Machine Learning Capstone Project

Convolutional Neural Networks: Dog Breed Classifier

Vindhya Avvari

Machine Learning Engineer Nano Degree Program

2020

DEFINITION

I. Project Overview

The convolutional neural network (CNN) is a class of **deep learning neural network**. CNNs represent a huge breakthrough in image recognition. They're most commonly used to analyze visual imagery and are frequently working behind the scenes in image classification. They can be found at the core of everything from Facebook's photo tagging to self-driving cars.

Image classification is the process of taking an **input** (like a picture) and outputting a **class** (like "dog" or a **probability** that the input is a particular class ("there's a 90% probability that this input is a). Computer Vision researchers have come up with a data-driven approach to design the algorithm to classify images into distinct categories. Instead of trying to specify what every one of the image categories of interest look like directly in code, they provide the computer with many examples of each image class and then develop learning algorithms that look at these examples and learn about the visual appearance of each class. In other words, they first accumulate a training dataset of labeled images, then feed it to the computer in order for it to get familiar with the data.

Given that fact, the complete image classification pipeline can be formalized as follows:

- Our input is a training dataset that consists of *N* images, each labeled with one of *K* different classes.

- Then, we use this training set to train a classifier to learn what every one of the classes looks like.

- In the end, we evaluate the quality of the classifier by asking it to predict labels for a new set of images that it has never seen before. We will then compare the true labels of these images to the ones predicted by the classifier.

Convolutional Neural Networks (CNNs) is the most popular neural network model being used for image classification problem. The big idea behind CNNs is that a local understanding of an image is good enough. The practical benefit is that having fewer parameters greatly improves the time it takes to learn as well as reduces the amount of data required to train the model. Instead of a fully connected network of weights from each pixel, a CNN has just enough weights to look at a small patch of the image. It's like reading a book by using a magnifying glass; eventually, you read the whole page, but you look at only a small patch of the page at any given time.

In this project the goal is to predict 133 different dog breeds. We were provided with labelled dog images as well as human images. The algorithm requires a dog image as input and outputs the prediction of what the breed is for the dog. And if a human image is given, the algorithm returns the dog name that most resembles to that human. The training of algorithm was done on GPU provided by the course which took significantly less time to train the model. The solution was achieved using two models- one that was built from scratch and the other used a transfer model. The predictions were compared and validated. It was a fun experience learning and implementing this project.

II. Problem Statement

To experiment, derive and develop an algorithm that could be used as part of a mobile or web app. In this project any user-supplied image acts as an input and is passed to the model. The trained model will then perform the logic to identify the image whether it is dog's image or human's image. If a dog is detected in the image, it will provide an estimate of the dog's breed. If a human is detected, it will provide an estimate of the dog breed that is most resembling.

In implementing the project different image classification models were analyzed, important decisions need to be made like how the architecture of your model is going to be for predicting results of high accuracy. All in all we build 2 models using CNN and compare the predicted results.

III. Metrics

The problem we try to solve is a classification problem. The performance is measured by using the test set accuracy score. After training is completed, I have tested it by loading couple of input images which the model has never seen, never been trained on. Hence, validating the results by comparing it to the actual breed name would give an idea of its performance. Also, looking at the accuracy score percentage clearly speaks how accurate the model is about to perform.

The accuracy is calculated as,
Accuracy = Correct predictions / Total predictions, in other words-
Accuracy = (true positives + true negatives) / Total dataset size

I've chosen Accuracy score as the metric for this image problem because it gives a sense of how often the model can make a correct prediction which is an easy & clear metric especially in the image classification spectrum. Also, in this project we were trying to optimize as well as compare two models on how well each behave/perform prediction so, I felt Accuracy score is the best bet.

In my transfer learning algorithm, the test accuracy attained was 84%.

ANALYSIS

IV. Data Exploration

Here are some statistics regarding the data that was provided for this project

```
There are 13233 total human images.
There are 8351 total dog images.
And..
There are 6680 total dog images in train dataset
There are 835 total dog images in valid dataset.
There are 836 total dog images in test dataset.
The total number of classes are 133
```

V. Exploratory Visualization

The one notable observation is that images are all of different sizes and are all colored images. You can take a glimpse of 20 images with their classification in the labels for each image Please refer to Figure 1.

Figure 2 shows a bar chart with the counts of images for each classification across the training set.

