Image Classifier: Dog Breed Classification

Introduction

Image classification is a powerful tool used by companies in the fields of marketing, social media, healthcare, security, and many more. Social media companies use algorithms that can identify specific individuals within a picture and recommend that picture to the person in it, for example. Tech companies trying to develop self-driving vehicles heavily rely on image classification for their vehicles to recognize pedestrians, traffic signals, signs, etc. Security and law enforcement departments have advanced facial recognition algorithms that can find people of interest using CCTV cameras. These algorithms are also being used to help doctors identify cancer in patients before doctors would have normally been able to provide a diagnosis.

In this project, I will create an image classifying algorithm that will determine the breed of a dog based solely on an image. The dataset is split into a train and test file with 120 dog breeds listed. The training set contains the images and breeds of the dogs in the image. A deep learning neural network will be "taught" to recognize the dog breeds then predict which breed pertains to the images in the test dataset. Neural Networks become more precise the more data it is provided in the training phase. Since the training dataset is limited and the number of examples per breed are even fewer, the challenge will be in producing an algorithm with a high accuracy rate of prediction on the dataset provided.

This analysis will use Python and TensorFlow to create the deep learning algorithm. To build the algorithm I will first pre-process the images. This is the preparation phase for the images to be used in the neural network. Then, if necessary, the images will be reshaped for analysis. This is followed by the creation of a convolutional layer and a pooling layer. These steps are repeated to provide additional layers. Next an activation function will be applied to the layers. This is followed by a layer that will predict the classification of the image. This is the general approach which will be used, however, unforeseen issues with the data may require additional measures to improve the accuracy of the prediction.

Once the algorithm is complete, a full report with the code will be provided. Along with this, a power point presentation will be available with highlights of the findings and delivery of the results.

The Dataset

The dataset used to create this neural network is Kaggle's Dog Breed Identification dataset. Kaggle compiled thousands of images and split them into the train and test dataset. Each image is given a unique ID. An additional spreadsheet is provided by Kaggle that matches the image ID to the breed of the dog in the image of the training set. The training set has 10,222 images and 120 dog breeds.

Exploratory Data Analysis

Exploratory data analysis, EDA, is one of the initial analyses that are performed on a dataset when searching for main trends, summarizing the data's characteristics, and using visual methods to get a better understanding of the data. John Tukey defined EDA as "Procedures for analyzing data, techniques for interpreting the results of such procedures, ways of planning the gathering of data to make its analysis easier, more precise or more accurate, and all the machinery and results of (mathematical) statistics which apply to analyzing data."

Data visualization tools were used to illustrate characteristics of the data. The histogram in Figure 1 shows the distribution of the breeds. There are about 70 to 120 images per dog breed. This results in a dataset that has a relatively uniform distribution. There are, however, very few images per dog breed. This will make it difficult to train a convoluted neural network that provides significant results.

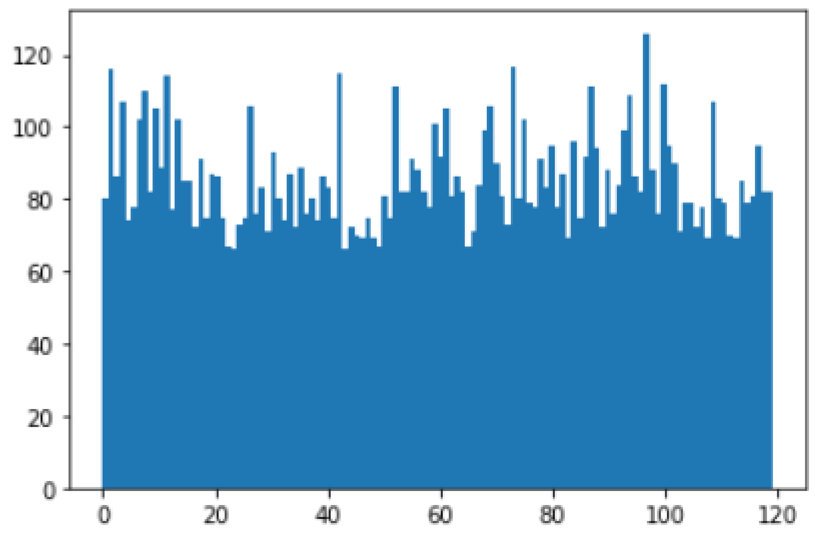


Fig 1. Histogram of Dog Breed Images

To see the sizes of the images a scatterplot of the number of pixels per image was created. Figure 2 shows the pixels per image.

