TDS 3301
Data Science Fundamentals

PROJECT

Birds Classifier

Prepared by

Mak Yen Wei, 1181203334, 018-9495849

1 Exploratory Data Analysis

1.1 Where to get the data?

The datasets comes from a repository of birds images that's compiled on a kaggle repository. The kaggle dataset link an be found here: https://www.kaggle.com/datasets/gpiosenka/100-bird-species. The original datasets contains 450 species of birds, each containing around 150 images of the species, however running such big datasets might consume quite a plenty of computing resources, while we don't have that, so we decided to use a smaller dataset which contains only 20 species of birds, each containing around 100 images of the species. The slicing of the datasets is made possbile by a python script that we've customly made for this purpose.

1.2 What is the data all about?

Our data set contains 20 bird species which included 3000 training images (150 images per species), 100 test images (5 images per species) and 100 validation images (5 images per species.) This is a very high quality dataset where there is only one bird in each image and the bird typically takes up at least 50% of the pixels in the image. As a result even a moderatly complex model will attain training and testing accuracies in the mid 90All images are 224 X 224 X 3 color images in jpg format. Data set includes a train set, test set and validation set. The data structure is convenient if you use the Keras ImageDataGenerator.flowfromdirectory to create the train, test and valid data generators. Images were gather from internet searches by species name. Once the image files for a species was downloaded they were checked for duplicate images using a python duplicate image detector program I developed. All duplicates detected were deleted in order to prevent their being images common between the training, test and validation sets.

After that the images were cropped so that the bird occupies at least 50% of the pixel in the image. Then the images were resized to 224 X 224 X3 in jpg format. The cropping ensures that when processed by a CNN their is adequate information in the images to create a highly accurate classifier. Even a moderately robust model should achieve training, validation and test accuracies in the high 90% range. Because of the large size of the dataset I recommend if you try to train a model use and image size of 150 X 150 X3 in order to reduce training time.

All files were also numbered sequential starting from one for each species. So test images are named 1.jpg to 5.jpg. Similarly for validation images. Training images are also numbered sequentially with "zeros" padding. For example 001.jpg, 002.jpg ....010.jpg, 011.jpg .....099.jpg, 100jpg, 102.jpg etc. The zero's padding preserves the file order when used with python file functions and Keras flow from directory.

The 20 types of birds can be listed here: American Redstart , Barred Puffbird, Belted Kingfisher, Cape May Warbler, Cream Colored Woodpecker, Crested Caracara, Eastern Meadowlark, Eastern Towee, European Goldfinch, Hoatzin, Hoopoes, Iberian Magpie, Imperial Shaq, Kookaburra, Red Bellied Pitta, Rufuos Motmot, Taiwan Magpie, Tree Swallow, Violet Green Swallow, White Tailed Tropic

1.3 How to visualize the content?

The visualtion of the data we've looked into is the number of images in each 20 classes. It is important to take note of the number of images in each column to be able to understand

what sorts up datasets we're working with and how we should treat the dataset. For example, if the dataset is having a class imbalance problem, meaning number of images in all of the classes are not around the same, we should consider using some sort of data augmentation techniques to balance out the dataset. In our case, we can see that the distribution is quite balanced, but we figured having a perfectly balanced dataset is in fact a good idea. Therefore, in the next section, we'll be talking about how we've augmented the dataset to make it more balanced.

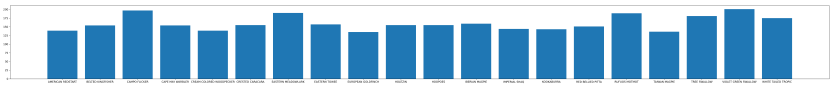

Figure 1: Bar Plot of images count for each class

The bar plot above shows the number of images in each class. As you can see, the distribution is quite balanced, but we figured having a perfectly balanced dataset is in fact a good idea. Therefore, in the next section, we'll be talking about how we've augmented the dataset to make it more balanced.

