Machine Learning Engineer Nanodegree Capstone Project - Final Report

Author: Ray Phan

Date: May 18th, 2020

Student provides a high-level overview of the project in layman's terms. Background information such as the problem domain, the project origin, and related data sets or input data is given.

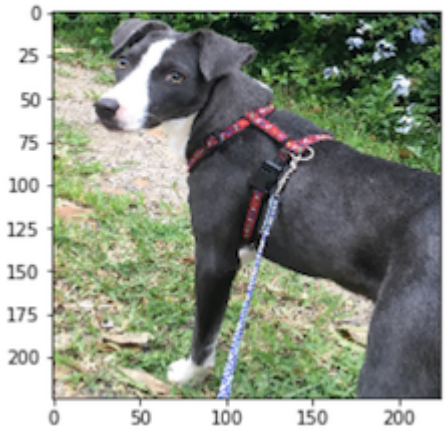
Project Overview

Introduction

The purpose of this project is to learn how to build a pipeline to process real-world, user-supplied images and shape it towards a real-world application, which is classifying dog breeds given an image of a dog. For this application, the proposed algorithm will identify an estimate of the canine's breed. If supplied an image of a human, the code will identify the resembling dog breed.

An example of what is to be achieved can be found below.

```
hello, dog!
your predicted breed is ...
American Staffordshire terrier
```



Domain Background

Classifying dog breeds just by image information is a very challenging problem. This problem is even hard for human beings without physically touching or assessing the dog in person. The end goal is to develop a model such that it infers what type of dog breed it is given a user-supplied image. After doing a cursory search through search engines, this topic is unfortunately not of academic interest but it has seen several surges in popularity on social media and on the Internet. For example, there is a renowned [Kaggle competition that was held two years ago for this challenge 1]1. Other examples can be found in [2], [3], [4], [5]. Not only is this a very challenging problem, the original competition had 120 different classes of dog breeds.

Project Origin

I found the discerning of 120 different classes to be intriguing, especially if there are certain nuances between the breeds that even humans would have trouble in discerning. I took on this challenge purely to see if machine learning and deep learning algorithms could automatically learn the right filters and weights to optimally decide this for us. Essentially, this is a relatively large multi-class classification problem where we fortunately can use supervised learning to solve this problem as we have the data available to us that is separated into the proper classes. Ultimately, the purpose of this is to explore the data available, design the right pre- and post-processing steps on the images and to decide which methodologies required to train a model and achieve a certain performance metric.

Datasets and Inputs

Fortunately, we have a dataset we can use for this challenge and is supplied by the Udacity Machine Learning Engineer Nanodegree Course. The dataset consists of already segregated training, validation and test groups. The input images must be colour images, though they can be of any resolution. Of course these must be images as the goal is to idenify what the canine breed would be in an image. In addition, the dataset consists of both dogs and humans.

For the dog images dataset, the images are of various resolutions but there are 8,351 images which are further segregated into 6,680 images for the training set, 836 images for the validation dataset and 835 images for the test dataset, approximately creating an 80%-10%-10% split. The images contain different backgrounds, and for each category the images are not balanced. Specifically, there are more images in one category than another. The number can vary between 4 and 8 images per category.

For the human images dataset, this contains 13233 total human images which are sorted by their corresponding names with each name in a directory. There are 5,750 unique names and each image is of size 250 x 250. The images also have different backgrounds and perspectives. The data is also not balanced as some directories have only one image when other directories have several images.

There are an abundance of dog and human images which should be enough for a data-heavy machine learning algorithm to properly bridge the gap between image to dog breed.

Problem Statement

The goal of this project is to build a machine learning - deep learning specifically - that can ultimately be used in a web application for example that can process real-world, user-supplied images. Specifically, this system will perform two tasks:

1. Given an image, if we detect that a dog is found the model will determine the canine's breed.
2. Given an image, if we detect that a human is found the model will specify that it's a human, then identify the closest dog breed that matches the human's appearance.

Because there is an abundance of images, a Convolutional Neural Network (CNN) would be the most optimal (and obvious) choice here as neural networks are data-hungry, but because we have a large amount of data available, we would be able to create a set of filters, weights and biases to help us automatically solve this problem for us. A CNN is designed to examine an image and extract an optimal feature representation that would be suitable for a classification algorithm to optimally choose the right class. The classification algorithm would naturally be a multi-layer perceptron (MLP). Therefore, the solution first involves training a CNN with the dog image dataset. The choice of architecture for the CNN will require some experimentation and study but this will naturally be part of the exploratory analysis of the problem. Secondly, we can use an already pre-trained

solution to detect whether we can find a human face. We can use OpenCV's Haar Cascades which have been carefully engineered to find human faces. Thirdly, in order to distinguish between whether we see a human face or a dog face, we have to determine whether we can find a dog in the image first. Creating a dog breed classifier only tells us the type of dog, but not if there's an actual dog appearing in the image. Thankfully, we can use already pretrained networks that have been trained on the ImageNet database that do detect the existence of dogs but not to the granularity of what we expect from the proposed application. A pretrained network like VGG16 [7] can work here. Therefore, the pipeline will be as follows:

1. Use OpenCV's [8] Haar Cascades to determine if we can find a human. If yes, run the dog breed classifier to see what the closest dog breed is given the human image.
2. Use a pretrained VGG16 network to determine if we see an actual dog in the image. If yes, run the dog breed classifier to determine what the dog breed is.
3. If neither (1) or (2) execute, produce an error.

Naturally since the solution to this would be CNNs, a benchmark model would be to try and create a CNN architecture from scratch to see if this solution is viable. If we can create a CNN model "from scratch" which can achieve a test accuracy of say 10%, we can confirm that we would be able to train a network to achieve the desired result as this would be better than random guess, which is 1 / 133 or roughly less than 1%. Once we confirm that we can create a CNN architecture from scratch that achieves this desired test accuracy, we would naturally transition to using a deeper network and transfer learning such that the deeper network has already been pretrained on the ImageNet database, but we change the final fully-connected layer that handles the classification of the image by fine-tuning the weights to account for the dog image dataset that we provide to it. In this case, we hope to achieve a test accuracy of 60% or more.

Metrics

Because there is a relatively good distribution of data for each dog class, we can simply use accuracy which is the fraction of the dog breeds the algorithm classifies correctly in proportion to the entire dataset. We do this for the training, validation and test datasets. However, because there are only a small number of images per class we may also opt to look at the magnitude of the loss function we impose on the CNN when optimally finding the parameters that can most accurately classify the dog breeds. In this case, we use the standard multi-class cross entropy loss function to help us do this. Also note that the data is highly imbalanced. Therefore, we can create a confusion matrix and calculate the precision and recall for each class individually by summing the entire row or entire column that the class resides in. In other words, for class `i` in the dataset with confusion matrix `M(i,j)` accessing a value at row `i` column `j`, we can calculate the precision for class `i` by `M(i, i) / sum(M[:, i])` with `sum(M[:, i])` summing over all rows for column `i`. Similarly, we can calculate the recall by `M(i, i) / sum(M[i, :])` with `sum(M[i, :])` summing over all columns for row `i`.

