in dataset
for x in range(z):
    '''
    Load Images
    '''
    #fig,ax = plt.subplots(1)
    a = loc[x][0]
    #print('\n','Images/' + a[0])
    #print(Class[x][0])
    photo = 'Images/' + a[0]
    img = mpimg.imread(photo)
    #ax.imshow(img)
    '''
    Set Bounding Box Around Dogs
    '''
    box = 'annotation/Annotation/'+a [0]
    x = len(box)
    box = box[0:x-4]
    e = ET.parse(box).getroot()
    file = '/' + str(a[0])
    #print(path+file)
    n = 5
    for el in e.findall('object'):
        #print('-----------------')
        for ch in el.findall('name'):
            #print(ch.tag,ch.text)
            pass

        for ch in el.findall('bndbox'):
            for chi in ch.getchildren():
               # print(chi.tag,chi.text)
               pass
        xmin = int(e[n][4][0].text)
        ymin = int(e[n][4][1].text)
        xmax = int(e[n][4][2].text)
        ymax = int(e[n][4][3].text)
        n = 6

        img_two = cv2.imread(photo)
        crop_img = img_two[ymin:ymax, xmin:xmax]
        resize = cv2.resize(crop_img,(300,300))
        cv2.imwrite(path+str(file),resize)
```

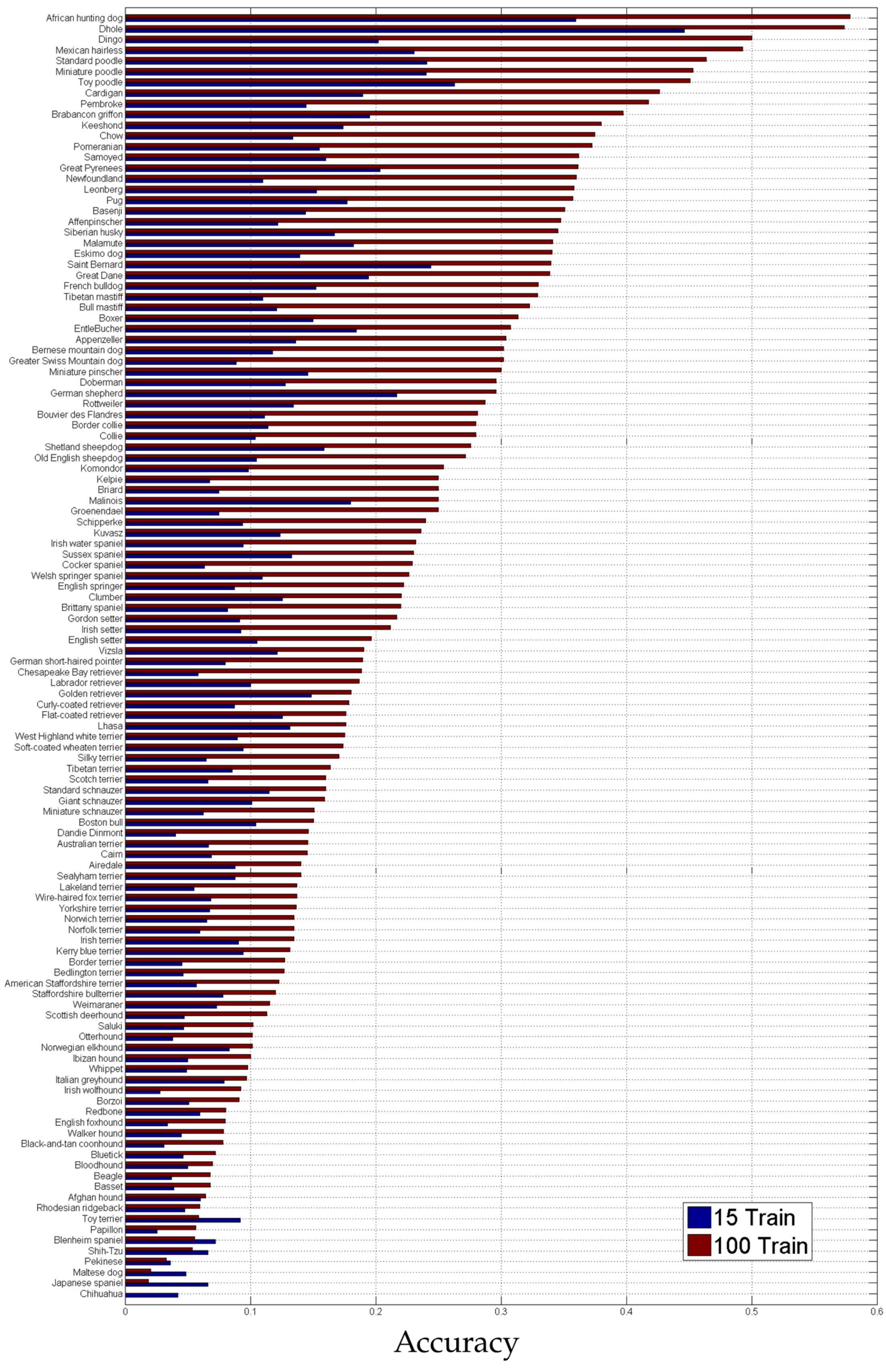
Appendix H - Gantt Chart



Project Name	Project Duration	Project Start Date	Project End Date		
Classification FYP	3 Months, 26 days	04/02/2019	29/05/2019		
Number	Task/Deliverable	Start Date	End/Due Date	Duration Estimation	Completion %
Phase 1	Project Plan & Background Review	04/02/2019	15/02/2019	11	
1.1	Database Investigation/Evaluation			2	
1.2	Asociated Background/Literature Research			5	
1.3	Produce Project Plan & Gannt Chart			1	
1.4	Required Resources Investigation			1	
1.5	Produce Report			2	
Phase 2	Presentation of PPBR	18/02/2019	01/03/2019	4	
2.1	Produce PPBR presentation			4	
Phase 3	Further Literature Research/Reading	18/02/2019	01/03/2019	14	
3.1	Colour Edge/ Feature Extraction Detection			3	
3.2	Principal Component Analysis			2	