1.4 What data transformation to the dataset?

The only data transformation part of the project is shrinking the 450 species birds images into just 20 classes. The reason for this is because we don't have enough computing resources to run such a big dataset. The slicing of the datasets is made possbile by a python script that we've customly made for this purpose.

1.5 What is the brightness distribution of the dataset?

Investigating the brightness of the images in our dataset gives us a better understanding of the dataset.

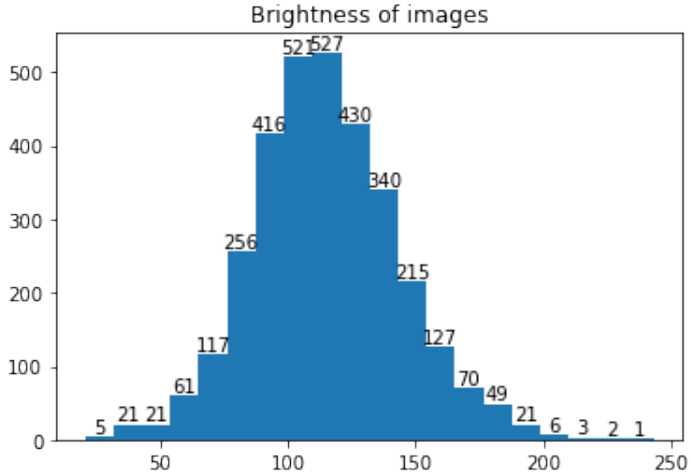

Figure 2: Brightness Distribution

We can see that the brightness of the images are quite similar, which can be a good thing, and also a bad thing. The good thing is that we don't have to worry about the brightness of the images being too different from each other, The bad side of it is that

the model will not be able to handle the brightness of the images that are too different from the training images. To counter this issue, there are several methods we can use, and in our finding, we used the ImageDataGenerator class from Keras to generate more images with different brightness, this is called data augmentation and will be explained in detailed in the later section

1.6 What is the color distribution of the dataset

The color distribution of the dataset is also important to look into. The reason is because the color of the images can affect the model's performance. For example, if the model is trained on images with a lot of green, and then it is tested on images with a lot of red, the model will not be able to perform well. Another point of view is that if the color distribution between different classes are found to be unique and distinctive, that can help us to improve the model's performance by doing feature selection.

The following plot is to distinguish whether color is a telling factor in the classification of the birds. In the below exmaple we plotted 5 images of American redstart.

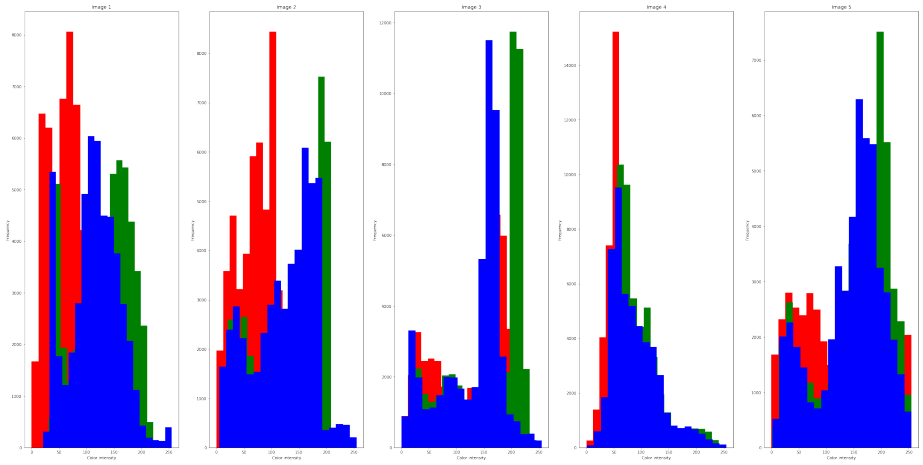

Figure 3: Color distribution

In this case, we see that for american redstart, the blue channel tend to spike up, this might be a telling factor in the classification of the birds. Let's look at more plots to really solidifies our understanding of the color distribution of the dataset.