Data Exploration

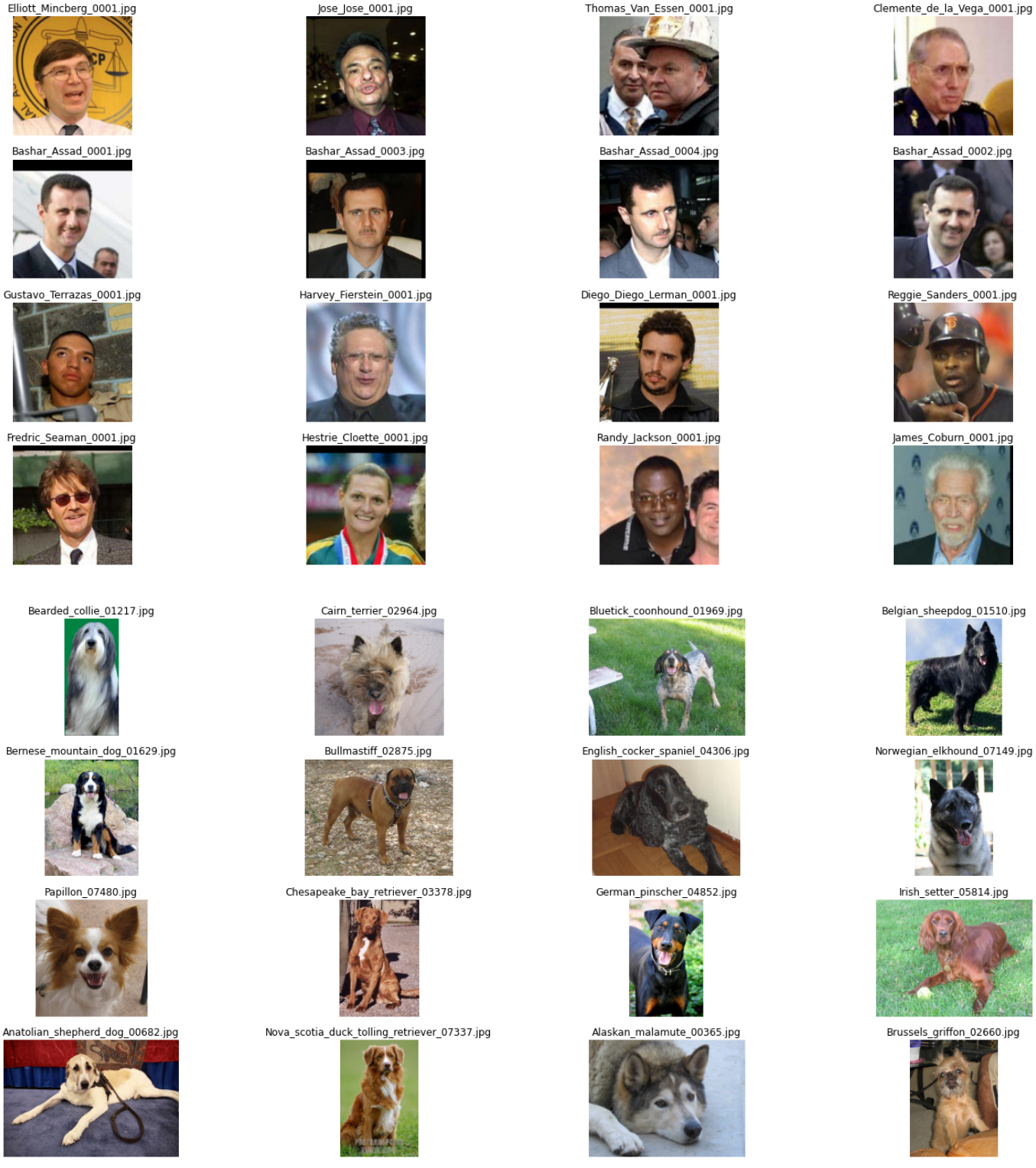
There are 8,351 images of dogs in the dog dataset in total. Of them, we have 6,680 images for the training set, 836 images for the validation dataset and 835 images for the test dataset, approximately creating an 80%-10%-10% split. The images contain different backgrounds, and for each category the images are not balanced. Specifically, there are more images in one category than another. There are 133 breeds in total in this dataset.

For the human face images dataset, this contains 13233 total human face images which are sorted by their corresponding names with each name in a directory. There are 5,750 unique names and each image is of size

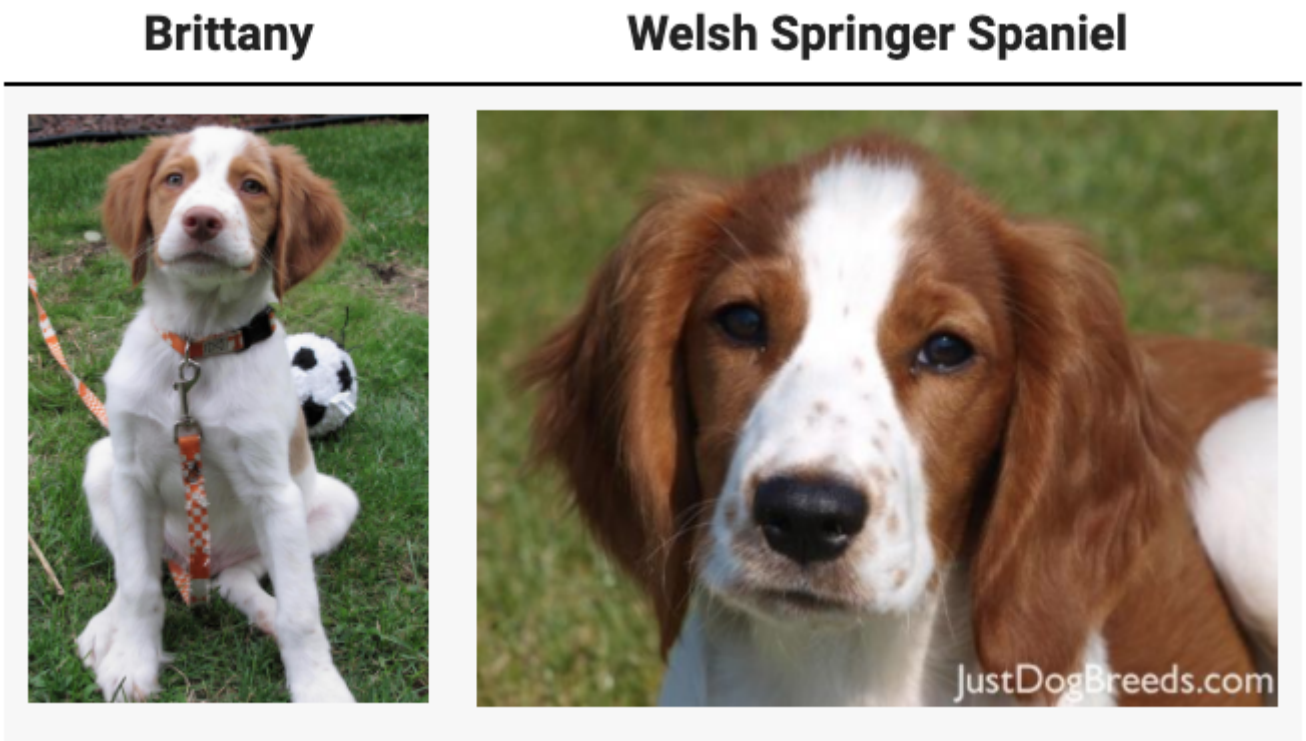
250 x 250. The images also have different backgrounds and perspectives. The data is also not balanced as some directories have only one image when other directories have several images.

There are an abundance of dog and human images which should be enough for a data-heavy machine learning algorithm to properly bridge the gap between image to dog breed.

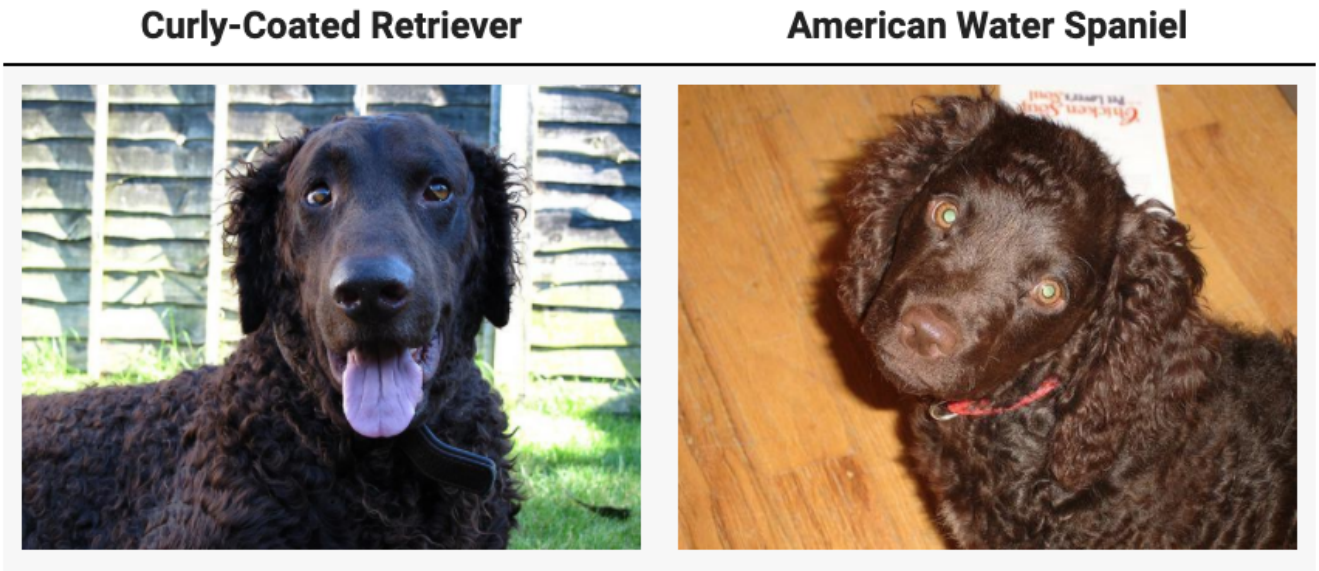
Because the datasets consist of just images, it would rather difficult to show plots and graphics that would be typical of tabular or one-dimensional data. However, we can most certainly extract a sample of each dataset and see what we're dealing with. The figures below show a some examples of dog images and human face images to give us this idea.



