CNN Project: Dog Breed Classifier
AWS ML Capstone Project

Hafeez Olafisayo Jimoh

1 Definition

1.1 Problem Statement

There are currently about 180 dog breeds in the world and dogs remain one of the most common domestic animals often used as pets also. Each dog breed has specific characteristic and health conditions them. In order to provide appropriate training to live peacefully with humans for them and even treat them during any medical condition, it is important that we are able to correctly classify dog breeds. It is very unlikely that a veterinary doctor would also know all the dog breed types that exist in the world. The knowledge is usually limited to the ones present in the local environment of such doctor. And even in certain cases, people just want to buy a particular dog breed but cannot distinguish one from the other. Thus, we can rely on deep learning algorithm to help us in identification of dog breeds. The focus of this project is to train thousands of image on a deep learning algorithm and be able to feed into our algorithm and get the result of the specific dog breed. This approach can also be applied in numerous other fields where we are seeing automation for example we now have agricultural robots, drone surveillance in farm lands. All these devices need to be able to identify plants breeds and diseases and animal breeds that would be encountered in the farm land. In situations like this, it would be impossible to rely on humans to classify different scenarios. Leveraging on state of the art computer vision algorithm is a way to provide answer to this

1.2 Problem Overview

In this project, I would build a pipeline to process real-world, user-supplied images. Given an image of a dog, the 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.

1.3 Metrics

There are different metrics for evaluating our model performance even though the most common is accuracy. However depending on our objective, problem type and nature of the data. we could use other metrics like F1 score, precision and recall. According to sklearn documentation precision and recall is defined a:

Precision is the ability of the classifier not to label as positive a sample that is negative, and recall is the ability of the classifier to find all the positive samples."

$$Precision = \frac{TP}{TP + FP} \tag{1}$$

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

Precision measures a model exactness and precision measures its completeness. The F1 score is can be defined as a weighted harmonic mean of the precision and recall. the F1 score is particularly useful for unbalanced dataset. In a highly unbalanced dataset, having an accuracy of 95% or more is not merely enough. It could just be that our classifier is doing a majority voting thus in this scenario, accuracy would not be the best metric to measure performance.

