Bird species Image Classification

Project Overview: This project focuses on developing a computer vision model to classify 200 different bird species using the Caltech-UCSD Birds 200 Dataset. Specifically, the ResNet (Residual Neural Network) architecture aims to create an accurate and efficient classification system. Key aspects of the project include data preprocessing, model selection and training, performance optimization, and result visualization. The goal is to achieve high classification accuracy while providing insights into the model's decision-making process through various visualization techniques.

Additionally, I added two test case results for better explanation, incorporating more advanced techniques.

Objective :Develop a ResNet(Residual Network)-based model to classify 200 bird species from the Caltech-UCSD Birds 200 Dataset.

Dataset :

- Caltech-UCSD Birds 200 (link to the official website)
- 200 bird species
- Split:
    - For splitting the dataset into train,test and val , used 'split-folders' python library
    - Balanced the dataset into train/val/test.
    - Test case 1 With the normal images without augmentation.
    - Test case 2 - Added augmentation/oversampling technique to increase the dataset.(Increased for each class images closely 100)

Methodology :

- Model :
    - Used ResNet50(official keras ResNet documentation)
- Framework : Tensorflow
- Data Augmentation :
    - keras ImageDataGenerator(keras ImageDataGenerator documentation) created a custom script .
- Training strategies :
    - Base Model - Resnet50
    - Custom layers -
        - Global average pooling - to reduce the complexity/spatial dimensions of the feature map output by the base model
        - Dense layer - Customized the number of neurons and used the 'ReLU' activation function.

        - Dropout layer - Implemented to prevent overfitting.
        - L2 regularization - Added to improve generalization.
        - Output layer - Customized the number of neurons based on the number of classes (200 neurons) and used the 'softmax' activation function.
    - Transfer-Learning and fine tuning -
        - Unfreezing the layers - The last 15 layers (out of 50) of the base model (ResNet50) are unfrozen to allow fine-tuning (layers.trainable = True).
    - Compile and training -
        - Optimizer - Adam optimizer to adaptively adjust the learning rate during training.
        - Loss Function - Categorical-cross-entropy .
        - Metrics - Accuracy is used as the primary metric to evaluate model performance.
        - Epochs - 30(due to limited GPU resources)
        - Additional -Used hyperopt library for hyper parameter optimization..

Results :

- After the training,

```
376/376 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 261s 626ms/step -
accuracy: 0.9677 - loss: 0.3108 - val_accuracy: 0.8630 -
val_loss: 0.7985
```

- Accuracy with test data :

