UNIVERSITY OF SOUTHAMPTON
Faculty of Engineering, Science and Mathematics
School of Electronics and Computer Science

A project report submitted for the award of
BSc Computer Science

Supervisor: Dr. George Konstantinidis
Examiner: Professor Tim Norman

Pet Breed Classification Problem using Convolutional Neural Networks

by Rares-Ovidiu Ficiu

April 30, 2019

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING, SCIENCE AND MATHEMATICS
SCHOOL OF ELECTRONICS AND COMPUTER SCIENCE

A project report submitted for the award of BSc Computer Science

by Rares-Ovidiu Ficiu

The last few years witnessed some major breakthroughs in the field of machine learning and computer vision which inevitably lead to an increasing demand for automation of jobs that rely heavily on processing large amounts of data. The focus of this paper is to offer a detailed view of the fields in which machine learning is currently being used, the different ways in which models operate and some of the general limitations they face. The discussion revolves around a step by step implementation of a classifier capable of differentiating between cats and dogs with human level accuracy, 99.6%, and between 139 breeds with 70% accuracy.

Acknowledgements

I would first like to thank my supervisor, Dr. George Konstantinidis for giving me the chance of pursuing this project, as well as assisting me throughout it.

Second, I would like to give credit to Professor Tim Norman for his contributions while designing the final model.

Finally, I want to mention Professor Mark Nixon and Dr. Jonathon Hare for their guidance in selecting finding valuable sources throughout this project.

Chapter 1

Introduction

The last 20 years have seen noticeable improvements across all disciplines, however, few of them experienced the same number of breakthroughs and growth in popularity experienced by the artificial intelligence domain. Even though the concept of artificial intelligence first appeared in literature over 100 years ago, in Erewhon written by **?**, and became an actual field of study in 1957, with the invention of the perceptron algorithm **?**, this field remained fairly quiet until recent years. The aim of this paper is to provide a better understanding into what is machine learning, how and where it is currently operating, as well as some general difficulties and limitations any machine learning algorithm faces and the ways in which some of them can can be overcome or at least mitigated.
In order to prove the effectiveness of the latest breakthroughs in computer vision and machine learning, we provide a way of building a classifier that, not only is able to differentiate cats from dogs with more than 82% accuracy as presented in **?**, but, it is also more accurate in classifying 139 breeds of animals (127 breeds of dogs and 12 breeds of cats) than the 59% reported by **?** on only 37 breeds of cats and dogs.

Because this paper is aimed at any reader, and because we cannot assume any prior knowledge, every step taken and the reasoning behind it in order to achieve this task is recorded. Moreover, because many concepts are only explained briefly, for a deeper understanding, it is highly suggested to follow the cited sources.

Chapter 2

Background and report of literature review

2.1 Current applications of machine learning and why it is important

Machine learning is already being used in a large number of fields in order to ease or augment human labor. Even though it has proven its effectiveness in some fields, this effectiveness is not automatically transferable to other domains as well. The one defect all machine learning algorithms have in common is that they all depend on large amounts of data in order to achieve acceptable performance levels (**?**). This means that the only fields in which machine learning is currently capable of operating with a greater than human accuracy, are typically the ones that have access to large amounts of processed data. Bellow we present two industries in which machine learning is currently operating and which are experiencing a growing demand for automation, namely Medicine and Financing. It should be noted that there are many other industries in which machine learning has been successfully deployed besides the two industries bellow presented bellow. (i.e. transport and food industries)

2.1.1 Medicine

Currently, in order to determine a potential prognosis for a patient, a doctor uses an algorithm which, after being fed all the relevant details about his health

condition, outputs an approximate score rating the gravity of the situation. Two models currently used with great success are the Acute Physiology and Chronic Health Evaluation [APACHE] score and the Sequential Organ Failure Assessment [SOFA] score.

Even though this field is currently under development there is clear evidence that by incorporating machine learning into medicine more accurate prognostics could be made. By using more information about the patient that, to the doctor managing the case, might seem unrelated, machine learning models were able to accurately predict the chance of dying for patients with metastatic cancer.(**?**)

Machine learning is also making the life easier for the radiologists and anatomical pathologists. Their main job is to interpret digitized images which, after being broken into pixels can be fed to a machine learning algorithm. Because hospitals around the world already possess a lot of this labeled images and because of the new advances in computer vision, the machine's precision will soon exceed human precision.(**?**)

2.1.2 Financing

There is a lot of interest in creating an algorithm that, if given the current situation of a business, is capable of predicting, with a high accuracy, the future of it. By using the top 35 technical indicators of the current status of a business as input features currently used by financial experts, researches have been able to achieve a precision of 61% in predicting the future price of the stock market(**?**).

Even if 61% does not sound too accurate for someone that is cautious with their money, it should be noted that this area is currently under heavy development and that there are many other factors than the 35 technical used that decide the future of a particular firm. Various political and economic factors and also market specific domain knowledge might further improve that accuracy if used as input.

2.2 The way machine learning works and the concepts behind it

2.2.1 The basics

Even though nature does not receive the credit it deserves, the latest advancements in computer vision forced us to realize the complexity of human vision and its power in recognizing patterns. According to **?**, humans can differentiate between cats and dogs with an accuracy of 99.6%. However, when computers are asked to differentiate between pictures, (like the ones presented below) they usually fail.

FIGURE 2.1: Image of a cat

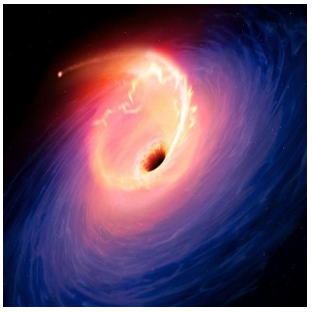
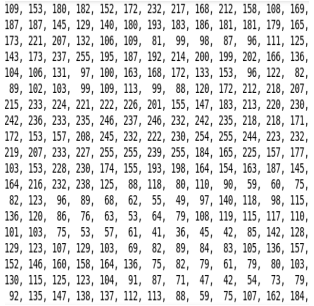


FIGURE 2.2: Random image



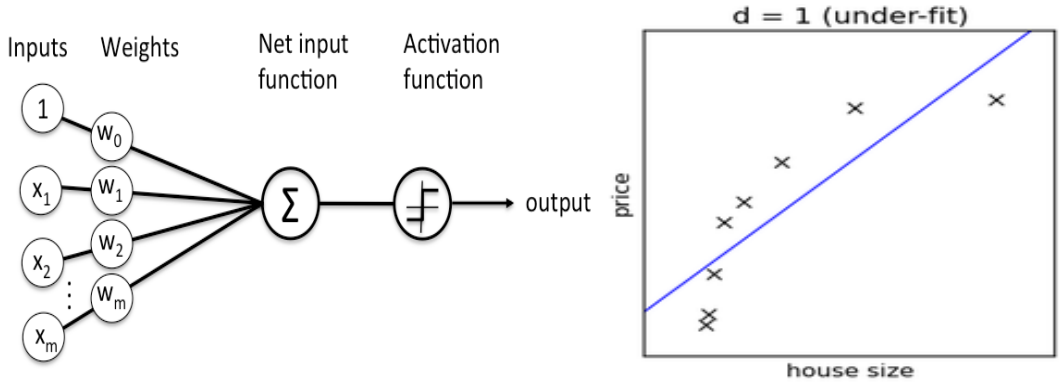
109, 153, 180, 182, 152, 172, 232, 217, 168, 212, 158, 108, 169,
187, 187, 145, 129, 140, 180, 193, 183, 186, 181, 181, 179, 165,
173, 221, 207, 132, 106, 109, 81, 99, 98, 87, 96, 111, 125,
143, 173, 237, 255, 195, 187, 192, 214, 200, 199, 202, 166, 136,
104, 106, 131, 97, 100, 163, 168, 172, 133, 153, 96, 122, 82,
89, 102, 103, 99, 109, 113, 99, 88, 120, 172, 212, 218, 207,
215, 233, 224, 221, 222, 226, 201, 155, 147, 183, 213, 220, 230,
242, 236, 233, 235, 246, 237, 246, 232, 242, 235, 218, 218, 171,
172, 153, 157, 208, 245, 232, 222, 230, 254, 255, 244, 223, 232,
219, 207, 233, 227, 255, 255, 239, 255, 184, 165, 225, 157, 177,
103, 153, 228, 230, 174, 155, 193, 198, 164, 154, 163, 187, 145,
164, 216, 232, 238, 125, 88, 118, 80, 110, 90, 59, 60, 75,
82, 123, 96, 89, 68, 62, 55, 49, 97, 140, 118, 98, 115,
136, 120, 86, 76, 63, 53, 64, 79, 108, 119, 115, 117, 110,
101, 103, 75, 53, 57, 61, 41, 36, 45, 42, 85, 142, 128,
129, 123, 107, 129, 103, 69, 82, 89, 84, 83, 105, 136, 157,
152, 146, 160, 158, 164, 136, 75, 82, 79, 61, 79, 80, 103,
130, 115, 125, 123, 104, 91, 87, 71, 47, 42, 54, 73, 79,
92, 135, 147, 138, 137, 112, 113, 88, 59, 75, 107, 162, 184,

FIGURE 2.3: Matrix

This is happening because computers cannot comprehend the patterns in the pictures in the same way we, humans, do. Computers 'see' images as matrices of numbers that represent the brightness of each particular pixel in each particular channel. To better understand the task computers face lets try to see a picture through their own 'eyes'. In the same way humans cannot comprehend what is being represented in the image 2.3, computers can not 'see' nor understand what is being presented in images 2.1 and 2.2 the way humans do. Even if it might seem impossible to teach computers, through long series of if-statements, to make sense from what seem to us random numbers, there are some ways in which they can be taught to find and recognize these patterns on their own. A brief description of how this is possible is presented bellow, however a more in-depth explanation is given by **?** in his online book.

2.2.1.1 Neurons: The building blocks of any neural network

A neuron is a function that takes several inputs, applies a weighted sum on all inputs, adds a bias and outputs the result. Then, by adding a condition on top of the outputted value, commonly referred to as applying a threshold, a decision can be made. For example, by first assigning a weight value to each item in a shopping list, and then by calculating the price of each item times the number of items, or weight, we can check if we can afford every item on this shopping list. Basically, all a neuron is doing, is placing a point in the data space in order for the activation function to make a decision. The weights help in calculating the actual price of the item, if a weight of 1 is applied it means the entire price has to be paid while for a weight of 0.75, it means that the product has a discount of 25%. If the total is situated above the line drawn by the activation function, which in this case is the amount of money the buyer has, it means that he cannot afford everything in the shopping list, if , however, it is under the line, it means he does.



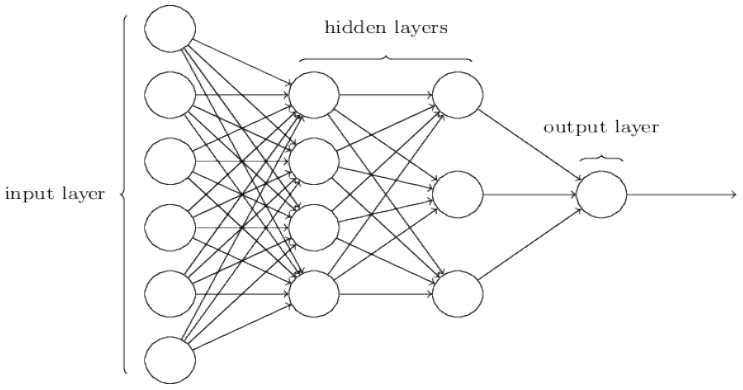
(Images sourced from **?** Left and **?** Right)

This is great for classification problems that have data that can be fitted with just one line. However, not many problems are simple enough to be solved by a single neuron.

2.2.1.2 Neural Networks

In the real world, however, people do not make purchases just based on the fact that they can afford a product or not. If we try to factor in the current mood of the buyer, for example, we need to add a second neuron that takes as input different details about the client current status. A good example is if he is smiling or not. The output can then be fed through a second activation function. If the buyers mood metric is above a certain threshold of happiness and if he has enough

money, the potential buyer will make a purchase, if, however, any of the conditions are bellow the required threshold, he will not. This is why problems that require more than one straight line to fit the data, also require more neurons to draw the complex line of combined thresholds that fits it.



(Image sourced from **?**)

Every extra neuron can be seen as another degree of freedom given to the model in order to draw the line based on which the network makes predictions.

2.2.1.3 The training

The fastest way of getting a model to perform an accurate prediction, is to first identify all the information required to make the prediction, and then feeding it to a model that has perfect parameters to process it. Even though this might seem easy for the example presented above, the amount of input nodes required to classify just one 200x200 colour image is 120000. Just manually assigning the perfect weight to each of them would be a difficult task for a human.

There are different ways of training a neural network. An inefficient approach is to randomly initialize all of it's parameters while keeping track of the best performing model. Even though mathematically the model would eventually be initialized with perfect parameters, statistically it would take 'forever'.