2 Analysis

2.1 Data Exploration

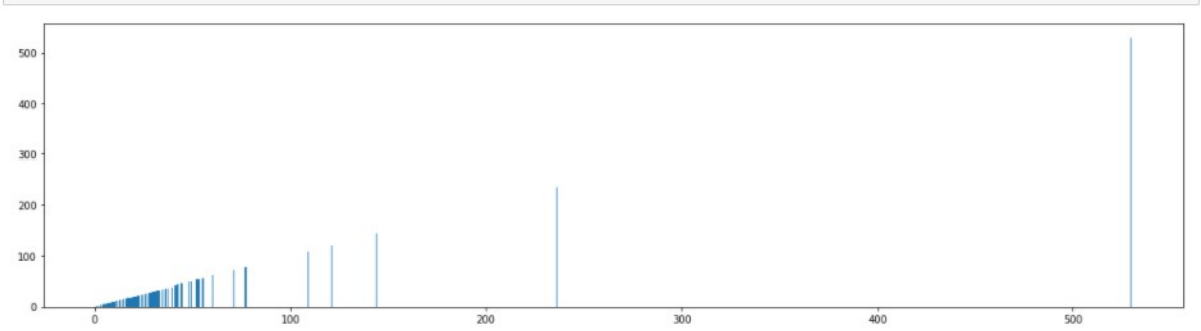

Figure 1: Human Frequency

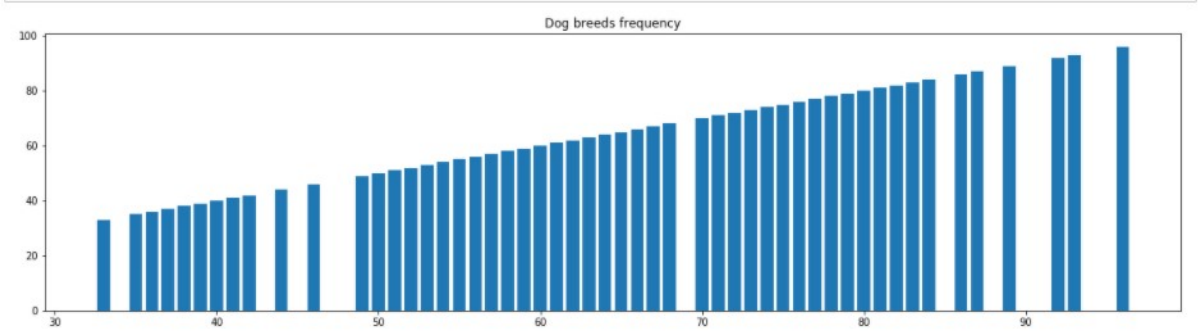

Figure 2: Dog Frequency plot

[h] All the data for humans and dogs were loaded as a data-frame. The following results summary were obtained

Figure 3: Dog Pictures

2.2 Exploratory Visualization

2.3 Algorithms and techniques

A small set of the images(100) was extracted from the main or whole data and analyzed. The human images were analyze to detect human faces in them and to identify human faces that are detected as dogs. Also, the dog images were analyze to identify how many dog images would be actually classified as dog and how many images would of dogs would be classified as humans.

An object detection method using Haar feature based classifier that was proposed by Paul Viola and Michael Jones in their paper, "Rapid Object Detection using a Boosted Cascade of Simple Features" in 2001. It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images.

Deep learning pre-trained algorithm was used on the short dog dataset. VGG16, AlexNet and ResNet were used on the short dog files. The short dog image sizes were resized to 224* 224, normalized and converted to tensor before applying the pre-trained models on them. The experiment was repeated for unnormalized images and the result obtained is shown below.

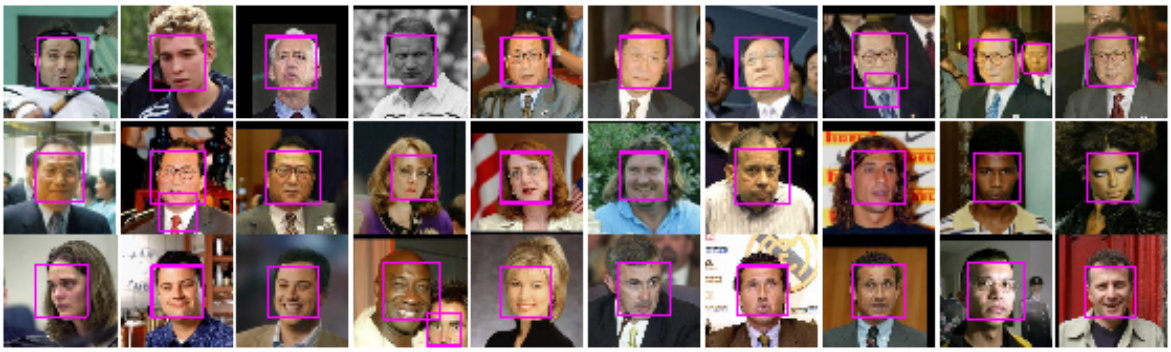

Figure 4: Human Pictures

2.4 Benchmark

The benchmark to assess the quality of this work against is the modified deep neural network in a work by Aydin and Sahand as described here. The reported accuracy from the paper on different modified state of the art models vary from 82 to 89.66%. State of the art models achieved the following for example: DenseNet-121:74.28% DenseNet-169:76.23% GoogleNet:72.11% ResNet-50: 73.28%. Thus, with data augmentation and fine tuning to be done in this work, it is expected that the accuracy that would be gotten would be more than 75% at least.

3 Questions and Answer

3.1 Question and Answer 1

- What percentage of the first 100 images in human files have a detected human face?
  Answer: percent of detected human face in 100 human images is 98.0%

- What percentage of the first 100 images in dog files have a detected human face?
  Answer: percent of detected human face in 100 dog images is 17.0%

3.2 Question and answer 2

Using VGG16:

- What percentage of the images in human files short have a detected dog?
  Answer: 1.0% of human images detected as dog

- What percentage of the images in dog files short have a detected dog?
  Answer: 100.0% of dog images detected as dog in normalized images

  96.0% of dog images detected as dog in un-normalized images.