Test case 1 result -

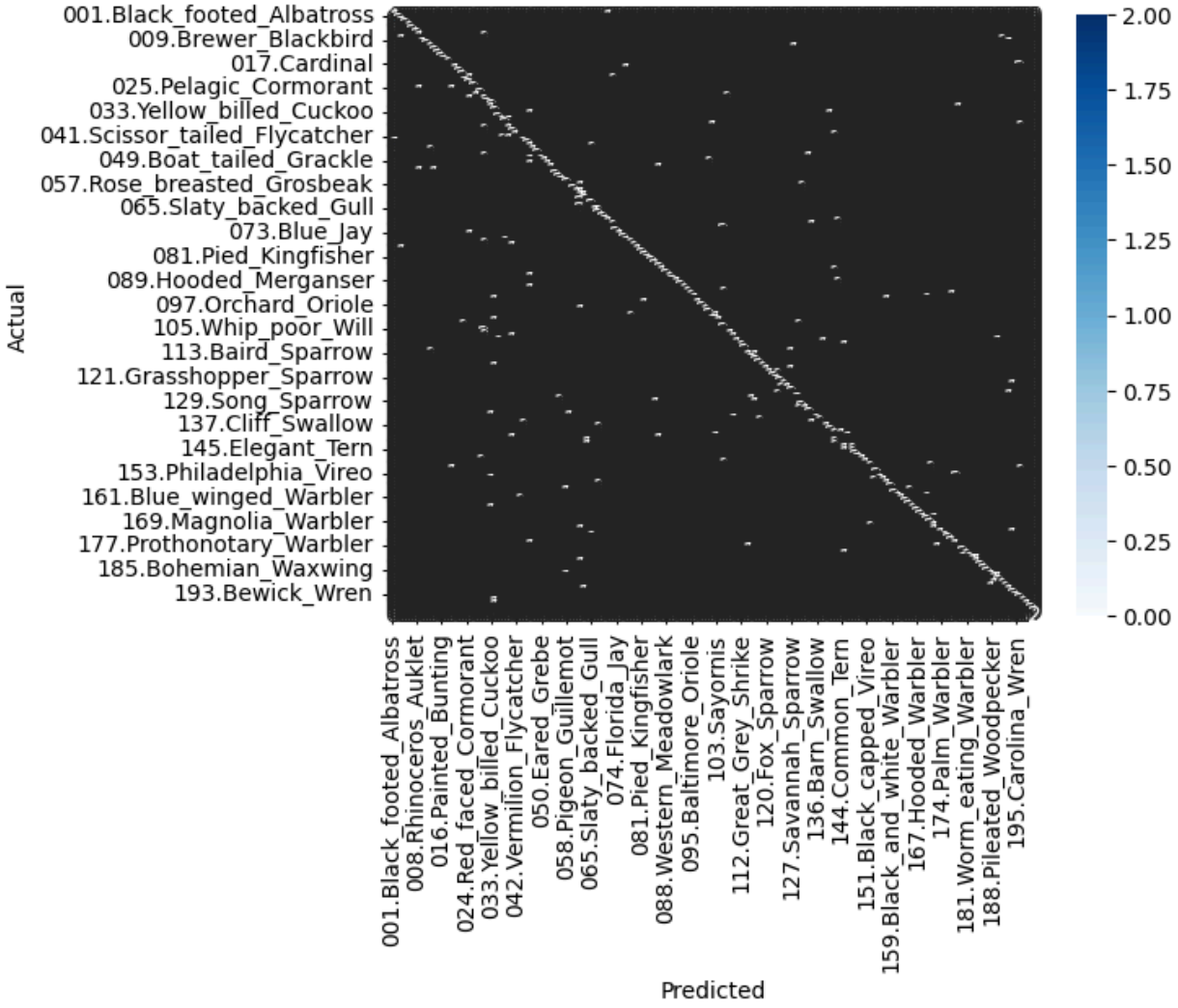
```
[ ] test_loss, test_acc = model.evaluate(test_generator,verbose=2)
    print('\n Test accuracy : ',test_acc)

398/398 - 8s - loss: 1.7464 - accuracy: 0.6030 - 8s/epoch - 20ms/step

 Test accuracy :  0.6030150651931763
```