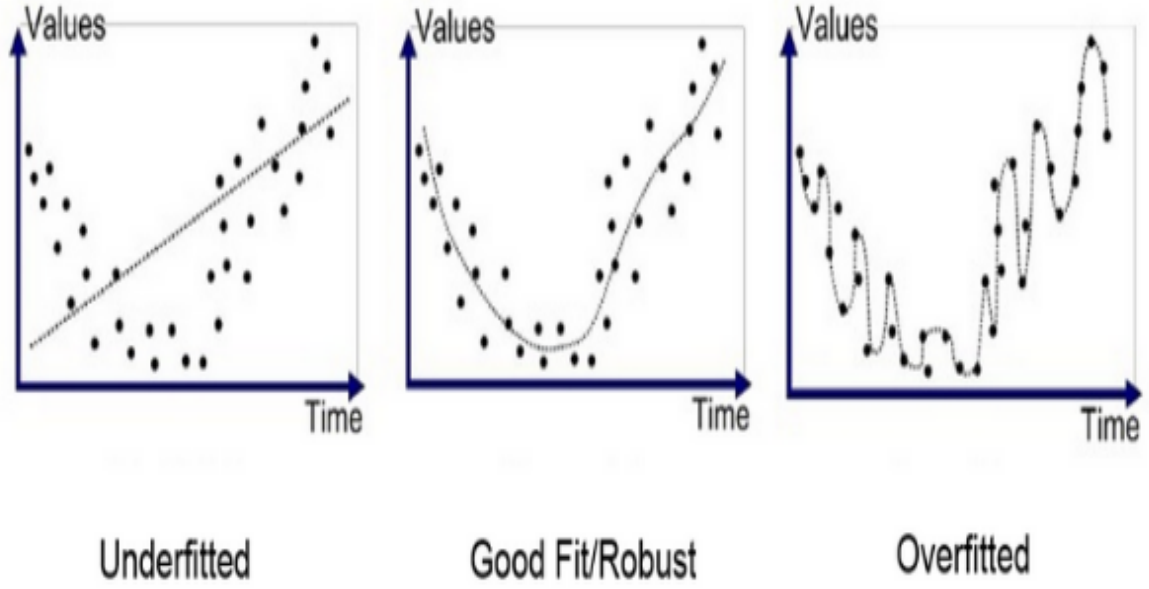
A more efficient approach is discussed in the 3.4.2.2. In short, the value outputted by the network, after being fed an instance to which the correct answer is already known, can be used to calculate how far the network is from the desired output, or how bad it is performing. Based on this distance, the weights and biases can be adjusted such that if the network is provided with similar input, this distance will be shorter, making the network perform slightly better.
Because of this the training and testing dataset should be split. The percentages

should be chosen based on the problem that is being solved. Bigger training dataset means more accurate model but it also means there is less data to reliably test on. While bigger testing dataset means more accurate testing but a less accurate model.

2.2.2 Common problems

2.2.2.1 Underfitting and Overfitting

Two of the biggest problems while training a neural network are underfitting and overfitting it. As previously stated, having more neurons inside the neural network gives the network the ability to draw more complex lines to fit the data. This can be seen as a good thing at first because by using complex lines the model is able to fit the training data better. In reality, however, if the model has too many neurons, it will just memorize every input given during the training phase and it will not be able to recognize the new input given during the testing phase. This is called overfitting and it manifests by the model having excellent results on the training dataset while performing poorly on the test dataset. If the network does not have enough neurons, or, if it is trained on a small dataset, it will not be able to 'understand' the data presented and it will do poorly on both training and testing datasets. This is called underfitting.



(Image sourced from **?**)

2.2.2.2 Dataset

As stated in **?** a problem with creating a working model is the availability of a reliable dataset.
First, a good dataset should contain data that accurately represents the world the model is going to be deployed in.
Second, the training dataset should be comprehensive enough such that the model learns how to react to any possible input received from the environment once deployed.

2.2.2.3 Lack of general understanding

Because models are trained only to solve specific problems and fed only similar inputs, they are not able to understand a change in their environment. As shown by **?** deep neural networks capable of classifying images can be easily fooled. For the experiment they used a deep neural network capable of classifying digits with 99.6% accuracy. They showed that when the deep neural network was given an image containing white noise it would still believe with a 99.99% confidence that it is seeing a digit, thus labeling it as one.

2.2.3 Types of neural networks

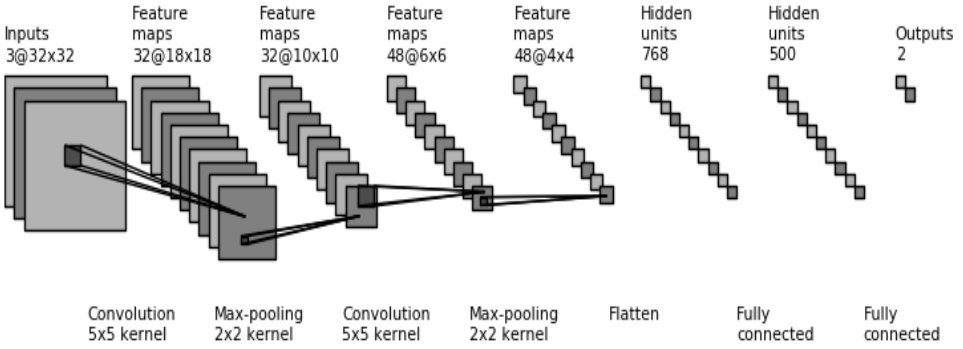
2.2.3.1 Deep neural networks

In the example used in 2.2.1.2 we showed how to neural networks can be used in order to draw 2 separate straight lines based on which the model makes a prediction. A deep neural network aims to combine multiple outputs from different neurons in order to draw just one complex line to fit the data.

2.2.3.2 Convolutional Neural Networks

In the case of models that are trying to classify pictures, having the photo flattened to a single array containing all the pixels in the original picture and then feeding it to the model is often ineffective because it removes the information contained inside the order of the pixels before the flattening taking place.
So instead of flattening the picture to a single array, a smaller neural network

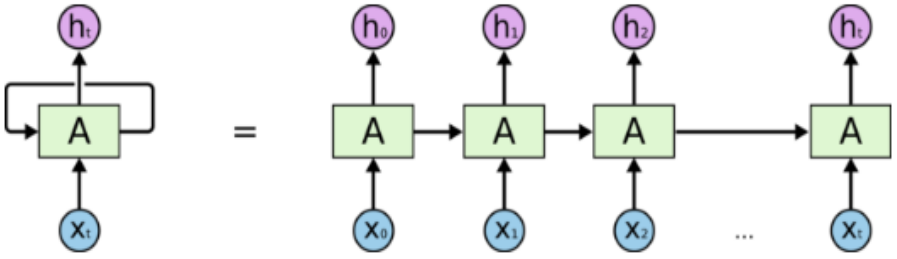
is created whose job is to only recognize patterns from the layers bellow. These smaller networks are then slid across the layers bellow them to check for any pattern learned during the training. If a pattern is found, these findings will be reflected in the activation of the layers above, thus making the neural network able to find and recognize shapes in images. This concept dates back to early 1970s, however the first published architecture that takes advantage of the spatial information contained inside the pixels is **?**.



(Image sourced from **?**)

2.2.3.3 Recurrent Neural Networks

In the case of models that are trying to predict data based on a previous sequence of events, in order to make an informed prediction, the last node, the one doing the actual prediction, must be somehow informed about the past trends exhibited by these past events. A good example of this is making a model able to predict and react to the stock market fluctuations. At any time step, the model 'sees' just one value, so it cannot be sure if the price is going to go up or down. This can be fixed by feeding the model some context as well.



An unrolled recurrent neural network.

(Image sourced from **?**)

Similar to how a convolution layer shares the weights across the entire layer, searching for the same pattern across it, and "informing" the layers bellow about its findings, a recurrent neural network shares the entire neural network across the entire sequence, providing information to the last network about its context.

Chapter 3

Implementation

3.1 Tools used

Because going in great detail about the mathematics behind neural networks is outside the scope of this paper, the entire implementation was done using python, taking advantage of the publicly available libraries. Most notable libraries are scikit-learn **?**, scikit-image **?**, tensorflow **?** and keras **?** which offer invaluable functionalities. From processing an image to designing the neural network, these libraries shortened the implementation by a great margin.

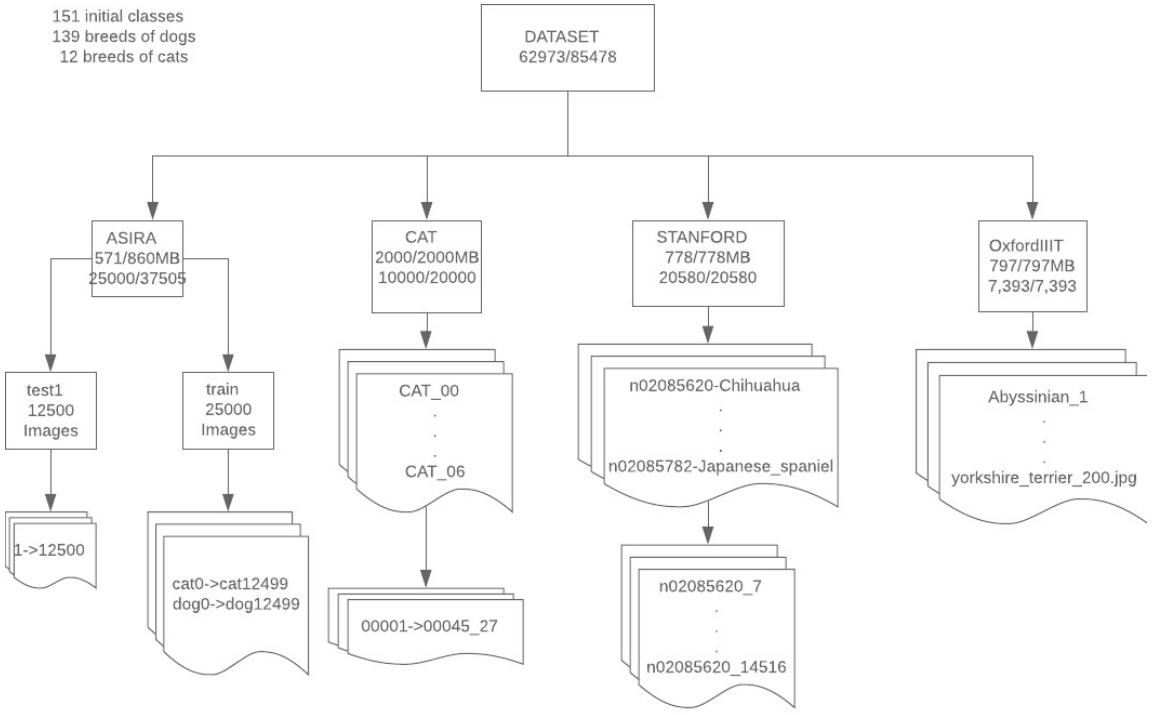
3.2 Preparing the datasets

As stated before the biggest problem in designing a working neural network is preparing the dataset in order to train the model. Gathering and labeling enough data in order to build something new would be a project in itself, that's why we opted for open source datasets. We also wanted to make sure the datasets are large enough so we decided to combine multiple datasets.
The following datasets have been combined:
?,
?,
?,
?
resulting in a dataset containing 62973 labeled photos.

3.2.1 Merging multiple datasets

Immediately after downloading it can be seen that there are some obvious problems with the resulting dataset. In this section we will discuss about them and propose some solutions as well as present some limitations.

The most immediate problem is that every dataset has its own particular structure, making their merging quite difficult.



The initial structure of the data set

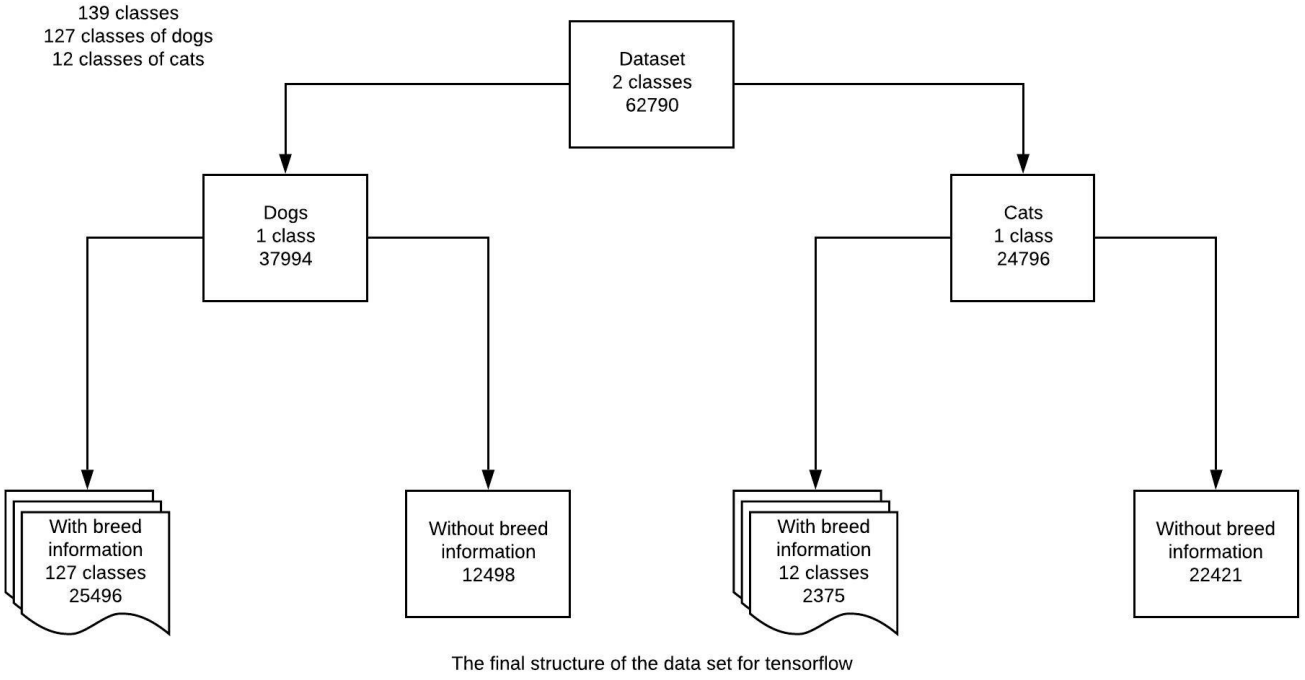
The first problem to emerge was the large number usable images contained inside the whole dataset. With a total of 62973/85478, it made even the handling quite difficult on a personal computer, the computer often freezing when an attempt to open one of the largest folders was made. Besides deleting the unlabeled images, there is no real work-around the large number of images. As stated in the sections above, this is more of consequence of the way in which machine learning works. More often than not, more images equals more training examples and implicitly higher test accuracy. Moreover, later we will propose some ways of increasing the size of the dataset instead of reducing it.