Using AlexNet:

- What percentage of the images in human files short have a detected dog?
  Answer: 0.0% of human images detected as dog
- What percentage of the images in dog files short have a detected dog?
  Answer: 99.0% of dog images detected as dog in normalized images

Using ResNet:

- What percentage of the images in human files short have a detected dog?
  Answer: 0.0% of human images detected as dog
- What percentage of the images in dog files short have a detected dog?
  Answer: 0.0% of dog images detected as dog in normalized images

3.3 Question 3

Question 3: Describe your chosen procedure for preprocessing the data.

How does your code resize the images (by cropping, stretching, etc)? What size did you pick for the input tensor, and why? Did you decide to augment the dataset? If so, how (through translations, flips, rotations, etc)? If not, why not?
Answer: This question is answered in section 4.1.

3.4 Question 4

Outline the steps you took to get to your final CNN architecture and your reasoning at each step.
Answer: this is described in section4.3

3.5 Question 5

Outline the steps you took to get to your final CNN architecture and your reasoning at each step. Describe why you think the architecture is suitable for the current problem.
Answer: this is described in section 4.3

3.6 Question 6:

Answer: This is answered in section 5.2

4 Methodology

4.1 Data Pre-processing

This subsection also answers to Question 3 asked in the Dog Project notebook. This pre-processing step involves data augmentation and transformation. This process can be done manually for all the datasets. But we can make use of inbuilt transformation methods in PyTorch to achieve this purpose which is faster and efficient. Te following techniques were used:

- Random Horizontal Flip: This transformation procedure flips the image randomly with a specified probability.
- Random Resize Cropping: This method randomly crops the image to a specified size. In our case, the images were resized to 224*224. Most of our pre-trained models also have this size has their expected input.
- Colour Jittering: The contrast and brightness of the images were slightly improved with this method.
- Normalization: helps get data within a range and reduces the skewness which helps learn faster and better. It makes the pixel value of the image to be between -1 and 1 according to the formula below:

$$image = \frac{(image - mean)}{std} \tag{3}$$

Note that Random Horizontal Flip and Colour Jittering were not applied on the validation and test data.

4.2 Implementation

4.3 Training with CNN Model from scratch

Given the project requirement of achieving 10% accuracy, this is by far the toughest and most time consuming part of this project. I have implemented four different versions before I found network that could achieve >10% accuracy

4.3.1 Network Initialization

The initialization of deep networks has a big on the overall accuracy of our model [1]. Previously,weights are initialized by the model randomly and this often impacts the performance of the algorithm. Effective initialization techniques are important for training this type of DNN. There are many efficient techniques that have been proposed during last few years. Some examples of initialization strategies are uniform weight initialization, Xavier initialization. In 2010, Y. Bengio [2] proposed the Xavier initialization which is a simple but an effective approach for weight initialization. In Xavier initialization, the weights are scaled by the inverse of the square root of number of input neurons of the layer. I have used xavier initialization sub

4.3.2 Network Architectures

Version1

```
Net(
  (conv1): Conv2d(3, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
  (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
  (conv3): Conv2d(64, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
  (conv4): Conv2d(32, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (dropout): Dropout2d(p=0.5)
  (fc1): Linear(in_features=3136, out_features=64, bias=True)
```

```
  (fc2): Linear(in_features=64, out_features=133, bias=True)
  )
```

After training this CNN for bout 50 epochs, I stopped the training as the accuracy on test set did not exceed 1%.