Test case 2 result -

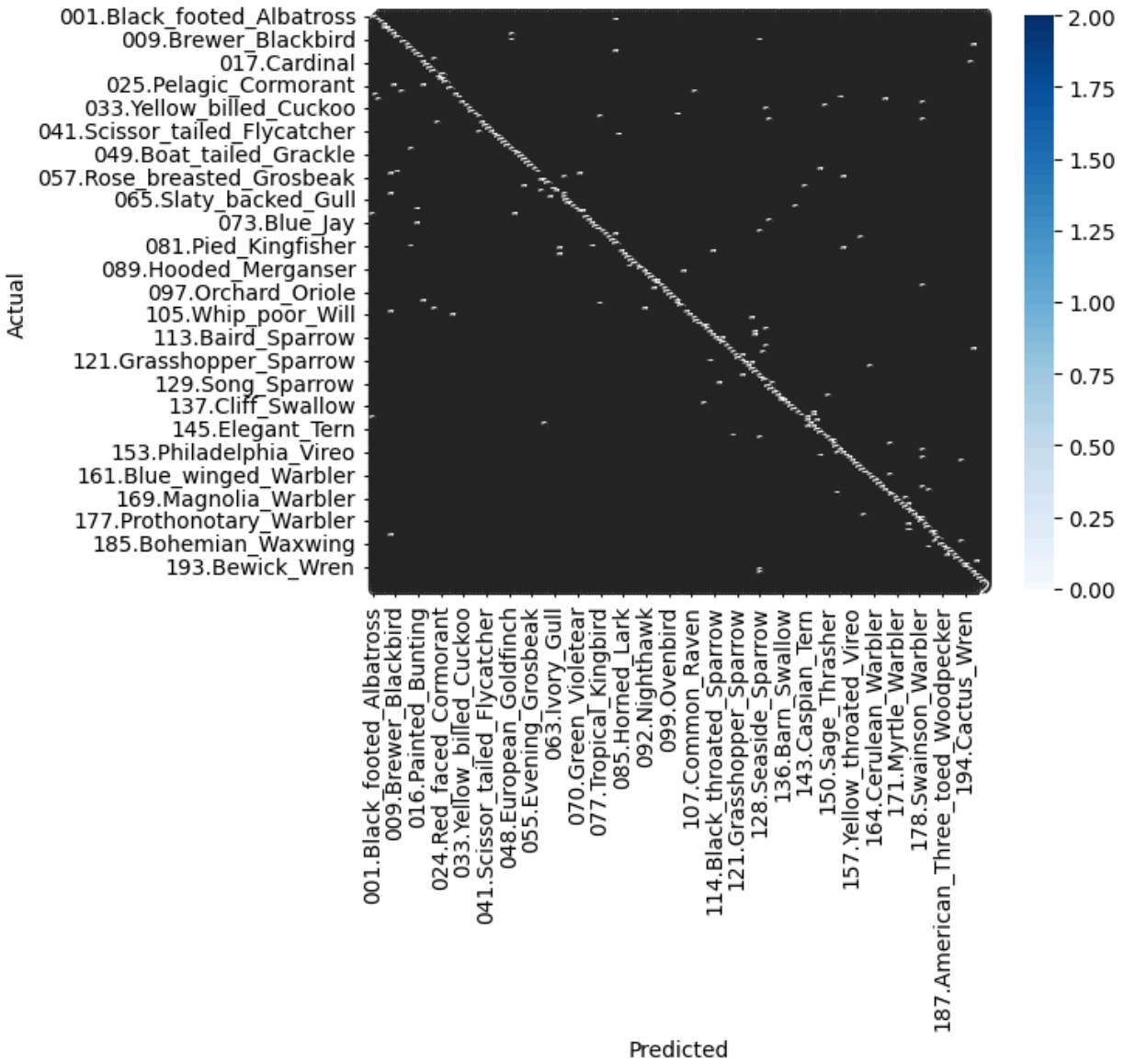
```
20s
[19] test_loss, test_acc = model.evaluate(test_generator,verbose=2)
     print('\n Test accuracy : ',test_acc)

397/397 - 14s - 36ms/step - accuracy: 0.6574 - loss: 1.8930

 Test accuracy :  0.6574307084083557
```

Visualization :

- Confusion matrix : Two test cases are presented, each with a confusion matrix visualizing the model's performance.
    - Test case 1, (before augmentation)
        - Confusion matrix csv link - [cm.csv]



