Classification of 315 Bird Species

Pizon Shetu

December, 2021

Contents

1 Problem

Numan Bird Watch is an international birds conservation and exploration organization. They are currently building a large database of images of different species of birds in order to help their explorers to identify and keep record of the species of birds. As it is difficult to identify various species of birds as many are similar in patterns, colors, and sizes it takes a keen eye to notice the slightest details to properly identify different birds.

Numan Bird Watch has tasked me to create an image classification which can identify and properly classify different species of birds, so when their explorers take images of birds across the globe, they can use our classifier to identify different birds properly.

2 Data

2.1 Data Collection

The data set is comprised of 45980 training, 1575 testing, and validating data. The data set was gathered from this kaggle Birds data set. The images are all 224x224x3 RGB photos in JPG format. The data set already comes with the data split into training, testing and validating, thus we are not required to do any splitting procedure.

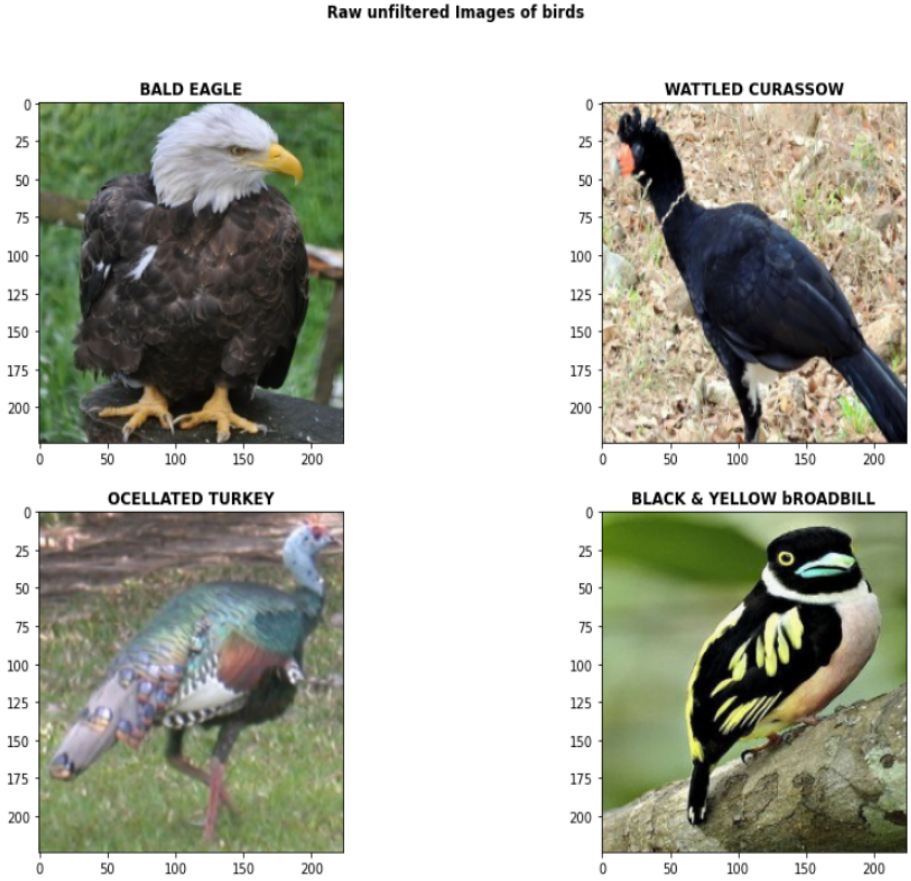
2.1.1 Data Cleaning

The data set is already clean and all images are 224x224x3, there were no anomalies with the images nor were there any duplicates, as far as data cleaning there was nothing required from my part as the data set was ready to go.

2.2 Data Exploration

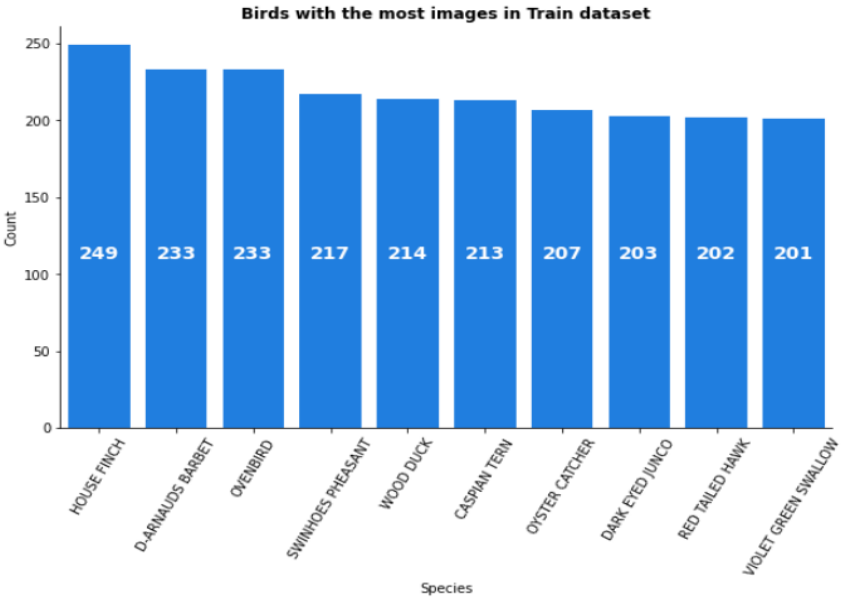
2.2.1 Images

The data set contain images for 315 different bird species, here we can see a sample of the images in the data set.

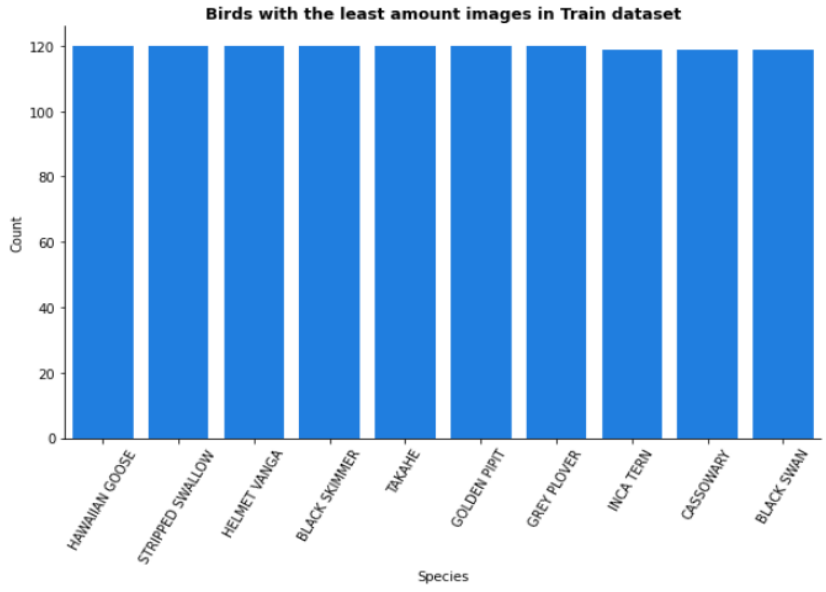


As humans we can clearly see that the images are quite clear and we would be able easily distinguish from different species of birds given the knowledge. But this is always not case for machines, as there are more details of images, more data to process, this makes it quite difficult to train neural network models as we face memory bottlenecks.