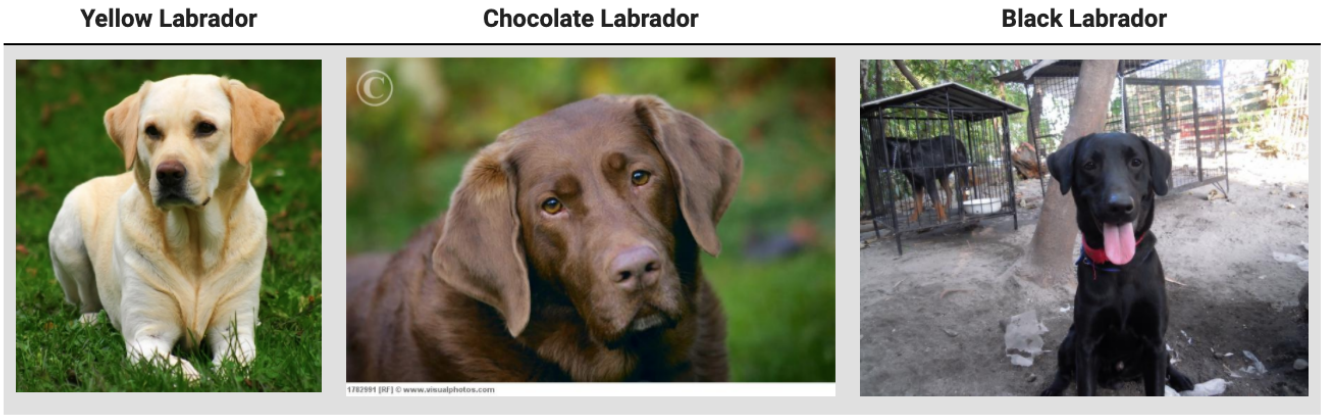
We can see that each dataset has varied backgrounds. In addition, what is extremely challenging about this dataset is that certain dogs look very similar to each other, but they are in fact different breeds. The figure below illustrates two different dog breeds, but they look extremely similar to each other. This is challenging even for humans!



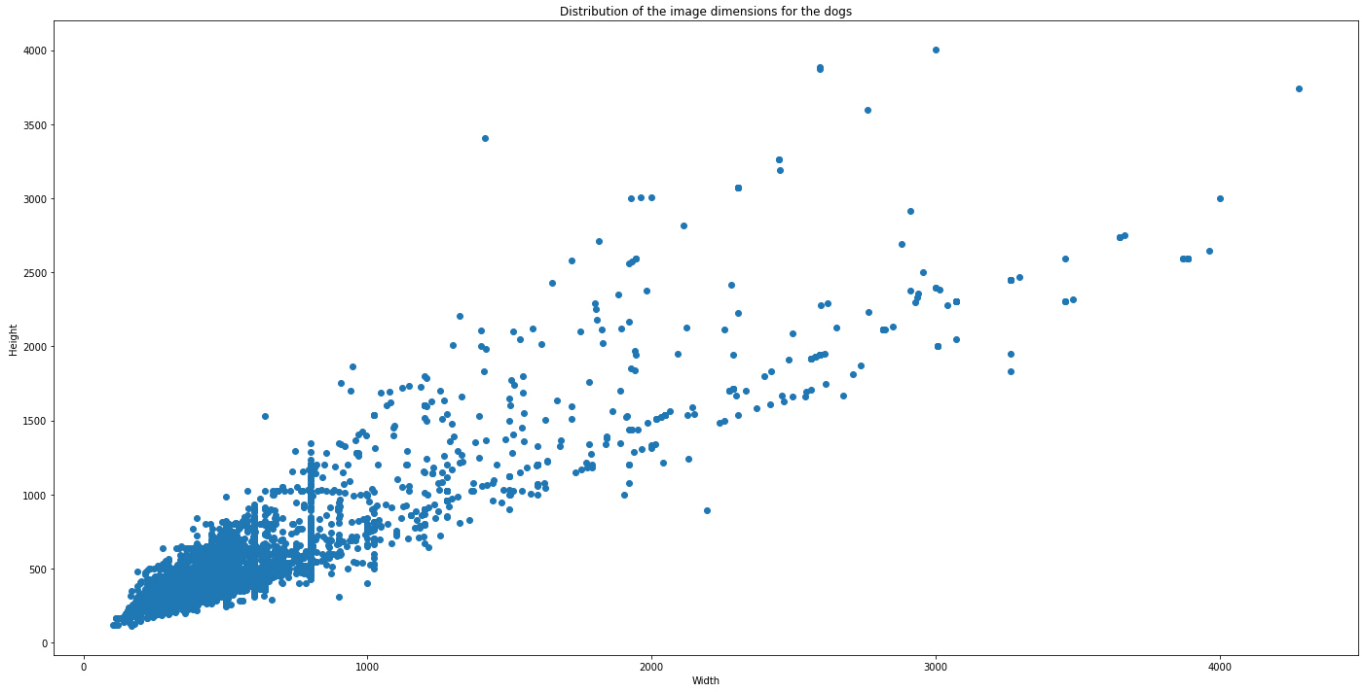
Here's another example where the fur is quite similar.



Finally, there are breeds where the physical appearance is the same, but the only exception to this is the colour.



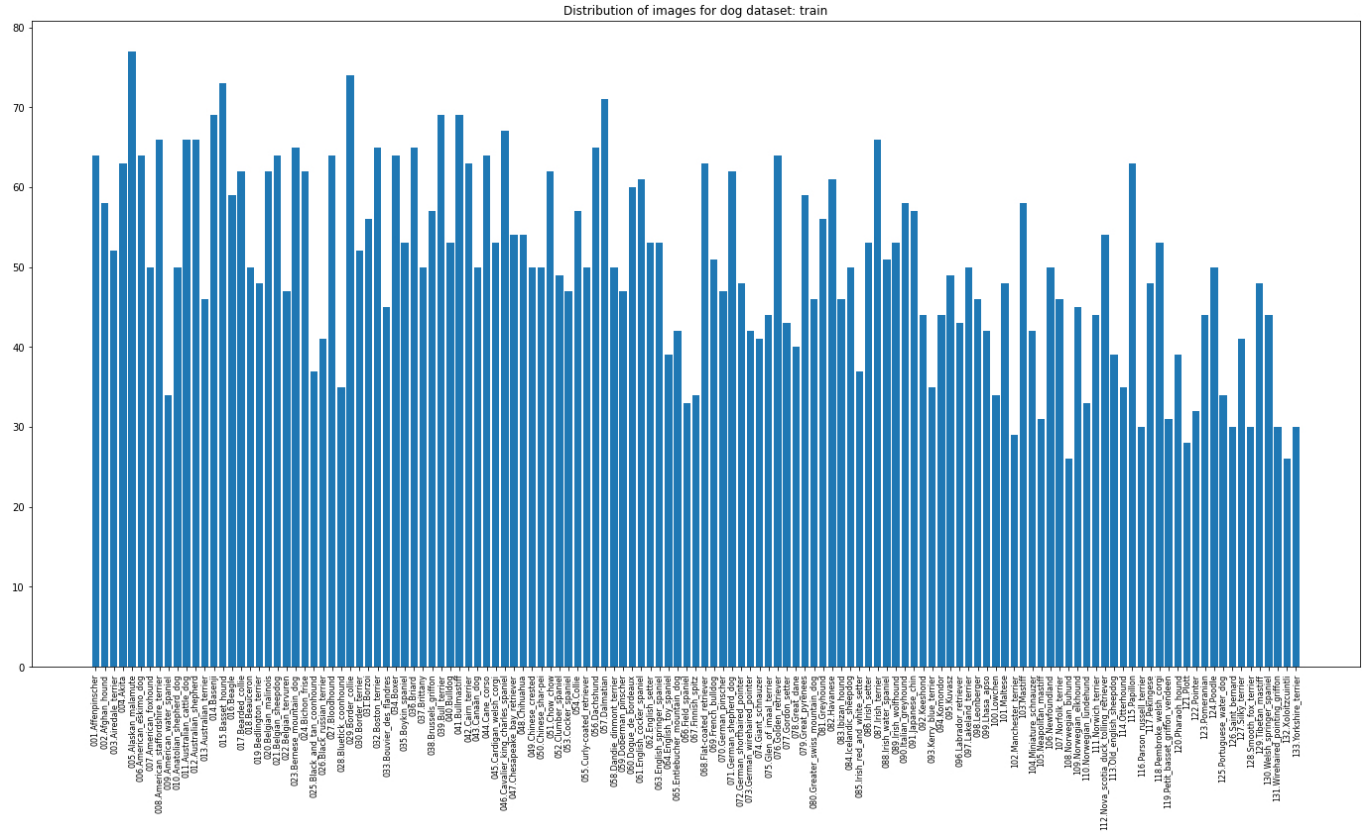
As an additional challenge, the dog dataset also can vary in resolution. The figure below illustrates a scatterplot showing the distribution of the image dimensions in the dataset. The horizontal axis records the width while the vertical axis records the height. There are a few high resolution images in this dataset, but the majority of the images are considered lower resolution.