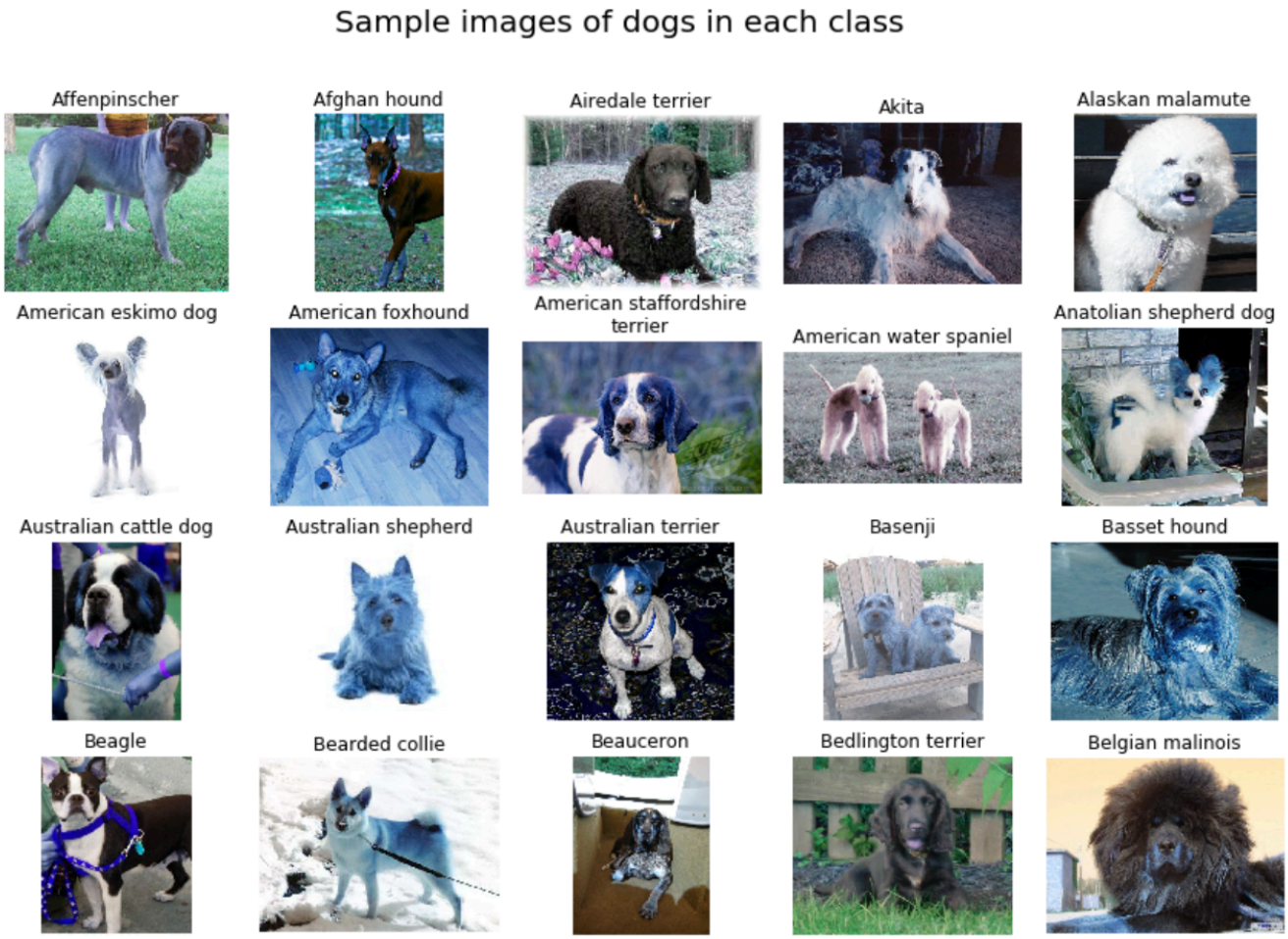


Figure 1.