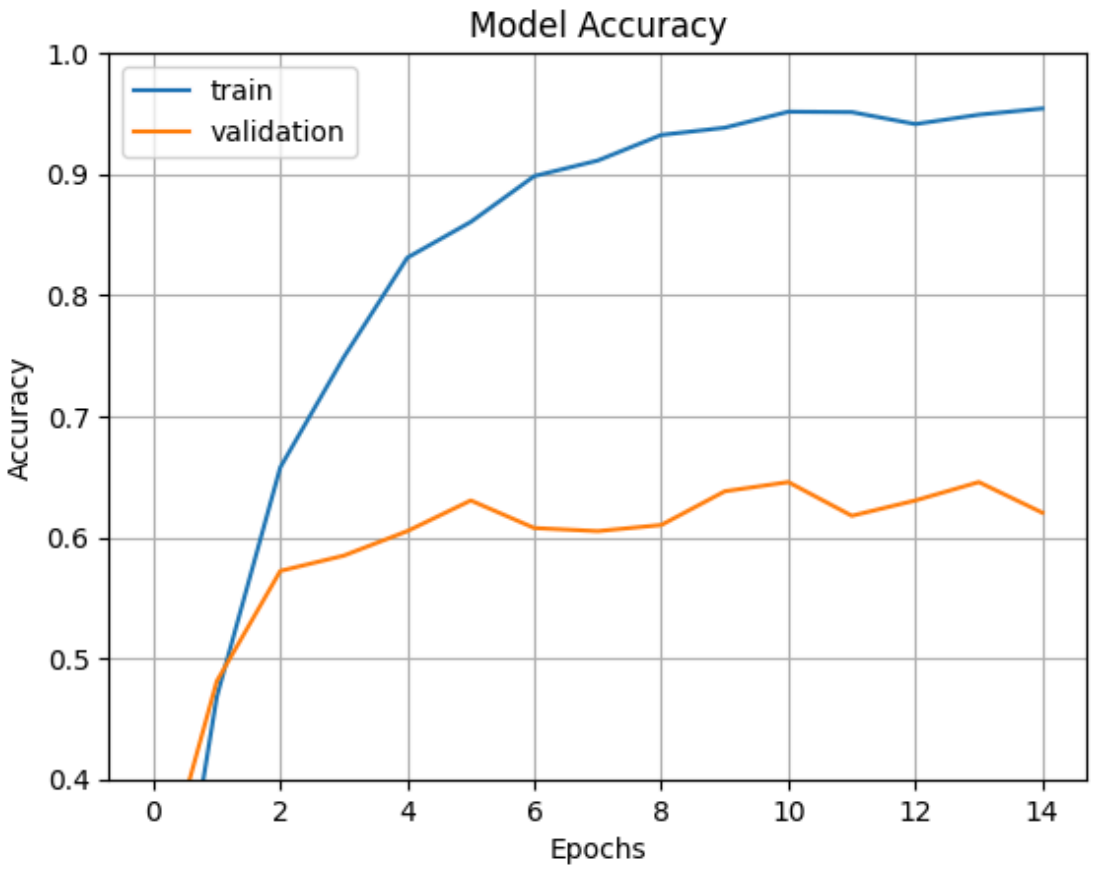
    - Test case 2 , after the augmentation, got a good amount of data for the train with resnet50. Also here added the L2 regularization method to prevent the overfitting.(but while researching the resnet architecture I understood that resnet architecture needs large amounts of data for better accuracy.)
        - Confusion matrix csv link -[cm.csv]