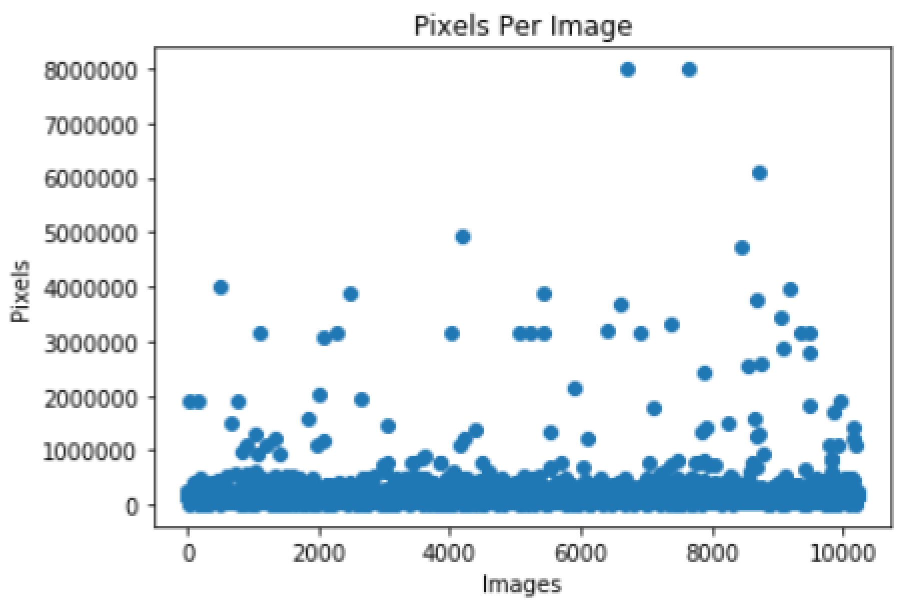


Fig 2. Pixels Per Image Scatterplot

The scatter plot shows that most of the images have less than 1 million pixels with the average being 184,176.35 pixels. However, there are many outliers. The minimum number of pixels per image is 12,240 and the maximum is 7,990,272. The mean is 184,176.35 with a median of 183,875.

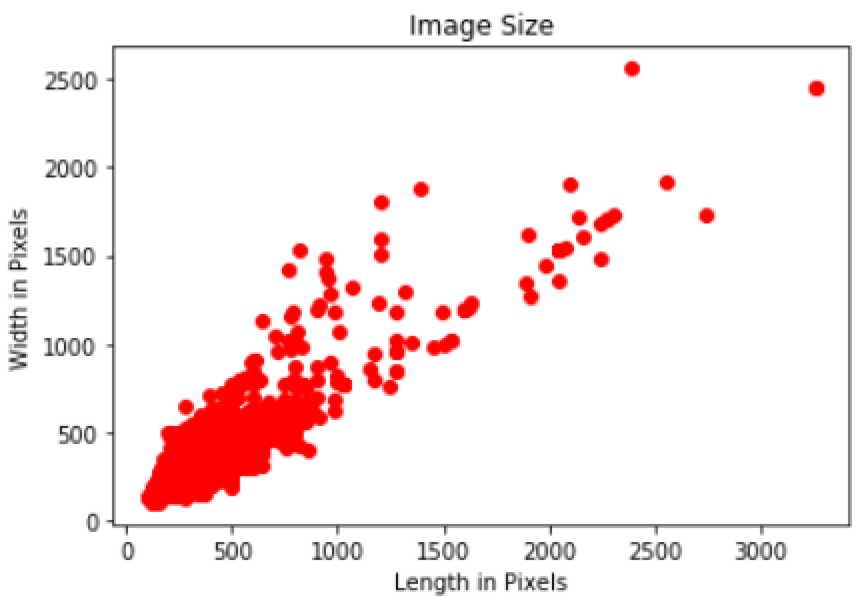


Fig 3. Image Size Distribution

The plot in Figure 3 illustrates the distribution of the image sizes. Most of the images seem to be around 500x500 pixels. To make the analysis of the images less computationally demanding, the images need to be resized. This, however, needs to be done on a trial and error basis to make sure the image integrity is upheld at lower sizes.

```
image shape: (2448, 3264)
number of pixels: 7990272
```