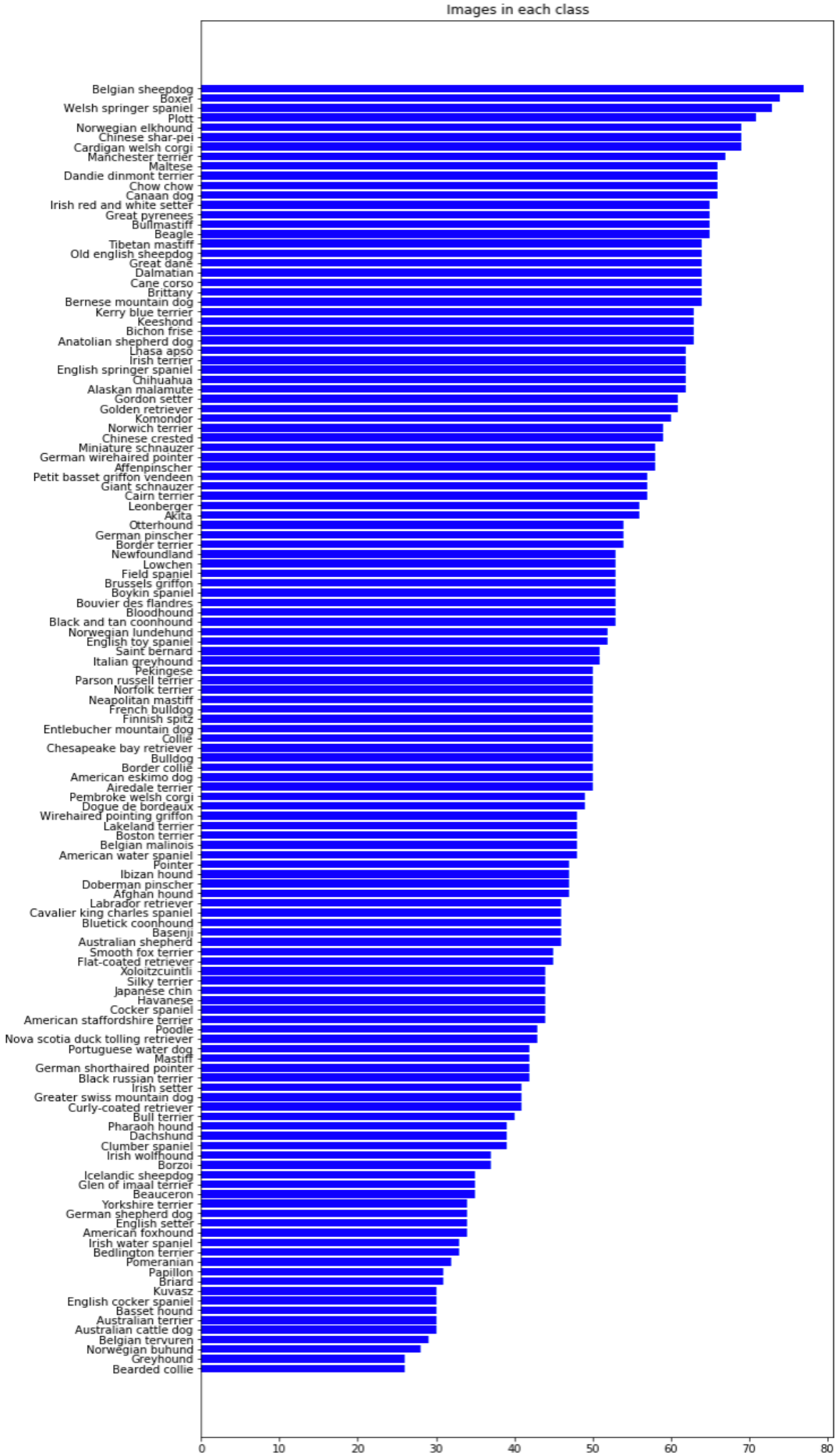

Images in each class
Belgian sheepdog
Boxer
Welsh springer spaniel
Plott
Norwegian elkhound
Chinese shar-pei
Cardigan welsh corgi
Manchester terrier
Maltese
Dandie dinmont terrier
Chow chow
Canaan dog
Irish red and white setter
Great pyrenees
Bullmastiff
Beagle
Tibetan mastiff
Old english sheepdog
Great dane
Dalmatian
Cane corso
Brittany
Bernese mountain dog
Kerry blue terrier
Keeshond
Bichon frise
Anatolian shepherd dog
Lhasa apso
Irish terrier
English springer spaniel
Chihuahua
Alaskan malamute
Gordon setter
Golden retriever
Komondor
Norwich terrier
Chinese crested
Miniature schnauzer
German wirehaired pointer
Affenpinscher
Petit basset griffon vendeen
Giant schnauzer
Cairn terrier
Leonberger
Akita
Otterhound
German pinscher
Border terrier
Newfoundland
Lowchen
Field spaniel
Brussels griffon
Boykin spaniel
Bouvier des flandres
Bloodhound
Black and tan coonhound
Norwegian lundehund
English toy spaniel
Saint bernard
Italian greyhound
Pekingese
Parson russell terrier
Norfolk terrier
Neapolitan mastiff
French bulldog
Finnish spitz
Entlebucher mountain dog
Collie
Chesapeake bay retriever
Bulldog
Border collie
American eskimo dog
Airedale terrier
Pembroke welsh corgi
Dogue de bordeaux
Wirehaired pointing griffon
Lakeland terrier
Boston terrier
Belgian malinois
American water spaniel
Pointer
Ibizan hound
Doberman pinscher
Afghan hound
Labrador retriever
Cavalier king charles spaniel
Bluetick coonhound
Basenji
Australian shepherd
Smooth fox terrier
Flat-coated retriever
Xoloitzcuintli
Silky terrier
Japanese chin
Havanese
Cocker spaniel
American staffordshire terrier
Poodle
Nova scotia duck tolling retriever
Portuguese water dog
Mastiff
German shorthaired pointer
Black russian terrier
Irish setter
Greater swiss mountain dog
Curly-coated retriever
Bull terrier
Pharaoh hound
Dachshund
Clumber spaniel
Irish wolfhound
Borzoi
Icelandic sheepdog
Glen of imaal terrier
Beauceron
Yorkshire terrier
German shepherd dog
English setter
American foxhound
Irish water spaniel
Bedlington terrier
Pomeranian
Papillon
Briard
Kuvasz
English cocker spaniel
Basset hound
Australian terrier
Australian cattle dog
Belgian tervuren
Norwegian buhund
Greyhound
Bearded collie
0
10
20
30
40
50
60
70
80

Figure 2.

VI. Algorithms and Techniques

In this project we have covered quite familiar and useful algorithms and techniques. For identifying the human faces, we have used the OpenCV's implementation of the Haar feature based cascade classifiers. The images, as a standard step are first converted into grayscale to be able to detect the faces easily. It is also equipped with the feature of drawing a bounding box around the detected faces. This image is then converted back to color image along with the bounding box. The performance of this function was also tested, it works really good on humans but a slight error in reading dog faces as humans. It was able to detect 98.0% of the first 100 images in human_files as human faces and 17.0% of the first 100 images in dog_files were detected as human face.

