**Bird Classification using CNN.**

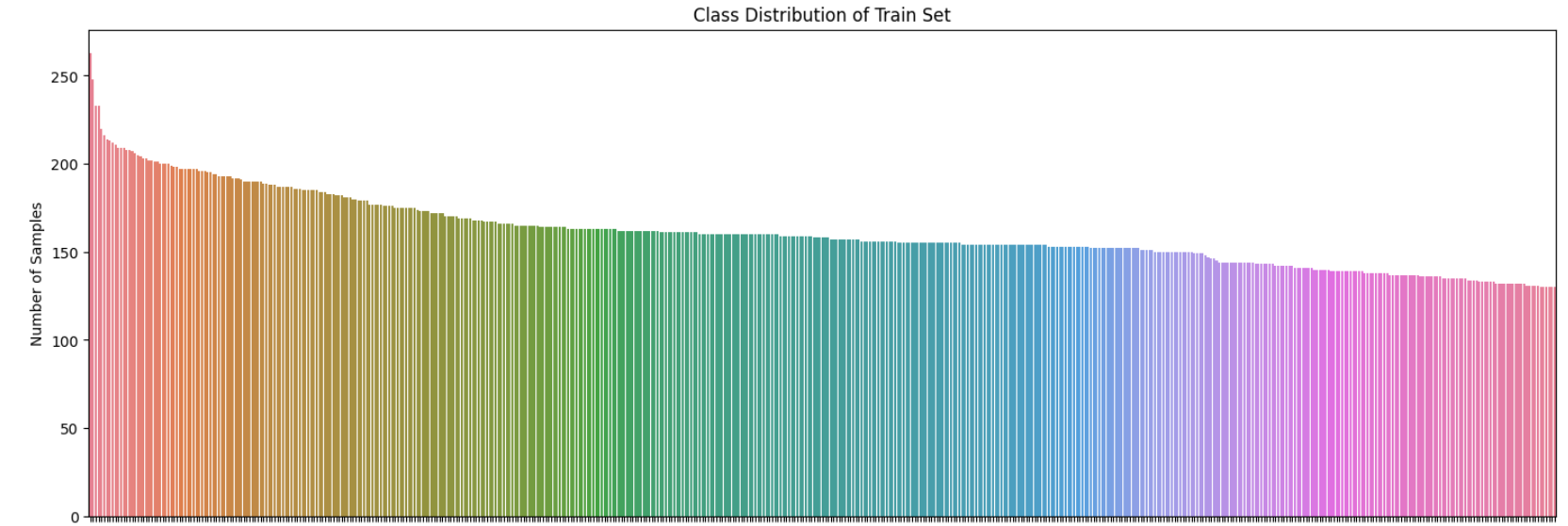
**EDA:**

Our data consists of 89,885 Images of 525 different types of birds, most of the images have the following dimensions [3,224,224].

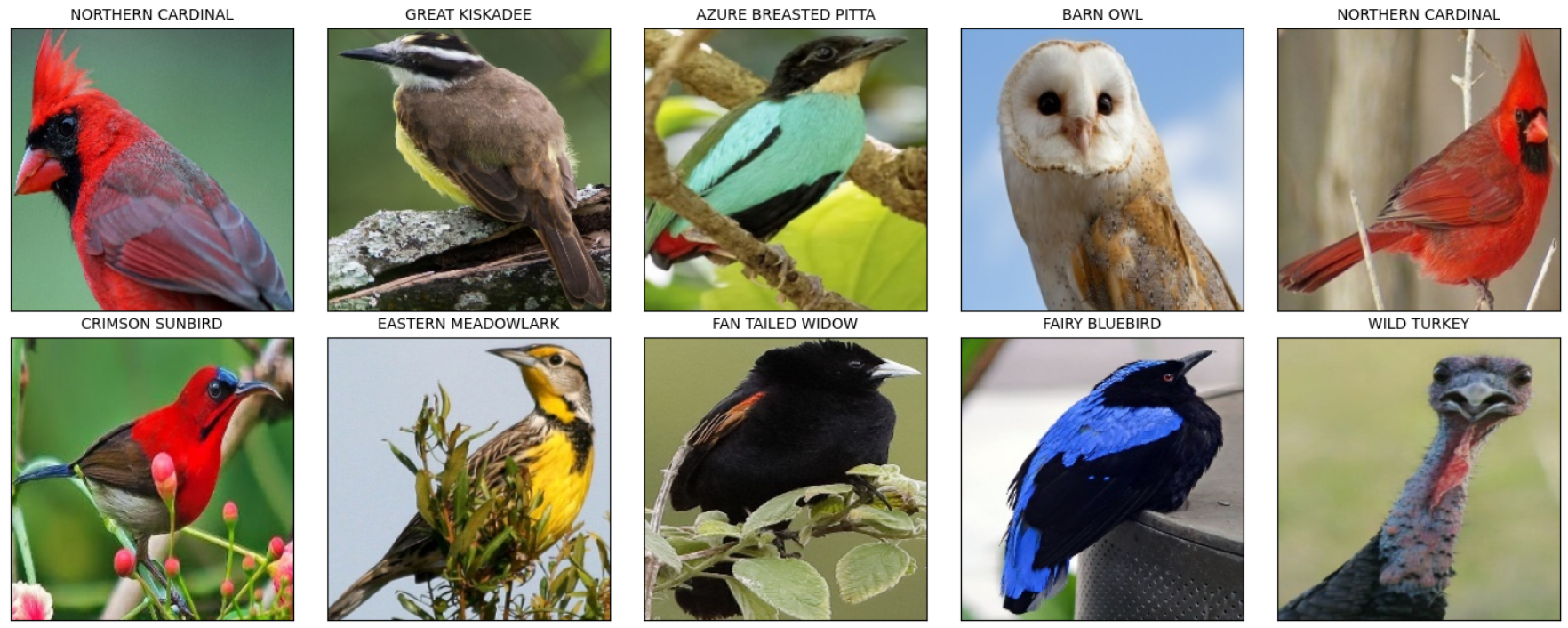
The dataset is divided into 84,635 samples for training, and 2625 samples each for both validation and testing, 5 images for each bird.