Second, it can be seen that there is not only a difference in the way picture are being named, making the manual handling of the images almost impossible, there is also a pretty notable difference in the way the labels are being handed.

ASIRA and OxfordIIT both have the labels contained inside the name of the image, while Stanford has the labels contained in the folder names. This can be fixed using python scripts to access, extract and manipulate the files. (Libraries used include glob to extract file names and both os and shutil to move, copy and delete unnecessary files.)

Third, there is a possibility of having a large number of duplicates, potentially causing the model to overfit on a small number of images. Fixing this seems quite straight forward at first. Just iterate through every picture and compare it with all other pictures pixel by pixel. This implementation has 2 major flaws: It assumes that the duplicates share the same size and it only works on small number of images. A more efficient approach would be to resize the images and then create a unique hash based on that particular image. Followed by a much faster comparison between the images. The option we opted for is somewhere in between the two. We first created a table containing all the images and their sizes. Then we compared the sizes in search for a match. When a match occurs, the two pictures are loaded into memory using sci-kit image, compared pixel by pixel, and if they happen to be perfect matches the second picture is deleted. This operation took around 20 minutes and resulted in around 100 duplicates being deleted.

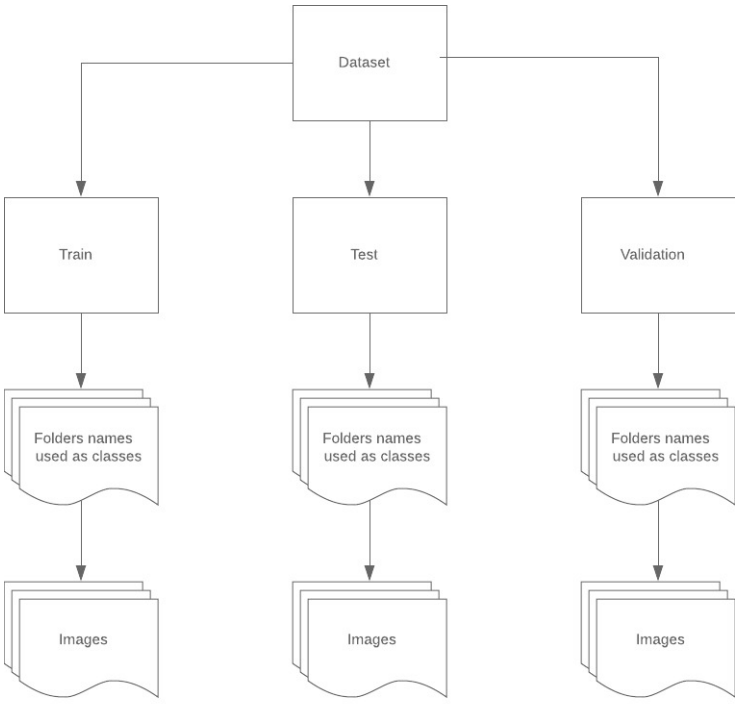
Lastly, the Stanford dataset and the OxfordIIIT dataset both contain a significant number of dog breeds inside them. 120 and 25 respectively. This means that there is a high chance of having duplicate breeds containing different pictures. After removing the code at the beginning of each class from the Stanford dataset it was clear there are only 7 unique classes inside the OxfordIIIT and, because the duplicates have been removed before, it was safe to merge the duplicate classes. Because of the small amount of images inside each of the class, the merging required no programming at all and was done manually, taking less than 5 minutes.



The final structure of the data set for tensorflow

3.2.2 Additional datasets for Keras

In later sections, we take a look at transfer learning using the MobileNet architecture. This can be done using tensorflow by downloading the publicly available weights and then loading them manually, however we think the user friendliness provided by the Keras library makes it more suited for the task. Keras makes it possible to feed images directly from a file just by providing the location of said file using the "flow_from_directory" function inside the "ImageDataGenerator" class. For this reason we also created multiple datasets of the following structure.



Dataset structure for Keras

3.3 Expanding the dataset

As stated in 2.2.2.2 a dataset has to accurately represent the real world conditions in which it will be deployed and while also being comprehensive enough such that the model generalizes well enough. The discussion bellow is meant to point out the reasoning behind expanding the already existing dataset, as well as present some ways in which this can be succesfully achieved.

3.3.1 Reasons to artificially expand the dataset

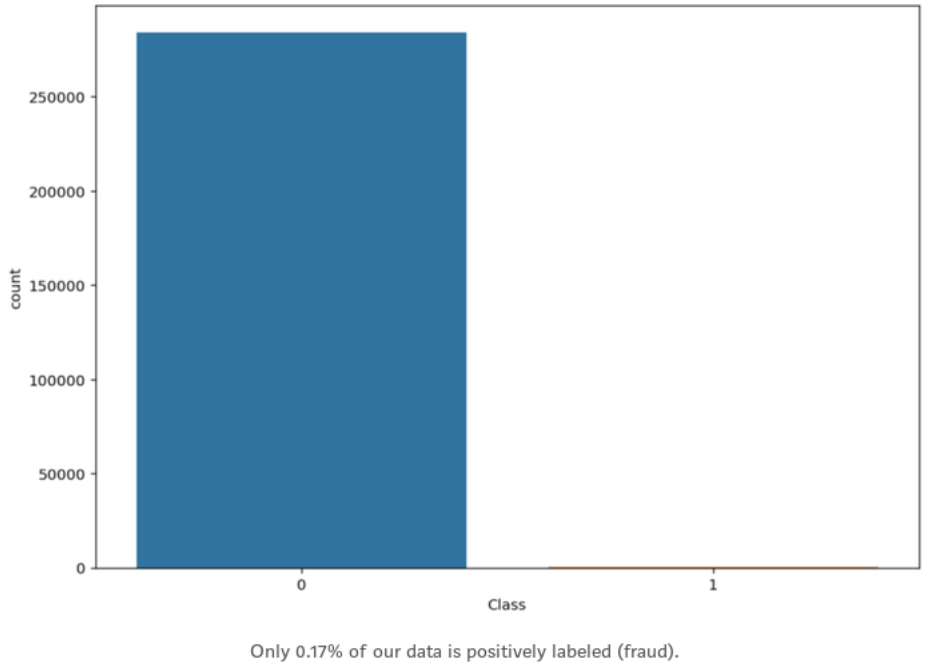
3.3.1.1 The difficulty of collecting real life data

The most straight forward and also the most effective way of expanding any dataset is by first collecting data from the real world and then manually labelling it. This method has some major drawbacks. In our case, even finding a dog and labeling it's breed with a 100% certainty is a difficult task. Also, while preparing any dataset one has keep in mind that even with a perfect model able to recognize 100% on the training dataset, by having 0.5% of data wrongly labeled data, the model will only have, in the best case scenario, a 99.5% accuracy in the real world.

The second major problem is the cost. For example, in the case of self-driving cars, every second of video costs at least the value of the gas required to keep the engine running.

3.3.1.2 Imbalanced dataset

Having an imbalanced dataset is a major problem while training a model, the reason being the models tendency to overfit on the classes containing the most instances. The most cited problem is the bank fraud problem. The number of fraudulent transaction is negligible small in comparison with the number of non-fraudulent transactions. This means that a model, labelling every transaction given as non-fraudulent, will still achieve a 99.83% accuracy.

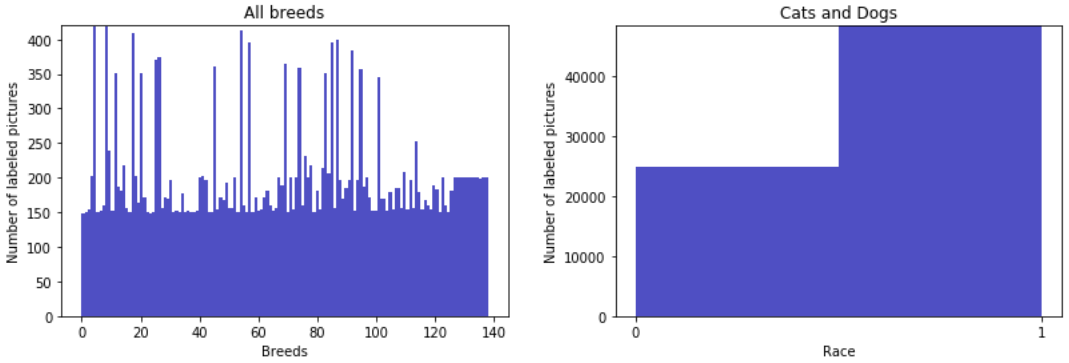


(Image sourced from **?**)

There are different ways of combating this imbalance. **?** have done a great job explaining some of these methods, outlining their strengths and weaknesses. However, their main concern is solving the imbalance in big data, on datasets containing at least 100,000 instances.
The two most common methods are over-sampling and under-sampling. The concepts behind them are easy to grasp. Just take less instances from the over-represented class, known as under-sampling, or take more of them from under-represented class, also known as over-sampling. Another thing worth noting is that

? reported better results while using Random Over-Sampling (ROS) over Random Under-Sampling or the Synthetic Minority Over-Sampling Technique (SMOTE).

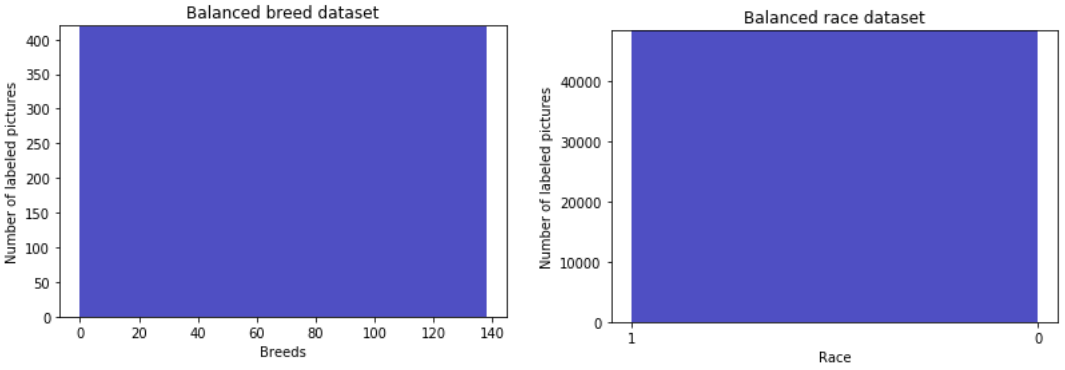
In our case, the individual datasets showed no sign of imbalance. However, after their merging, there were clear signs of imbalance.



In the first picture it can be observed the imbalance present between the breeds while in the second, the over-representation of dogs between the cat and dog classes. It is easy to understand what caused the imbalance between the breeds. The Standford dog dataset contains 120 breeds with only 150 pictures/breed while the OxfordIIIT contains 25 breeds of dogs and 12 breeds of cats with around 200 pictures/breed. Moreover, some of the breeds are duplicates causing the huge spikes in the number of images/breed.
There are 2 obvious limitations by balancing the datasets through over-sampling. First, there is inevitably going to appear an imbalance between the images of the same class. If for examples the class containing the highest number of instances contains 300 instances, a class with 260 images will leave the class with 260 unique images and 40 duplicates causing the model to overfit on those over-sampled 40 unique images. Second, this must be done only on the training dataset, after splitting the dataset into training and test. Some of the doubled instances might end up in both the train and test dataset, making the accuracy meaningless.

Another way of dealing with imbalanced dataset is by assigning different weights to each class, making it possible to over-sample or under-sample percentages of instances instead of entire instance, thus solving the problem stated above. In the example above, the class containing 260 images would have its weight adjusted to 1.15 instead of 1. In theory this would make the model train like it would normally do on a balanced dataset. However, even though this method can improve the classification performance, by changing the desired output of a network to be biased for the under-represented class, it is possible to make it more unstable, resulting in a high degree of variance in the performance measures **?**.

For our problem, we have to point out that, none of the datasets contain over 100,000 instances and that the number of instances contained inside the class with the highest number of instances is on average 2 times that of the other classes. We do not think choosing down-sampling over over-sampling, done by trowing away a decent percentage of the dataset, could have helped the model achieve a higher accuracy. Lastly, instead of using the random over-sampling presented in **?**, we opted for a simpler over-sampling algorithm. We first found the class containing the highest number of instances and copied instances from the other classes until they reached the same number of instances. Also, because of the long training times, we could not afford to train multiple models to deal with the higher variance posed by assigning different weights.