As a next step a pre-trained VGG-16 model was implemented to detect the dogs and here is how it performed.(It was perfect!)

The percentage of the detected dogs in human_files_short: 0% The percentage of the detected dogs in dog_files_short: 100%

The main part of the project was to create CNN in 2 ways- from scratch as well as using transfer learning.

1) **Building from Scratch model** : In this neural network, the input image was passed as a 3D array (224x224x3) with 3 representing the 3 color channels and 224x224 being the image dimensions. The CNN architecture comprises of different layers – Convolution, Rectified Linear Unit (ReLU) and pooling. **Convolution** layer extracts the features from an input image by preserving the relationship between pixels by learning image features using small squares of input data. It is a mathematical operation that takes two inputs such as image matrix and a filter or kernel. I have used a 3x3 filter for all my three convolution layers. The **activation function** (ReLu) is applied to each convolution layer to introduce non-linearity to a system. It was found that doing so would make the network to train lot faster(Eg. Make all -ve values to zero). Now this transformed output is then sent to the next layer- **pooling layer**. Here I've used 3 max pooling blocks immediately after each convolutional layer to preserve the most present features in the transformed image output.

At the end of a CNN, you implement fully connected layer which gives you the desired output. In our algorithm the fully connected layer outputs 133 features corresponding to 133 breed classes. Apart from the methods mentioned above I've also made use of dropout regularization (30%) between the layers in order to prevent overfitting. While setting the learning rate to 0.05, no. of epochs to 25 and giving batch size as 32, and iteratively training the model over this neural network gave the accuracy of 13%. However, this model has been trained only on the 6680 training images to predict 133 breeds of dogs. Had it been trained on even larger set, it may have been able to increase its accuracy score by adjusting some of the other hyperparameters.

2)**Transfer learning model** : Transfer learning is a method for reusing a model trained on a related predictive modeling problem. It can be used to accelerate the training of neural networks.. Using a pre-trained model which has prior knowledge not only saves time, computing resources, speeds up the training process but also will have higher accuracy in predicting. This is what transfer learning exactly does. It is the improvement of learning in a new task through transfer of knowledge from a task that it has already learnt.