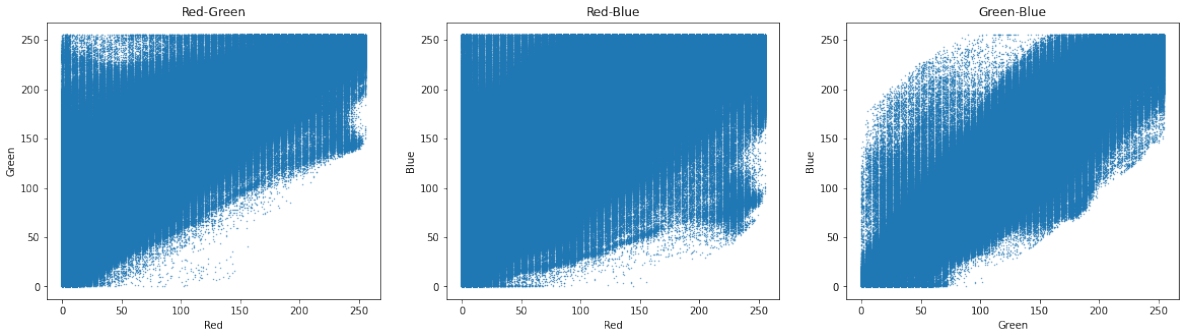

Figure 4: Cream Coloured WOODPECKER

As you might tell, the scatter plot of the color distribution of the images are quite similar to each other. This means that there are no distinctive color distribution between

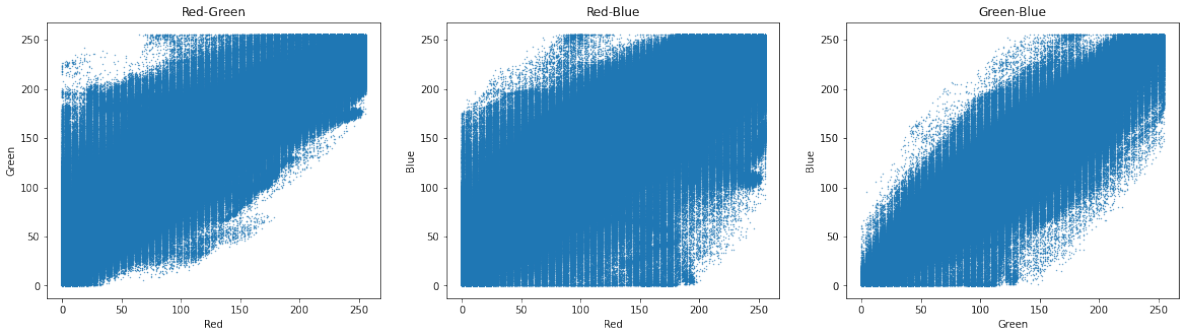

Figure 5: KOOKABURRA

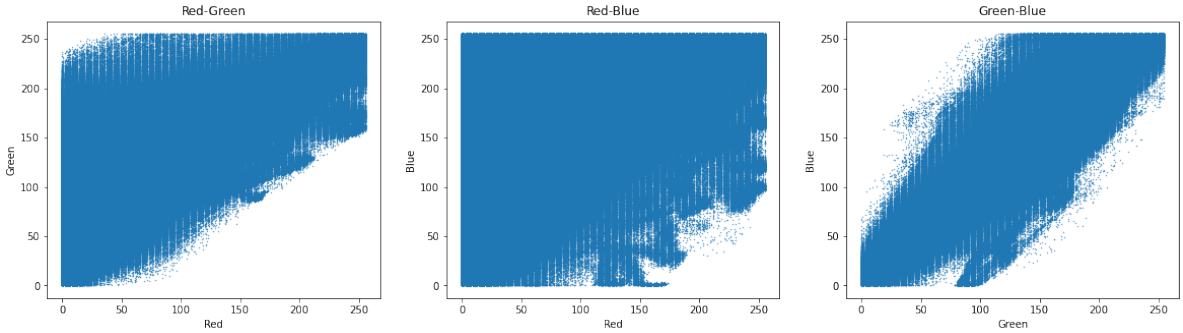

Figure 6: Campo Flicker

the classes which tells us that the color of the images are not a telling factor in the classification of the birds.