3.3.	Machine Learning Methods (SVM, ANN, KNN)			3	
3.4	Database Research & Evaluation			2	
3.5	Supervised vs Unsupervised Deep Learning			2	
3.6	Data Cleaning (Noise/Occclusion/Size /Perspective)			2	
Phase 4	Programming Networks	01/03/2019	24/03/2019	21	
4.1	Create ANN/CNN			6	
4.2	Create KNN			6	
4.3	Create SVM			6	
4.4	Apply Colour Edge Detectors			3	
Phase 5	Evaluating/Testing Networks	23/03/2019	06/04/2019	14	
5.1	Test run the CNN, Debug & Evaluate			2	
5.2	Test run the KNN, Debug & Evaluate			2	
5.3	Test run the SVM, Debug & Evaluate			2	
5.4	Compare and contrast the different Methods			4	
5.5	Apply Data Wrangling Techniques with Actual Data			4	
Phase 6	Convert Optimum Technique into an Application	06/04/2019	19/04/2019	14	
6.1	Investigate /Research App Programming			2	
6.2	Produce code for app			7	
6.3	Dubug, Test and Evaluate			3	
6.4	Cosmetic and Further Tuning			2	
Phase 7	Produce Final Report	20/04/2019	10/05/2019	21	
7.1	Produce Document, Sections & Layout			5	
7.2	Literature Review			4	
7.3	Technical Content and Research Findings			10	
7.4	Summary, Abstract & Overall Findings			2	
Phase 8	Viva and Exhibition	10/05/2019	29/05/2019		
7.1	Prepare Viva Presentation			10	
7.2	Produce Poster and Demonstration/App for Exhibition			5	

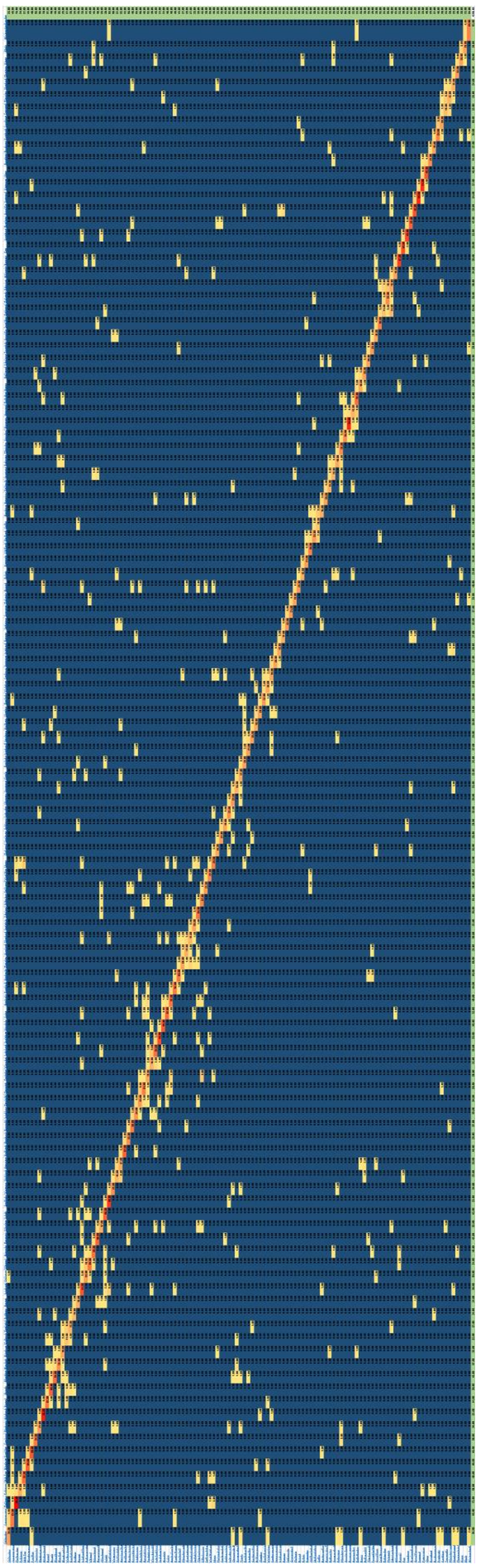

Week 5 Starting 04/03
Week 6 Starting 11/03
Week 7 Starting 18/03
Week 8 Starting 25/03
Week 9 Starting 01/04
Week 10 Starting 08/04
Week 11 Starting 15/04
Week 12 Starting 22/04
Week 13 Starting 29/04
Week 14 Starting 06/05
Week 15 Starting 13/05
Week 16 Starting 20/05
Week 17 Starting 27/05
Week 18 Starting 23/06
M T W T F S S

Appendix I – Linear Classifier SIFT Results [17]



Appendix J – Final Confusion Matrix

Note: Due to the size of the confusion matrix, the entire table was not included in this document only a screen shot is shown. The final matrix can be given upon request.