For each different bird we have an average of 161 images with a max of 263 and a min of 130 per class, the data is of very high quality, all of the images only contain 1 bird and in most of them the bird takes up around 50% of the pixels, also, the images have not been augmented and we kept it that way for multiple reasons the main reason being that our dataset is already very big and we didn’t have enough computational power to process all of the original images as well as the augmented.

The data is relatively balanced, with an average of 161 per class, a max of 263 and a low of 130, looking at the graph below we can see that it is quite balanced:



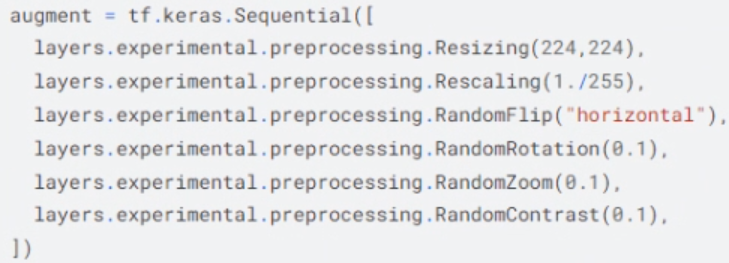
Here are some sample images:



As we can see, there are some birds that are relatively similar to one another (e.g., Fan Tailed Widow and the Fairy Bluebird) but there are also birds that are very distinct from the rest (e.g., Wild turkey and the Barn Owl).

Benchmark:

We found a [notebook](https://www.kaggle.com/code/vencerlanz09/bird-classification-using-cnn-efficientnetb0) that used the same dataset, they used weights from a pre-trained model (EfficientNetB0) and heavily augmented the data by flipping, rotating, zooming and changing the contrast



And this is their network structure:

A screenshot of a computer code

Description automatically generated

And these are their test results:

A white background with black text

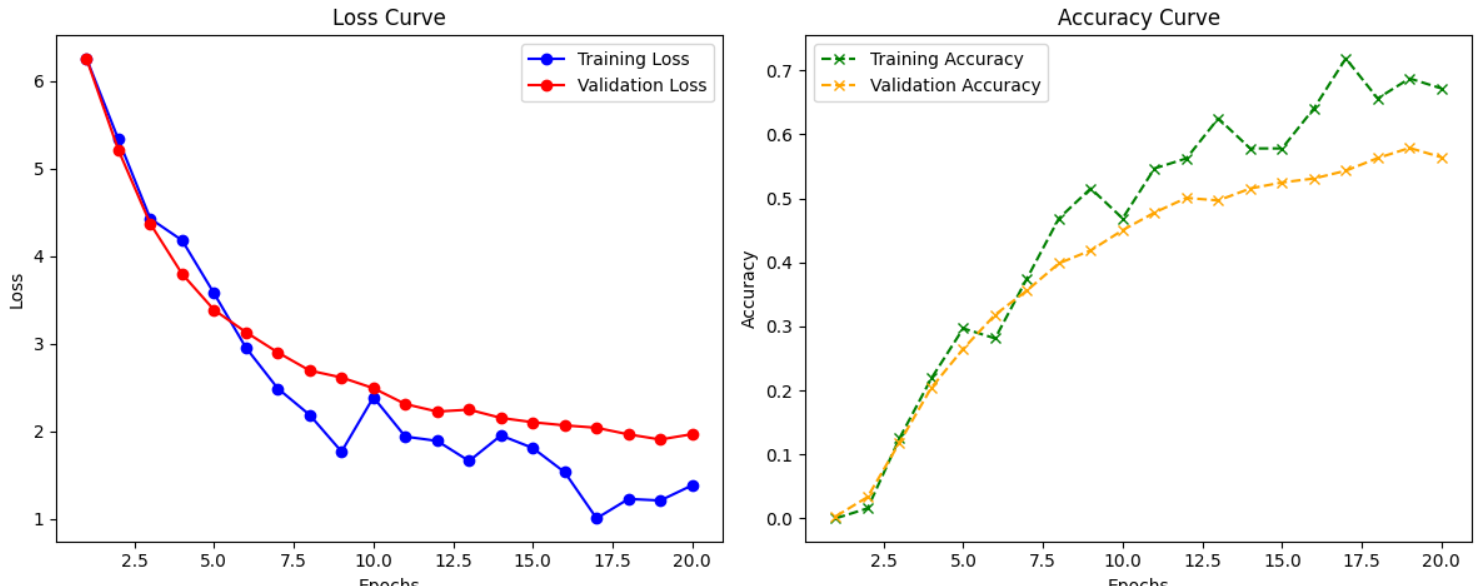
Description automatically generated

**Initial Implementation and Results:**

a.

After implementing our Convolutional Neural Network and training it on the train dataset with k-fold validation (k=5) these are our results:

**Fold 1:**

****

**Fold 2:**

**A comparison of graphs with numbers and symbols

Description automatically generated with medium confidence**

**Fold 3:**

**A graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of

Description automatically generated**

**Fold 4:**

**A comparison of graphs with red blue and green lines

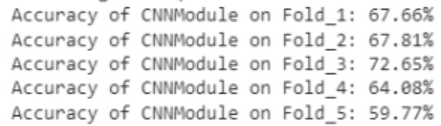
Description automatically generated**

**Fold 5:**

**A comparison of graphs with red and blue lines

Description automatically generated**

We ran all the folds on 20 epochs and got the following accuracies on the test dataset:



Average accuracy: 66.4%

Here are some good and bad classification examples:

|  |  |  |
| --- | --- | --- |
| Train images | Test images | Classification |
|  |  | Correct |
|  |  | Correct |
|  |  | Incorrect |

Our 3 suggestions for improvements:

1. Using batch normalization, since it helps in reducing internal covariate shift, thus making optimization more stable and accelerating training.
2. Learning rate scheduler, because an adaptive learning rate strategy can help the model converge faster and avoid overshooting or getting stuck in local minima.
3. Dropout, since it helps the model avoid overfitting by dropping out random neurons during training.

In the end, we chose to implement the batch norm and the learning rate scheduler and not dropout because by looking at the loss curve graphs from before, we saw that our model is not overfitting.

After the implementation, we suggest the following three:

1. Deepen the network, by deepening the network our model can pick up even more details.
2. Initialize our weights with weights from pre-trained networks.
3. Dropout, after deepening the network, our model might overfit, so adding dropout now will be much more beneficial.