Version2:

```
Net(
  (conv1): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (conv3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (conv4): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (conv5): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (fc1): Linear(in_features=25088, out_features=512, bias=True)
  (fc2): Linear(in_features=512, out_features=512, bias=True)
  (fc3): Linear(in_features=512, out_features=133, bias=True)
  (dropout): Dropout(p=0.3, inplace=False)
    )
```

Version3:

After several trials of the previous 2 of training and retraining, the accuracy didnt exceeed 1% on test set. Thus, I did not bother to try it on the validation set. I proceeded to draw my motivation of the architecture to use from a less complex state of the art algorithm. I implemented the architecture below which is very similar to AlexNet. Simple modification was done to the ALexNet found here in order for the model to address our exact dog breed identifier need.

```
Net(
  (net): Sequential(
    (0): Conv2d(3, 96, kernel_size=(11, 11), stride=(4, 4))
    (1): ReLU()
    (2): LocalResponseNorm(5, alpha=0.0001, beta=0.75, k=2)
    (3): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (4): Conv2d(96, 256, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (5): ReLU()
    (6): LocalResponseNorm(5, alpha=0.0001, beta=0.75, k=2)
    (7): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (8): Conv2d(256, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU()
    (10): Conv2d(384, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU()
    (12): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU()
    (14): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace)
    (1): Linear(in_features=6400, out_features=4096, bias=True)
    (2): ReLU()
    (3): Dropout(p=0.5, inplace)
    (4): Linear(in_features=4096, out_features=2048, bias=True)
```

```
    (5): ReLU()
    (6): Linear(in_features=2048, out_features=133, bias=True)
  )
)
```

4.3.3 Convolution layer

The architecture is made up of 5 convolution layers with kernels of size 11*11, 5*5 and 3*3. The conv layer is denoted by the Conv2d notation. It typically requires four hyper-parameters:

- Number of filters K,
- their spatial extent F,
- the stride S,
- the amount of zero padding P.

It accepts an input which is our image. In this case it is an image with 3 channels. For our image in this project, it accepts 224*224*3 denoted as a volume of size W1×H1×D1. Padding is a simple process of adding additional border of pixels to the edge of the image. It allows us to preserve and decide the size of the output volume we want. Sometimes it is most commonly as it would be used to exactly preserve the spatial size of the input volume so the input and output width and height are the same. The last parameter is the stride. It helps us determine how much the filter moves over the original image/input volume. When the stride is 1 then we move the filters one pixel at a time. When the stride is 2 (or uncommonly 3 or more, though this is rare in practice) then the filters jump 2 pixels at a time as we slide them around. The essence of this is that it will produce smaller output volumes spatially. A lot of learning usually takes places here as the layer determines its weight and biases by itself through back propagation. The output shape of each convolution layer given an input volume size (W), filter size of F, stride of S and padding size P on the border is given as:

$$W2 = \frac{(W - F + 2P)}{S + 1} \tag{4}$$

4.3.4 ReLU Activation Layer

Ac activation layer accepts an input and threshold it between a value. The relu is a computationally efficient activation that allows our network to converge very quickly. The relu is a Non-linear—although it looks like a linear function, ReLU has a derivative function and allows for back-propagation. The figure in Fig.5 shows how relu works. The architecture implemented has a total of 7 RelU activation layers each at the output of each convolution layer and fully connected layer.

$$A(x) = max(0, x) \tag{5}$$

4.3.5 Local Response Norm

LRN is a non-trainable layer that is no learning happens here. It is basically a form of normalization that square-normalizes the pixel values in a feature map in a within a local

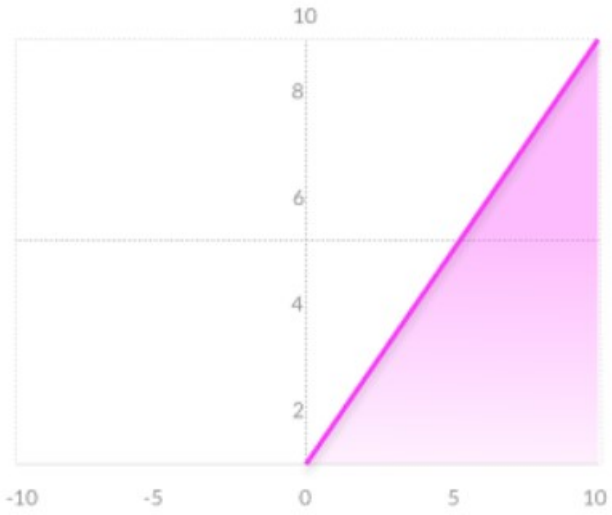

Figure 5: ReLU activation layer.Source: here

neighborhood. As described in the original work [3], the LRN is given as

$$b_c = a_c \left(k + \frac{\alpha}{n} \sum_{c'=max(0,c-n/2)}^{min(N-1,c+n/2)} a_{c'}^2 \right)^{-\beta} \tag{6}$$