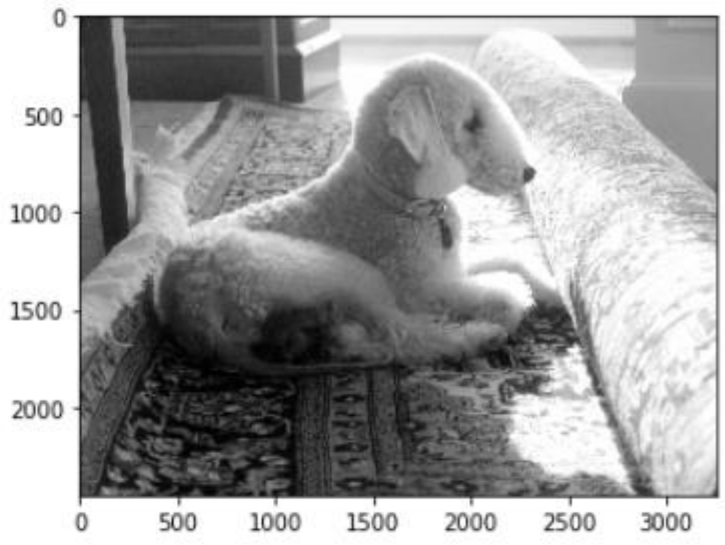


Fig 4. Largest Image at Original Size

```
image shape: (200, 200)
number of pixels: 40000
```

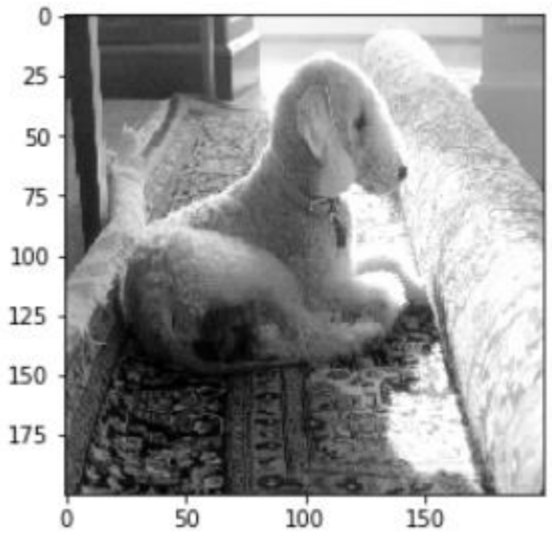


Fig 5. Largest Image Resized

After trial and error, a 200x200 pixel image maintained the integrity of the image without too much distortion of both the largest image in figure 4 and the smallest in figure 6.

```
image shape: (134, 97)
number of pixels: 12998
```

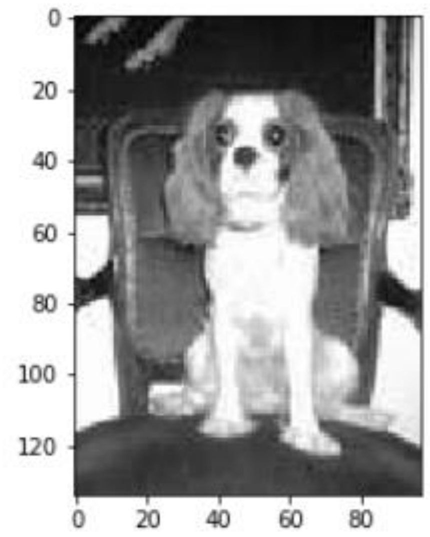


Fig 6. Small Image at Original Size

```
image shape: (200, 200)
number of pixels: 40000
```