For perspective I wanted view some distribution of different species to see how many images are there for each. Here is a distribution for birds with the most images.



I was also concern about certain species and if they would be under-fitted, here is a distribution of birds with the least amount of images.



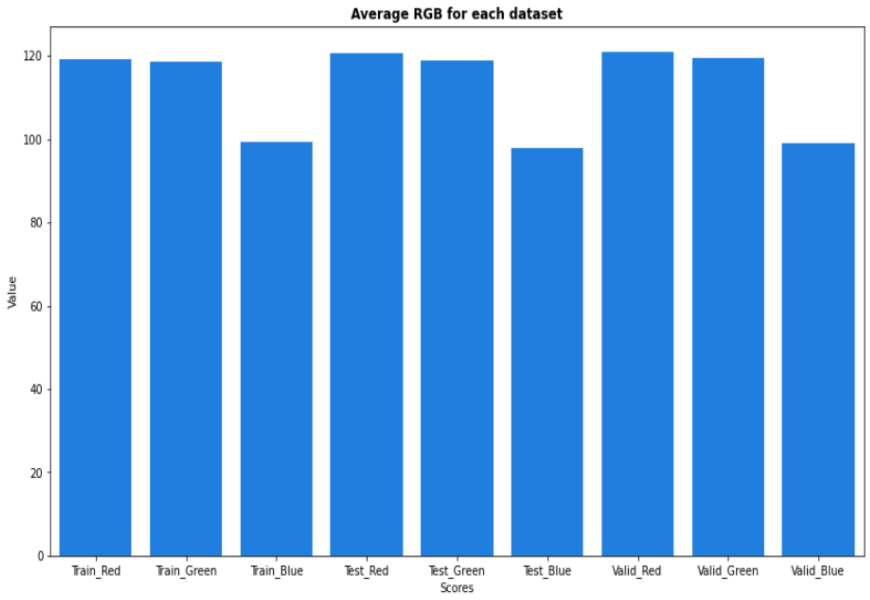
Its good to see that the minimum number of photos we have is 120, this will suffice to avoid under-fitting, while I could plot the number of images for each species it is unnecessary and not to insightful for the task at hand.

2.2.2 RGB Distribution

I wanted to check if there was any sort bias towards any of the 3 RGB channels in our data set, granted this was a unnecessary step I was curious nevertheless. This step required me to utilize PIL Image library, to get the image data for each image in our data set and append those data into a data frame. Below is a snippet of the data frame.

	Species	Train_Red	Train_Green	Train_Blue	Test_Red	Test_Green	Test_Blue	Valid_Red	Valid_Green	Valid_Blue
0	AFRICAN CROWNED CRANE	116	118	90	113	114	80	89	97	91
1	AFRICAN FIREFINCH	126	109	86	108	95	72	127	119	87
2	ALBATROSS	109	123	134	128	135	146	119	118	116
3	ALEXANDRINE PARAKEET	122	127	101	127	131	105	118	121	86
4	AMERICAN AVOCET	125	127	128	126	127	121	124	119	112

Here is the distribution of the average RGB for each channel in each data set



While this was to be expected we can see there is not a significant deviation from the RGB channels in our data sets while we could've guessed this outcome but now we know for sure when making our model perhaps we won't have to worry model prediction inaccuracy due to RGB.

2.2.3 Word Cloud

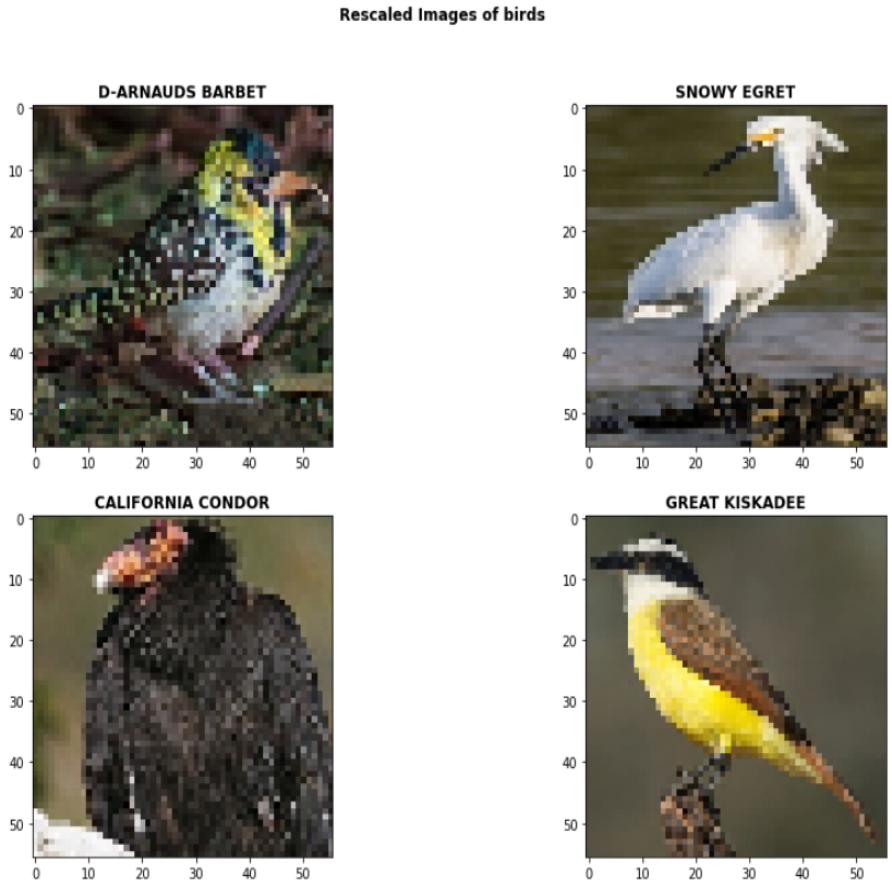
I was curious what was the most frequent specie of bird, not just in specific specie type but their origin and color, for this I decided to utilize a word cloud. A word cloud can generate the most frequent words appearing in any given string thus, I combined all the specie names into a single string and generated this word cloud.

It's important to point out that this has no correlation to the number of images of each bird has, as these are all the unique specie names in the data set, we can see there are a lot of birds from the family of Finch, Pheasant, Warbler and we can also see many birds are from the region of America.