References

[1] A. H. D. S. H. A. B. J. e. a. Donner J, “Frequency and distribution of 152 genetic disease variants in over 100,000 mixed breed and purebred dogs,” *PLOS Genetics,* vol. 15, no. 1, p. e1007361. , 2019.

[2] U. o. S. Faculty of Veterinary Science, “Online Mendelian Inheritance in Animals, OMIA,” Sydney School of Veterinary Science, [Online]. Available: http://omia.org/. [Accessed 15 03 2019].

[3] e. Martín Abadi, “TensorFlow: A System for Large-Scale Machine Learning,” in *The 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI ’16).*, 2016.

[4] P. L. X. W. &. X. T. Ziwei L., “Deep Learning Face Attributes in the Wild,” Hong Kong, 2015.

[5] W. a. J. F.Yan, “A Tennis Ball Tracking Algorithm for Automatic Annotation of Tennis Match,” in *British machine vision conference*, Surrey, 2005.

[6] D. D. T. D. D. e. a. M Bojarski, “End to end learning for self-driving cars,” arXiv preprint, Holmdel, 2016.

[7] D. K. S Tong, “Support vector machine active learning with applications to text classification,” Stanford, 2001.

[8] R. F. F. G. E Osuna, “Training support vector machines: an application to face detection,” Cambridge, Massachusetts, 1997.

[9] V.Vapnik, The Nature of Statistical Learning, New York: Springer, 1995.

[10] V. V. Corinna Cortes, “Support-Vector Networks,” *Machine Learning ,* vol. 20, pp. 273-297, 1995.

[11] C.-J. L. Chih-Wei Hsi, “A Comparison of Methods for Multiclass Support Vector Machines,” *IEEE Tansactions on Neural Networks,* vol. 13, no. No.2, pp. 415-425, 2002.

[12] M. G. J. G. JM Keller, “A fuzzy k-nearest neighbor algorithm,” *IEEE Transaction on Systems, Man and Cybernetics,* vol. 15, no. 4, pp. 580-585, July 1985.

[13] E. S. M. I. O Boiman, “In Defense of Nearest-Neighbor Based Image Classification,” IEEE, 2008.

[14] B. T. Navneet Dalal, “Histograms of Oriented Gradient for Human Detection,” in *International Conference on Computer Vision*, San Diego, 2005.

[15] T. T. L. V. G. Herbert Bay, “SURF: Speeded Uo Robust Features,” in *Computer Vision – ECCV*, Leuven, 2006.

[16] D. G. Lowe, “Object Recognition from Local Scale-Invariant Features,” in *International Conference on*, Vancouver, 1999.

[17] N. J. B. Y. F.-F. L. Aditya Khosla, “Novel Dataset for Fine-Grained Image Categorization: Stanford Dogs,” Computer Science Department, Stanford University, Stanford, 2011.

[18] C. Kanan, “Fine-Grained Object Reconition with Gnostic Fields,” in *IEEE Winter COnference on Application of Computer Vision* , Steamboat Springs, 2014.

[19] J. Y. H. J. e. a. Guang Chen, “Selective Pooling Vector for Fine-grained Recognition,” in *IEEE Winter Conference on Applications of Computer Vision*, Waikoloa, 2015.

[20] K. Fukushima, “Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position,” Springer, Tokyo, 1980.

[21] W. D. R. S. L.-J. L. K. L. L. F.-F. Jia Deng, “ImageNet: A Large-Scale Hierarchical Image Database,” in *In CVPR09*, 2009.