Where the size is the number of neighbouring channels used for normalization, α is multiplicative factor, β an exponent and k an additive factor.

4.3.6 Pooling Layer

In convolutional neural nets, the input size decreases as we move deeper in to the network. It happens due to filter size and stride. Further reduction in dimensions can be achieved via pooling. In maximum pooling used in the architecture, it processes a region of size f x f and reduce it to single value: maximum value in that region. A total of 7 max pooling layers were used also each at the input of each convolution layer and fully connected layer.

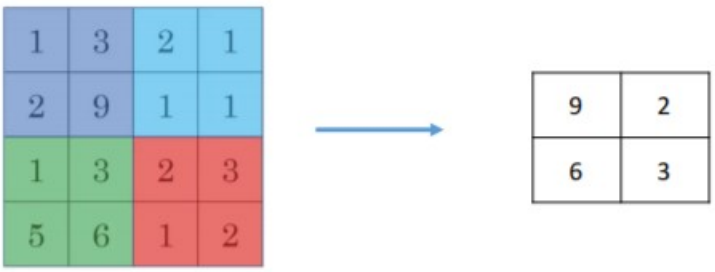

Figure 6: Maximum pooling

4.3.7 Drop Out Layer

The drop out layer is a very simple and intuitive way to reduce overfitting in our architecture [4]. The key idea in drop out layer as described in the original paper is to randomly drop units (along with their connections) from the neural network during training. This prevents units from co-adapting too much. At test time, the effect of approximating the averages of all the predictions of all these dropped or thinned networks are done by simply using a single unthinned network that has smaller weights. This significantly reduces overfitting and gives major improvements over other regularization methods [4].

4.3.8 Fully connected(FC) Linear layer

The fully connected layers are simply linear layers that we have in our artificial neural network. It is also referred to as a feedforward network or a perceptron. The learning decision for classification task is usually made at these layers. The previous convolution layers are majorly for feature extraction. The architecture makes use of three feedforward layers with a final output size of number of classes of our dog which is 133.

4.4 Transfer Learning approach

Transfer learning is the process of taking a pre-trained neural network and adapting the neural network to a new, different data set. In this project, we would make use of pre-trained state of the art networks on ImageNet dataset with 1000 classes. For this project, I would experiment with 3 different pretrained models and report their accuracy.

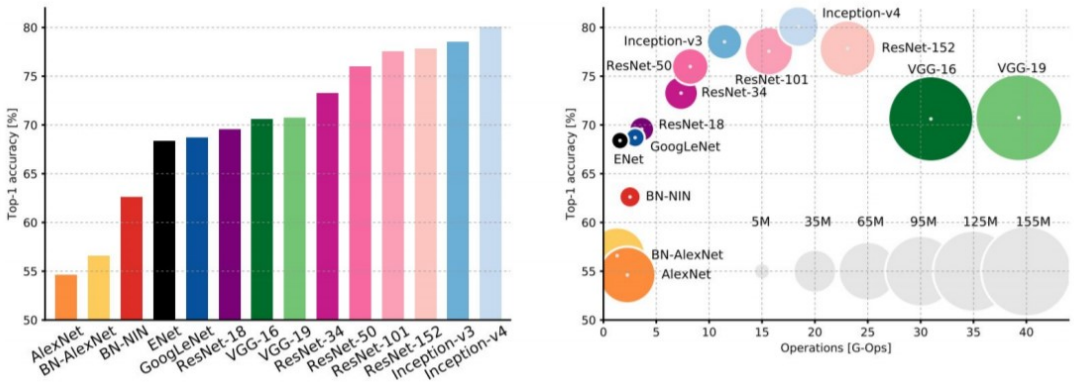

Figure 7: An Analysis of Deep Neural Network Models for Practical Applications, 2017
Source: CS231 lecture note: Fei-Fei Li et al Lecture 9 - 1 May 2, 2017

The above image shown in Fig. 7 compares different models accuracy and complexity(number of operations). Given the complexity and accuracy, I would experiment with VGG16, AlexNet and ResNet to compare how the results vary on our dog breed identifier project.