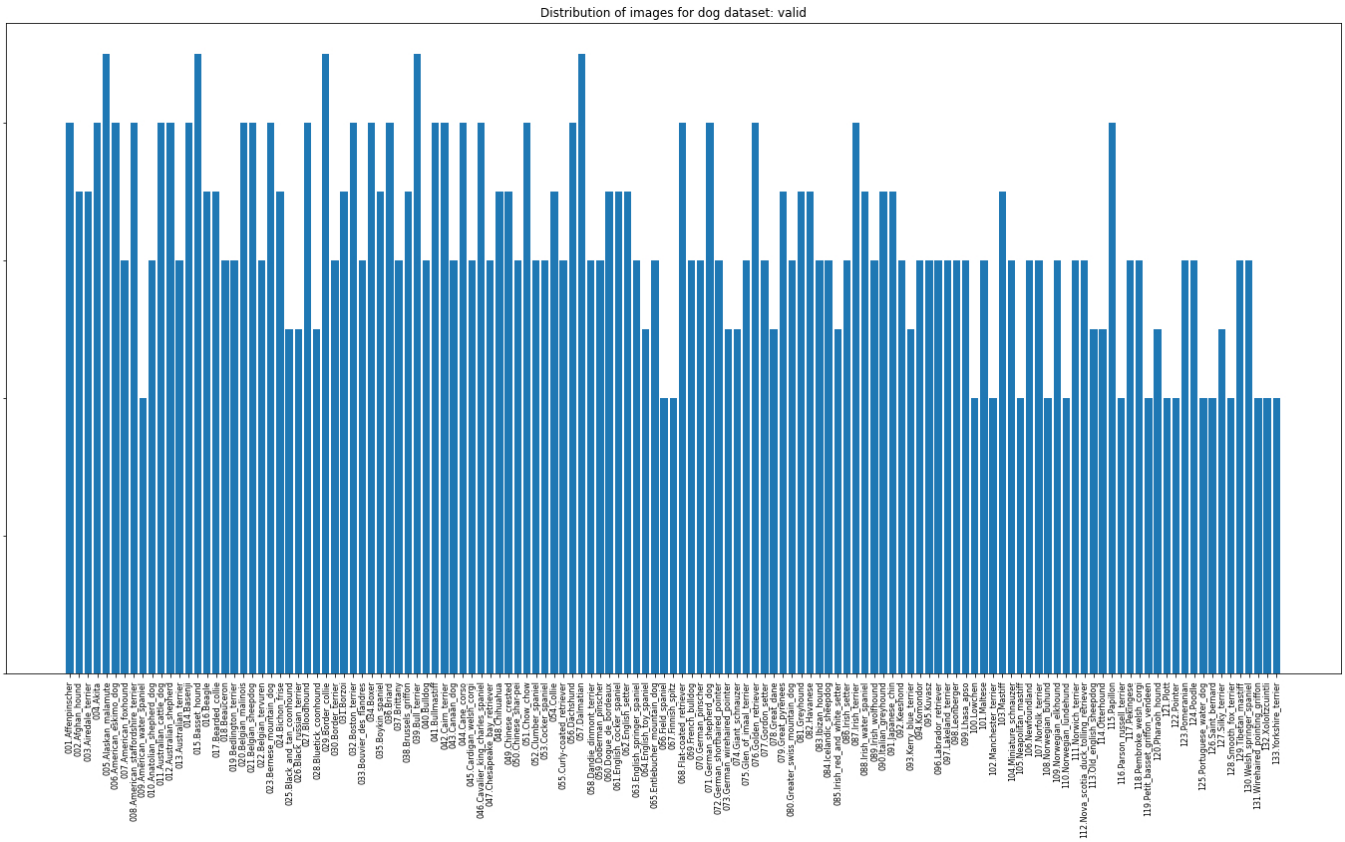


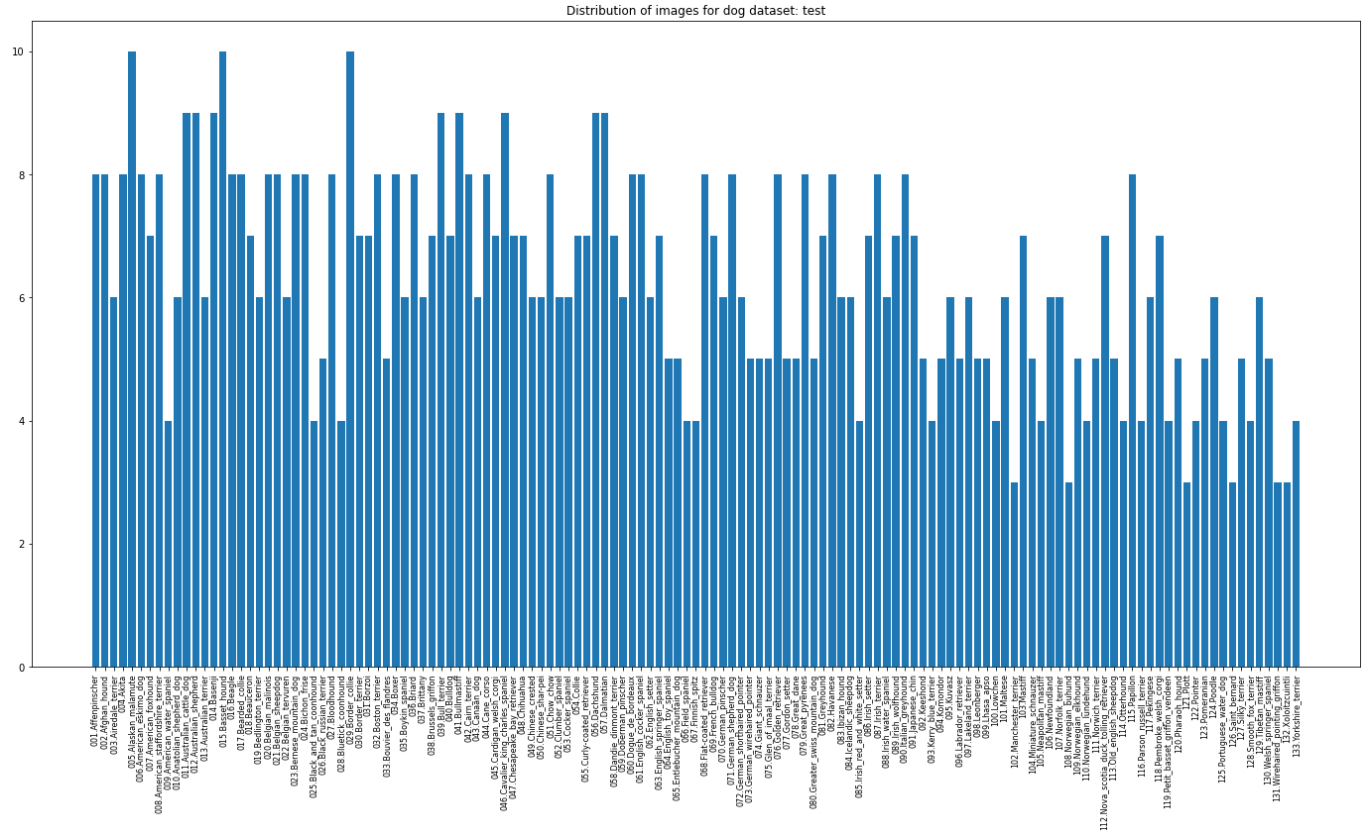
Interestingly, if we took the average width and height of all of the dog images in the dataset, we get approximately 567 pixels for the width and 529 pixels for the height. Since we now see that the image resolution for the dog images can vary, it is important that we develop a model that is resolution agnostic. That is, we should be able to come up with a model where the resolution is immaterial to the performance of the model.