In the project that I submitted, I've used the pre-trained residual learning framework : ResNet-152 model to create a convolutional neural network based on transfer learning. On the ImageNet dataset the neural network evaluates residual nets with a depth of up to 152 layers---8x deeper than VGG nets. To make use of this model, we replace the fully connected layer of ResNet-152 with the 133 features set to use its knowledge in predicting the 133 classifications. And it has satisfied results with accuracy of 84% which is far better prediction score than the model that was built from scratch.

VII. Benchmark

For our benchmark model, we will use the Convolutional Neural Networks (CNN) model created from scratch with an accuracy of more than 10%. This should be enough to confirm that our model is working because random guess would be 1 in 133 breeds which are less than 1. So, I will be comparing the 1st model (built from scratch) to validate the performance against the optimized model built using transfer learning.

METHODOLOGY

VIII. Data Preprocessing

The dogs dataset has been divided into 3 separate folders for training, validation and testing. And each dataset also has 133 folders for each of the dog classification it has.

These set of augmentation methods have been applied to each data set as part of the preprocessing.

A. RandomResizedCrop

B. RandomHorizontalFlip

C. CenterCrop

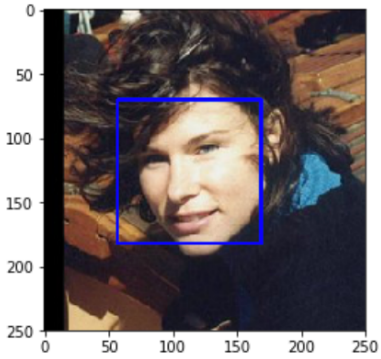
D. Normalize the images using mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])

IX. Implementation

Step 1— Human Face Detection:

This is step is used to identify human face in image. The project uses Opencv's haarcascade xml for detecting face. The performance of this function was also tested, it works really good on humans but a slight error in reading dog faces as humans.

```
Number of faces detected: 1
```



Step 2—Dog Face Detection:

We use the pre trained VGG-16 model to return true or false depending on whether dog was detected in the image or not.

Output:

The percentage of the detected dogs in human_files_short: 0% The percentage of the detected dogs in dog_files_short: 100%

Step 3 — Creating CNN from Scratch:

Designing a CNN architecture from scratch. I stacked up 3 convolutional layers after input layer and adding max pooling layer in between convolution layers. And then added dropout methods in between.

The challenges during this phase were not being able to get accuracy above 10% in the initial trials. Also, when increasing the number of epochs or layers in the CNN or reducing the learning rate, the algorithm has taken a lot of time to train each time.

```
Net(
  (conv1): Conv2d(3, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
  (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
  (conv3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=
False)
  (fc1): Linear(in_features=6272, out_features=500, bias=True)
  (fc2): Linear(in_features=500, out_features=133, bias=True)
  (dropout): Dropout(p=0.3)
)
```

Briefly this is how my pipeline looks like & works:

My choice of CNN architecture consists of 3 Convolutional layers and 2 Fully Connected layers. ReLU activation function was applied to every layer except the last one. An input image of size 224x224x3 is passed and output of 133 is obtained.

1st Layer: Consists of Conv2d with 3 Input Channels and 32 Output Channels and a Kernel size of 3x3 and stride 2x2 with padding = 1 It is followed by ReLU and Maxpool2d.

After this step the image is downsized.

2nd Layer: Consists of Conv2d with 32 Input Channels and 64 Output Channels and a Kernel size of 3x3 and stride 2x2 with padding = 1. It is followed by ReLU and Maxpool2d.

After this step the image is downsized.

3rd Layer: Consists of Conv2d with 64 Input Channels and 128 Output Channels and a Kernel size of 3x3 and stride 1x1 with padding = 1.

This layer will not reduce input image dimensions. Now, the process is followed by ReLU and Maxpool2d which downsizes the image to 128.

4th Layer: Applied drop out of 30%

5th Layer: Consists of fully connected layer with 6272 Input Features and 500 Output Features.

6th layer: Applied drop out of 30% to avoid over fitting.

7th Layer: Consists of Linear with 500 Input Features and 133 Output Features. The desired amount of output features.

Step 4 — Transfer learning using Pre-trained ResNet-152:

We use a pre trained model ResNet152 to predict the dog breeds. With this algorithm the accuracy has exceeded my expectations giving accuracy rate up to 84% .