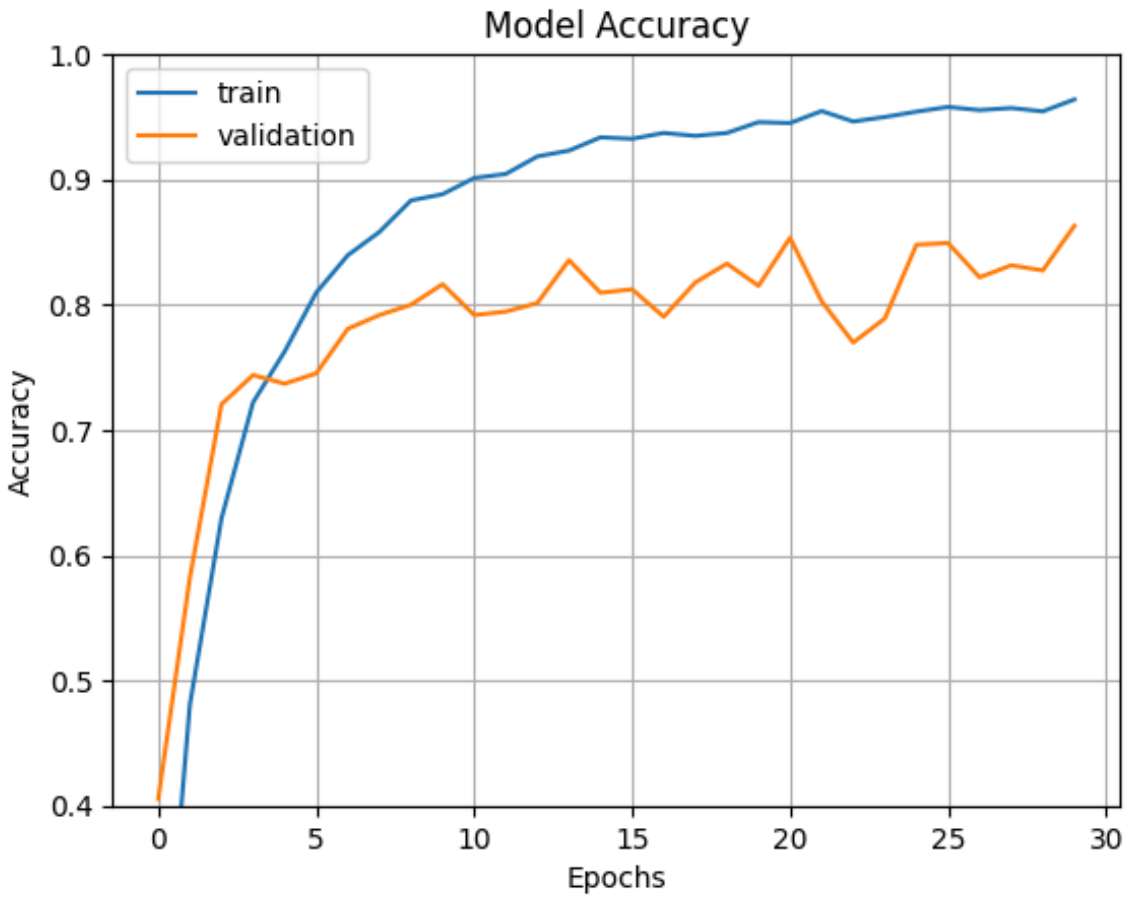
➔ Note :

1. Both matrices show a strong diagonal line, indicating good overall classification accuracy for many species.
2. Test case 2 (Image 2) appears to have more off-diagonal points, suggesting slightly more misclassifications compared to test case 1.
3. The color scale ranges from 0.0 to 2.0, likely representing the number or proportion of predictions.

- Accuracy graph:
  - Test case1,


Model Accuracy
train
validation
Accuracy
Epochs
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0
2
4
6
8
10
12
14