2.3 Pre-processing

2.3.1 Rescaling Images

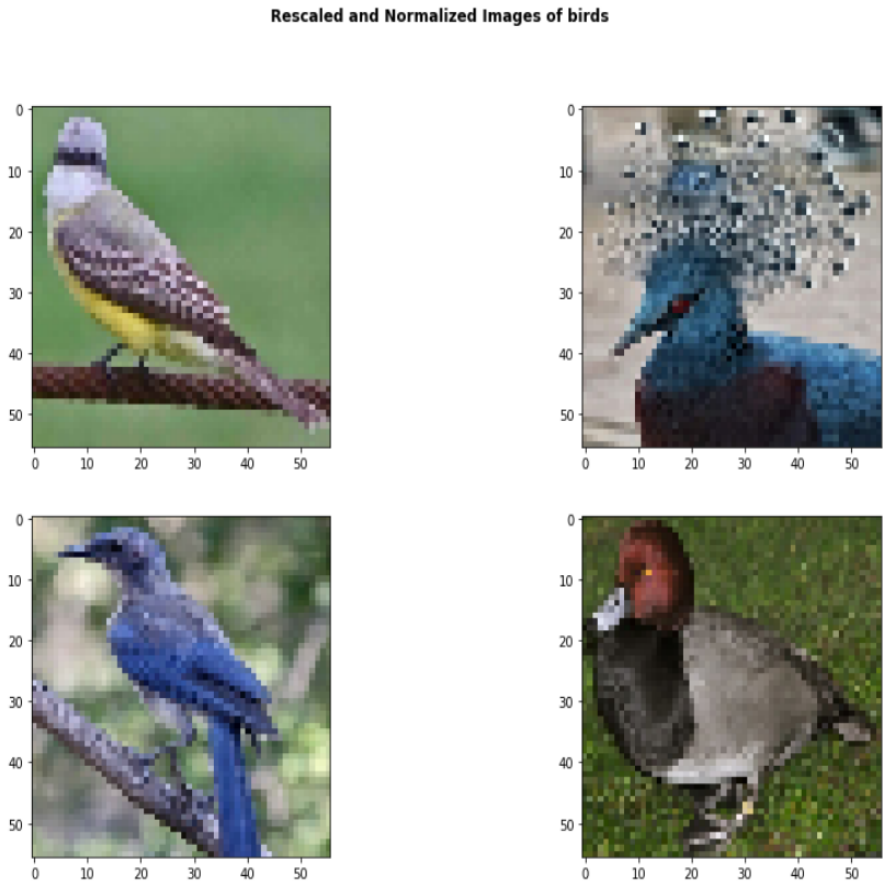
Our image size is 224x224 and this can be troublesome for our models to train, as the larger the image size, more time is required to train, I attempted to run my models with this image size without pre-processing and ran into memory bottleneck and kernel crashes. In my case I needed to reduce the images by a factor of 4 as 112 pixel size still led to overload of my GPU memory, I am running RTX 2080 8GB for reference. To negate this I will be re-sizing all the images to 56x56. Here is a sample of the rescaled images, we can see this resulted in a much more pix-elated but still distinguishable image.



The images are much more pix-elated but still distinguishable.

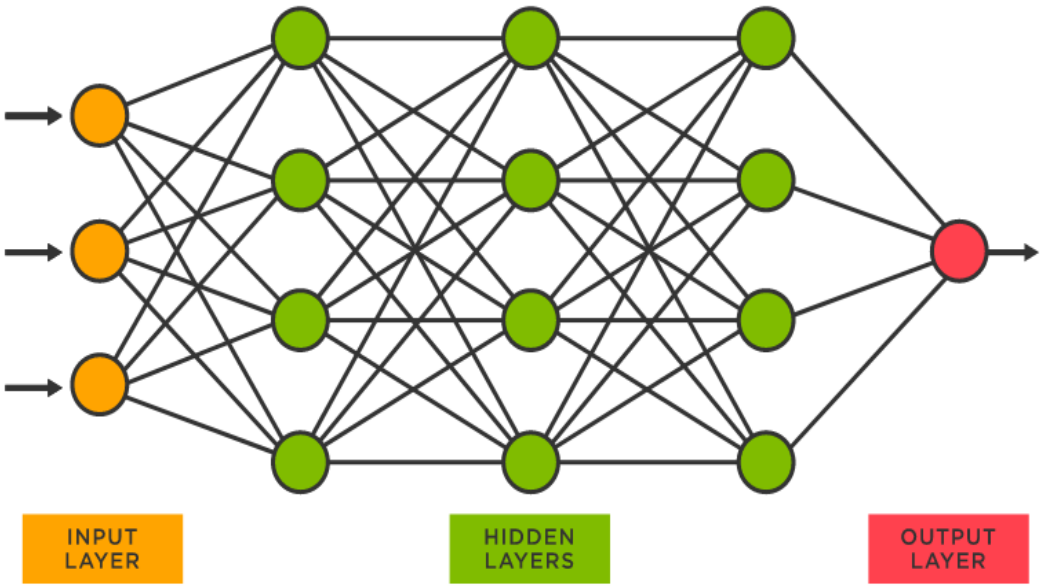
2.3.2 Normalizing Images

Neural Networks gain a big boost when the data is normalized, which means the values of the data have a mean close to 0 making the range of our inputs between -1 and 1. This process generally speeds up any learning process in training leading to a quicker convergence, since machines generally lose accuracy when computing math operations on large numbers and have a difficult time learning when different features vary wildly, this allows for getting all our data on the same scale. Since it is difficult to get a distribution of our input data since they are in essence will be an array of images with each image having a shape of 56x56x3 it is ideal to normalize our data. Below is a set of images after going through a re-scaling and normalizing process.



3 Neural Network

Neural Networks are the centerpiece of this project and how we will accomplish a good classifier, a neural network are a set of algorithms, modeled after the way human brains work, and their ability to recognize patterns. In the brain there are many neurons connected to each other communicating and sharing information and giving instruction from one set of neurons to the next. In the same sense that is how a neural network is designed a layer of neurons or nodes receiving inputs and outputting results to the next set of layers. Below is an depiction of what a neural network looks like. The diagram was taken from tibco.com