X. Refinement

In the pre-trained model we have frozen the parameters so that back propagation is eliminated. Replaced the last fully connected layer to have only 133 features matching the desired 133 dog breeds that are in the train set. The model has been trained for 25 epochs after trying different set of runs iteratively. The learning rate is set to 0.05 which gave. The best results after different trials. The loss function is set to Cross Entropy Loss.

Initial Epochs--→

```
Epoch 1, Batch 1 loss: 4.888731
Epoch 1, Batch 101 loss: 4.884851
Epoch 1, Batch 201 loss: 4.876753
Epoch: 1      Training Loss: 4.876593      Validation Loss: 4.854926
Validation loss decreased (inf --> 4.854926).  Saving model ...
Epoch 2, Batch 1 loss: 4.804243
Epoch 2, Batch 101 loss: 4.830817
Epoch 2, Batch 201 loss: 4.810738
Epoch: 2      Training Loss: 4.807353      Validation Loss: 4.721612
Validation loss decreased (4.854926 --> 4.721612).  Saving model ...
Epoch 3, Batch 1 loss: 4.763125
Epoch 3, Batch 101 loss: 4.707907
Epoch 3, Batch 201 loss: 4.682672
Epoch: 3      Training Loss: 4.680340      Validation Loss: 4.564233
Validation loss decreased (4.721612 --> 4.564233).  Saving model ...
Epoch 4, Batch 1 loss: 4.732466
Epoch 4, Batch 101 loss: 4.597262
Epoch 4, Batch 201 loss: 4.584637
Epoch: 4      Training Loss: 4.584277      Validation Loss: 4.451067
Validation loss decreased (4.564233 --> 4.451067).  Saving model ...
Epoch 5, Batch 1 loss: 4.635141
Epoch 5, Batch 101 loss: 4.534521
Epoch 5, Batch 201 loss: 4.535649
Epoch: 5      Training Loss: 4.535378      Validation Loss: 4.447823

.
. .
```

```
. . .
. . .
```

Final Epochs→

```
.
. .
. . .
. .
.
Epoch 20, Batch 1 loss: 3.571919
Epoch 20, Batch 101 loss: 3.900408
Epoch 20, Batch 201 loss: 3.892611
Epoch: 20      Training Loss: 3.897042      Validation Loss: 3.780419
Epoch 21, Batch 1 loss: 3.684646
Epoch 21, Batch 101 loss: 3.864353
Epoch 21, Batch 201 loss: 3.882832
Epoch: 21      Training Loss: 3.876737      Validation Loss: 3.716901
Validation loss decreased (3.764051 --> 3.716901).  Saving model ...
Epoch 22, Batch 1 loss: 3.403298
Epoch: 22      Training Loss: 3.826715      Validation Loss: 3.907371
Epoch: 23      Training Loss: 3.805639      Validation Loss: 3.698256
Validation loss decreased (3.716901 --> 3.698256).  Saving model ...
Epoch: 24      Training Loss: 3.768080      Validation Loss: 3.601655
Validation loss decreased (3.698256 --> 3.601655).  Saving model ...
Epoch: 25      Training Loss: 3.707456      Validation Loss: 3.646589
```