- Test case2

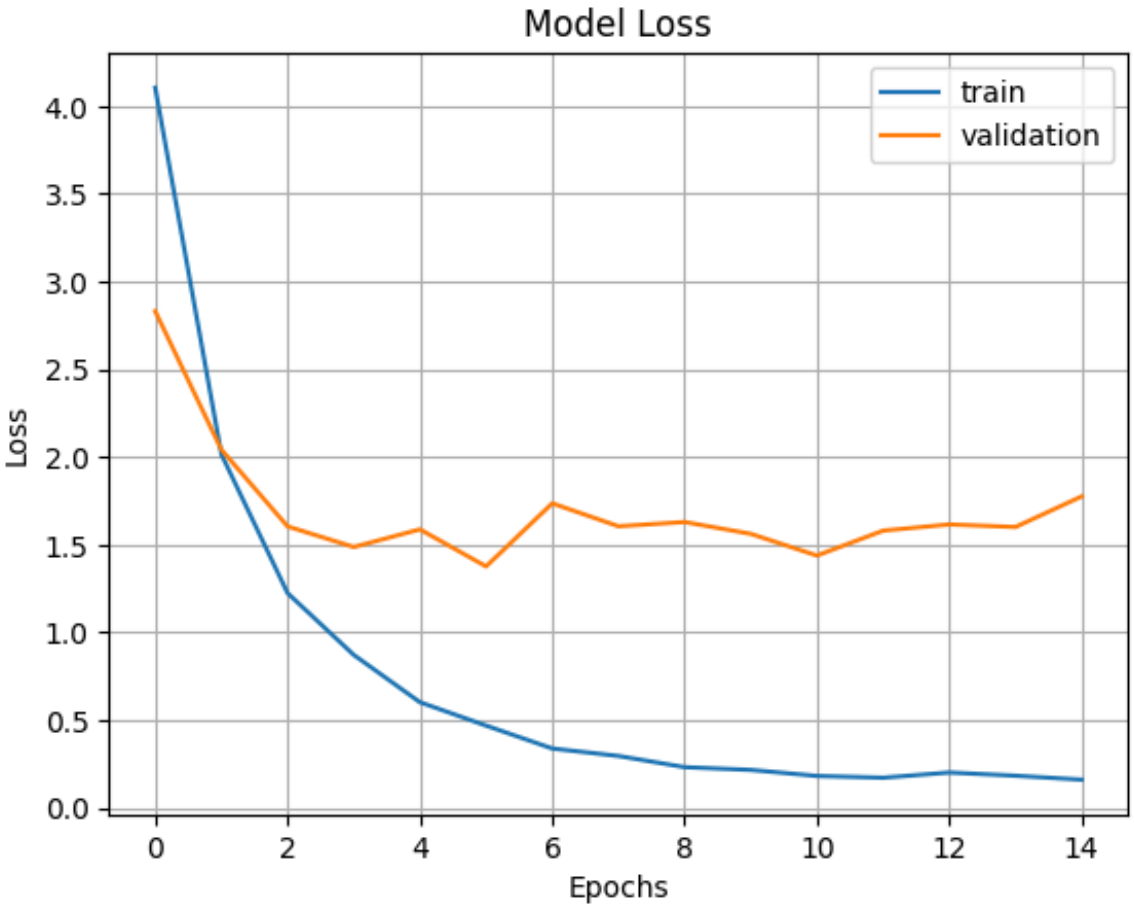


➔ Note :