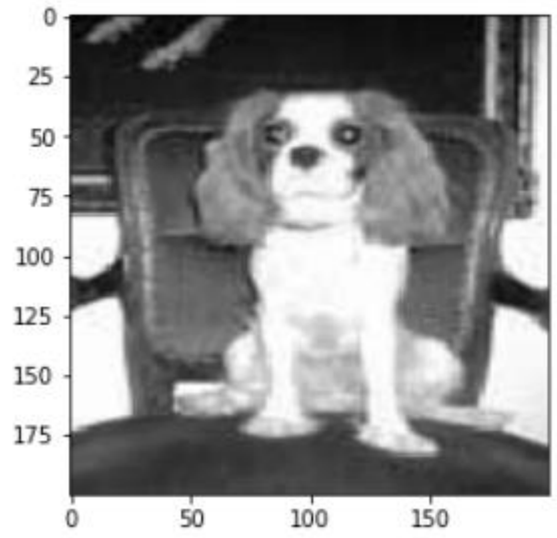


Fig 7. Small Image Resized

Since both the largest image and the smallest resized well, it is fair to assume that the images in between should resize without major distortions.

Hypothesis Testing

Table 1 below shows that there are differences in the average number of pixels per breed. We want to determine whether the difference between the average number of pixels per breed is statistically significant. This will be done by comparing the p-value to the significance level to the null hypothesis.

Breed	Average Pixels Per Breed
affenpinscher	126256.812500
afghan_hound	187398.060345
african_hunting_dog	174261.453488
airedale	177532.841121
american_staffordshire_terrier	165741.337838
appenzeller	175589.282051
australian_terrier	153140.058824
basenji	214494.936364
basset	219222.048780
beagle	189387.819048

Table 1. Average Number of Pixels Per Breed

The test used for this analysis is a One-Way ANOVA.

- Null-Hypothesis: There is no difference between the means of pixels of each breed
- Alt-Hypothesis: The means of some of the breeds are different
- Alpha= .05

```
               df        sum_sq       mean_sq         F        PR(>F)
breed       119.0  1.187827e+13  9.981743e+10  1.901349  1.525382e-08
Residual  10102.0  5.303371e+14  5.249822e+10       NaN           NaN
```

Table 2. Results of the One-Way ANOVA Test

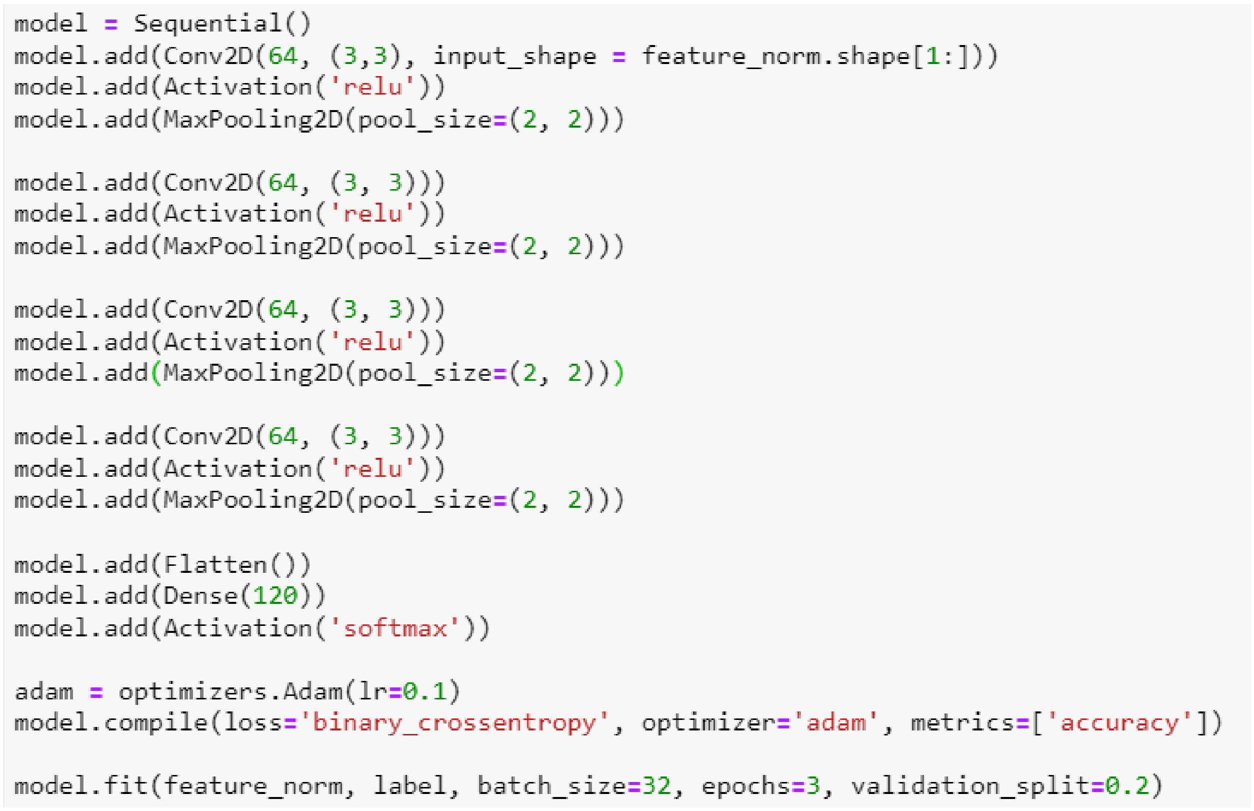
The p-value on our F-statistic is much smaller than the significance level of .05. Therefore, the null hypothesis is rejected. It is concluded that some of the differences between the pixel means of the breeds are statistically significant.