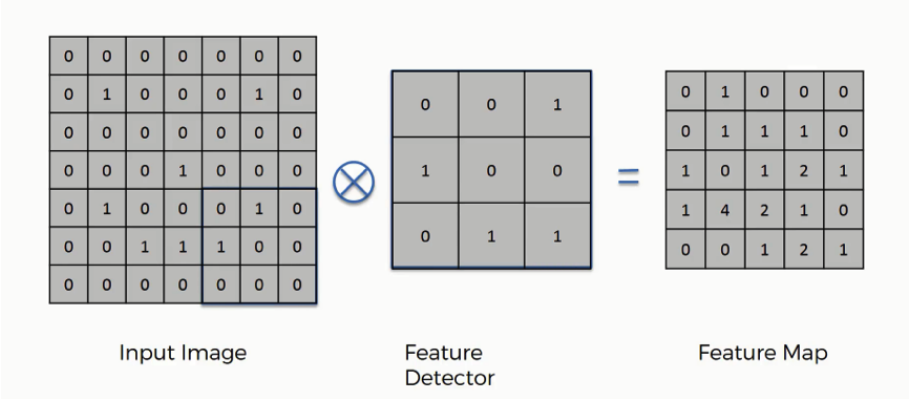
In this illustration we can think of the input layer containing our data or image values, which is then relayed to the hidden layers where each hidden layers process those inputs and extracts some information from this input, once one hidden extract the information it passes it along to the next and in the end it final makes a decision on what those inputs mean and provides us with an output. For demonstration purposes below I will show the results of a sequential model in Keras with one hidden layer, don't worry about the details yet I will explain later in this report.

```
Epoch 1/5
1437/1437 [==============================] - 6s 3ms/step - loss: 5.7500 - accuracy: 0.0047 - val_loss: 5.7577 - val_accuracy:
0.0032
Epoch 2/5
1437/1437 [==============================] - 4s 3ms/step - loss: 5.7437 - accuracy: 0.0053 - val_loss: 5.7612 - val_accuracy:
0.0032
Epoch 3/5
1437/1437 [==============================] - 4s 3ms/step - loss: 5.7427 - accuracy: 0.0053 - val_loss: 5.7611 - val_accuracy:
0.0032
Epoch 4/5
1437/1437 [==============================] - 4s 3ms/step - loss: 5.7424 - accuracy: 0.0054 - val_loss: 5.7623 - val_accuracy:
0.0032
Epoch 5/5
1437/1437 [==============================] - 4s 3ms/step - loss: 5.7422 - accuracy: 0.0054 - val_loss: 5.7607 - val_accuracy:
0.0032
50/50 [==============================] - 0s 2ms/step - loss: 5.7607 - accuracy: 0.0032
Test Accuracy: 0.317
```

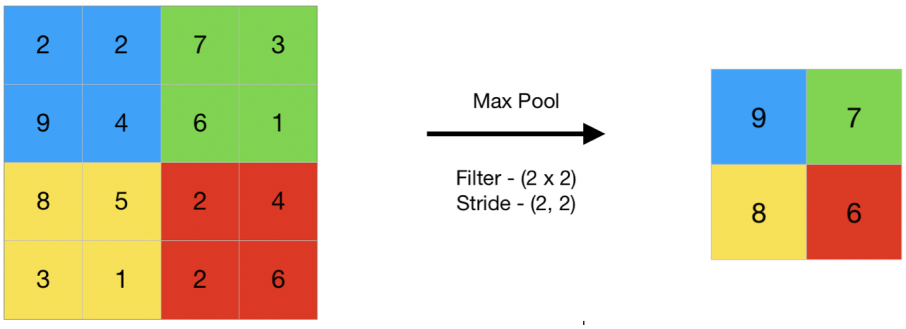
As we can see this model does a horrible job at predicting bird species with less than 1% but that's okay as this is for demonstration purposes.

3.1 Convolutional Neural Networks

A Convolutional Neural Network (CNN) is a different flavor of neural network and it specializes in being able to take an input image and assigning learnable weights and biases to various aspects of the image to be able to learn and differentiate that image from others and this is specially important for us since we need to be able to differentiate various bird species. CNN does linear operations which involves multiplying a set of weights and an array of input data which creates a filter or a kernel which is much smaller than the original input data. And this filter is applied throughout the original input data or image left-to-right then top-to-bottom to distinguish key features of an image. Below is an image which shows a good representation of how convolution works on a 7x7 input image, the image was obtained from superdatascience here is a link.

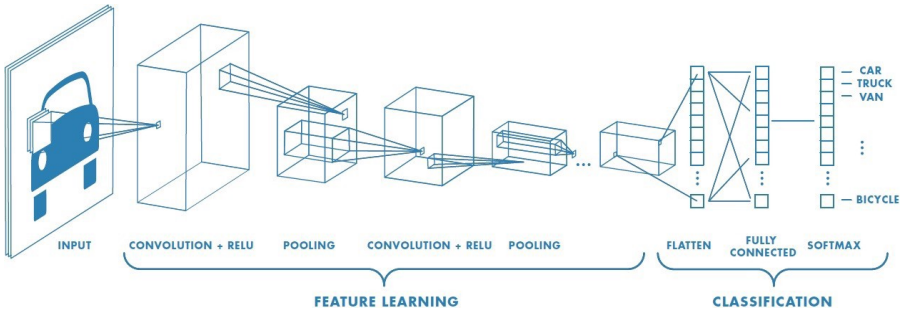


A key aspect of a CNN is the pooling layer which is responsible for reduction in the spatial size or the dimensions of the feature map, leading to a reduction in the number of parameters needed to learn and this allows for less computation needed to perform training in the network. While there are many types of pooling in our case we will be using max pooling which selects the maximum element from the area of the feature map which is hovered over by filter. This leads to a new feature map containing the most important features of the prior. Below is a image representing max pooling by geeksforgeeks



Another key aspect of CNN is most hidden layers use an activation function which essentially tells a node or neuron if it should be calculating the weighted sum and further adding its bias with it, without it every neuron would be performing a linear transformation on the inputs to use the weights and biases in many cases leading to a less powerful network. While there are many activation functions we will be using 'ReLU' which is a non-linear activation function ReLU stands for Rectified Linear Unit. ReLU function does not activate all the neurons at the same time it does this by de-activating all neurons whose linear transformation is less than 0, leading to all negative values to be zero. This enables ReLU to be much more computational efficient compared to other activation functions.

To truly visualize the whole dynamics of a CNN here is a illustration obtained from towardsdatascience blog credits to Sumit Saha



3.2 Sequential Model

Now that I have explained some of the complexities of a CNN, I will now explained the Keras Sequential model which we will be using to build our CNN. The Sequential API from Keras allows us to create models layer-by-layer, and is much simpler than their functional models which does for allow for more flexibility but lacks the simplicity of the Sequential model. Many of these definitions are from Keras own website.