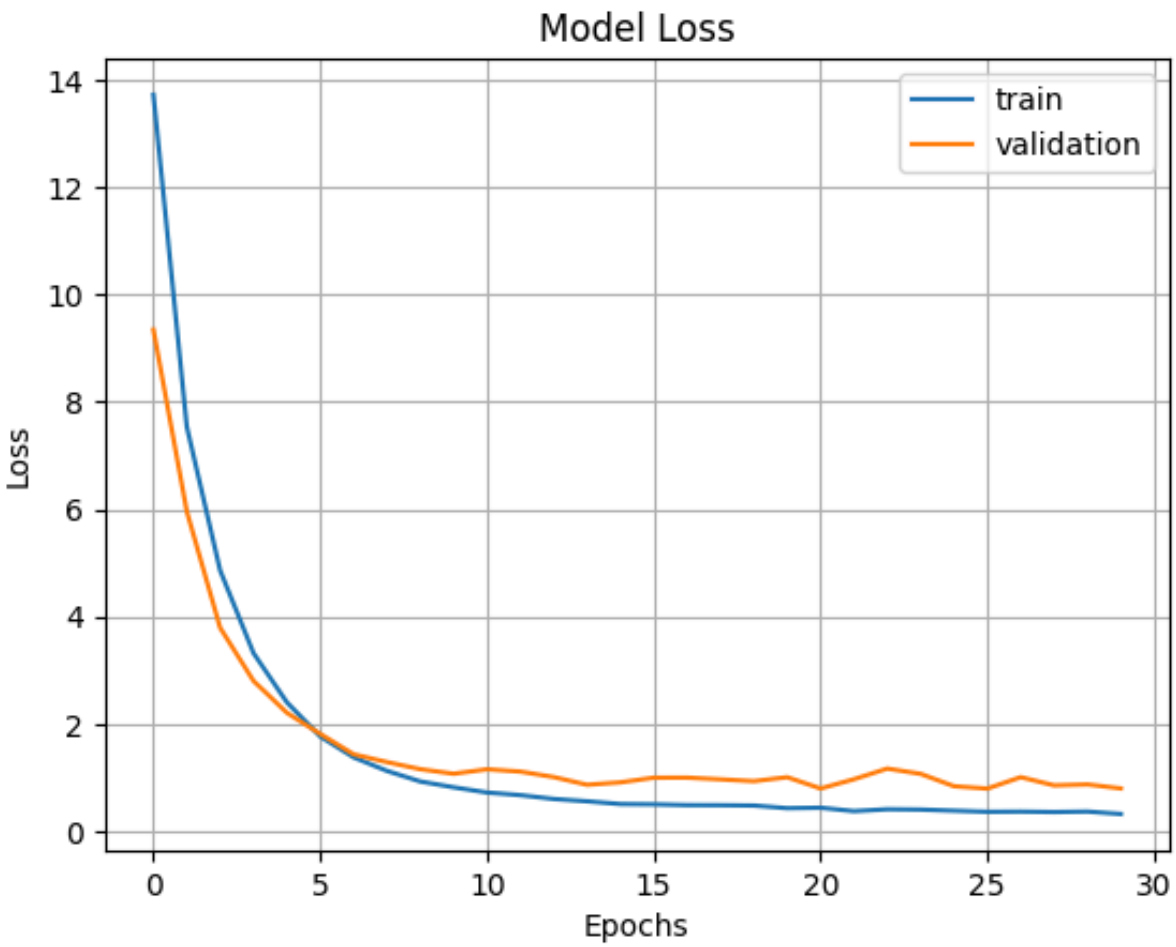
1. Training Accuracy:
    a. Both models show increasing training accuracy over epochs.
    b. Image 1: Reaches ~95% accuracy by epoch 14.
    c. Image 2: Achieves higher accuracy, approaching 97% by epoch 30.
2. Validation Accuracy:
    a. Image 1: Plateaus around 60-65% after initial improvement.
    b. Image 2: Significantly better, fluctuating between 80-87% in later epochs.
3. Overfitting:
    a. Image 1: Large gap between training and validation accuracy indicates overfitting.
    b. Image 2: Smaller gap suggests better generalization.
4. Model Performance:
    a. Image 2 shows superior performance in both training and validation accuracy.
    b. Image 2's model likely generalizes better to unseen data.

Error graphs:

- Test case 1:



- Test case 2 :



➔ Note :

1. Overfitting:
    a. Image 1: Large gap between training and validation loss indicates overfitting
    b. Image 2: Training and validation loss curves are closer, suggesting better generalization
2. Stability:
    a. Image 1: Validation loss fluctuates after epoch 4
    b. Image 2: Both losses stabilize after about epoch 10, with minor fluctuations
3. Final Loss Values:
    a. Image 1: Training loss near 0, validation loss around 1.5-2.0
    b. Image 2: Both training and validation loss converge to about 0.5-1.0
4. Overall, the model in Image 2 demonstrates better performance in terms of loss reduction and generalization. It shows a more consistent decrease in both

training and validation loss, with less indication of overfitting compared to the model in Image 1.

Implementation :

Created a custom Python script to use the trained model for classifying images. Added all the code files to the GitHub repository.

Trained model(download from here) - bird Species Classification.h5

Complete project - github repository link

Challenges faced :

- GPU - The Graphical Processing Unit is a major part of this project. The limitations are significantly affecting my work because, for training purposes, we need a minimum amount of GPU resources. I have used Google Colab, but the assigned GPU there is limited. As a result, I have created different accounts to access the GPU.
- Time management - Only after training and testing can we determine how the model performs, which allows us to make decisions for further tuning or new ideas. Here, completing one training session takes hours.
- Dataset - After researching, I realized that our given dataset is very small for the case of a Residual Neural Network. Therefore, after preprocessing, I needed to apply augmentation to balance and increase the dataset.

Future Improvements :

- Based on the resource availability we can improve the accuracy.