This, however, will not affect the neural network as the images will be resized to 40,000 pixels for each image. Our comparison of the images before and after resizing demonstrate that image integrity is maintained when resized.

In-Depth Analysis

The method used to classify the dog breed images is a convolutional neural network (CNN). This was done using Keras and Tensorflow in Python. The simplest model to use in Keras for a CNN is Sequential(), therefore this was used to build the model. An AWS machine was used to increase computational capacity. This allowed for the use of five layers in the CNN. The activation functions used were rectified linear unit (relu) on the first three layers and softmax used for the last layer. This combination provided the best results, figure 8.

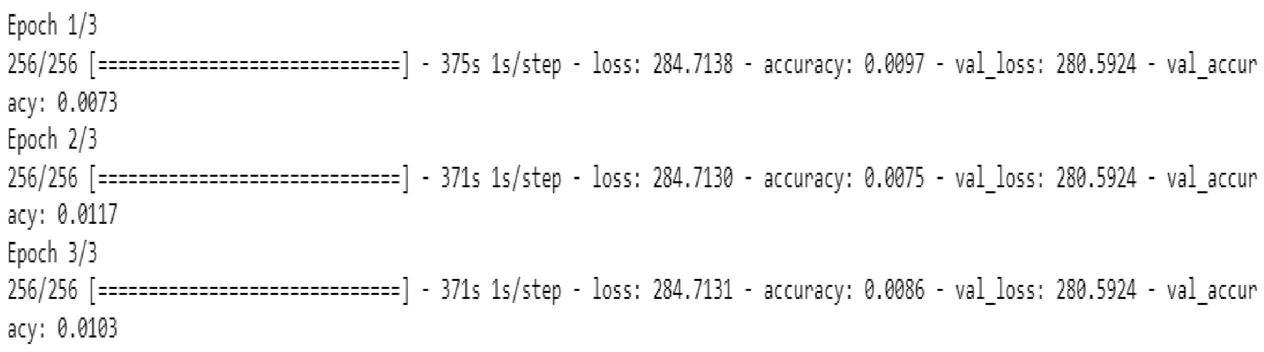
The learning rate was set at .1 as this was the smallest learning rate possible with the computational power available. Testing the different hyperparameters also resulted in Adam being the best optimizer.



```python
model = Sequential()
model.add(Conv2D(64, (3,3), input_shape = feature_norm.shape[1:]))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(120))
model.add(Activation('softmax'))

adam = optimizers.Adam(lr=0.1)
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

model.fit(feature_norm, label, batch_size=32, epochs=3, validation_split=0.2)
```