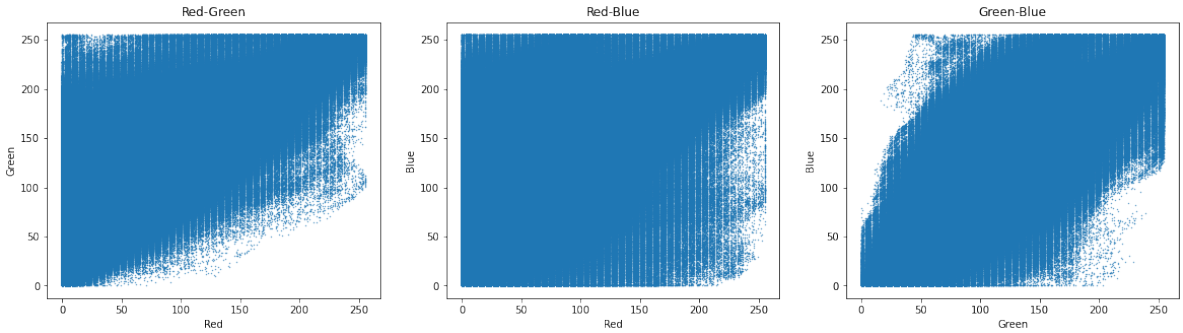

Figure 7: Rufous Motmot

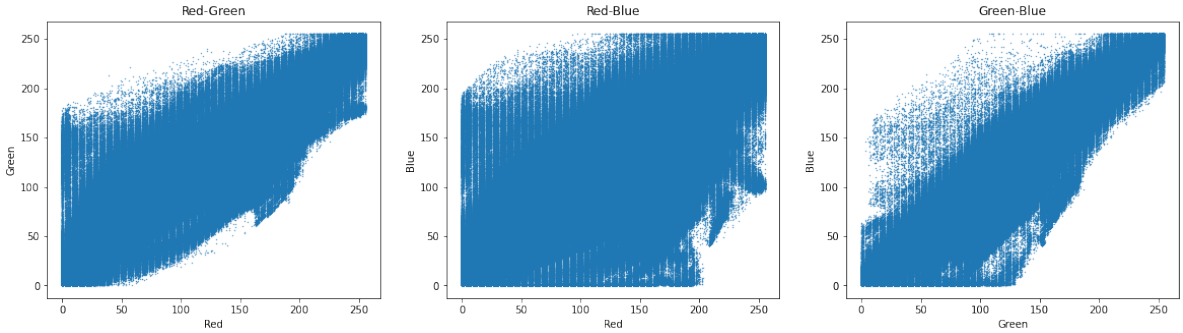

Figure 8: White-tailed Tropicbird

1.7 Size of the image?

Next up, we look into the size of the images. Below we are plotting the size of one of the images in the dataset. The images are all in the same size, which is 224x224. This is because we are using the VGG16 model, which is a pre-trained model that is trained on images of size 224x224. However, if we are training it on the original size, you can see that the images are quite big, and it will take a lot of time to train the model. The bigger the image the larger the details, and the more time it takes to train the model.

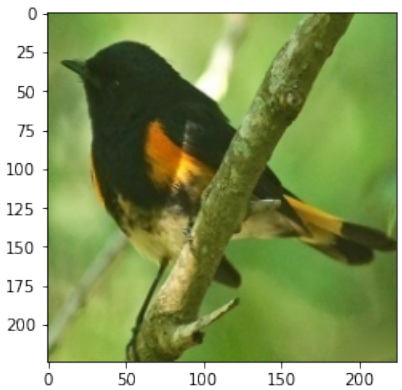

Figure 9: Sizes of the images

In order to counter this issue, we attempt to use a gaussian blur filter to blur the images, and then resize it to 224x224. The reason for this is because we want to reduce the details of the images, and also reduce the size of the images.