Exploratory Visualisation

As noted above, the dataset is quite imbalanced. To quantify this properly, let us explore how many images there are for each category for the training, validation and test groups for the dog dataset. Those are shown in the images below.


Distribution of images for dog dataset: train
001.Affenpinscher
002.Afghan_hound
003.Airedale_terrier
004.Akita
005.Alaskan_malamute
006.American_eskimo_dog
007.American_foxhound
008.American_staffordshire_terrier
009.American_water_spaniel
010.Anatolian_shepherd_dog
011.Australian_cattle_dog
012.Australian_shepherd
013.Australian_terrier
014.Basenji
015.Basset_hound
016.Beagle
017.Bearded_collie
018.Beauceron
019.Bedlington_terrier
020.Belgian_malinois
021.Belgian_sheepdog
022.Belgian_tervuren
023.Bernese_mountain_dog
024.Bichon_frise
025.Black_and_tan_coonhound
026.Black_russian_terrier
027.Bloodhound
028.Bluetick_coonhound
029.Border_collie
030.Border_terrier
031.Borzoi
032.Boston_terrier
033.Bouvier_des_flandres
034.Boxer
035.Boykin_spaniel
036.Briard
037.Brittany
038.Brussels_griffon
039.Bull_terrier
040.Bulldog
041.Bullmastiff
042.Cairn_terrier
043.Canaan_dog
044.Cane_corso
045.Cardigan_welsh_corgi
046.Cavalier_king_charles_spaniel
047.Chesapeake_bay_retriever
048.Chihuahua
049.Chinese_crested
050.Chinese_shar-pei
051.Chow_chow
052.Clumber_spaniel
053.Cocker_spaniel
054.Collie
055.Curly-coated_retriever
056.Dachshund
057.Dalmatian
058.Dandie_dinmont_terrier
059.Doberman_pinscher
060.Dogue_de_bordeaux
061.English_cocker_spaniel
062.English_setter
063.English_springer_spaniel
064.English_toy_spaniel
065.Entlebucher_mountain_dog
066.Field_spaniel
067.Finnish_spitz
068.Flat-coated_retriever
069.French_bulldog
070.German_pinscher
071.German_shepherd_dog
072.German_shorthaired_pointer
073.German_wirehaired_pointer
074.Giant_schnauzer
075.Glen_of_imaal_terrier
076.Golden_retriever
077.Gordon_setter
078.Great_dane
079.Great_pyrenees
080.Greater_swiss_mountain_dog
081.Greyhound
082.Havanese
083.Ibizan_hound
084.Icelandic_sheepdog
085.Irish_red_and_white_setter
086.Irish_setter
087.Irish_terrier
088.Irish_water_spaniel
089.Irish_wolfhound
090.Italian_greyhound
091.Japanese_chin
092.Keeshond
093.Kerry_blue_terrier
094.Komondor
095.Kuvasz
096.Labrador_retriever
097.Lakeland_terrier
098.Leonberger
099.Lhasa_apso
100.Löwchen
101.Maltese
102.Manchester_terrier
103.Mastiff
104.Miniature_schnauzer
105.Neapolitan_mastiff
106.Newfoundland
107.Norfolk_terrier
108.Norwegian_buhund
109.Norwegian_elkhound
110.Norwegian_lundehund
111.Norwich_terrier
112.Nova_scotia_duck_tolling_retriever
113.Old_english_sheepdog
114.Otterhound
115.Papillon
116.Parson_russell_terrier
117.Pekingese
118.Pembroke_welsh_corgi
119.Petit_basset_griffon_vendeen
120.Pharaoh_hound
121.Plott
122.Pointer
123.Pomeranian
124.Poodle
125.Portuguese_water_dog
126.Saint_bernard
127.Silky_terrier
128.Smooth_fox_terrier
129.Tibetan_mastiff
130.Welsh_springer_spaniel
131.Wirehaired_pointing_griffon
132.Xoloitzcuintli
133.Yorkshire_terrier


Distribution of images for dog dataset: valid



As we can see from the the graphs above, there are some categories in the training set that have a good abundance of images. For example, the Akita dog breed (label number 4) has close to 77 images whereas the Norwegian buhund (label number 108) has close to 26 images. For typical machine learning algorithms, this data imbalance could pose as a problem. Fortunately for CNNs, we can introduce perturbations to the existing images through means of data augmentation so that the problems with data imbalance are reduced.

Algorithms and Techniques

Human Face Detector

To detect whether there is a human face in the image, we can use OpenCV's Haar Cascade Face Detector. If this detector finds that there is a human face in the picture, we notify the user there is a human here and determine which breed of dog they resemble.

Dog Detector

To detect whether there is a dog in the image, we will use a pretrained VGG16 network with ImageNet weights. The ImageNet database consists of 14 million images belonging to 1000 classes. Of these 1000 classes is a subset of classes that belong to a dog. Because our dog dataset is only a fraction of images compared to the ImageNet database, and because this network is able to discern other classes of images, it would make sense to use this pretrained network to see if there is a dog in the image first. The actual breed does not need to be known as the granularity of classifying the dog breed will be achieved with our proposed dog breed classification model. We only need to identify whether there is a dog in the image first. The choice of VGG16 is due to the fact that it's a more lightweight model compared to the more recent deeper networks with many more layers and thus many more parameters. Also remember that we only need to detect the existence of a dog.

Dog Breed Classifier - From Scratch

As noted earlier, we will explore two sets of CNN architectures. The first one will determine if a CNN model is viable in classifying dog breeds. As such, we will design a very simple CNN model using a combination of 2D convolutional layers, 2D max pooling layers, ReLU activation functions, fully-connected layers and dropout layers. The implementation of this network will be done in PyTorch.

Dog Breed Classifier - Using Transfer Learning

Once we verify that the task of classifying the dog breed is possible, we will use transfer learning on a deeper network, ResNet50 where we only change the last layer of the network, the fully-connected layer to have the same number of output neurons as there are classes. ResNet50 was also trained with the ImageNet database. Therefore, it would make sense to "freeze" the layers so that the images undergo the optimal feature extraction of the images while only tuning the fully-connected layer as that is responsible for doing the classification of the dog breed.