3.3.2 Image augmentation

One way of cutting down the cost of expanding the dataset is by taking multiple pictures of each instance, from a multitude of angles and in different lighting conditions. While collecting pictures of dogs, for example, taking multiple pictures of the same dog can bring the cost down to manageable amounts. This is not necessary going to improve the accuracy of the model, it is possible that this will cause the model to overfit on that particular dog. However, if the images are different enough from one another, while also being close to what the model is designed to predict, it can help it generalize better. A faster and cheaper way of achieving a comparable result, is by digitally altering the already labeled images. This must be carefully done with the 2 conditions from 2.2.2.2 in mind.

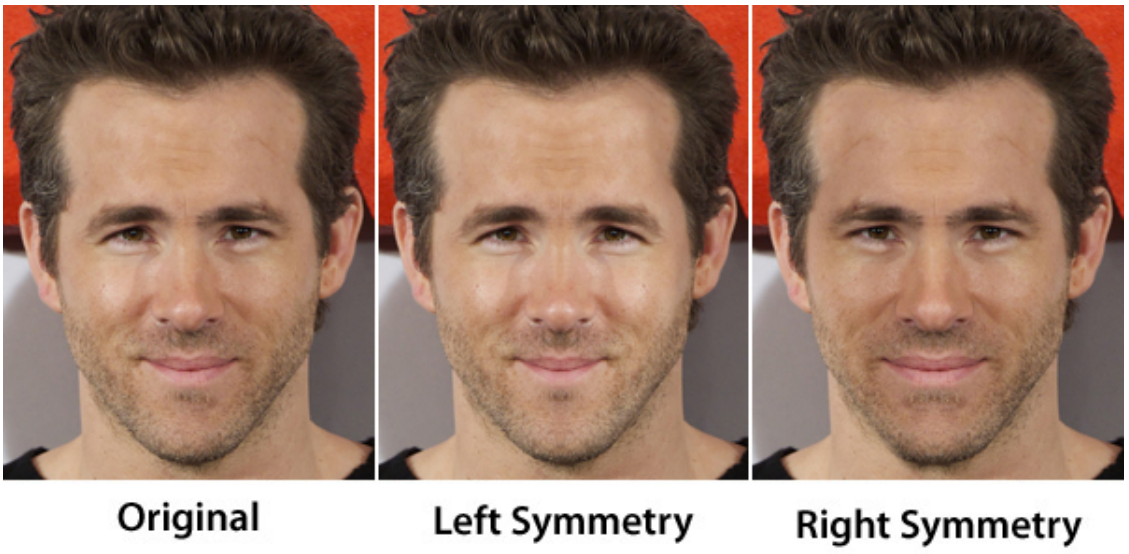
Orientation flip

FIGURE 3.1: Original

FIGURE 3.2: Horizontal Flip

FIGURE 3.3: Vertical Flip

In the example presented above, flipping the orientation horizontally as in 3.2, can make the model generalize better. The reason being, that it makes sure that the model learns to search for features in more than one spot across the image, and, in the same way that human faces are not symmetric, by changing the orientation of a face, a different face or dog can be created.



(Image sourced from ?)

This means that while classifying images taken by humans, the horizontal flip has the potential of doubling the dataset, with what we believe to be valuable information.

It should be noted that a vertical flip can achieve a similar result on some classification problems. But it poses no real value in solving our problem because the

resulting image, seen in 3.3, fails to represent real life examples, resulting in a confused model.

This kind of augmentation, even if it seems basic at first, can, in cases in which images can appear in any orientation (e.g. skin cancer lesions), quadruple the available dataset.

Rotation

Similar to vertical flips, which are achieved by rotating an image by 180 degrees, images can be rotated by any degree. Our dataset contains some images that could be augmented by a 90 degree rotation, as seen in figure 3.5. By rotating the original image of the dog lying down, a new image of a dog staying up is created. However, the dataset contains a small proportion of images that can successfully represent real examples after this kind of flip, making it more likely to confuse the model instead of increasing its accuracy.

Other ways of altering the images

Images can also be zoomed in and out 3.6, brightened or darkened 3.7 and in some cases altered such that the recognition becomes difficult even for humans 3.8.

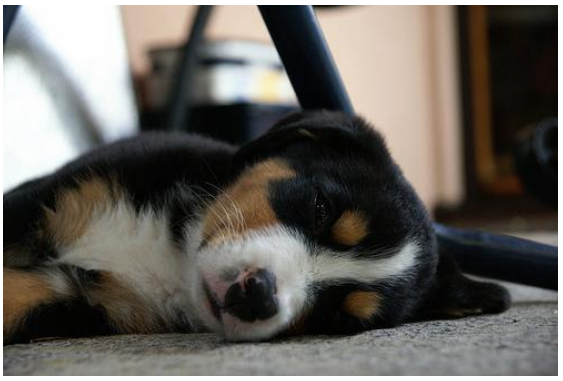


FIGURE 3.4: Original

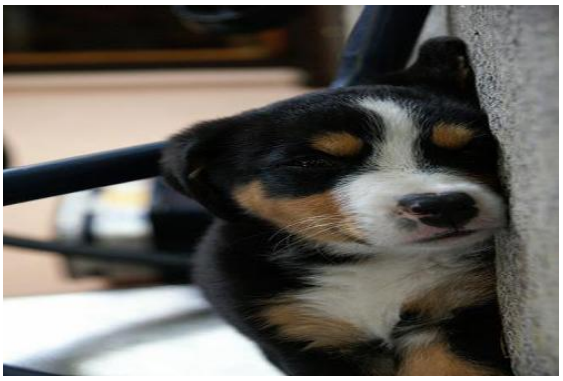


FIGURE 3.5: 90 degree rotation

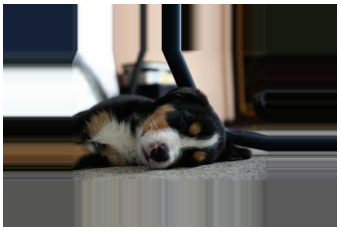


FIGURE 3.6: Zoomed out

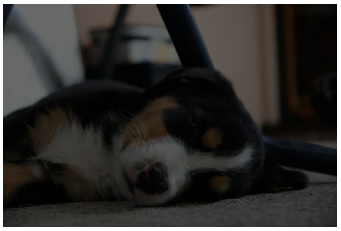


FIGURE 3.7: Darkened

FIGURE 3.8: Multiple filters

Conclusions
It should be clear by now that images can be digitally altered to create a more comprehensive dataset. However, in order to make a successful augmentation images must be carefully altered to avoid copying the original while still keeping it's meaning.

3.4 Deep Neural networks

3.4.1 Introduction

The following sections are a continuation of the discussion started in 2.2.1.3. The discussion revolves around a step by step implementation of a classifier with human level accuracy. The reasoning behind every step taken towards achieving this goal is documented, making these findings highly reproducible. The first step is to build and evaluate a deep neural network, followed by a convolutional neural network, and then 2 state of the art architectures.

First Deep Neural Network	
Type	Neurons
Input	400x400x3 flattened
FC	240000
FC	96000
FC	38400
FC	11520
FC	2304
FC	460
FC	92
FC	19
FC	5
FC	2

3.4.2 First Deep Neural Network

In order to feed a colour image, which is a 3 dimensional matrix (height*width*channels), to a normal neural network 2.2.1.2, the matrix has to be flattened down to an uni-dimensional array. Because of the large number of images contained in cats and dogs classes, the first models were designed, trained and evaluated only these 2 classes.

3.4.2.1 Hyperparameters

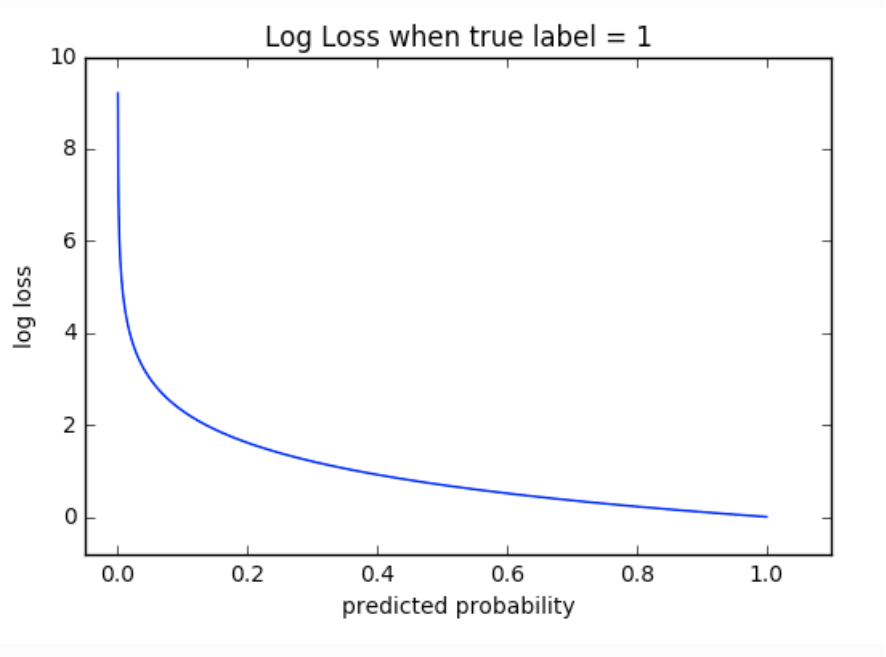
Knowing that a having a higher resolution to the images is likely to yield a higher accuracy, we decided to use the 400x400 size for our images. We also tried to make it fairly deep while also trying to keep a pyramid shape. As a loss function we used cross-entropy and to optimize it we chose AdaM optimizer with a learning rate of 0.01.

3.4.2.2 The loss function and it's optimization

In 2.2.1.3 was briefly mentioned that by feeding both the predicted and expected values to a function designed to asses how close these two values are to one another, it is possible to make an informed decision about the direction of the next training step.

Loss Function

The function making the assessment is called the loss function. The most simple loss function is the difference between the predicted and the expected values. Different loss functions can yield different results on the same prediction, this is why the loss function is chosen based on the tasks that the algorithm is trying to solve. Cross-entropy is a popular loss function used for solving classification problems, the reason being that the outputted values are between 1 and 0.



Log loss function (Image sourced from **?**)

Optimization

At every learning step, using the derivative of the loss function presented above, the best direction along which the weights should updated in order to follow the steepest descend in the parameter space is calculated. Every parameter is then updated based on how much it influenced the result of the loss function scaled by the learning rate. This is known as gradient-descent.

The learning rate can be improved by keeping in mind the direction of the previous steps taken. By taking bigger steps for every consecutive step taken in a particular

direction the convergence time can be improved significantly. Similar to how a ball rolling down a hill builds momentum, the momentum parameter in our algorithm increases in magnitude.
Adam or Adaptive Moment Estimation uses an exponentially decaying average of past squared gradients to calculate the size of the step and an exponentially decaying value of past gradient to calculate its direction. More details about Adam as well as more optimization algorithms can be found in **?**.

3.4.2.3 Problems

The obvious problem was the sheer size of this neural network. The entire network had around 140 billion parameters. However, 80% of them were contained by the first fully connected layer which required the adjustment of over 115 billion parameters. It was clear that either the size of the images had to be reduced, or the size of the first fully connected layer.

Second Deep Neural Network	
Type	Neurons
Input	224*224*3 flattened
FC	4000
FC	4000
FC	1000
FC	300
FC	100
FC	30
FC	2

3.4.3 Second Deep Neural Network

3.4.3.1 Hyperparameters

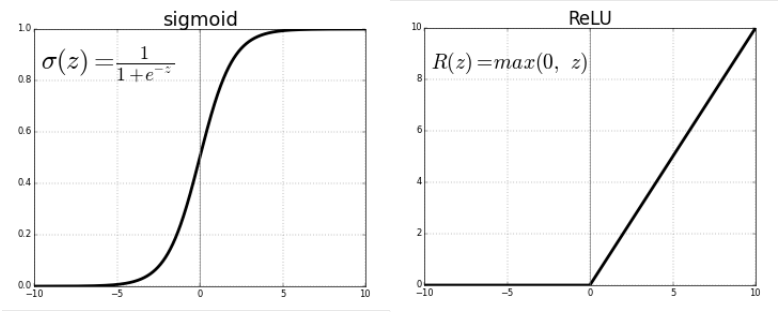
In order to make the network smaller, we used the picture size of 224*224*3 used by **?** in their MobileNet architecture. We also brought down the number of nodes in each layer, bringing the total number of parameters to around 600 million making the loading of the model into memory possible. Every layer had ReLU as the activation function.

3.4.3.2 Problems

This implementation had 2 problems. First, every epoch took around 70 minutes and second it was stuck at 62% accuracy on both the training and the test datasets. The reason it was stuck at 62% instead of 50%, was because the dataset contained 37994 images of dogs and just 24796, making the dataset imbalanced.
After balancing the dataset using the technique discussed in 3.3.1.2 the model remained stuck at 50% accuracy, pointing to a flaw in the design.

3.4.3.3 Dying ReLU

After monitoring the activations on each layer, we observed that a large amount of neurons in a layer had 0 activation. This is a known problem caused by the ReLU activation function.