VGG16 network shown in Fig. 9 is a CNN architecture which was used to win the ImageNet Challenge 2014 submission, where the team used to secure the first and the second places in the localisation and classification tracks respectively. The 16 in the name suggests that it has 16 layers with weight. Most unique thing about VGG16 is that instead of having a large number of hyper-parameter they focused on having convolution layers of 3x3 filter with a stride 1 and always used same padding and maxpool layer of 2x2 filter of

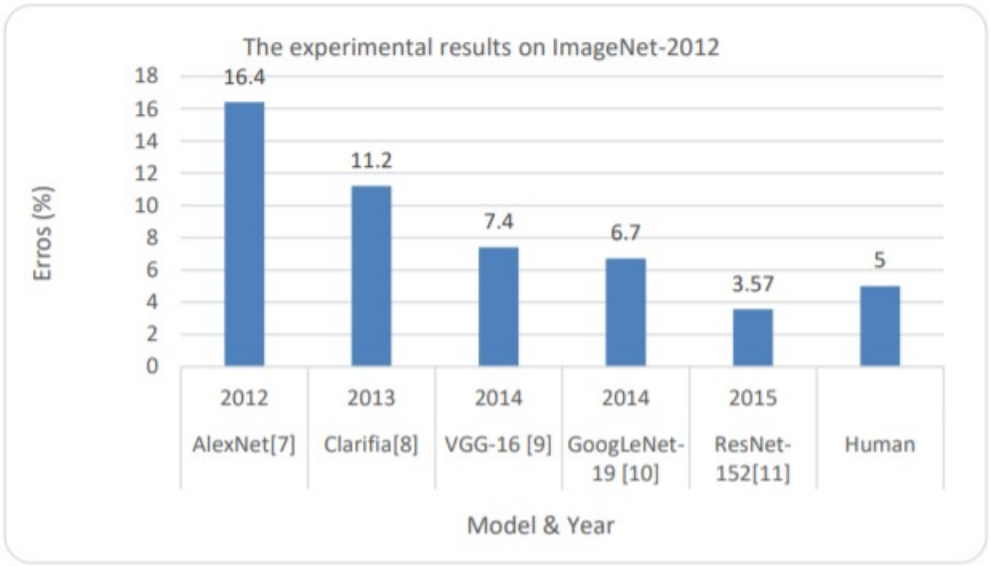