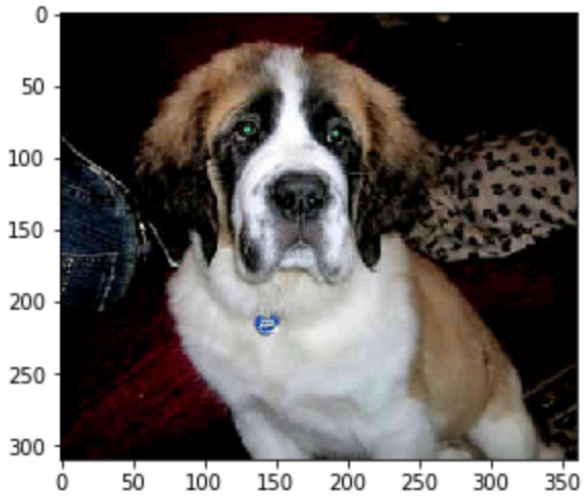
RESULTS

XI. Model Evaluation and Validation

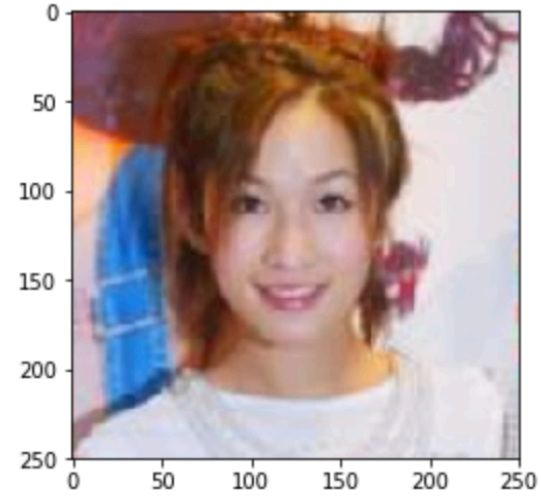
The accuracy that the model has achieved is 84%.

In the evaluation of the algorithm- If an image is given 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. If neither of them are provided then, it will say it is unable to predict or produce an error.

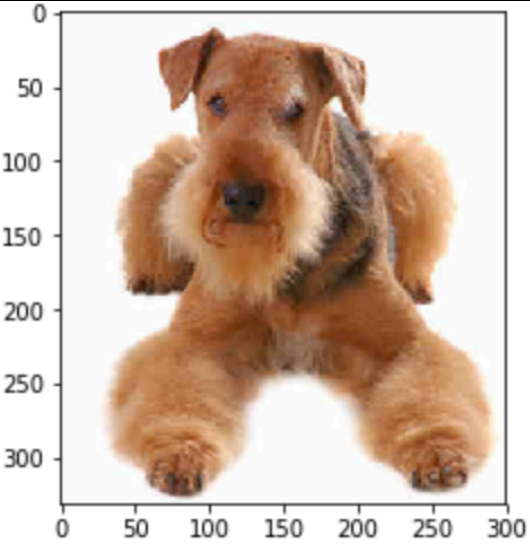
The test results are below from couple of runs using the trained final model-



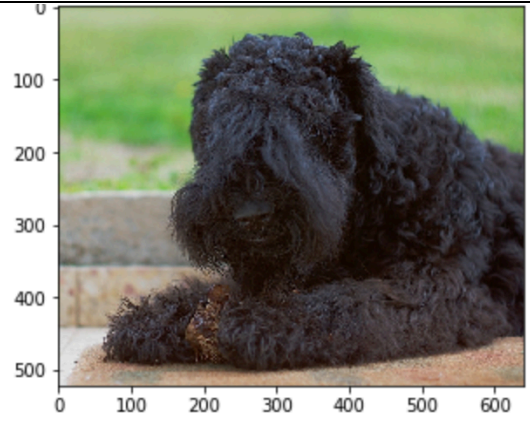
```
Hello, dog!
Your breed is Saint bernard
```



```
Hello, human!
 ..You look like a Silky terrier
```



```
Hello, dog!
Your breed is Airedale terrier
```



```
Hello, dog!
Your breed is Kerry blue terrier
```

I then validated the results against the actual dog images provided in the data set of the training model and found them to be accurate after carefully analyzing the label of each image.

XII. Justification

When compared the accuracy scores of both customized CNN model created from scratch by me and the CNN model created using the Transfer learning, the accuracy rates differ by a large percentage. The CNN model created using the transfer learning performed the best with 84% accuracy while the scratch model was giving 13% accuracy. Hence, it is justifiable to say that the later CNN model has solved the problem of identifying/predicting the breed classification of dog's way more accurately than the first model.

References

1. https://towardsdatascience.com/wtf-is-image-classification-8e78a8235acb
2. https://en.wikipedia.org/wiki/Convolutional_neural_network
3. https://www.kaggle.com/c/dog-breed-identification/overview/description
4. https://github.com/udacity/dog-project
5. https://www.kdnuggets.com/2018/04/right-metric-evaluating-machine-learning-models-1.html
6. https://www.analyticsvidhya.com/blog/2020/02/learn-image-classification-cnn-convolutional-neural-networks-3-datasets/
7. https://becominghuman.ai/understanding-the-basics-of-cnn-with-image-classification-7f3a9ddea8f9
8. https://towardsdatascience.com/the-4-convolutional-neural-network-models-that-can-classify-your-fashion-images-9fe7f3e5399d
9. https://towardsdatascience.com/wtf-is-image-classification-8e78a8235acb
10. https://pytorch.org/
11. https://scikit-learn.org/stable/modules/classes.html