Benchmark

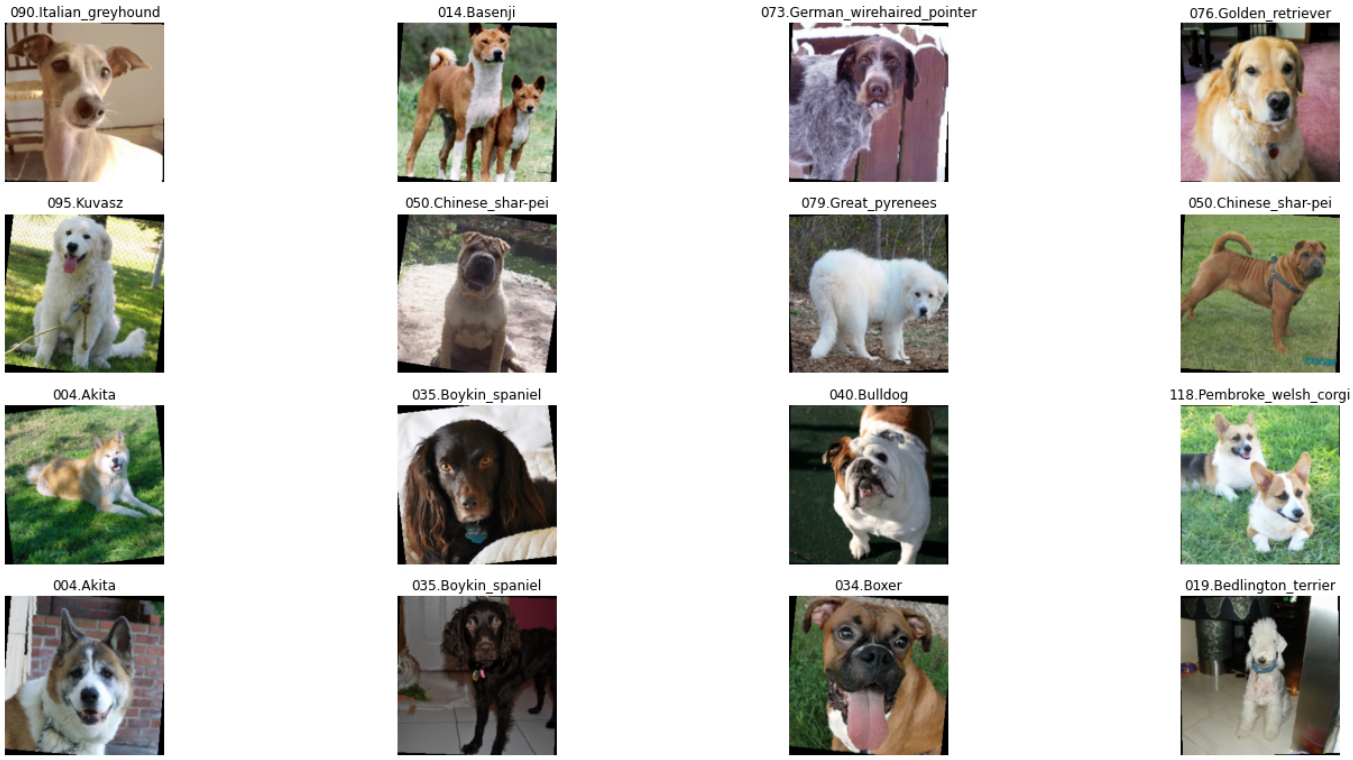
As noted in the proposal, for the CNN "from scratch", we hope to achieve a test accuracy of 10% which is better than random chance to prove to us that this task is viable. For using transfer learning, we hope to achieve a test accuracy of 60%, but we optimistically are hoping for higher. In addition to the accuracy, we are hoping that the precision and recall for the majority of classes in the dog dataset are also around the same magnitude, as well as the average precision and recall over all dog classes.

Data Preprocessing

To ensure that we tackle the data imbalance problem that is part of the dog dataset, we seek to impose augmentations to the images before they are introduced to the CNN so that the weights, biases and filters are tuned so that they can generalise to images it has not seen before. Firstly, in order to be image resolution augnostic, we resize the image down so that the smallest dimension is of size 256 in order to maintain the same aspect ratio as the original image. We then perform a centre crop of the image so that we extract a 224 x 224 image. 224 x 224 is the image resolution size used for VGG16 and ResNet50 so we will also use that same resolution. Also, to make this compatible for PyTorch, we convert the image into a floating-point tensor so that the colour channels are scaled to the range of `[0, 1]` and we also permute the shape of the image such that instead of the image being `H x W x C` with `H`, `W` and `C` being the height, width and channels, PyTorch processes images in `C x H x W` format, so this permutation of channels must be performed.

After, at random I apply rotation of up to 10 degrees as we don't expect dogs to have wildly different orientations other than portrait or landscape. Usually the photos are taken by a human who want the photo to look sensible. I also perform random horizontal flips, but not vertical as we don't expect the dogs to be upside down. I also resize the image so that the smaller dimension is 256 so that we keep the same aspect ratio, then used random resized crops of size 224 x 224 where the scale of the crop is between 0.96 to 1.0 as I don't want to cut out a lot of the dog in the augmentation. Doing this may make the breed more ambiguous to discern. I also change the aspect ratio a bit to try and capture nonlinearities in the lens. Finally, I normalise the image using the mean and standard deviation derived from ImageNet weights as a final post-processing step and also advised in the PyTorch docs.

The figure below illustrates a small sample of dog images that undergo the data augmentation step above. Take note that the mean and standard deviation normalised is undone to bring the image back into normal pixel space so we can visualise the images properly.



Implementation

Human Face Detector

Implementing this was quite straight forward. The cascade classifcation XML metafile was first obtained through OpenCV's Github repository [10] and using OpenCV, we determine whether we can find a face. OpenCV's `cv2.detectMultiScale` method once you construct a cascade classifier can be used to isolate out faces. Since we're only checking for the existence of the face, the exact localisation of the face doesn't matter. Therefore, we only need to check if the number of localisations is greater than 0. On a randomly selected 100 faces from the dataset, using the Haar Cascade detector resulted in a 99% accuracy, so this is convincing enough to proceed to the next step.

Dog Detector

We can use PyTorch's VGG16 pretrained model on the ImageNet database and simply determine out of the 1000 classes whether the most probable class is a dog. In fact, in the ImageNet database a dog corresponds to the range of indices between 151 and 268 inclusive. Therefore, as long as the most probable class is between this range of indices, we will know that a dog is visible in this image. However, before we can use the pretrained model, we must ensure that the image is resized to 224 x 224, then normalized using the mean and standard deviation derived from the ImageNet weights as VGG16 was trained on ImageNet. To make this more robust, we perform a resizing so that the smaller dimension becomes 256, then extract a centre crop of size 224 x 224 so that we don't completely distort the image when resizing it. After this pre-processing, we simply run it through the model and see whether the index is within the aforementioned ranges. On 100 randomly selected dog images from the dog dataset, we were able to achieve 99% accuracy, so this is convincing enough to proceed to the next step.

Dog Breed Classifier - From Scratch

Using PyTorch, I created an architecture that was relatively shallow to quickly assess whether the task is viable, but deep enough to use the right combination of layers to perform an adequate feature extraction. Specifically, I followed a guideline made by Andrej Karpathy in his blog post about different custom CNN architectures: https://cs231n.github.io/convolutional-networks/#architectures

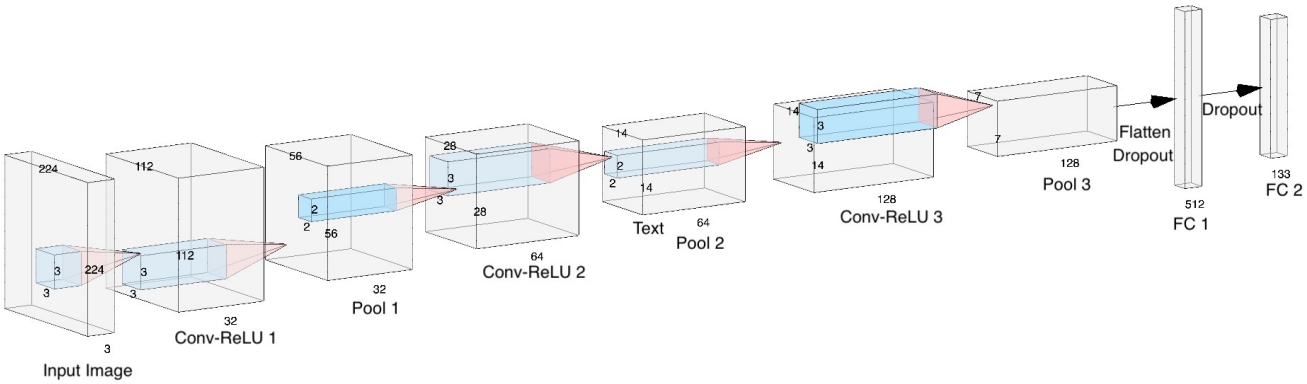
There was one architecture that spoke to me:

`INPUT -> [CONV -> RELU -> POOL]*2 -> FC -> RELU -> FC`

Note that `FC` means fully-connected / linear layers and `CONV` means 2D convolutional layers. I liked this architecture in particular because not only is it somewhat deep (but not as deep as say ResNet) for quick testing, but there are multiple feature extraction layers so we can develop more complex features for the given input volume. I also inserted dropout layers before every fully connected layer to minimise overfitting. I added in an extra combination of `[CONV -> RELU -> POOL]` to really try and make a difference, so the final architecture is:

`INPUT -> [CONV -> RELU -> POOL]*3 -> DROPOUT -> FC -> RELU -> DROPOUT -> FC`

A pictorial representation of this architecture can be found below:



In more detail:

- The input image is of size 224 x 224 x 3.
- Conv-ReLU 1: Convolutional 2D with kernel size 3 x 3, stride 2 x 2 and 32 filters. We employ padding so the height and width of the resulting output tensor is the same. We also apply the ReLU activation function to the output values of the tensor.
- Pool 1: Apply Max Pooling using a 2 x 2 window and a stride of 2 x 2
- Conv-ReLU 2: Convolutional 2D with kernel size 3 x 3, stride 2 x 2 and 64 filters. We employ padding so the height and width of the resulting output tensor is the same. We also apply the ReLU activation function to the output values of the tensor.
- Pool 2: Apply Max Pooling using a 2 x 2 window and a stride of 2 x 2
- Conv-ReLU 3: Convolutional 2D with kernel size 3 x 3, stride 2 x 2 and 128 filters. We employ padding so the height and width of the resulting output tensor is the same. We also apply the ReLU activation function to the output values of the tensor.
- Pool 3: Apply Max Pooling using a 2 x 2 window and a stride of 2 x 2
- Flatten: The output tensor after Pool 3 is flattened so that we obtain one vector.
- Dropout: We also apply dropout after the flattening operation
- FC 1: This is a fully-connected layer with 512 neurons. ReLU is also used after the output of this.