- Sequential is a model in Keras which allows to create models layer by layer, according ot Keras it is a model with linear stack of layers.
- Conv2D: This creates a convolutional kernal which is convolved with layer input to create a tensor of outputs

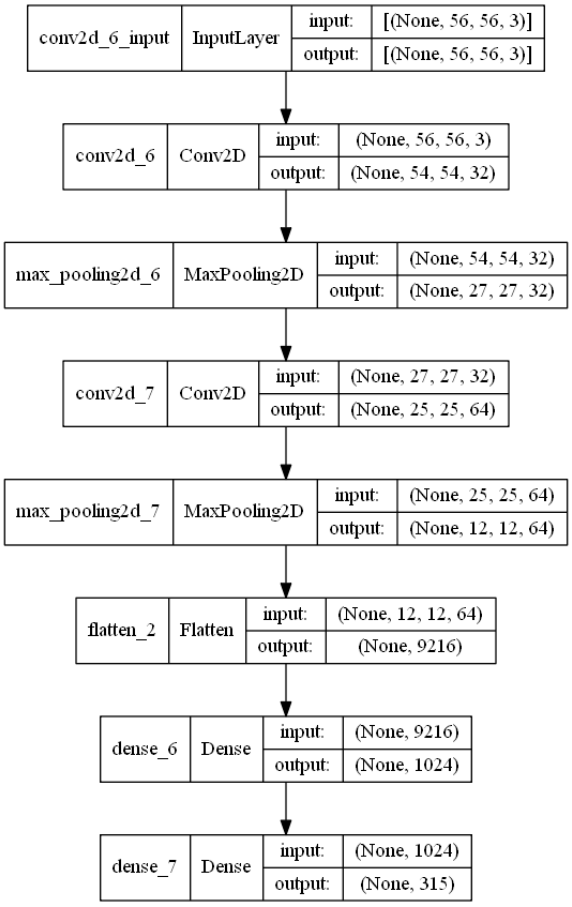
  - Filters Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution).

  - Kernel size: An integer or tuple/list of 2 integers, specifying the height and width of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions.

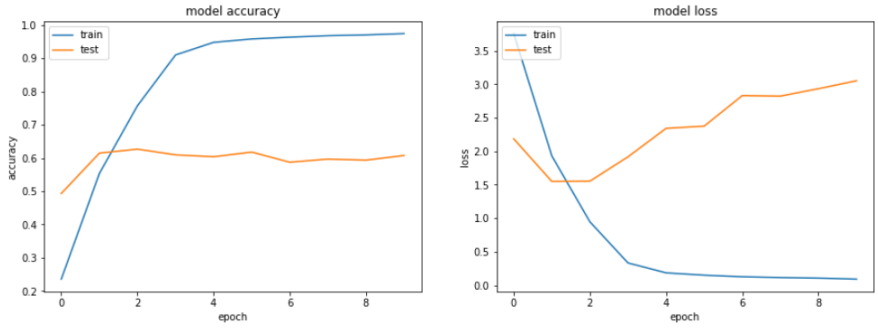
  - Strides An integer or tuple/list of 2 integers, specifying the strides of the convolution along the height and width. Can be a single integer to specify the same value for all spatial dimensions.

- MaxPool2D The maxpooling layer will reduce the size of each feature map by a factor of 2, reducing value of the pixels in the feature maps. Max pooling is said to allow for extraction of the sharpest features of a image and it is also faster to compute than convolutions.
- Flatten Flatten takes all a N X N dimentional feature map and turns it into a FLAT 1 * (NxN) for example: 3x3 into 1x9
- Dense Dense is the essential neural network layer, this decides the number of neurons in a layer and it receives all its input from the previous layer.
- Dropout is utilized to prevent overfitting of a neural netowrk, it works by randomly setting the neurons that make up the hidden layers to 0 at each update of a training phase.
- Optimizer Optimizers are essentially methods used to change the weights and learning rate of your model to avoid reduction in losses. In our case we will be using the Adam optimizer
- Loss function This is our value which the model attemps to minimize during its training.
- Metric This is a metric of evaluation to see how well our model does, a method to judge the performance of our model.
- Epoch Epoch is essentially the number of times you go through the training set in more accurate definition it is an arbitrary cutoff, generally defined as "one pass over the entire data set", used to separate training into distinct phases, which is useful for logging and periodic evaluation.

Here is a CNN sequential model with two convolutional layer and two max pooling followed right after the Conv2d layers. This model was able to achieve a accuracy score of 60% after 10 epochs. I will also plot the model accuracy and model loss through each epoch.



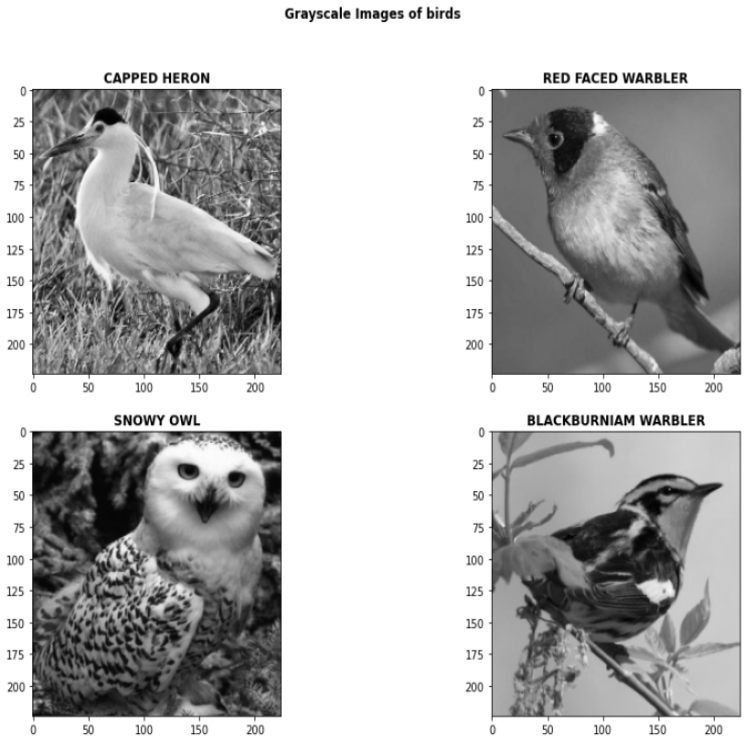
Here is a plot of the model accuracy and model loss through each epoch.



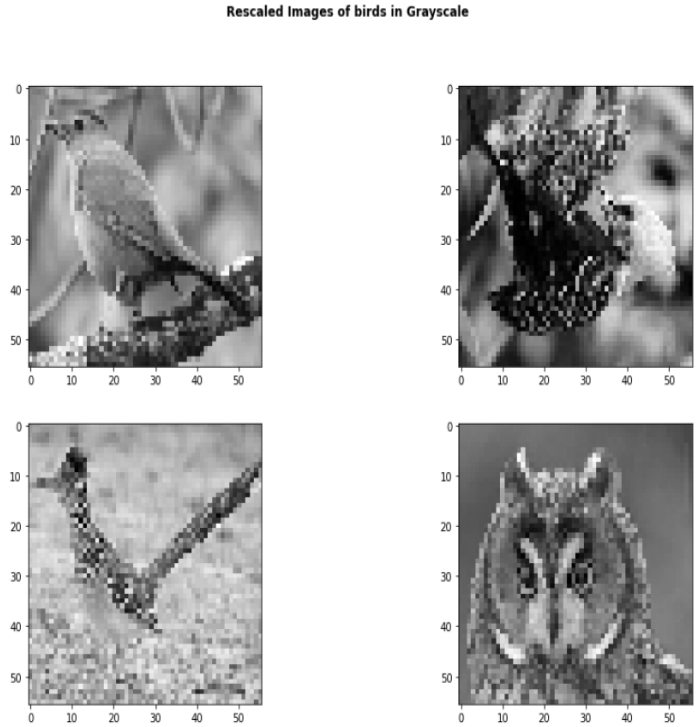
3.3 Gray-scale

As I was doing this classification problem I also wonder what would happen if I turned our images into gray scale and did the same procedure as we have done for our standard RGB images. Hence I attempted to see which would yield better classification before I jumped into tuning our model from scratch.