[22] I. S. G. E. Alex Krizhevsky, “ImageNet Classification with Deep Convolutional Neural Networks,” in *Advances in Neural Information Processing Systems*, Tornoto, 2012.

[23] Y. Y. Y. W. S. Z. Q Zhang, “Interpreting cnns via decision trees,” arXiv, Hong Kong, 2018.

[24] P. H. L. B. Y. B. Yann LeCun, “Object Recognition with Gradient-Based Learning,” in *Shape, Contour and Grouping in Computer Vision* , Berlin, Springer, 1999, pp. 319-345.

[25] E. D. V. A. C. M. P. A. V. L. Rosasco, “Are Loss Functions All the Same?,” *Neural Computation,* vol. 16, no. 5, pp. 1063-1076, 2004.

[26] M. R. S. Zhilu Zhang, “Generalized Cross Entropy Loss for Training Deep Neural Networks with Noisy Labels,” in *32nd Conference on Neural Information Processing Systems*, Cornell, 2018.

[27] L. Bottou, “Large-Scale Machine Learning with Stochastic Gradient Descent,” in *COMPSTAT'2010*, Princton, 2010.

[28] E. H. Y. S. John Duchi, “Adaptive Subgradient Methods for Online Learning and Stochastic Optimization,” *Journal of Machine Learning Research,* vol. 12, pp. 2121-2159, 2011.

[29] J. L. B. Diederik P. Kingma, “ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION,” in *ICLR 2015*, 2015.

[30] F. S. T. K. W. Z. C. Y. C. L. Chuanqi Tan, “ A Survey on Deep Transfer Learning,” in *The 27th International Conference on Artificial Neural Networks*, 2018.

[31] F. Chollet, “Xception: Deep Learning with Depthwise Separable Convolutions,” arXiv:1610.02357, 2017.

[32] A. Z. Karen Simonyan, “VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION,” in *ICLR 2015* , Oxford, 2015.

[33] X. Z. S. R. J. S. Kaiming He, “Deep Residual Learning for Image Recognition,” in *arXiv:1512.03385*, 2015.

[34] V. V. S. I. J. S. Z. W. Christian Szegedy, “Rethinking the Inception Architecture for Computer Vision,” in *arXiv:1512.00567*, 2015.

[35] M. Z. B. C. D. K. W. W. T. W. M. A. H. A. Andrew G. Howard, “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications,” in *arXiv:1704.04861*, 2017.

[36] D. R. Sargan, “IDID: Inherited Diseases in Dogs: Web-based information for canine inherited disease genetics,” *Mammalian Genome,* vol. 15, no. 6, pp. 503-506, 2004.

[37] R. M. S. Kevin K. Dobbin, “Optimally splitting cases for training and testing high dimensional classifiers,” in *BMC medical genomics*, 2011.

[38] S. D. R. G. A. F. Joseph Redmon, “You Only Look Once: Unified, Real-Time Object Detection,” in *arXiv:1506.02640v5*, Washington, 2016.

[39] Q. C. S. Min Lin, “Network in Network,” arXiv:1312.4400, Singapore, 2013.

[40] G. H. A. K. I. S. R. S. Nitish Srivastava, “Dropout: A Simple Way to Prevent Neural Networks from Overfitting,” *Journal of Machine Learning Research,* vol. 15, pp. 1929-1958, 2014.

[41] C. S. Sergey Ioffe, “Batch Normalization: Accelerating Deep Network Training b y Reducing Internal Covariate Shift,” arXiv:1502.03167, 2015.

[42] J. Schmidhuber, “Deep Learning in Neural Networks: An Overview,” in *Neural Networks*, Elsevier, 2015, pp. 85-117.

[43] Y. L. Cun, “A Theoretical Framework for Back-Propagation,” Toronto, 1988.

[44] A. F. M. Agarap, “Deep Learning using Rectified Linear Units (ReLU),” arXiv:1803.08375, 2018.

[45] M. M. A. R. Corinna Cortes, “L2 Regularization for Learning Kernels,” 2009.

[46] A. V. A. Z. C. V. J. Omkar M Parkhi, “Cats and Dogs,” in *IEEE Conference on Computer Vision and Pattern Recognition*, Oxford, 2012.

[47] M. M. e. a. AS Abdul-Nasir, “Colour image segmentation approach for detection of malaria parasites using various colour models and k-means clustering,” WSEAS TRANSACTIONS on BIOLOGY, Perlis, 2013.

[48] V. B. Deepesh Rawat, “A Steganography Technique for Hiding Image in an Image using LSB Method for 24 Bit Color Image,” in *International Journal of Computer Applications (0975 – 8887) Volume 64– No.20, February 2013 15* , UTTARAKHAND, 2013.