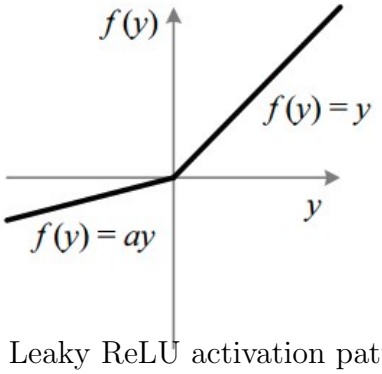
(Image sourced from **?**)

ReLU turns any negative input into 0 while keeping any positive input unchanged. This has been proven by **?** to improve the convergence time of convolutional neural networks by up to 6 times . However, on our neural network it was causing entire layers to die, making the learning impossible.

3.4.4 Final Deep Neural Network

3.4.4.1 Hyperparameters

To fix the dying ReLU problem, we changed the activation functions on all layers to Leaky ReLU. The difference between ReLU and Leaky ReLU is that if the activation is negative, Leaky ReLU scales it by a chosen value instead of outputting 0.



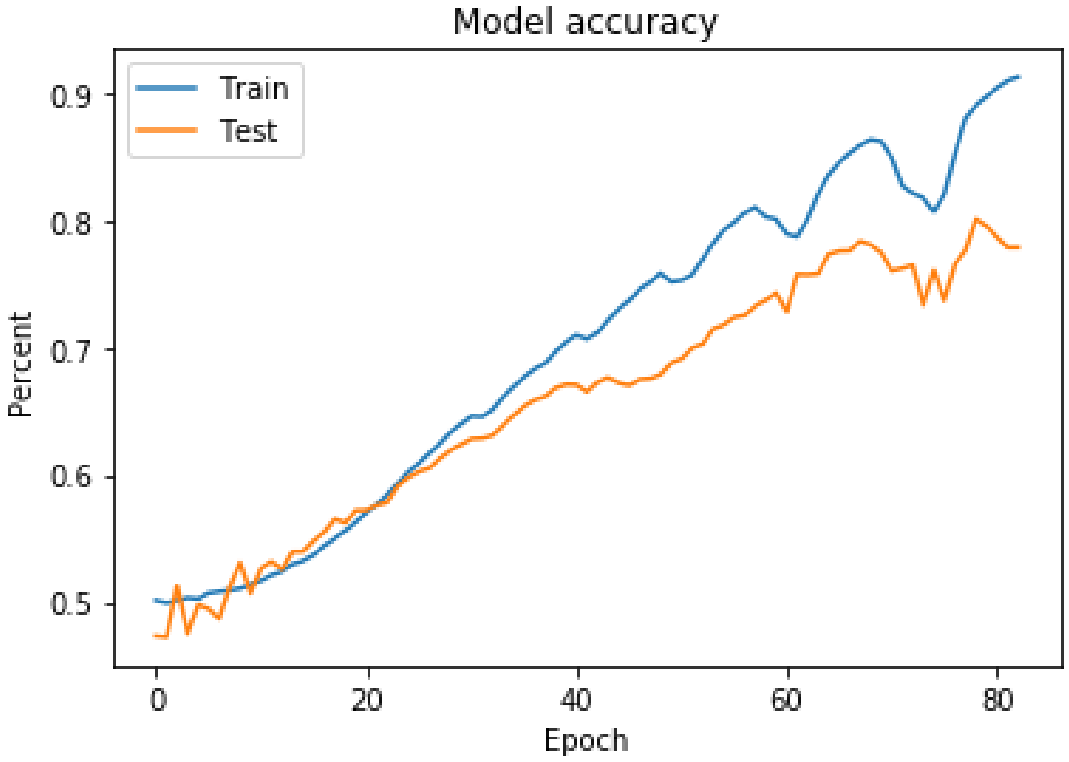
Leaky ReLU activation pattern (Image sourced from **?**)

3.4.4.2 Quick Fixes

The following problems were minor and required little changes in the model design to fix. The first problem was that the model was still not learning. However, after checking the neurons activation, it became apparent that the gradients were exploding. Exploding gradients happen when the optimizer overcompensates for the calculated loss, propagating large values across the network, making the next step overcompensate in return, starting a chain reaction. In our case this was happening because of the poorly chosen value for the learning rate. After reducing the learning rate from 0.01 to 0.00001 the network started behaving normally.

Final Deep Neural Network	
Type	Neurons
Input	100*100*3 flattened
FC	4000
FC	4000
FC	1000
FC	300
FC	100
FC	30
FC	2

The second issue was more difficult to trace. As can be seen in graph bellow, not only the model was learning, it also did not show any signs of underfitting or overfitting. First successful run achieved 91% accuracy on the training dataset and just over 80% on the test dataset.



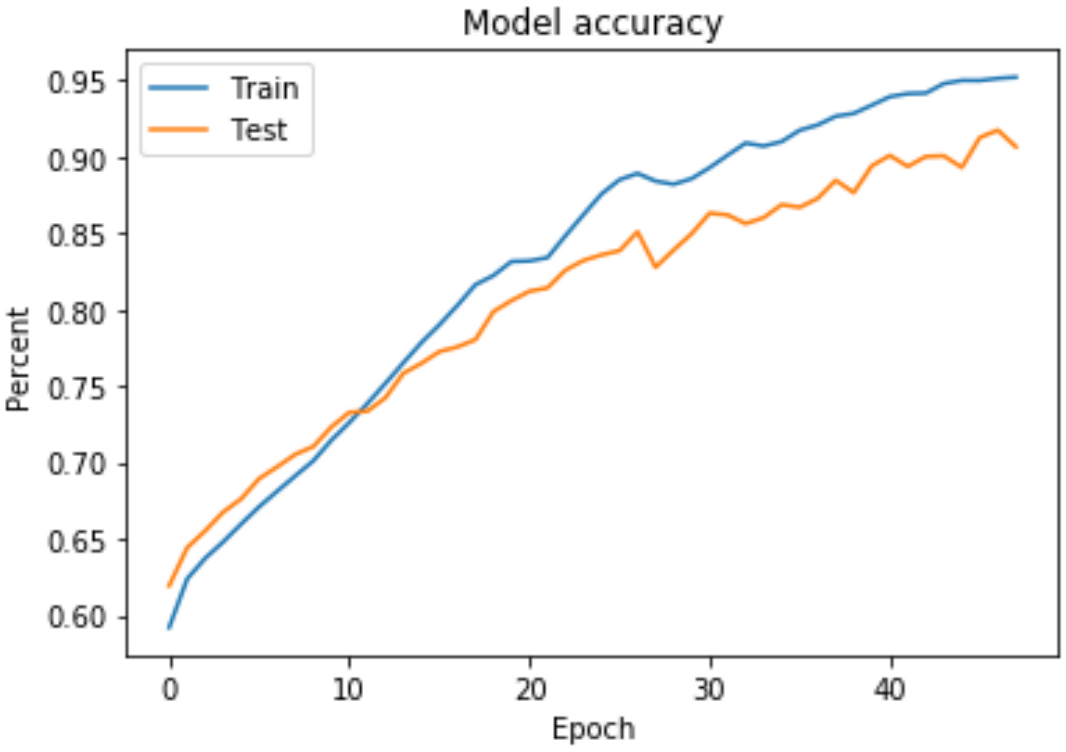
Random Labels

However, when we manually tried to test it on some examples, it was performing

very poorly. After a thorough analysis of the code we found the following lines of code:

```
random.seed(125)
random.shuffle(X_cats_dogs)
random.shuffle(y_cats_dogs)
```

It turns out that, we were not only shuffling the train labels independently from images, making the model learn from whatever the resulting dataset was, because we were splitting the dataset into train and test after the shuffling was done, we were also testing on a randomly shuffled dataset.
After fixing this problem, we also doubled the learning rate from 0.00001 to 0.00002 because even after speeding it up from 70 to 40 minutes/epochs, 87 epochs still took 58 hours.



Final Deep Neural Network

With the fixed dataset and the doubled learning rate, it achieved 91% accuracy before starting to overfit in less than 50 epochs.

3.4.5 Conclusions

Although deep neural networks can be really effective in solving some particular problems, they face serious limitations when it comes to classifying images. The

major problem is the input layer. Besides the fact that the image must be flattened to an uni-dimensional array, resulting in a major loss of spatial information, the input vector grows exponentially with the size of the image, making the use of high resolution images highly unpractical.

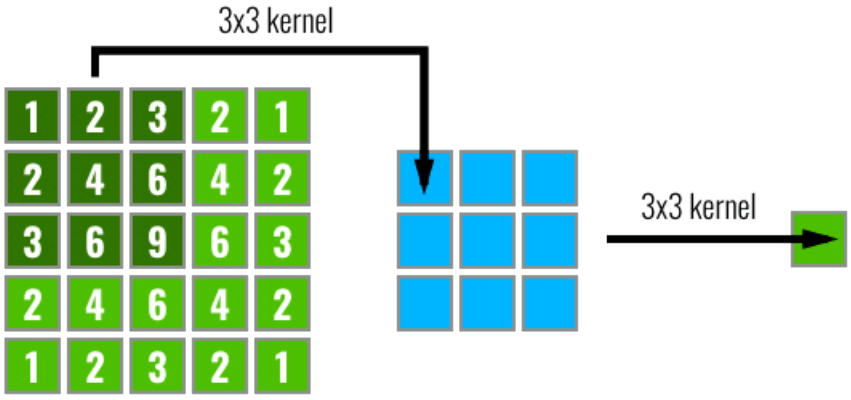
3.5 Convolutional Neural Networks

3.5.1 Introduction

In this section we seek to solve both problems posed by deep neural networks using convolutional neural networks. A brief description was already given in 2.2.3.2. However, in this section we offer an more in depth view of the ways in which they operate and why they offer a good solution to these problems.

3.5.2 Components

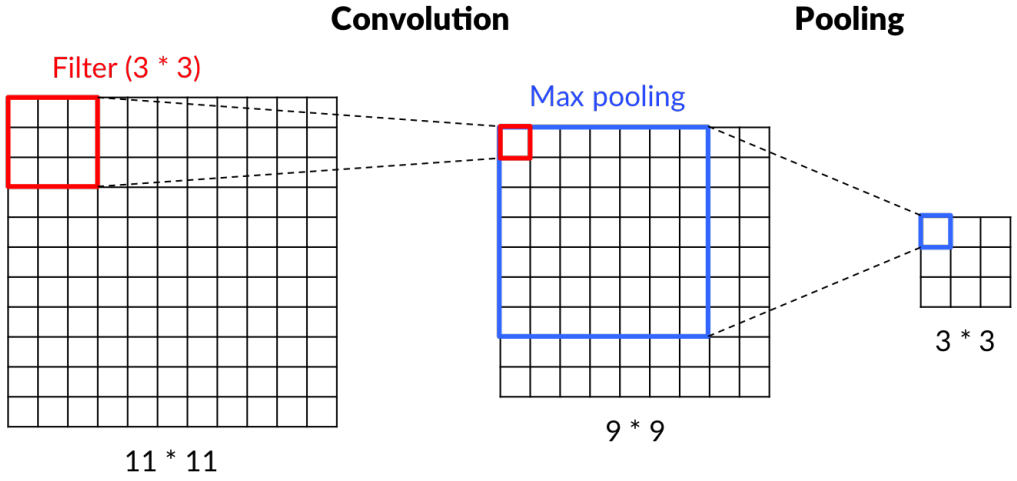
Local receptive fields.
The main difference comes from the way in which the images are processed. Instead of having a neural network in which every neuron is fully connected to all the neurons from the previous layers, the neurons in convolutional neural networks are only connected to a small, localized region in the layers bellow.
Shared weights
Instead of having different weights attached to each local receptive field, the same matrix of weights and biases is shared between all the neurons of an entire layer, this means that each neuron is searching for same features in its own local receptive field. This is why the output is often referred to as a feature map. A good representation of how a feature map is created and further used can be seen in the image bellow. 3.5.2
By combining these 2 concepts it is possible to create a layer that not only retains the spatial information of the previous layer, but also reduces the number of parameters considerably.



Convolution Layer (Image sourced from **?**)

Pooling Layer

Each convolution layer is usually followed by a pooling layer. This is meant to reduce the dimension of the output by summarizing the information generated by the convolution layer.



Pooling Layer (Image sourced from **?**)

The most common pooling procedure is max-pooling which only keeps the highest value from a particular region. As can be seen in the image above, a 9x9 max-pool layer on a 11x11 feature map, will reduce its dimension to 3x3. However, a problem with the operation above is the ratio between the size of the feature map and the size of the pooling kernel. A pixel placed centrally, might contain the highest global value, resulting in a 3x3 matrix composed only of 1 value, erasing all the information contained inside the feature map instead of summarizing it.

Other common pooling procedures include : average pooling and L2 pooling.

3.5.3 Convolutional network

3.5.3.1 Hyperparameters

In order to check the amount of lost information during the flattening process, the resolution of the images was kept at 100x100.
Due to how the max-pooling works, all convolution layers use ReLU activation. However, based on the findings in the sections above, in order to avoid a dying ReLU problem, all fully connected layers use Leaky ReLU activation.

For the first filters we selected a size of 50x40 because we believed that, as can be seen in the examples bellow, most of the important features needed for a successful classification can be fitted inside a window this big, making the model 'understand' the data better. Moreover, because of the small image resolution, only a 2*2 max-pool operation was applied after every convolution.