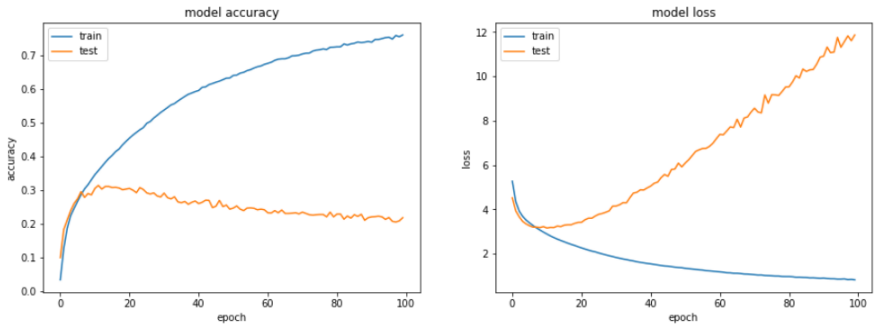
First lets look at some gray scale images of our data set.



As we did previously we must re-scale and normalize these images as well, here are the results.



After inputting these gray scale data into our previously created CNN model we get an accuracy of 21.78% after 100 epochs.



Intuitively I was expecting such results but it never hurts to try. Interestingly as epochs get larger our model gets less accurate and has a higher model loss, this was also the case for our RGB model when I ran for 100 epochs but I neglected to save the model thus unable to show its results. Now we will continue back on our venture in maximizing our model for RGB data.

4 Hyperparameter Tuning

Hyperparameter tuning is process in which we optimize parameters within the model in an attempt to yield better results, it works with parameters are variables which govern the training process itself, for example epoch is an hyperparameter. Hyperparameter tuning works by running multiple trials in a single training job. Each trial is a complete execution of your training application with values for your chosen hyperparameters, set within limits you specify.

Before I start talking about specific types of hyperparameter tuning options we have for our case I am going to explain callbacks and how it will help us save time.

- Callbacks is a type of object in TensorFlow and Keras allows users to monitor the performance in metrics at certain points of a training run and execute actions depending on the performances based on the metric values.
- Early Stopping is part of callbacks where it allows the model to stop training when our metric stops improving.
- ReduceLROnPlateau is a callback to reduce the learning rate when a metric has stopped improving.
- Monitor is the metric that is used to determine when to activate these methods, in our case its validation accuracy loss for early stopping and validation model loss on Reducing learning rate on plateau
- Patience is the number of epochs with no improvement after which training will be stopped.
- Factor is the amount the learning rate will be reduced by.

4.1 RandomSearch

RandomSearch's tuning methods revolves around randomly trying a combination of hyperparameters from a given serach space. In our case of our previous model our search space involves the number of input units for each Conv2d blocks, the number of neurons in our hidden layer and the dropout rate. I will be using Keras Tuner to tune my model, these are the following parameters I will be controlling in the tuner, max trials which is how many model variation should the tuner test, and executions per trail which is how many trail should be run per variation.

Having a large max trials allows us to test many variations to get the check the most optimal parameters but it takes some time and selecting a executions per trial high will take a significant amount of time while the same model may yield different results I am okay with the first result so we will set that to 1, as far as max trials I will set this to 50 to test many parameters. Score is the the accuracy of our model with those parameters.

RandomSearch yielded these results

```
Hyperparameters:
input_units: 64
n_layers: 1
conv_0_units: 224
num_of_neurons: 64
num_of_neurons_2: 160
dropout: 0.1
conv_1_units: 160
Score: 0.7301587462425232
```

4.2 Hyperband

While RandomSearch is a great method to find good parameters but it wastes a lot time training bad parameters since RandomSearch has no bias towards the parameters it uses and hence the name the parameters are randomly selected so instead we will try Hyperband, which avoids this major problem in random search. Hyperband will randomly sample the parameters but instead of running full max epochs on them they will initially run few epochs to test out what type of results it is getting. Depending on the results it will then do a re-run of the parameters which yielded the best results and run the full epoch then.

Hyperband yielded these results

```
Hyperparameters:
input_units: 96
n_layers: 2
conv_0_units: 224
num_of_neurons: 96
num_of_neurons_2: 160
dropout: 0.0
conv_1_units: 512
conv_2_units: 480
conv_3_units: 480
tuner/epochs: 25
tuner/initial_epoch: 0
tuner/bracket: 0
tuner/round: 0
Score: 0.6901587247848511
```

4.3 Bayesian Optimization

While Hyperband is more efficient at getting rid of bad parameters it is still choosing at random, so there is another tuner we can utilize which is Bayesian Optimization(BO). BO address the random selection problem of both RandomSearch and Hyperband and selects first few parameters randomly, then depending on the performance on of those hyperparameters it select the next best hyperparameters. Using the performances of the previous hyperparameters which were used, BO sets the next hyperparameters based on past historical performance and this continues till the tuner reaches optimal hyperparameters or exhausts maximum number of trials allowed.

Bayesian Optimization yielded these results

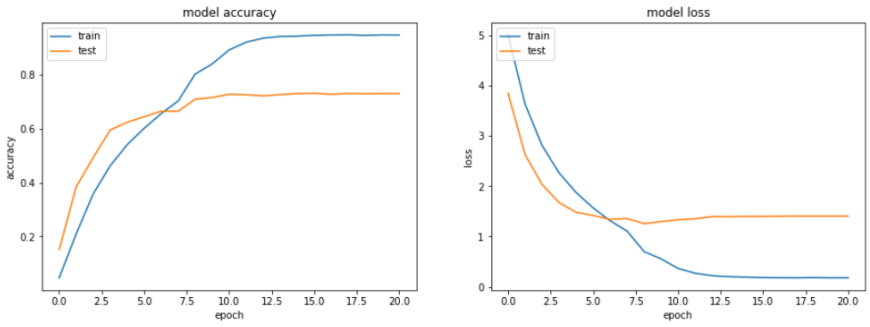
```
Hyperparameters:
input_units: 32
n_layers: 2
conv_0_units: 160
num_of_neurons: 256
num_of_neurons_2: 256
dropout: 0.45
conv_1_units: 512
conv_2_units: 64
conv_3_units: 64
Score: 0.7117460370063782
```

4.4 Best Model

In this whole process of hyperparameter tuning it was a time extensive task and throughout it all it took long computation time and sometimes the results might not garner much boost in performance. After some theory crafting and combining and more research I found the best parameters for our model would be below. There might have been a few changes here and there but for the most part this yielded the best results out of all models with a accuracy score of 73.2%

```
model = Sequential()
model.add(Conv2D(32, (3, 3), activation="relu", input_shape=(56, 56, 3), padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation="relu", padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, (3, 3), activation="relu", padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(2048, activation='relu'))
model.add(Dropout(rate=0.2))
model.add(Dense(2048, activation='relu'))
model.add(Dropout(rate=0.2))
model.add(Dense(315, activation='softmax'))

model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics="accuracy")
```



Now this is much more ideal, as we don't see a decline in performance as epochs increase, and we have avoided over-fitting. We have also increased overall accuracy due to a more complex neural network. Rather further fine-tuning the

model, it is safe to say our neural network is near its optimal accuracy, perhaps it can get to $75 - 80\%$ but the time and computational cost of achieving such a model would not be most optimal. An alternative solution is using large pre-trained models to train on your data set.