Fig. 8. Best Performing CNN Model

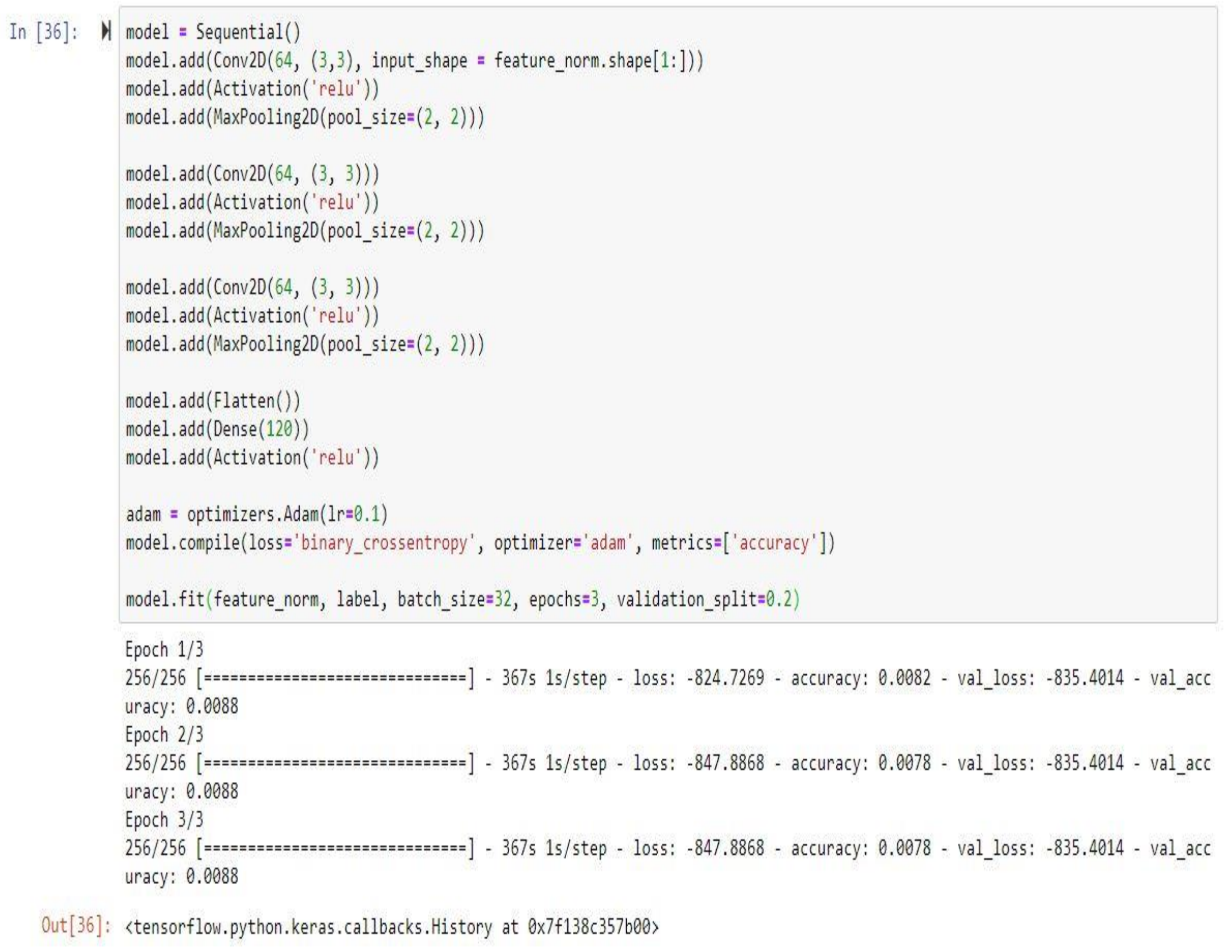
To measure the accuracy of the model, the validation accuracy is used. The training set was split into 80% training data and 20% validation. The accuracy used to assess the model is the validation accuracy.



```
Epoch 1/3
256/256 [==============================] - 375s 1s/step - loss: 284.7138 - accuracy: 0.0097 - val_loss: 280.5924 - val_accur
acy: 0.0073
Epoch 2/3
256/256 [==============================] - 371s 1s/step - loss: 284.7130 - accuracy: 0.0075 - val_loss: 280.5924 - val_accur
acy: 0.0117
Epoch 3/3
256/256 [==============================] - 371s 1s/step - loss: 284.7131 - accuracy: 0.0086 - val_loss: 280.5924 - val_accur
acy: 0.0103
```