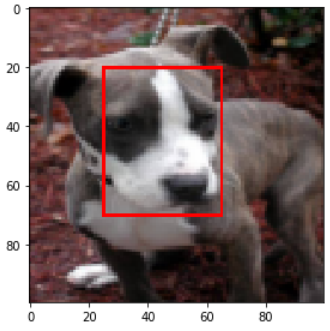


FIGURE 3.9: Example 1

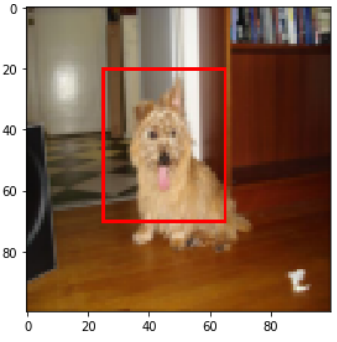


FIGURE 3.10: Example 2

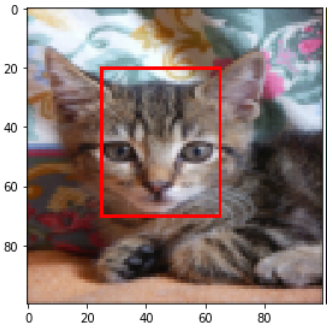
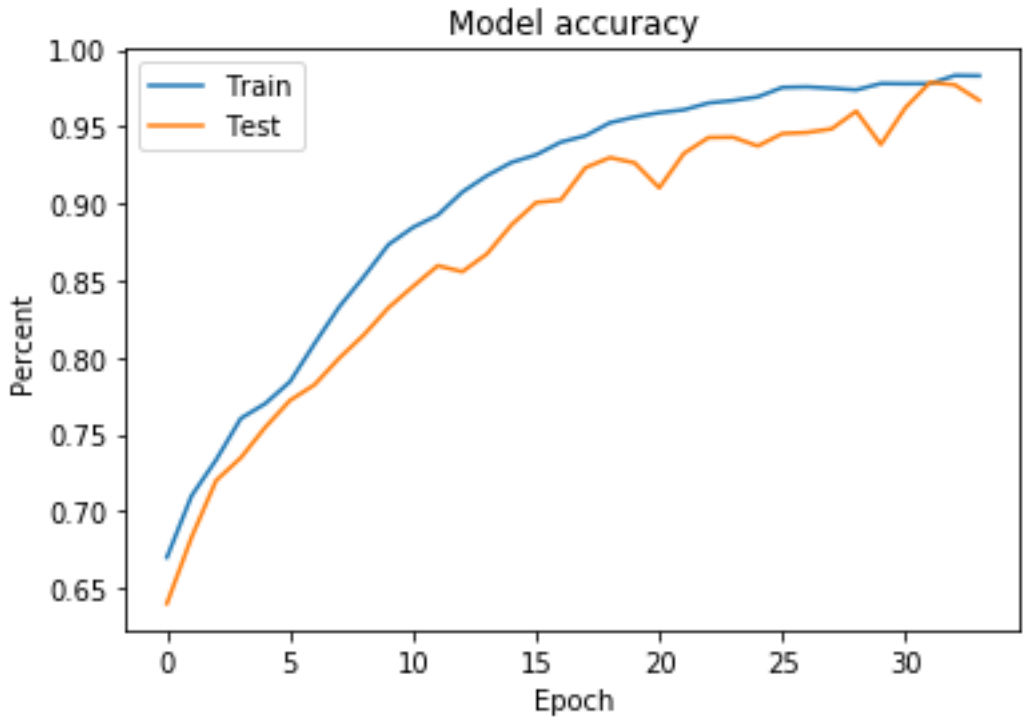


FIGURE 3.11: Example 3

To train the model, we used AdaM optimizer and 0.0001 learning rate.

3.5.3.2 Results

The accuracy improved by 6%, from the 91% to 97%, proving that there is a large amount of information wasted by deep neural networks while classifying images. Also, the model converged in just 37 epochs instead of 50. However, because the first 2 layers had 25 and 50 really large filters, the model still had a large number of parameters, making the training operation on one epoch 40 minutes long.



Convolutional Neural Network		
Type	Filter shape	Input
Conv	50*40*3*25	100*100*3
Max-Pool	2*2	100*100*25
Conv	25*20*25*50	50*50*25
Max-Pool	2*2	50*50*50
Conv	5*5*50*50	25*25*50
Max-Pool	2*2	25*25*50
Conv	5*5*50*50	13*13*50
Max-Pool	2*2	13*13*50
FC	2450*2450	7*7*50 flattened
FC	2450*200	2450
FC	200*2	200

3.6 Transfer Learning

3.6.1 Introduction

What if instead of randomly initializing all parameters of a model, we could use an already trained model, using its knowledge to our advantage. This concept is known as transfer learning. For example a model that was previously trained to classify multiple races of animals could transfer that knowledge to a model that only tries to differentiate between cats and dogs. This model will then learn that there are some similarities between cats and lynxes, or between dogs and wolves, making the resulting model more accurate.

FIGURE 3.12: Cat

FIGURE 3.13: Lynx

Pretraining

The first step in performing a successful transfer learning is to get a model that was previously trained on a similar task. There are 2 ways in which this can be achieved: a new randomized model can be created and then trained using a similar but more comprehensive dataset, or an open-source one could be use instead. This pretrained model will extract more general feature maps from the images, making the new classifier less prone to overfitting.

Fine-tuning

Second step is to transform the pretrained model to fit the new data. This is achieved by freezing the first layers, the ones that create the feature maps, preventing them from further changing. Then, by replacing the last layers, the ones responsible of the classification, with new randomly initialized layers, the new

model learns to interpret the information extracted by the pretrained layers to fit the new dataset.

3.6.2 Popular Architectures

In an attempt to further speed up the model, as well as improve the accuracy, we will have a look at 2 the popular architectures that are publicly available, namely MobileNet and InceptionV3. This section aims to present, as well as explain, the concepts that make these architectures so efficient compared with the regular convolutional neural network we designed earlier.

3.6.2.1 MobileNet

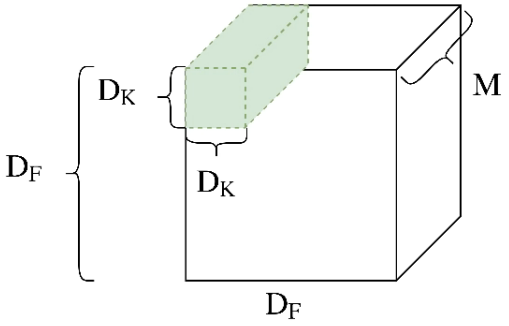
MobileNet is a neural network architecture designed mainly for mobile and embedded vision applications. In the **?** it was mathematically proven to be 8-9 times faster than typical convolutional networks with minor losses in accuracy. The main feature that makes this possible is the concept of depthwise separable convolution.

Depthwise separable convolution

In the case of a kernel of size DK*DK*M and an image with size DF*DF*M, a regular convolution operation with N filters takes DK*DK*DF*DF*M*N operations.

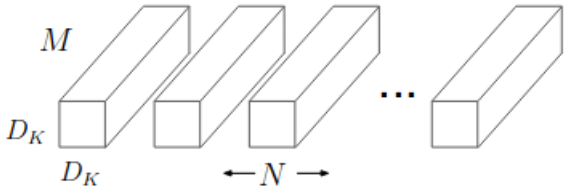
$kernelWidth*kernelHeight*imageWidth*imageHeight*nr.ofchannels*nr.offilters$

As can be seen in the image bellow, the green block is superimposed over the image, performing the convolution operation on the entire block. This means that features are not only extracted from the differences between pixels in one channel, they are also being extracted from the differences between one pixel, across every channel.



Regular Convolution Operation (Image sourced from **?**)

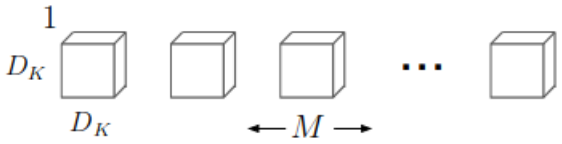
A depthwise separable convolution seeks to split the extraction of features from 1 channel and the extraction of features between the channels. To achieve this the convolution operation is split into 2 steps.



Standard convolution filters (Image sourced from **?**)

First, a depthwise convolution is performed. This is a similar convolution to the one presented above, but instead of a block, a square is used instead. One depthwise operation takes DK*DK*DF*DF*M operations.

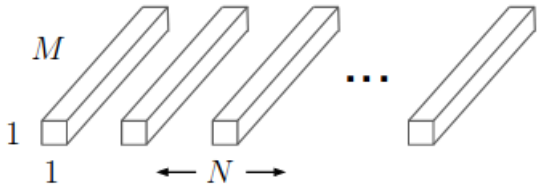
$$kernelWidth * kernelHeight * imageWidth * imageHeight * 1$$



Depthwise convolution filters (Image sourced from **?**)

The depthwise operation is followed by a pointwise convolution which has a complexity of DF*DF*M*1 operations. This takes the location of one pixel and performs a convolution on all of its layers. If multiple depth-wise filters are used, then the complexity of the pointwise operation is further multiplied by the number of channels used.

$$1 * 1 * imageWidth * imageHeight * nr.ofchannels * nr.offilters$$



Pointwise convolution filters (Image sourced from **?**)

$$DK * DK * DF * DF * M + DF * DF * M * N$$

Total complexity of the depth-wise separable convolution

$$\frac{DK * DK * DF * DF * M + DF * DF * M * N}{DK * DK * DF * DF * M * N} = \frac{1}{N} + \frac{1}{DK^2}$$

Comparison between the complexity of regular and depthwise separable convolutions

$$\frac{1}{1024} + \frac{1}{3^2} = 0.112$$

Comparison between regular and depthwise separable convolutions with 1024 filters and 3*3 kernel size

It can be observed in 3.6.2.1 that a depth-wise separable convolution requires only 11.2% of the operations required by a regular convolution.

Architecture

MobileNet combines the increased effectiveness of feature extraction from regular convolution operations with the scalability of depth-wise convolutions. This results in a fast model with minimal losses in accuracy.

As can be seen in 3.6.2.1, the size of the filters are considerably smaller than those of the convolutional network presented above 4.4. Moreover, it should be noted that the input size is more than 4 times bigger than what we previously used.

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

(Image sourced from **?**)

Pretraining

In order to pretrain this network the ImageNet dataset was used **?**, ImageNet is a dataset containing 14 million images and 22 thousands classes. The adjusted parameters are publicly available and can be easily loaded into memory using keras.applications.MobileNet.

Fine-tuning

For the fine-tuning process, the first 20 layers have been frozen, and the last fully connected layer was replaced with 3 dense layers.

3.6.2.2 Inceptionv3

Similar to how MobileNet architecture was designed to reduce the computational complexity of the convolutions by factorizing the convolution process into 2 distinct processes, depthwise and pointwise convolutions, InceptionV3 **?**, proposes the use of 'bottleneck' layers. Moreover, instead of using 1 filter size per layer, inception layers use multiple filters with different sizes, concatenating the resulting feature maps. This gives the model the freedom to decide what type of features maps are important. A good example where this might be useful is when training a model to classify images that have a black dot right in the center, using a 5x5 filter on the image might diffuse the information coming from that black dot too much, however a 1x1 filter would get a strong activation resulting in a model that not only is more efficient but also converges faster.

'Bottleneck' Layers

Using the equation for the regular convolution operation 3.6.2.1, we can see that one convolution operation with a kernel size of 5x5 requires 2.78 times more multiplications than a 3x3 convolution operation.

$$\frac{5 * 5}{3 * 3} = 2.78$$

However, a 5x5 convolution operation can be speed up by first applying a 1x1 convolution and then being followed by a 5x5 convolution.

$$5 * 5 * 28 * 28 * 3 * 32 = 60211200$$

Number of operations needed with 32 5x5 filters on a 28x28x3 image resulting in a 28x28x32 image

$$1 * 1 * 28 * 28 * 3 * 16 = 37632$$

Number of operations needed with 16 1x1 filters on a 28x28x3 image resulting in a 28*28*16 image

$$5 * 5 * 28 * 28 * 16 * 32 = 10035200$$

Number of operations needed with 32 5x5 filters on a 28x28x16 image resulting in a 28*28*32 image

$$\frac{10035200 + 37632}{60211200} = 0.1672$$

Complexity difference between a normal 5x5 convolution operation and one with a 1x1 bottleneck

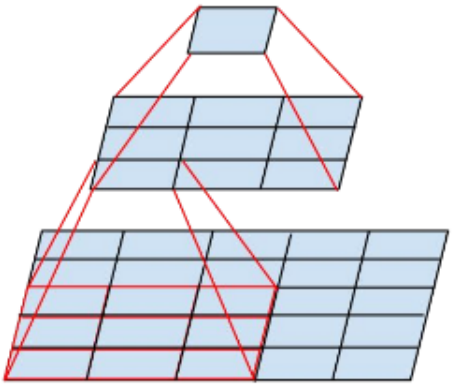


Figure 1. Mini-network replacing the 5×5 convolutions.

(Image sourced from **?**)

$$\frac{225 + 81}{625} = 0.4896$$

Inception Layers

As stated in the beginning, the structure of some convolution layers is also changed, instead of using a regular convolution layer, inception convolution layers combine multiple filters.