Figure 8: Baseline accuracy on ImageNet

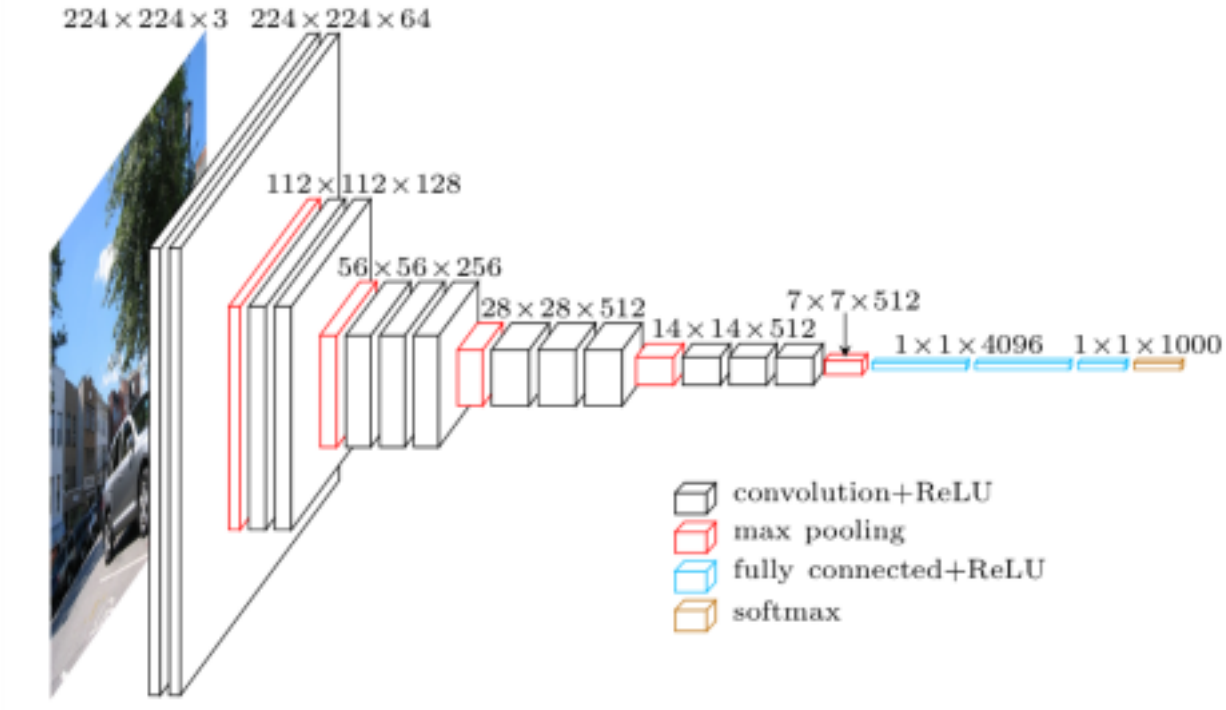

Figure 9: VGG16 network architecture

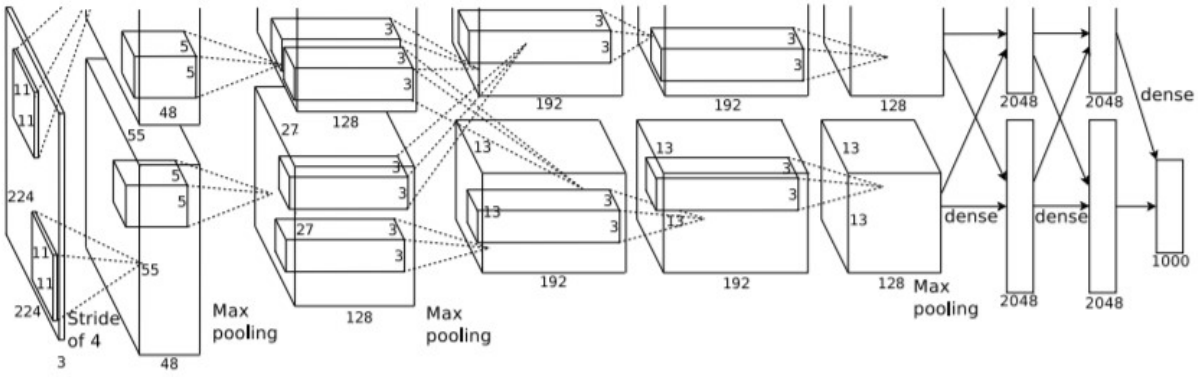

Figure 10: AlexNet Network architecture

stride 2. This network is a large with high number of operations as displayed in fig.7. It has about 138 million (approx) parameters.

As shown in Fig.7, we can see that the AlexNet has one of the least level of complexity and this is evident as shown in its architecture shown in Fig.10. AlexNet has a total of 5 convolutional layers with 3 convolution layers and 2 fully connected layers.