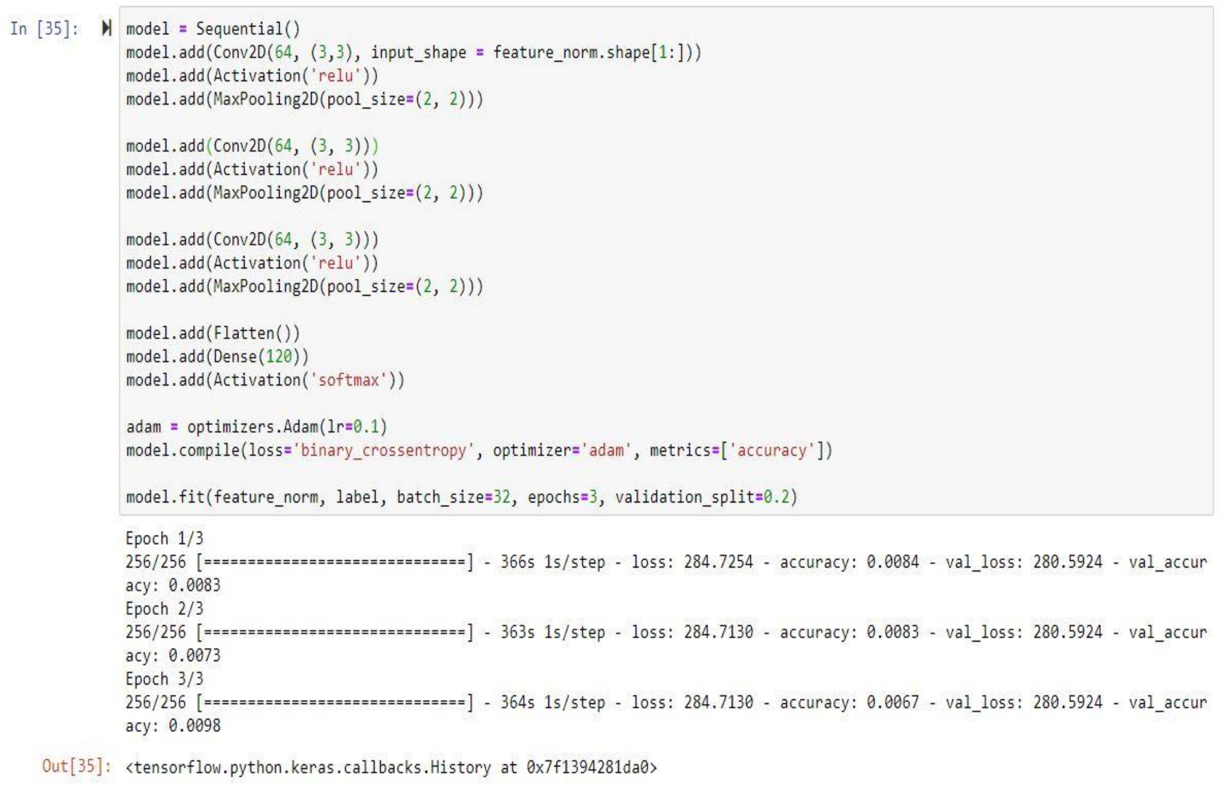
Fig. 9. Results of The CNN Model

The best validation accuracy of the three epochs was 1.17%. For comparison purposes, figures 10 and 11 below show the validation accuracies of the weaker models. CNN Model #2's best result was .88%. This model had all activation functions set to relu. CNN Model #3's best result was .98%. This model used softmax as the last activation layer but only had four layers.



```
In [36]:  model = Sequential()
          model.add(Conv2D(64, (3,3), input_shape = feature_norm.shape[1:]))
          model.add(Activation('relu'))
          model.add(MaxPooling2D(pool_size=(2, 2)))

          model.add(Conv2D(64, (3, 3)))
          model.add(Activation('relu'))
          model.add(MaxPooling2D(pool_size=(2, 2)))

          model.add(Conv2D(64, (3, 3)))
          model.add(Activation('relu'))
          model.add(MaxPooling2D(pool_size=(2, 2)))

          model.add(Flatten())
          model.add(Dense(120))
          model.add(Activation('relu'))

          adam = optimizers.Adam(lr=0.1)
          model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

          model.fit(feature_norm, label, batch_size=32, epochs=3, validation_split=0.2)

          Epoch 1/3
          256/256 [==============================] - 367s 1s/step - loss: -824.7269 - accuracy: 0.0082 - val_loss: -835.4014 - val_acc
          uracy: 0.0088
          Epoch 2/3
          256/256 [==============================] - 367s 1s/step - loss: -847.8868 - accuracy: 0.0078 - val_loss: -835.4014 - val_acc
          uracy: 0.0088
          Epoch 3/3
          256/256 [==============================] - 367s 1s/step - loss: -847.8868 - accuracy: 0.0078 - val_loss: -835.4014 - val_acc
          uracy: 0.0088

Out[36]:  <tensorflow.python.keras.callbacks.History at 0x7f138c357b00>
```

Fig. 10. CNN Model #2



```
In [35]:  model = Sequential()
          model.add(Conv2D(64, (3,3), input_shape = feature_norm.shape[1:]))
          model.add(Activation('relu'))
          model.add(MaxPooling2D(pool_size=(2, 2)))

          model.add(Conv2D(64, (3, 3)))
          model.add(Activation('relu'))
          model.add(MaxPooling2D(pool_size=(2, 2)))

          model.add(Conv2D(64, (3, 3)))
          model.add(Activation('relu'))
          model.add(MaxPooling2D(pool_size=(2, 2)))

          model.add(Flatten())
          model.add(Dense(120))