- Dropout: We apply dropout after FC 1.
- FC 2: This is a fully-connected layer with 133 neurons - this is the final output layer which corresponds to the confidence of each dog breed. Choosing the neuron that has the largest response corresponds to the dog breed the network is most confident in choosing.

Dog Breed Classifier - Transfer Learning

Using PyTorch, we access the computer vision helper utilities to create a ResNet50 that is pretrained on ImageNet weights. The modification we make here is to replace the last fully connected output layer so that instead of 1000 neurons, there are 133 neurons. All layers are frozen so that no updates are made when training and are used solely for feature extraction. The last layer is the only layer that will change with backpropagation so that the weights for this layer can be updated to adapt to the dog dataset.

Refinement

As discussed earlier, we start with an initial "from scratch" architecture which we hope to achieve a test accuracy that is better than random chance. Once we determine this is viable, we can refine our architecture by choosing a deeper network (i.e. ResNet50) and fine-tuning this network. We expect that this network performs better than the scratch version.

Model Evaluation and Validation

To evaluate the performance of our models, we will first have a look at the test accuracy, then when required we'll move to precision and recall. Take note that when we evaluate the system, we ensure that no augmentations are performed and only the resizing, centre cropping and normalisation is done. The other augmentations are not performed so that we have a base of reference when evaluating the performance of the model. Random augmentations at each epoch will not allow us to determine whether we are improving as the random augmentations may artificially produce a higher accuracy as they may not be as extreme as other epochs.

From scratch

For the scratch architecture, the hyperparameters I chose were using SGD with a learning rate of 0.01 and a momentum parameter of 0.9. The loss function we will use is the standard multi-class cross entropy loss. We train for 100 epochs and use a batch size of 64 images. The training and validation loss achieved after 100 epochs is 0.129851 and 8.977805. We also achieve an accuracy of 96% for the training set and 15% for the validation set. This indicates that the model can indeed recognise the dog breeds in the training set but fails to generalise to images that are outside of the training set, coming to the conclusion that the model is not sophisticated enough to generalise to unseen data. On top of this, if we had a good starting point (as in a pretrained network) we have a better chance at obtaining good performance. For closure, the test accuracy was 11% with a test loss of 3.959711. Therefore, we achieved our benchmark accuracy of 10% for the scratch architecture.

Transfer Learning

Using ResNet50 and undergoing the same hyperparameters as the previous section provides a training and validation accuracy of 98% and 89% respectively. The loss values were 0.061848 and 0.372436. We can see that the model can better generalise and especially with the deeper network with the pretrained weights. For

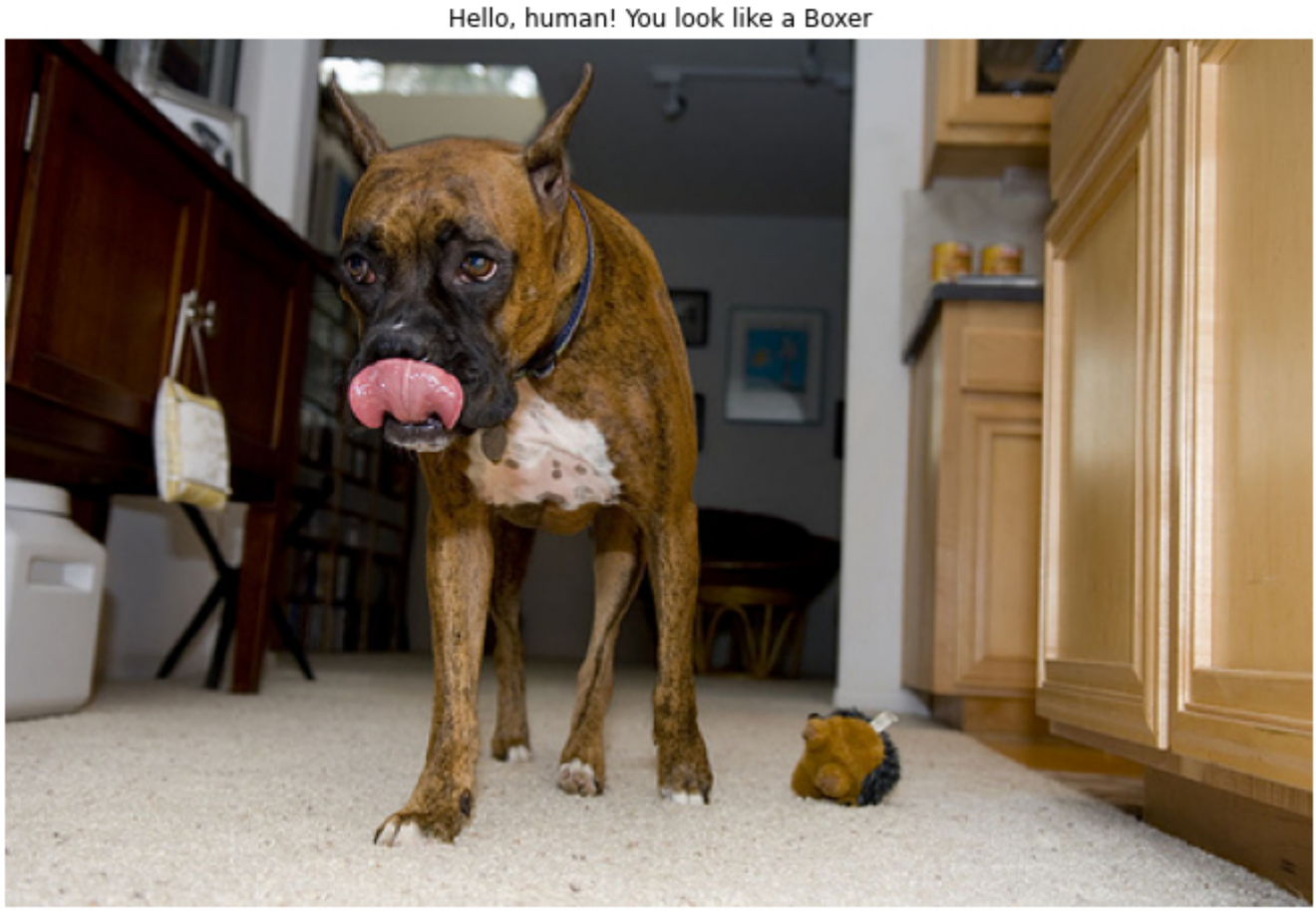
closure, the test accuracy was 87% with a loss of 0.552524. Therefore, we achieved our benchmark accuracy of 60% for the ResNet50 architecture.