Figure 10: Gaussian blur comparison

The above figure shows a comparison between the original image and the blurred image. As you can tell, the details had been reduced by a lot. The gaussian distribution is a bell-shaped curve, and the center of the curve is the mean, and the width of the curve is the standard deviation. The standard deviation is the measure of how spread out the numbers are. The higher the standard deviation, the more spread out the numbers are. In this case, we are using a standard deviation of 5, which means that the numbers are quite spread out.

1.8 What can we predict, and how can we do that?

The next step is to look into the dataset and see what we can predict. In this case, we are predicting the bird species, we'll be looking at 3 models which are VGG16, VIT and lastly MobileNet.

2 Pre-Processing Methods.

2.1 Rescale, Normalization

Before we use the data for model training and inference, we can implement some pre-processing methods which are the steps taken to format the images. Firstly, for normalisation, we have to rescale the training data, validation data as well as test data.We rescale all of the data with the value 1/255. The reason is that our original images consist of RGB coefficients in the 0-255, but such values would be too high for our models to process (given a typical learning rate), so we target values between 0 and 1 instead by scaling with a 1/255 factor.

2.2 Data Augmentation

Data augmentation is a well-known technique to artificially create new training data from existing training data. It is useful to enhance performance and outcomes of a machine learning model by creating new and different training examples. For instance, image resizing, image rotation, image flipping and so on. Therefore, we are artificially expanding the available dataset for training the deep learning models which are all of the selected models. Additionally, we apply the technique only on training data not on validation data, nor on testing data.

2.3 Flipping, Rotation, Zooming, Brightness

The data augmentation methods we are using is ImageDataGenerator in Keras. We have implemented a few augmentation techniques which are flipping, rotation and zooming. First of all, we set horizontal_flip and vertical_flip as true which let the image randomly flip half of the images horizontally and vertically. Secondly, image rotation allows the model to become invariant to the orientation of the object. In this case, we rotate the image by setting rotation_range value between 0 to 180 degrees. Thirdly, the zoom augmentation either randomly zooms in on the image or zooms out of the image. For that, we set the zoom_range with the value 0.1 to let it zoom inside the image. Lastly, we set brightness_range for brightening or darken bird images. The value is set between 0.1 to 0.9.

The following is a plot of the augmented images. The original images are on the left and the augmented images are on the right. As you can see, the augmentation technique introduces skewness, rotation, brightness varities to the image. This will tremendously helps the model to learn the features of the images better and fit edge cases better.

Figure 11: Augmented Images

3 Model Construction and Comparison

Models Used

We have used 3 models which are VGG16, VIT and MobileNet. The reason for using these 3 models is because they are all pre trained models, and they are all state of the art models. The VGG16 model is a convolutional neural network that is trained on more than a million images from the ImageNet database. The VIT model is a vision transformer model that is trained on more than a million images from the ImageNet database. The MobileNet model is a convolutional neural network that is trained on more than a million images from the ImageNet database.

Model Training

The model is primarily trained on 2 different metrics, Learning rate of 0.001 and 0.0001. All models are trained on epoch of 30 and the batch size of 32

Hyperparemeters Tuning

The hyperparameters that we are tuning are the learning rate, the optimizer, the activation function, the loss function and the metrics. The learning rate is the step size at each iteration while moving toward a minimum of a loss function. The optimizer is the algorithm used to update the weights of the neural network. The activation function is the function that is applied to the weighted sum of the inputs of a node. The loss function is the function that is used to evaluate a set of weights. The metrics is the function that is used to judge the performance of the model.