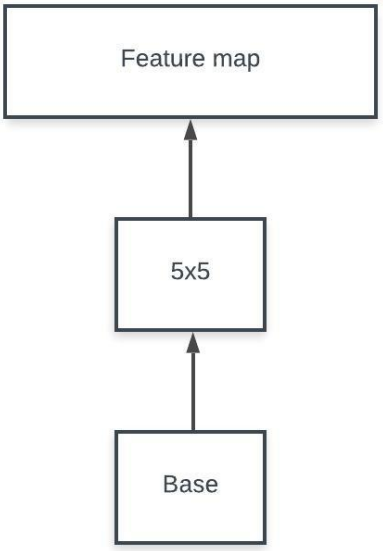


FIGURE 3.14: Regular Layer

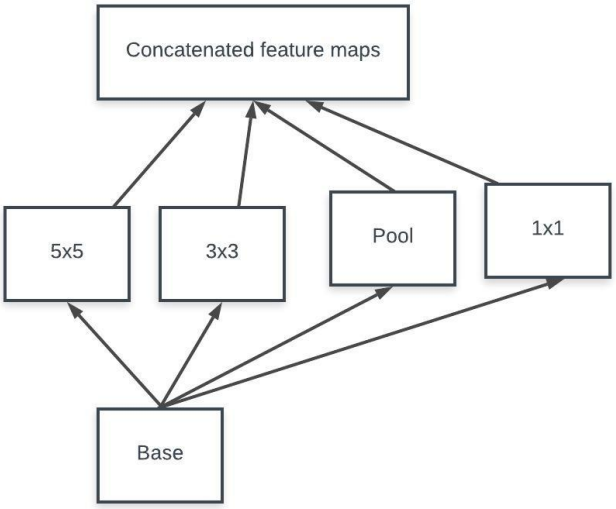
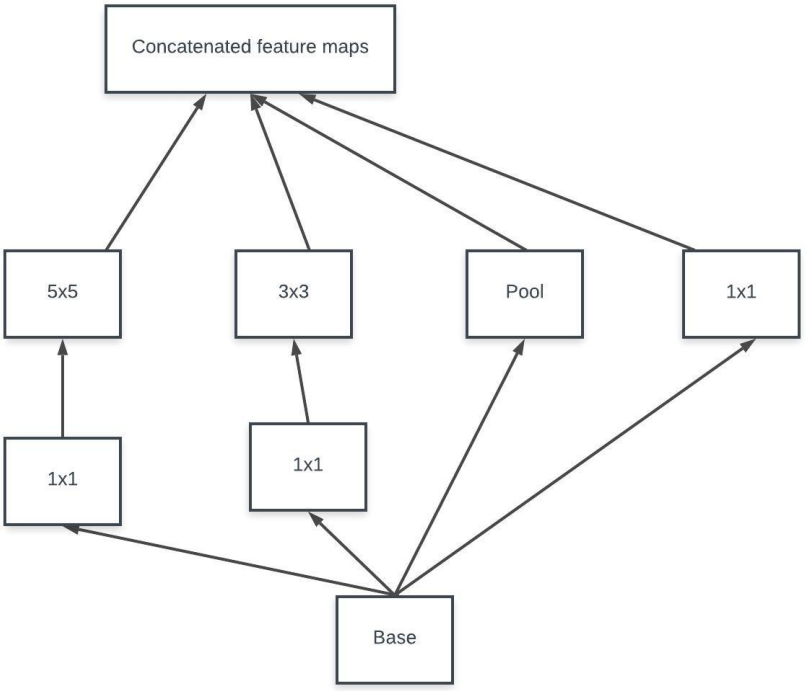


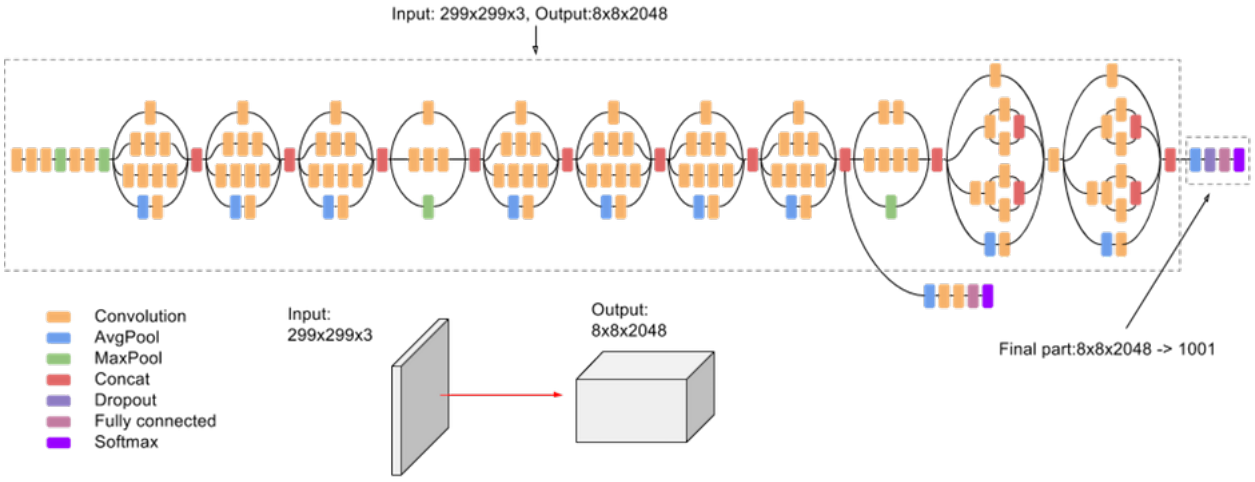
FIGURE 3.15: Naive Inception Layer

The major draw back is the extra computational cost. Besides the fact that the convolution operation is more complex, the resulting image created by the concatenation of all the intermediary convolutions is also significantly larger, increasing its computation complexity of the following layer as well. However, by adding a bottleneck layer to each intermediary convolution in the inception layer, the computation cost can be reduced.



Inception Layer

Architecture



InceptionV3 architecture (Image sourced from **?**)

There are few important things to note here, first the standard resolution chosen by **?** is 299x299, resulting in an image almost 2 times bigger than the image used as input by **?** for the MobileNet model and almost 10 times bigger than the one we experimented with in previous chapters. Second, there are no inception layers at the beginning. Because each inception layer contains some 'bottleneck' layers, it would be inappropriate to restrict the flow of information from the very beginning.

Pretraining and Fine-tuning

Similar to MobileNet 3.6.2.1, an already pretrained model on the ImageNet dataset is publicly available and can be easily loaded using keras.applications.InceptionV3. However, because of the high number of hidden layers, in order to fine-tune the model, the first 130 hidden layers had to be frozen.

Chapter 4

Evaluation

4.1 Introduction

In previous chapter, we only presented the accuracy achieved by the models implemented by us, 91% achieved by the deep neural network 3.4.4 and its improvement to 96% achieved by using a convolutional neural network 2.2.3.2. This chapter aims to compare our models with the state-of-the-art models in both accuracy and evaluation time.

4.2 Accuracy as evaluation

Accuracy is a great evaluation metric when it is applied to balanced datasets 3.3.1.2. However, in our case, the dataset is not balanced, this means that, either the test dataset has to be balanced, or a second evaluation metric for the models has to be chosen. It is not advised to expand the test dataset by neither duplicating nor augmenting existing images. When a model, that learned to classify an image, is presented with 5 instances of that particular image, it will predict it correct 5 times, skewing the accuracy towards that image. Also testing on augmented images is not a good solution either. The resulting images are not representing real world examples in the same way the original images did.

$$\frac{TP + TN}{TP + TN + FP + FN}$$

4.3 Alternatives

Precision, Recall, F1-score

All three can only be used to evaluate binary classifiers.(i.e.toxic-nontoxic comments, malign-benign tumors). One thing to note is that some problems might require higher precision, while others require a high recall score. For example while building a model to auto-play videos for kids, it is better to have some genuine high-quality videos flagged as false-negatives in order to make sure that no dangerous videos get through.

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

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

$$F1 = 2 * \frac{precision * recall}{precision + recall}$$

Confusion table

A better alternative for evaluating a non-binary classifiers is to create a confusion table or matrix. By plotting the desired class on the x-axis and the predicted class on the y-axis, we can visualized how the model fits each class individually. Also, because the results for each class is plotted individually, imbalance in the test dataset is not a problem.

4.4 Race classifiers

Before moving forward to building the breeds classifier, we first compared the race classifiers built in the previous chapter.

Comparison						
Model	Input	Classes	Parameters	Train time(s)	Evaluation Time(s)	Acc
DNN	100x100x3 flattened	2	140,333,066	1.089	1.006	91.7%
CNN	100x100x3	2	7,392,903	1.456	0.990	97.8%
MobileNet	224*224*3	2	5,924,171	0.901	0.288	99.2%
InceptionV3	299*299*3	2	25,476,386	2.2	0.676	99.6%

Based on the findings presented in the previous chapters, the reasons behind some of the results should be evident. The efficiency of both ways of shortening the evaluation time implemented by MobileNet and InceptionV3 is apparent. Because MobileNet was designed for mobile use, it is compact and fast, an evaluation taking 5 times less than our CNN architectures while only having 15% less parameters and using an image 5 times bigger. On the other hand, InceptionV3 was designed to reduce the complexity of the convolution operation with minimal impact to the models performance, resulting in a bigger but still more efficient model.

Another interesting result is the fact that 1 train operation on 140 million parameters took only 1.089 seconds for the deep neural network, while the same operation took 1.4 seconds on only 7 million parameters. We believe this is caused by the way in which the training operation is performed. In a convolution neural network, all the in-between steps are stored in memory to be later used to calculate the gradients. This means that even if only 7 million parameters are being adjusted at the end, the number of calculations done during the training time is way higher.

Lastly, the first accuracy increase is attributed to the extra information used, while the second and third are attributed to a combination of using a pretrained network and the use of higher resolution images. Also, not only the accuracy achieved by the InceptionV3 model beat the 82% accuracy reported by **?**, it equaled the human level accuracy of 99.6% reported by **?**. Even though accuracy is not a good metric,

after verifying these findings using the confusion matrices bellow, we are confident that these findings are accurate.

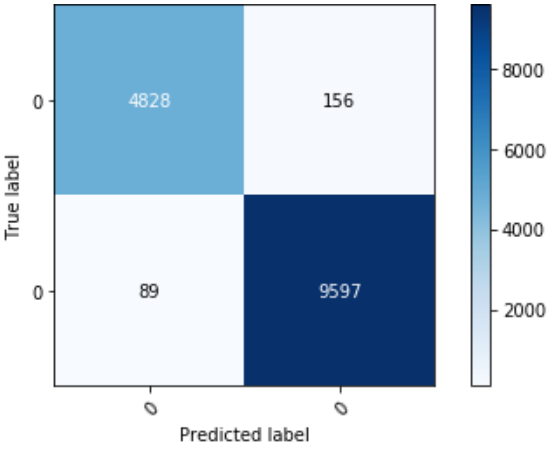


FIGURE 4.1: MobileNet Test

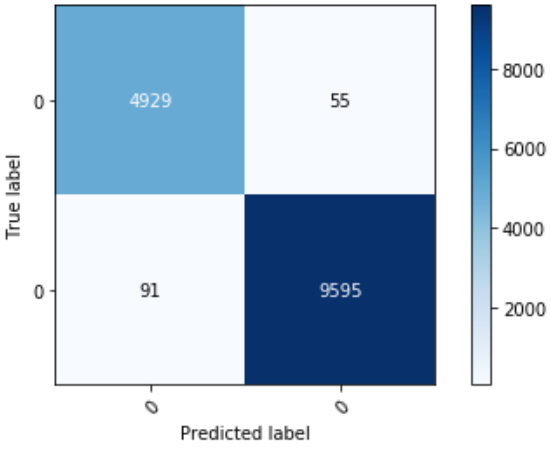


FIGURE 4.2: InceptionV3 Test

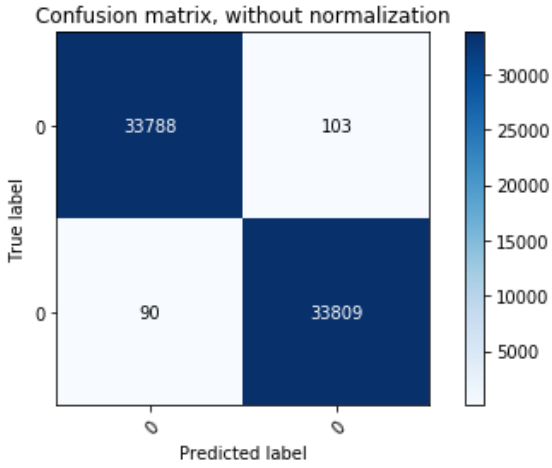


FIGURE 4.3: MobileNet Train

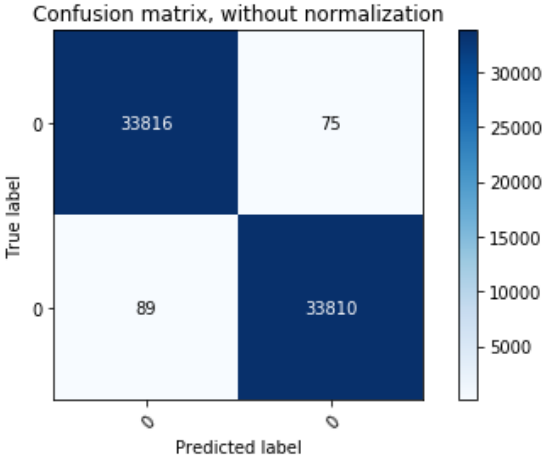


FIGURE 4.4: InceptionV3 Train

FIGURE 4.5: Predicted:Cat Actual:Dog

FIGURE 4.6: Predicted:Dog Actual:Cat

FIGURE 4.7: Predicted:Cat Actual:Dog

4.5 Breed Classifiers

All breeds classifier

Using the 2 best performing models, we further trained 2 classifiers on all 139 breeds. Both MobileNet and InceptionV3 models achieved 70.3% accuracy, which is 11% higher than the accuracy reported by **?**. We attribute this higher accuracy to the higher number of images used. They only used 50 images/breed and they tested on 100 images/breed, while we trained on around 150 images/breed and evaluated the model on 50 images/breed. To achieve a higher accuracy, we strongly believe that more data is required.