5 Transfer Learning

Transfer learning is the idea which a model is developed to learn and identify a certain task and it can be res-used for similar task and tuned to refinement. The difficulty with Neural Network models are they are computation expensive, take long time, and take large data set I am talking about over millions of data to train a fully fledged neural network. Companies such as Google with large data sets create robust models such as ResNet, mobilenet, EfficientNet and etc known as pre-trained models which can utilized these to fine tune to our own models. The analogy that is widely used is imagine your ancestors who once learned skills, language and ability to use tools which helped them survive, they have this vast knowledge since they lived their life but once their offspring are born, the babies are blank slate they know nothing but the parents pass over that knowledge so the babies don't have to start from ground zero, they utilize their parents knowledge and adapt to their environment. Such is the case of transfer learning.

Transfer learning is using past models knowledge to increase the learning capabilities of new models. Pre-Trained models are models created by someone or others to solve a similar problem but rather than building a model from scratch you utilize and tweak their already built model to your own problem. Two of the most popular neural networks are EfficientNet and VGG16 and I will be utilizing their networks to incorporate and tune to my data set.

5.1 Freezing/Unfreezing of Layers

Before we continue our modeling with pre-trained models I like to talk about the idea of freezing and unfreezing layers in a model.

Training a pre-trained model to your own large data set will take time and especially if you want a high accuracy score, we also have to train each layer of the model to our specific data set. This is where freezing and unfreezing layers of a model comes in, what freezing and unfreezing allows us to do is train specific layers which is 'unfreeze' and 'freeze' not train the other layers. This specially crucial when accounting for how long training takes.

In most cases most pre-trained models already have a good weights and training on detecting patterns of lines, and edges in their first few layers, but the subsequent layers are more adept to the data set they were trained on for this reason we freeze some these layers and keep the first few unfrozen so we can get a good training done on our specific data set without wasting time learning the early nuisances of lines and edges.

5.2 EfficientNet

EfficientNet is a CNN architecture where the scaling method uniformly scales the dimensions of depth, width and resolution using a compound coefficient. Traditional practice scales these factors arbitrary but EfficientNet utilizes this uniform scaling method to set a fixed scaling coefficients. For instances if we needed 2^N more computational power, we can just increase the network depth by α^N, width by β^N, and image size by γ^N where α, β, γ are coefficients which are constanted and determined by a grid search on the original small model.

After loading EfficientNetB0 from Keras it is an functional model, but recall we have been doing Sequential models, so we will take its layers and transfer to a Sequential model. Once we set up our model we yield a score of approximately 70%. As far as unfreezing and freezing certains did not yield to better results in EfficientNet

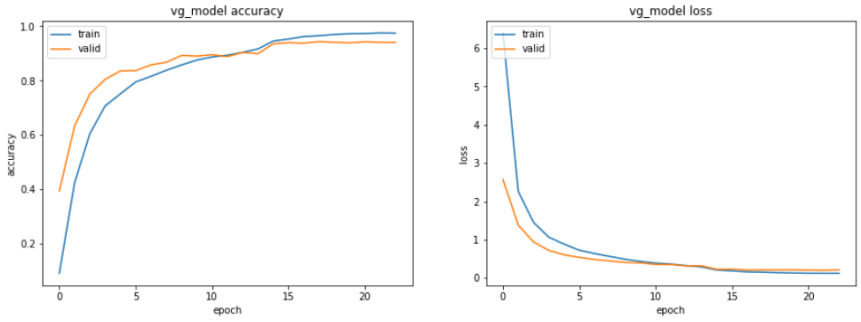
A side note about EfficientNet, I was able to achieve a 94% accuracy rate with this model, when I trained 100 epochs, but along the way I wanted to try some theory crafting and messed around with the model without saving it, and I was not willing to re-spent that time training the model all over again since I have already gotten 97%+ from the VGG model.

5.3 VGG16

VGG16 was invented by the Visual Gemoetry Group of Oxford University and proposed by Karen Simonyan and Andrew Zisserman, it was used to win 1st place at 2014 ILSVRC callenge. A key distinction with VGG16 is rather than have large number of hyper-parameters they instead focused on having many convolution layers of 3v3 filters and always used same padding with a maxpool layer of 2x2. The 16 in its name represents the 16 layers which have weights, the network itself has approximately 138 million parameters.

After loading the VGG16 model I repeated the similar steps that I took with EfficientNet taking the layers and adding it to a Sequential model. I needed to remove the last layer which was an output layer of 1000 and add a layer of 315.

After 40 epochs of training our VGG16 model had a accuracy score of 94% Significantly improve any result we had gotten prior. Below is the plots of our accuracy and model loss notice that it ends near 20 epochs and that was due to our callbacks doing an early stop since we the model was not improving past that, for reference we had to do 50 epochs to even get 73% with our model build from scratch but this has done it in less than 23 epochs.



5.3.1 VGG16 Tuning

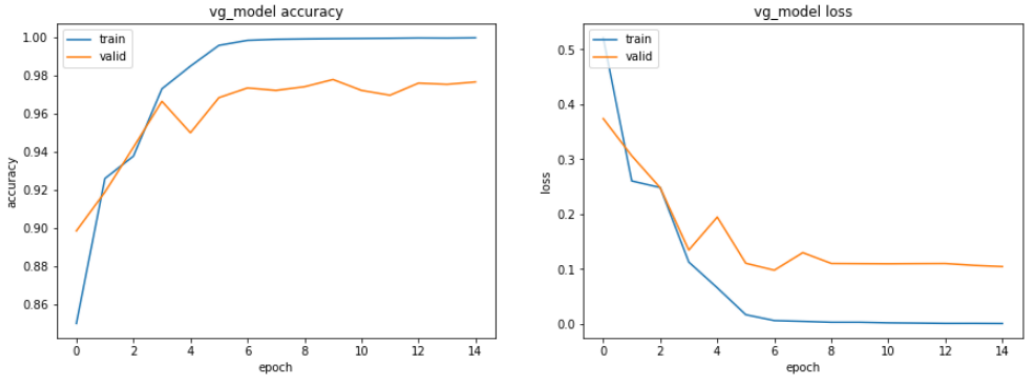
While we were not able to get better results from utilizing the freezing and unfreezing method with EfficientNet, we will try with our VGG16 model. Here is what our VGG16 model looks like

Even though we had done preprocessing earlier I wanted to utilize Keras ImageDataGenerator which is able to take images from directory and generate image data in batches, this method is much more efficient more our pre-trained models and in general. In the case of VGG16 I have also preprocessed the input using Keras VGG16 preprocess input library. This essentially allows VGG16 model to digest the data easily.