**Results after Improvements:**

**Fold 1:**

**A graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of

Description automatically generated**

**Fold 2:**

**A graph of different colored lines

Description automatically generated with medium confidence**

**Fold 3:**

**A graph of different colored lines

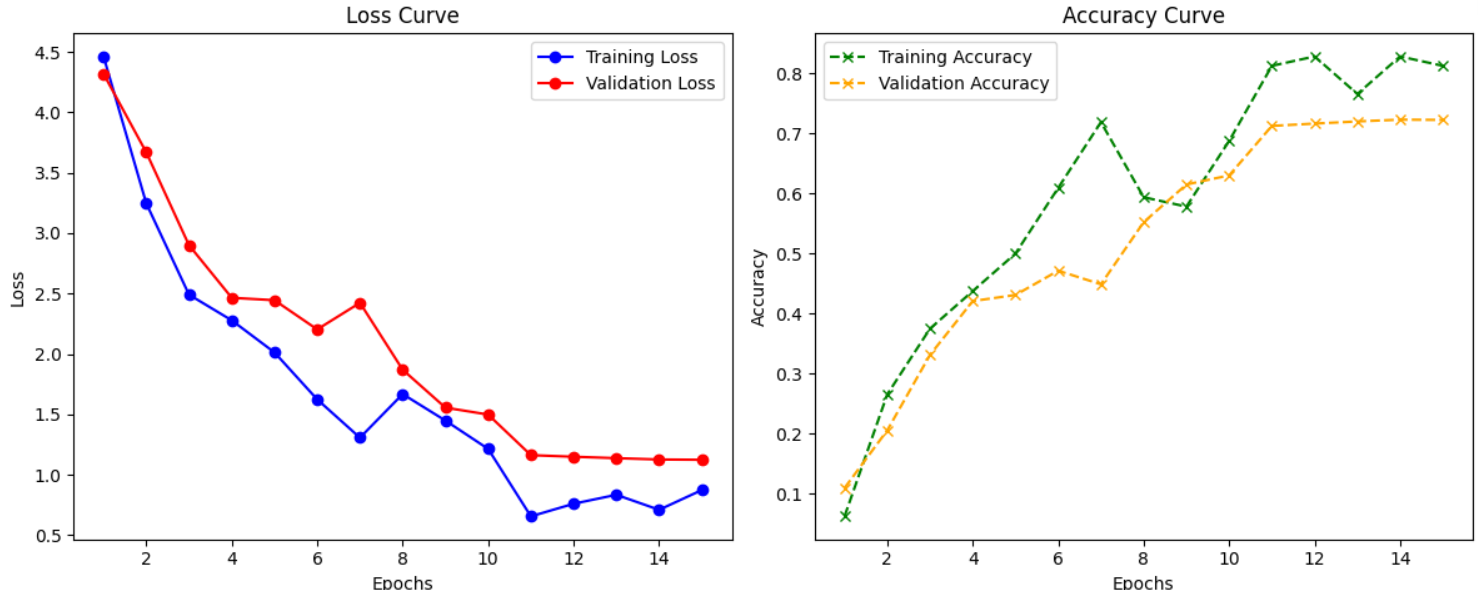
Description automatically generated with medium confidence**

**Fold 4:**

**A graph of different colored lines

Description automatically generated with medium confidence**

**Fold 5:**

****

And here are the accuracies:

A black background with white text

Description automatically generated

Average accuracy: 80.28%

Note: here we only ran 15 epochs per model because we couldn’t run them on 20 epochs like previously, however, we trained a model on 25 epochs and got a test accuracy of 90%.

Augmentation:

here are the augmentations that we ran on the test data.



And this is the resulting accuracy:



As you can see, the accuracy dropped by 6% because our training data was not augmented, so seeing a rotated bird or a flipped bird (not that kind of bird) was completely new to our model.