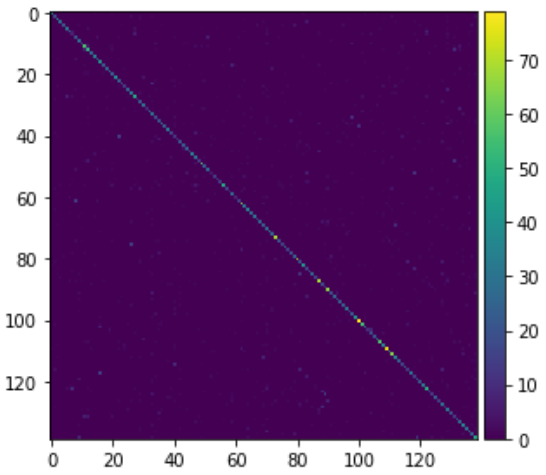


FIGURE 4.8: MobileNet Test

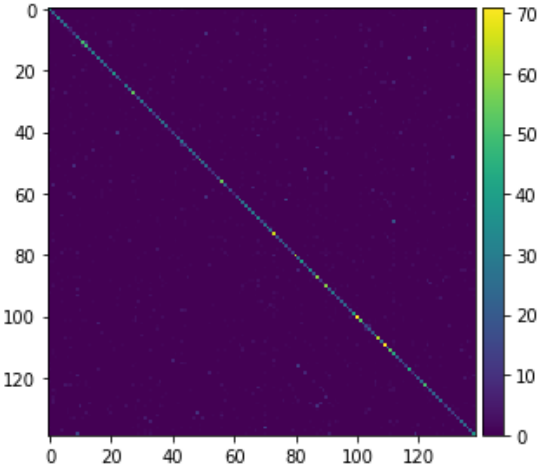


FIGURE 4.9: InceptionV3 Test

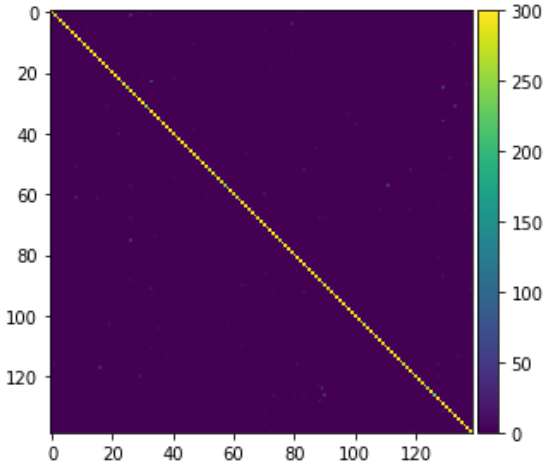


FIGURE 4.10: MobileNet Train

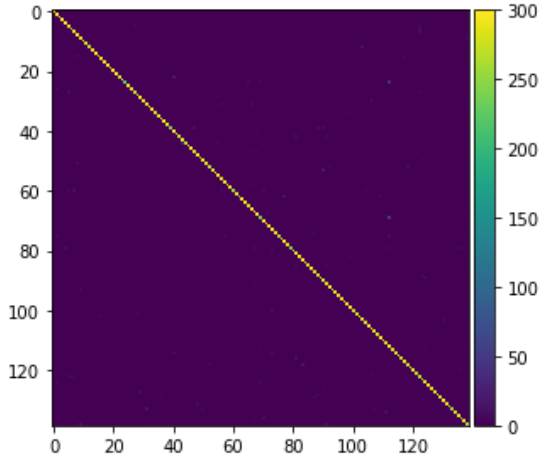


FIGURE 4.11: InceptionV3 Train

FIGURE 4.12: Bernese Mountain Dog

FIGURE 4.13: Appenzeler

FIGURE 4.14: Basset

FIGURE 4.15: Beagle

The most confusing pairs of classes for both models are the Beagle-Basset and Bernese Mountain Dog-Appenzeler, however, we can see in the images above that there are a high number of similarities between these breeds, making the classification difficult even for a human.

Dogs and Cats only classifiers

Similar to how we trained the "all breeds" classifier, hoping to create more accurate overall classifier, we also trained 2 separate models for the 12 breeds of cats and 2 for the remaining 127 breeds of dogs. Because of the lower number of classes, we expected the separated classifiers to have an overall accuracy higher than the "all breed" classifier. Both cats classifiers achieved 86% accuracy while the dogs

classifiers achieved 74% accuracy, this means that if the race classification is done prior to feeding the image to one of these 2 models, we can expect at least an accuracy increase of 3%.

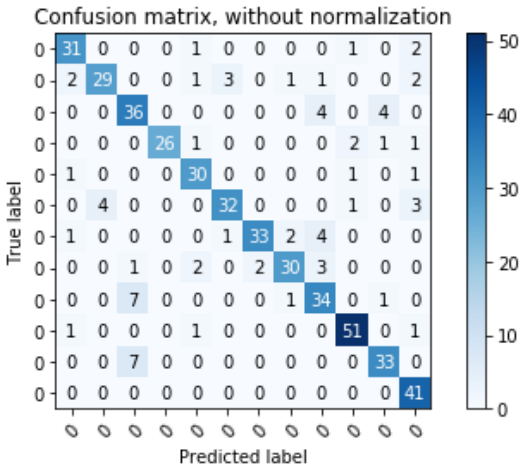


FIGURE 4.16: Cats MobileNet Test

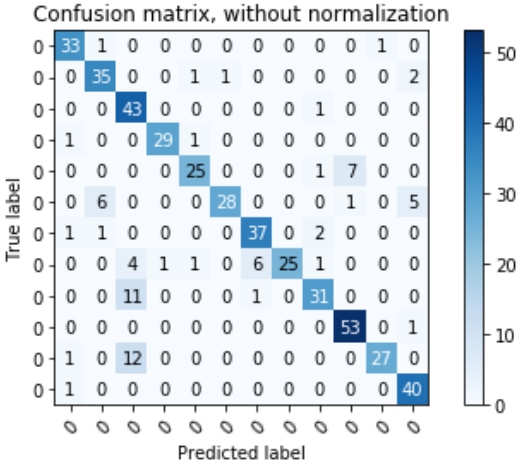


FIGURE 4.17: Cats InceptionV3 Test

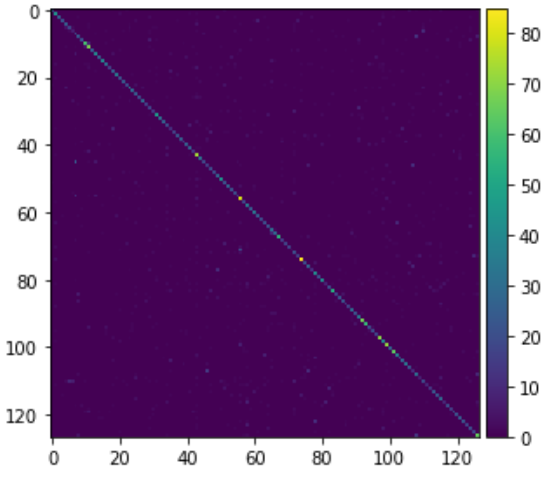


FIGURE 4.18: Dogs InceptionV3 Test

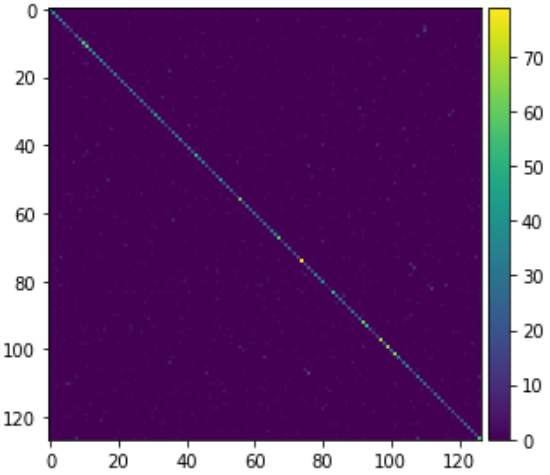
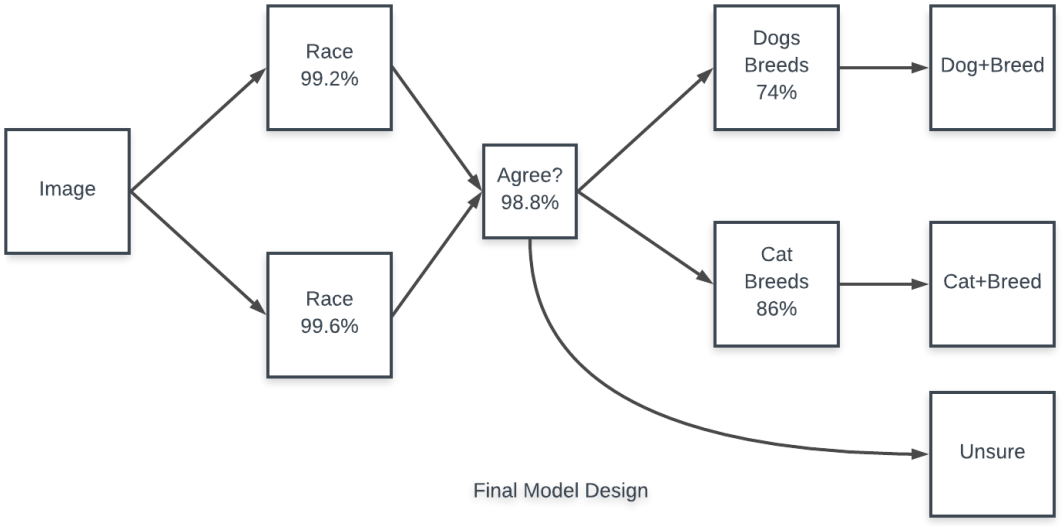


FIGURE 4.19: Dogs MobileNet Test

Chapter 5

Final Design

Because the intent of this paper is to offer a detailed perspective of what machine learning is currently capable and not just to build a high-performing breeds classifier, this design was chosen with flexibility and reliability in mind. We strongly believe that this machine learning algorithm can be transferred into any field that first requires a general prediction (i.e.cancerous or non-cancerous, toxic or non-toxic comments) followed by a more specific prediction as long as there is enough labeled data available for the more general classes.



Final Model Design

First, we combined the 2 models that predict the race. We used 2 models in order to reduce the possibility of a false positive getting through. If these 2 models do not predict contradictory results, then the result is further fed to the corresponding 2 models that will further predict the breed. If the last 2 models contradict, however, then both results will be shown. We believe that in the medical field, at

least, instead of printing an "unsure" message any time any 2 models contradict during a prediction, by printing the 2 most probable diseases, or breeds in this case, the time required for a doctor to research the options and provide an accurate diagnostic could be significantly shortened.
It should be noted that there is no strong requirement for 6 models to make the predictions. Similar results could be achieved with just 2 classifiers. However, it is highly likely that by using just 2 models the number of false positives would be significantly higher, which, in some fields could prove problematic.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

There are several achievements in this paper that are worth noting. Not only we built a model capable of classifying with human level accuracy (99.6%) cats and dogs, beating the proposed 82% accuracy reported by **?**, we also beat the 59% accuracy achieved by **?** by building a model with 70% accuracy on 139 different breeds instead of just 37, proving the unquestionable effectiveness of both having a large dataset and an accurate pretrained model. Moreover, we believe that, because the steps were recorded in a clear and concise way, these findings are easy to understand and highly reproducible by any reader.

6.2 Future Work

There are still a lot of things that can be added or improved in our implementation. First, we noticed that all our models hit a plateau after just 30 epochs, if more time was available, we would have explored some regularization methods in order to make the model less prone to overfitting, possibly improving the accuracy slightly. It is worth mentioning that a similar result could have been already implemented using image augmentation, however, this idea was dropped during the implementation because the results were already above the proposed threshold so we saw no reason implementing something that would inevitably lead to a considerable increase in the already high training times.

Also, we need to admit that there are still a lot of things that went uncovered. Machine learning is a vast domain and uncovering everything requires more than what one single paper is able to offer. In this paper we only managed to explain the general concepts behind machine learning, and offer some general information about some more complex concepts that were required to understand how we achieve these results. Even though the general limitations covered by us apply to any machine learning algorithm, in order to get a more complete picture many more concepts would need explanation.