Validating correctness

By randomly choosing a few human images and dog images, I used the pipeline discussed above when reacting to a detected human, a detected dog or neither. Some examples of this are shown below:

Hello, human! You look like a Silky terrier

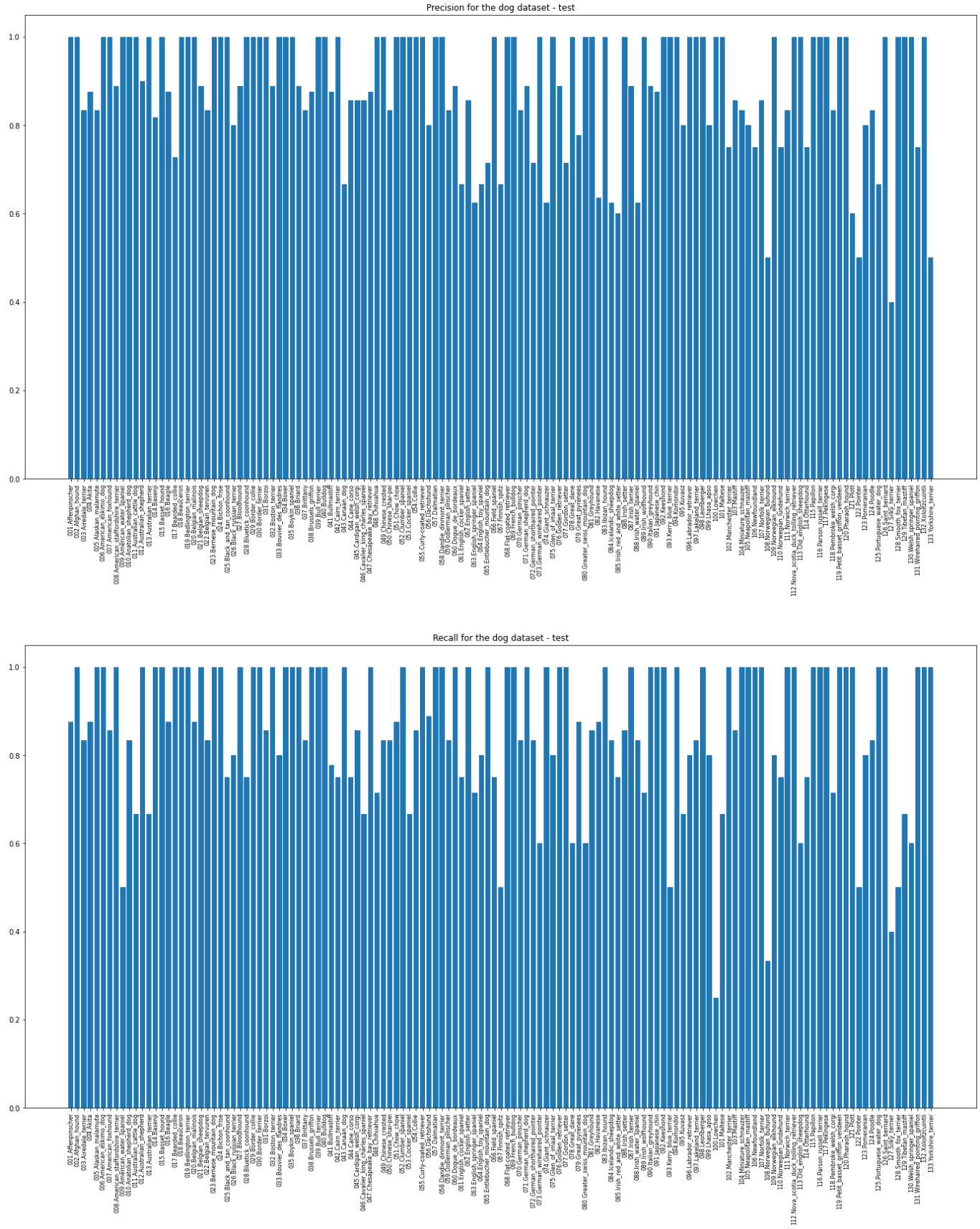
Hello, dog! You look like a Pomeranian



Cross-referencing the dog images, the predicted labels do match the ground truth ones. Amusingly, the human images do tend to look like the predicted dog breed too `:D`.

Justification

Here we will justify that the model is working to our expectations. To this end, we will compute the per-class precision and recall. Plots for both of these on the test dataset are shown below



At a cursory glance, the precision and recall for most of the classes are above the 60% mark, which is great for an application like this. On computing the average precision and recall over all classes results in 89% and 86% respectively. Having roughly the same magnitude as the test accuracy bodes very well despite the class imbalance. However, what is most interesting are the cases where the precision and recall are low. The class with the lowest precision corresponds to a Silky Terrier while the class with the lowest recall is the Lowchen. We examine some examples of when the system misclassified each of the Silky Terrier images below.

Hello, dog! You look like a Yorkshire terrier

Hello, dog! You look like a Yorkshire terrier

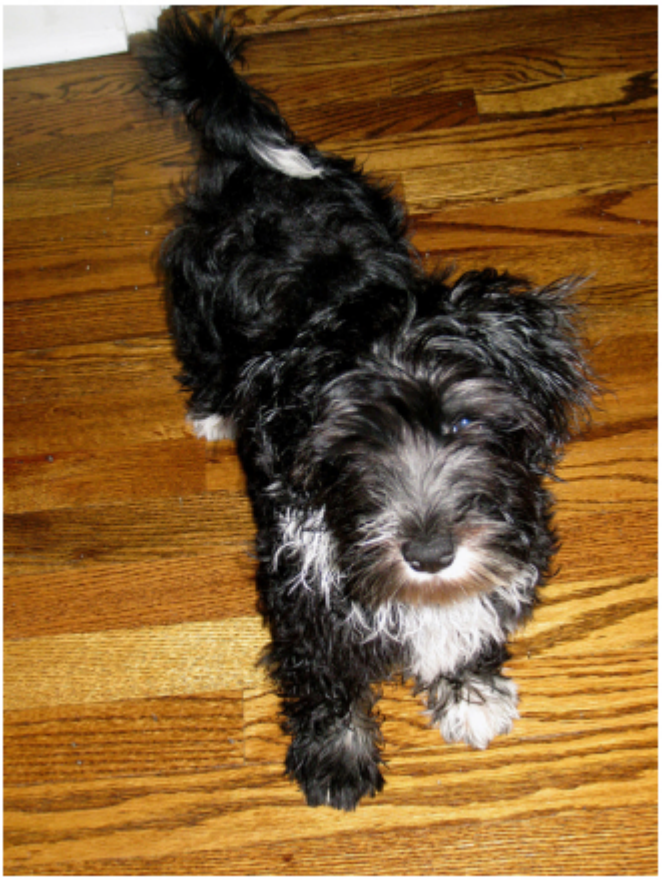
We can see that the system predicted the dogs as being a Yorkshire Terrier. Let's see what a correct classification looks like in the test dataset.

These are so similar to the Yorkshire Terrier that I'm even having trouble classifying them! I'm convinced that this is an edge case so this isn't something to worry about.

We examine some examples when the system misclassified Lowchen dogs from the test dataset shown below.

Hello, human! You look like a Lhasa apso

Hello, dog! You look like a Miniature schnauzer



I can certainly see the first one being confusing as it looks very similar to a Lhasa Apso, but the second one looks like it was mislabelled, so this is attributed to human error. Comparing with an actually correctly labelled

Lowchen shows that the similarities with a Lhasa Apso:

With this small exploration into the misclassifications, I am convinced that the performance is acceptable for public use. With the accuracy, precision and recall for the test dataset exceeding the expected 60%, this was most certainly a success.

Conclusion

The objectives for creating a system for processing dog breed images to ascertain its canine breed was successful. We explored the use of CNNs by training a "from scratch" architecture and transfer learning to achieve high accuracy despite dealing with a difficult dataset and with class imbalance. On top of this, we used a combination of human face and dog detectors to help us accomplish the task of providing a fun user experience for those who want to use a system for automatically determining dog breeds and for humans to see how close they resemble a particular dog breed. Future work could be established such that more deeper architectures could be used for transfer learning or perhaps bootstrapping our dataset with the one found on Kaggle. A further extension to this could be to report say the top 3 breeds that the system identified with their corresponding confidences to provide a more fun user experience.

References

1. Kaggle Competition - Dog Breed Classification
2. How to easily build a dog breed image classification model - James Le - Medium
3. Dog Breed Classification using CNNs - Deniz Doruk Nuhoglu - Towards Data Science
4. Dog Breed Classification - mc.ai
5. Dog Breed Classification using CNN and Transfer Learning - Li Liping - Gitconnected

6. PyTorch
7. Deep Residual Learning for Image Recognition - Kaiming He et al. - arXiv
8. Very Deep Convolutional Networks for Large Scale Image Recognition - Karen Simonyan and Andrew Zisserman
9. OpenCV - Cascade Classifier
10. ImageNet Database
11. OpenCV's Haar Cascade Classifier - XML File
12. Andrej Karpathy - Convolutional Neural Network Architectures