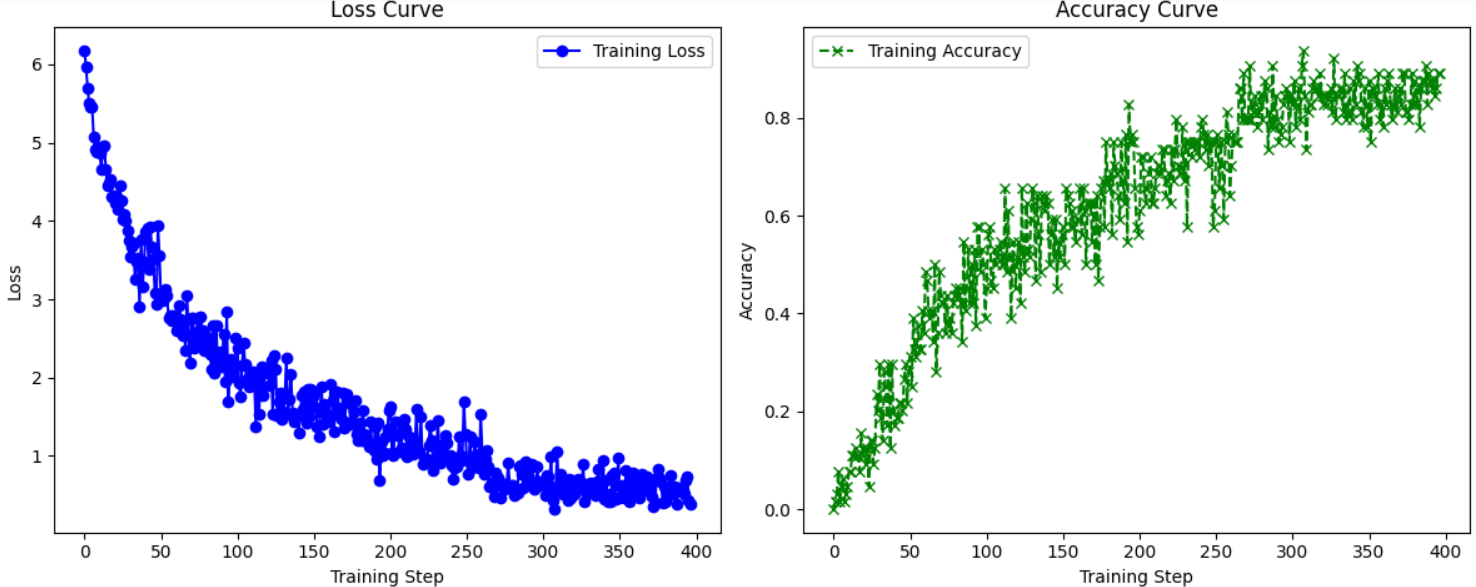
**Adding a new bird:**

For this section, we added 200 images of a new bird called “Common Kingfisher” to the train data from a dataset called [“Indian birds species image classification”](https://www.kaggle.com/datasets/ichhadhari/indian-birds).

Here is the bird:



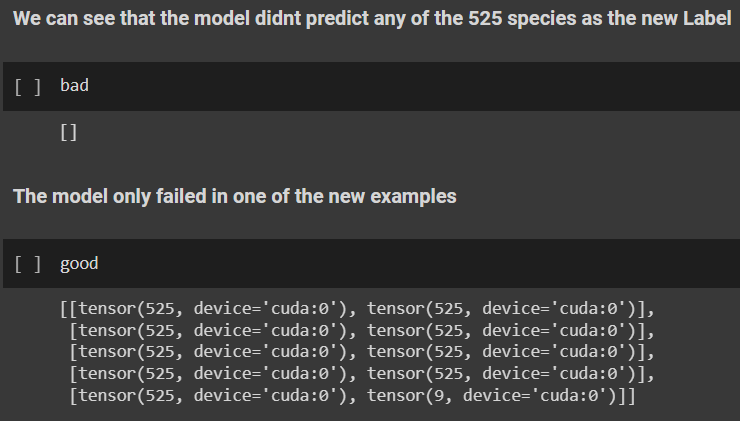
And here is the learning curve as well as the accuracy (only on train and not validation)



And this is the result on the test dataset with the added 5 images of the new kind



We can see that the model accuracy increased which is kind of surprising, but by digging deeper we found out the reason, and its because the model didn’t misclassify another bird as the new one, and it only failed to classify 1 of the new birds.



Deep learning models as feature extractors:

We ran each of the following models with epochs=10 & batch\_size=64

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Model Name | #parameters | Val loss | Val acc | Test loss | Test acc | # unique correct samples | # unique errors |
| 1. resnet18 – v1 | 11689512 | 0.7155 | 0.9006 | 0.282 | 09209 | 4 | 141 |
| 2. AlexNet | 59154765 | 2.3491 | 0.7573 | 1.012 | 0.7832 | 0 | 441 |
| 3.googleNet | 6624904 | 0.8805 | 0.8907 | 0.3533 | 0.9078 | 5 | 156 |
| 4.resNet34 | 21797672 | 0.6467 | 0.907 | 0.2344 | 0.9362 | 6 | 108 |

We chose two ML models: RandomForest and KNN. And chose googleNet neural network. So the googleNet sends 1024 features (one layer before the output) to the ML models  as input.

Results:

**Random Forest(max\_depth=10)**A black screen with white text

Description automatically generated

**KNN (n\_neighbors=30):**



We can see huge decrease in the performance for test and validation, but we knew that KNN would perform better because it will look for images with “similar” features (features extracted from googleNet)

3.e)

The only significant changes are changing the last layer, and manual hyperparameter tuning for the classic ML models. We tried to show confusion matrix of each model but they all looked the same because of the large number of classes.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Model Name | #parameters | Test loss | Test Acc | F1 score | Train Time |
| 1. resnet18 | 11689512 | 0.282 | 09209 | 0.9222 | 59 mins |
| 2. AlexNet | 59154765 | 1.012 | 0.7832 | 0.7648 | 49 mins |
| 3.googleNet | 6624904 | 0.3533 | 0.9078 | 0.91 | 62 mins |
| 4.resNet34 | 21797672 | 0.2344 | 0.9362 | 0.9348 | 64 mins |
| 5.randomForest | 0 |  | 0.376 | 0.2691 | 24 mins |
| 6. KNN | 0 |  | 0.7398 | 0.7 | 7 mins |

**All of the confusion matrix looked like this for 2 reasons:**

1. Very large number of classes to show in a heatmap
2. The test set size (2k samples) is very small compared to 525X525 matrix