To ease our work, we came up with 2 functions, a function to tune the learning rate as the number of epoch passes, and a early stopping function. The learning rate function is used to reduce the learning rate as the number of epoch passes. The early stopping function is used to stop the training process when the validation loss is not decreasing anymore. To further understand the learning rate schedule, we first talk about the issue if you have a high learning rate, and a low learning rate. If you have a high learning rate, the model will converge faster, but it will not be able to find the global minimum. We can see that in the plot below

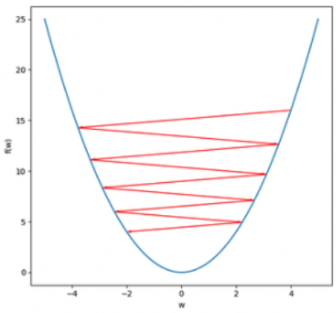

Figure 12: Learning rate Plot

If you have a low learning rate, the model will converge slower, but it will be able to find the global minimum, which is a better way to train the model. We can see that in the plot below

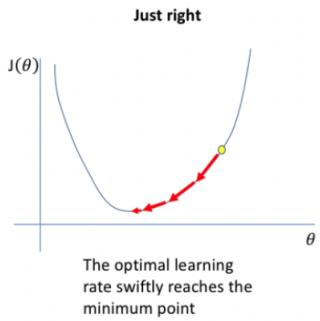

Figure 13: Learning rate Plot

On the other hand, early stopping is a method that allows you to specify an arbitrary large number of training epochs and stop training once the model performance stops improving on a hold out validation dataset. The early stopping function is used to stop the training process when the validation loss is not decreasing anymore.

Model Evaluation

VGG16

VGG16 is a convolutional neural network that is 16 layers deep. The pretrained network can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals. The network has an input image size of 224-by-224.