4.5 Other Parameters of the model used

4.6 Loss Function

The cost function which we want to minimize is calculated using the Cross Entropy Loss. There are several other loss functions but since this is a classification problem, I chose cross entropy loss. It is given by:

$$L = -(ylog(p) + (1-y)log(1-p)) \tag{7}$$

for a binary classification problem.

$$L = -\sum_{c=1}^{10} y_{o,c} log(p_{o,c}) \tag{8}$$

Cross-entropy loss, or log loss, measures the performance of a classification model whose output is a probability value between 0 and 1

4.7 Optimizer

The optimizer used in this task is the Adam optimizer. It is an stochastic optimizer which is an extension of stochastic gradient descent (SGD) presented at ICLR 2015 that is well suited for deep learning problems with little memory requirement. The parameter updates

for the Adam optimizer as defined here is:

$$r_t = (1 - \gamma_1) f'(\theta_t) + \gamma_1 r_{t-1} \tag{9}$$

$$p_t = (1 - \gamma_2) f'(\theta_t)^2 + \gamma_2 r_{t-1} \tag{10}$$

$$\tilde{r_t} = \frac{r_t}{(1 - \gamma_1)^t} \tag{11}$$

$$\tilde{p_t} = \frac{r_t}{(1 - \gamma_2)^t} \tag{12}$$

$$v_t = \alpha \frac{\tilde{r_t}}{\sqrt{\tilde{p_t}}} \tag{13}$$

$$\theta_{t+1} = \theta_t - v_t \tag{14}$$

4.8 Learning Rate

The learning rate determines the step size in the parameter update rule. It has been carefully chosen. It should not be a small value but not too small and should not be too high also. We want our loss to reduce over time and converge eventually to a minimum. Thus, in order to arrive at a global minimum, we need a learning rate that satisfies this requirement.

4.9 Refinement

Note that I have not trained all the models at once, I train for small epoch, save the model, load the model and resume training again. At the start of doing this, I was doing it in the same cell, this it clears the previous output, but subsequent ones were done in different cells using epochs of 20 and 30. I have experimented with some other models too like densenet and inception but the accuracy on validation test was too small for epochs of 20. So i did not bother to train for a longer time. This can be checked in the notebook here All trained models can be found here

	Number of epochs	Learning rate	Optimizer
VGG	50	0.001	Adam
RESNET	50	0.001	Adam
ALexNet	50	0.005	SGD
Model scratch	200	0.001	Adam

5 Results

Model	Valid. Accuracy
VGG	69%
RESNET	13%
AlexNet	53%
Model Scratch	20%

5.1 Justification

This model and results gotten in this satisfies the minimum requirement and predicts fairly the dog breed in an image. Although, with much more computation time int he GPU worksapace, it would be possible to experiment more and train for longer time and posssibly get higher accuracy in the process. Although, in the result in a research paper [5], accuracy between 72-87% were gotten for different models. And this work achieved 69 accuracy for VGG16 which is relatively fair

5.2 Improving the project

- Currently now, if an image has a dog and human in it, the model would predict only one of the objects in the image. The network can be improved by using object detection technique that identifies the individual classes and their location. Then, we can now identify for a particular boundary box, which cat breed is in it and which dig the human being detected resembles.
- Another improvement is to develop this into a web app.

- It is currently difficult to run experiment as it involves running all the juypter notebook cells one by one. This work can be improved by turning it into an executable python file that accepts the image to be classified and return the image as an output.
- We can also create an endpoint by running all this in a cloud environment. This makes deployment easy and updating our model.
- Finally, identifying the actual/right hyper-parameters to use to achieve high test accuracy. This is often an art than a science as it involves running different experiments to know which would give high accuracy. Meanwhile, there is a current time limitation to how long and how much experiments can be ran in the GPU workspace. The hyper-parameters value can also be gotten from papers, sadly a number of these reported values are not easily reproducible and repeatable in other environment.
- Use another metric like precision, recall and F1 score apart from the accuracy used.
- Also, when an image with an input channel of 1 ie a gray scale image is used for testing, the model can not be used. We can handle this case by having to convert our gray scale image to RGB coloured image and not modifying our transfer learning model.

References

[1] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *International conference on machine learning*, pp. 1139–1147, 2013.

[2] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, 2010.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[4] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[5] A. Ayanzadeh and S. Vahidnia, "Modified deep neural networks for dog breeds identification," *Preprints*, 2018.