Layer (type)	Output Shape	Param #
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
fc1 (Dense)	(None, 4096)	102764544
fc2 (Dense)	(None, 4096)	16781312

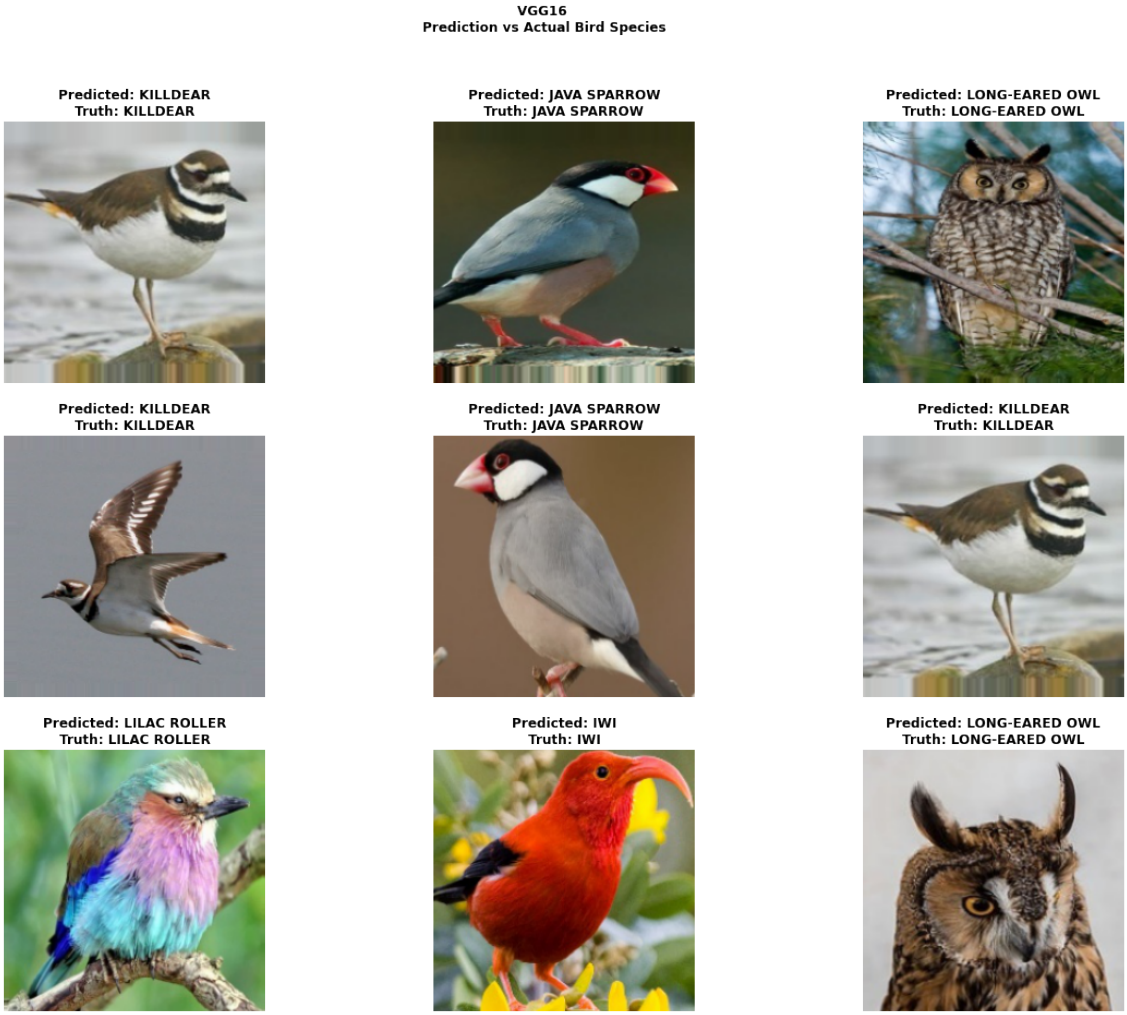
Total params: 134,260,544
Trainable params: 0
Non-trainable params: 134,260,544

I will be unfreezing the all the layers from black4conv1 and any layers below it, this will allow for our model to better train on our data set in the later part of the model while the model already has a well-establish understanding of edges and lines it might need to train more on the finer quirks of our data set. Unlike EfficientNet we were able to increase our accuracy score on unseen testing data to 97.5%



6 Predictions

After finally completing our VGG16 tuned model, I wanted to plot some images of actual and predicted bird species. In order to plot these images I needed to go through a deprocessing of our data, recall we had processed our inputs which led to the colors of our image to deviate from original colors, we I was able to find a helpful function from Sanchit Vijay which deprocessed the images and return their original colors. Since our data is in a ImageDataGenerator we will be taking random samples from batches. Here is a sample size of the predictions from the first batch.



7 Conclusion

Working on this image classification problem has been a massive learning experience as it has taught me how to work with Keras API's and different libraries as well as diverting from your traditional tabular data. Ironically the easiest task on this project was working with the data, as it was the cleanliest data I have worked with unlike my prior tabular projects where data is dirty and difficult to operate on. Other quirks I had to tackle was just implementing different ways of loading and setting up the data set for my models other than that it was a smooth transition from loading the data and implementing it into my models.

The most challenging part of this project was fully understanding the complexities of a Neural Network and how to implement Convolutional blocks along-

side max pooling and etc. While most time was spent building a Keras model from scratch and fine-tuning that model, it was also the most rewarding. Initially working with pre-trained models seemed simple, but properly configuring them for your data set is key, and tuning them is a even bigger factor. Through the tuning process I was able to learn and implement the freezing and unfreezing technique to squeeze even more performance of pre-trained models.

I hope Numan Bird Watch will be pleased with the model I have created for them as it can identify 315 different species with a 97.5% accuracy and it can always be trained on new species of birds introduced.

8 Personal Takeaways and Further Study

I was quite surprised at the data set as many of the bird species while different look very similar, for instance to even a human eye the Pygmy Kingfisher looks identical to a Rufous Kingfisher, but the models did a great job of extracting key features to distinguish different species with similar looks and features.

This is only a stepping stone, as the world of classification of images is endless, while this project was a learning experience I believe there a lot more I can learn when it comes to classification and recognition of images. This data set was very usable, I can only imagine how difficult it would've been if the data set required a lot of rework to able to process into models. I believe projects working on object detection and facial recognition would an interesting challenge and I will continue what I've learned here into those problems.