We can compare the model on different learning rates of 0.001 and 0.00001

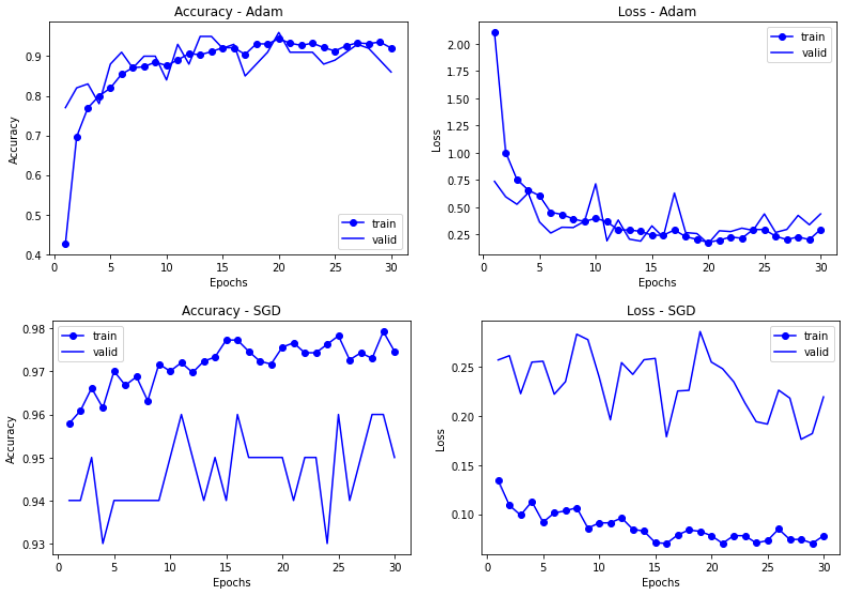

Figure 14: Adam, SGD Acc, Loss - Learning rate Comparison

The above plots are trained on the learning rate of 0.001

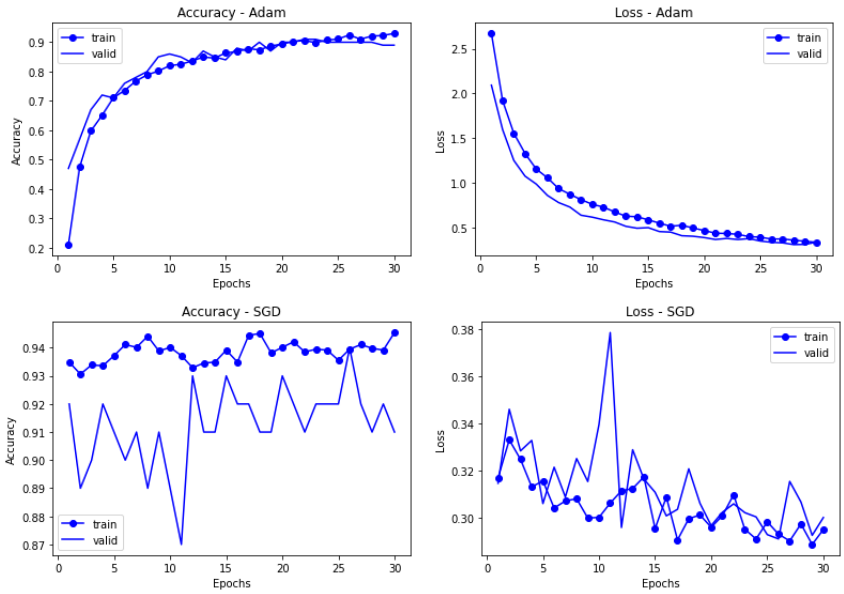

Figure 15: Adam, SGD Acc, Loss - Learning Rate Comparison

The above plots are trained on the learning rate of 0.0001.

As you can tell from the plot, the model with the learning rate of 0.001 is able to converge faster than the model with the learning rate of 0.0001. The model with the learning rate of 0.001 is able to achieve a higher accuracy than the model with the learning rate of 0.0001. The model with the learning rate of 0.001 is able to achieve a lower loss than the model with the learning rate of 0.0001. Also, the model using the Adam optimizer is able to achieve a higher accuracy than the model using the SGD optimizer.

3.0.1 Model Performance Table — VGG16

	Optimizer	Best accuracy	Best Accuracy Epochs	Best loss	Best Loss Epochs
0	SGD	0.96	10	0.17	27
1	ADAM	0.96	19	0.17	19

Figure 16: Model Performance Table Using VGG16

Mobile Net

Mobile Net is a convolutional neural network that is 16 layers deep. The pretrained network can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals. The network has an input image size of 224-by-224.

We can compare the model on different learning rates of 0.001 and 0.00001

The above plots are trained on the learning rate of 0.001.

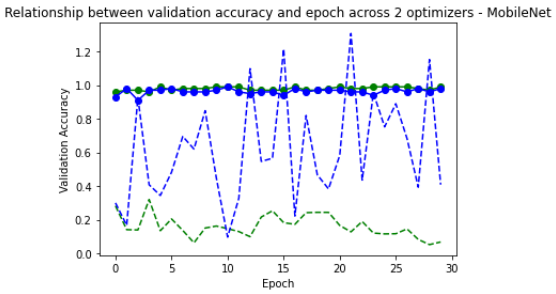

Figure 17: Adam, SGD Acc, Loss - Learning Rate Comparison

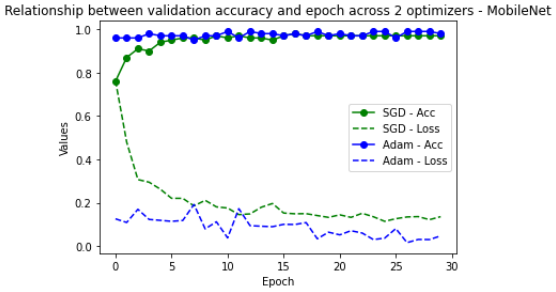

Figure 18: Optimizer Comparison using VGG16

The above plots are trained on the learning rate of 0.0001,

The plots shows us that the learning rate of 0.0001 gives us a stable and smooth decent to the global minima, and both of the optimizers worked really well, achieveing a high accuracy and low loss.