          model.add(Activation('softmax'))

          adam = optimizers.Adam(lr=0.1)
          model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

          model.fit(feature_norm, label, batch_size=32, epochs=3, validation_split=0.2)

          Epoch 1/3
          256/256 [==============================] - 366s 1s/step - loss: 284.7254 - accuracy: 0.0084 - val_loss: 280.5924 - val_accur
          acy: 0.0083
          Epoch 2/3
          256/256 [==============================] - 363s 1s/step - loss: 284.7130 - accuracy: 0.0083 - val_loss: 280.5924 - val_accur
          acy: 0.0073
          Epoch 3/3
          256/256 [==============================] - 364s 1s/step - loss: 284.7130 - accuracy: 0.0067 - val_loss: 280.5924 - val_accur
          acy: 0.0098

Out[35]:  <tensorflow.python.keras.callbacks.History at 0x7f1394281da0>
```

Fig. 11. CNN Model #3

Conclusion

Clearly a validation accuracy of 1.17% is not a good result. It is far worse than simply guessing. Multiple hyperparameters were used, however, the best ones were not able to overcome the main limiting factors. Given the available computing power, it would have taken weeks process more epochs. To process more layers would not have required weeks of training time. Using a computer with more GPUs would greatly decrease the training time from weeks to hours.

The second factor limiting the results of the model is the number of images. 10,000 per dog breed may have been enough to raise the results above 50%, but that was the total number of images available. Each breed had only around 100 images each. This is not enough to properly train a neural network.

To improve on this model without having to invest in a more powerful computer, a pre-trained model can be used to boost the classification ability of the current model. This is what transfer learning does. A new model used to train on a certain task builds on a previous model the was trained to do a similar task by using its weights and features. There are a few transfer learning models available to try on the dog breed classification dataset, VGG-19 and ResNet-50, amongst others.

On a more positive note, many entries into the Kaggle competition for dog breed classification were not able to reach 1% accuracy on the test set without using transfer learning. Once transfer learning was used, results range from 60% accuracy and higher.