3.0.2 Model Performance Table — Mobile Net

	Optimizer	Best accuracy	Best Accuracy Epochs	Best loss	Best Loss Epochs
0	SGD	0.98	16	0.11	24
1	ADAM	0.99	10	0.015	26

Figure 19: Model Performance Table Using Mobile Net

VIT

The ViT is a visual model based on the architecture of a transformer originally designed for text-based tasks. The ViT model represents an input image as a series of image patches, like the series of word embeddings used when using transformers to text, and directly predicts class labels for the image.

With what we can understand from the above graphs, VIT is by far the best model we've tested, with a high accuracy and low loss, and both of the optimizers worked really well.

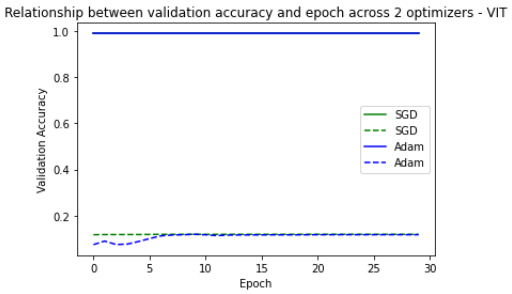

Figure 20: Adam, SGD Acc, Loss

3.0.3 Model Performance Table — VIT

	Optimizer	Best accuracy	Best Accuracy Epochs	Best loss	Best Loss Epochs
0	SGD	0.99	0	0.05	1
1	ADAM	1.0	1	0.009	1

Figure 21: Model Performance Table Using VIT

Conclusion

After implementation of the model, we can evaluate it by comparing the resulting accuracy and loss. With computer resources as a concern in mind, in our opinion, we'll look more towards the Best Validation accuracy and also the number epoch needed to achieve the results. With that being said, Vision Transformer (ViT) will be the best model for the image classification of bird species in this project. MobileNet will be ranked in second place which means it is better than VGG16 but worse than Vision Transformers (ViT). Lastly, VGG16 will be the last option which has the worst performance in this project.

4 Deployment

4.1 How to connect to your web API?

API is hosted here publicly: https://birds-classifier-api.herokuapp.com/

The API is hosted using a framework called FastAPI. We've extracted the best model — VIT and saved it using joblib. The API is able to take in a image and return the prediction of the bird species.

However the api endpoint might not always be available to serve requests, if you are interested in using the API, this is due to the heroku is not a free product, and we are not able to keep it running 24/7.

4.2 Hosted streamlit?

Streamlit is hosted here publicly: https://shaunmak-birds-classifier.streamlit.app/

The streamlit application contains all of our findings in a more detailed format, feel free to look it through.

4.3 Do I have real-time prediction and visualization?

Yes both API and Streamlit provides real time prediction but only streamlit application provides visualization

4.4 How should I improve the performance? What strategy?

There are primarily three things we hope to improve in the future.

As we know, one of the strengths of Transformer models is their flexibility to scale to high parametric complexity. Scaling up on compute, model and size of training samples improves performance only large models can benefit from more training data, and the performance of smaller models plateaus quickly and cannot leverage from additional data. This indicates that large scale models have the capacity to further enhance their representation learning capabilities. However, with the current designs, scaling upon Transformer models is expensive and compute prohibitive, thus an efficient design is required.

We would like to suggest that it reduces the high complexity of Transformer models. For example, explore selective or sparse attention to previous layer tokens while updating each next layer token. The Reformer model employed locally-sensitive hashing (LSH) to minimise the complexity of self-attention from O(n2) to O(nlog(n)). In similar pursuit, the recent Lambda Networks propose to model local context as a linear function which helps reduce complexity of self-attention.

Thirdly, we might face the problem of the object not always being in the center of the image, hence we can shift the pixels of the image either horizontally or vertically to overcome it. This can be done easily with the API which keras provides, namely "ImageDataGenerator", by adding a parameter called "shift".