

## COS30082 - Applied Machine Learning Bird Species Classification

---

### Contents

<b>1</b>	<b>Methodology</b>	<b>2</b>
1.1	Resize Image . . . . .	2
1.2	Model Architecture . . . . .	4
1.3	Model Discussion . . . . .	5
1.4	Normalize Training Images . . . . .	5
1.5	Hyperparameters . . . . .	5
<b>2</b>	<b>Result</b>	<b>6</b>
2.1	Loss and other Metrics . . . . .	6
2.2	Evaluation Metric . . . . .	7

# 1 Methodology

## 1.1 Resize Image

Firstly, I will resize the dataset so that every image has a size of  $(224 \times 224)$ . The Python below showing how I resized the whole dataset and save them in Google Drive (Because I perform Training on Google Colab).

```
1 train_data_dir = '/content/drive/My Drive/COS30082/Assignment1/Dataset/Train'
2 output_dir = '/content/drive/My Drive/COS30082/Assignment1/Dataset/Train_Resized'
3
4 # Create the output directory if it doesn't exist
5 os.makedirs(output_dir, exist_ok=True)
6
7 # Loop through all files in the source directory
8 for filename in os.listdir(train_data_dir):
9     if filename.endswith(('.png', '.jpg', '.jpeg', '.bmp', '.gif')): # Check for image file
10         extensions
11         # Open an image file
12         with Image.open(os.path.join(train_data_dir, filename)) as img:
13             # Resize the image to IMAGE_SIZE x IMAGE_SIZE
14             img = img.resize((IMAGE_SIZE, IMAGE_SIZE))
15             # Save the image to the output directory with the same name
16             img.save(os.path.join(output_dir, filename))
17
18 print("Resizing complete!")
```

In Google Drive, if I check randomly 2 images, they will have the same shape as Figure 1 and 2

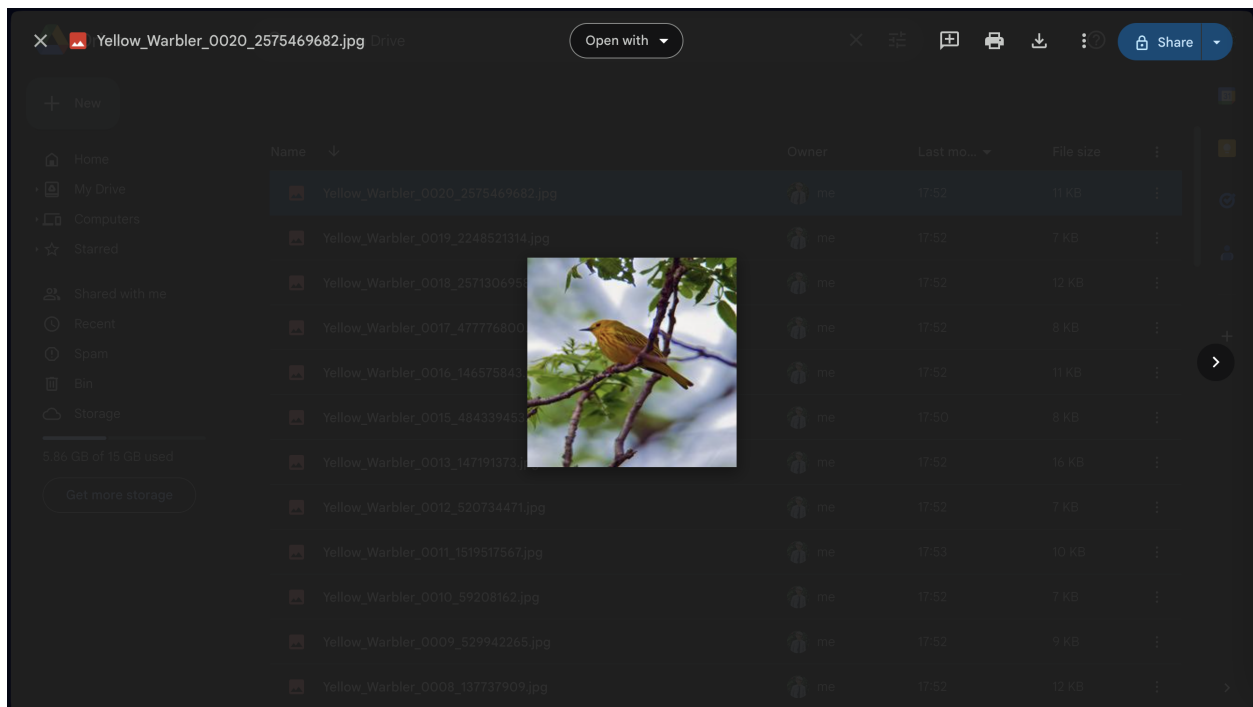


Figure 1: Sample Image 1 with the shape of  $(224, 224, 3)$

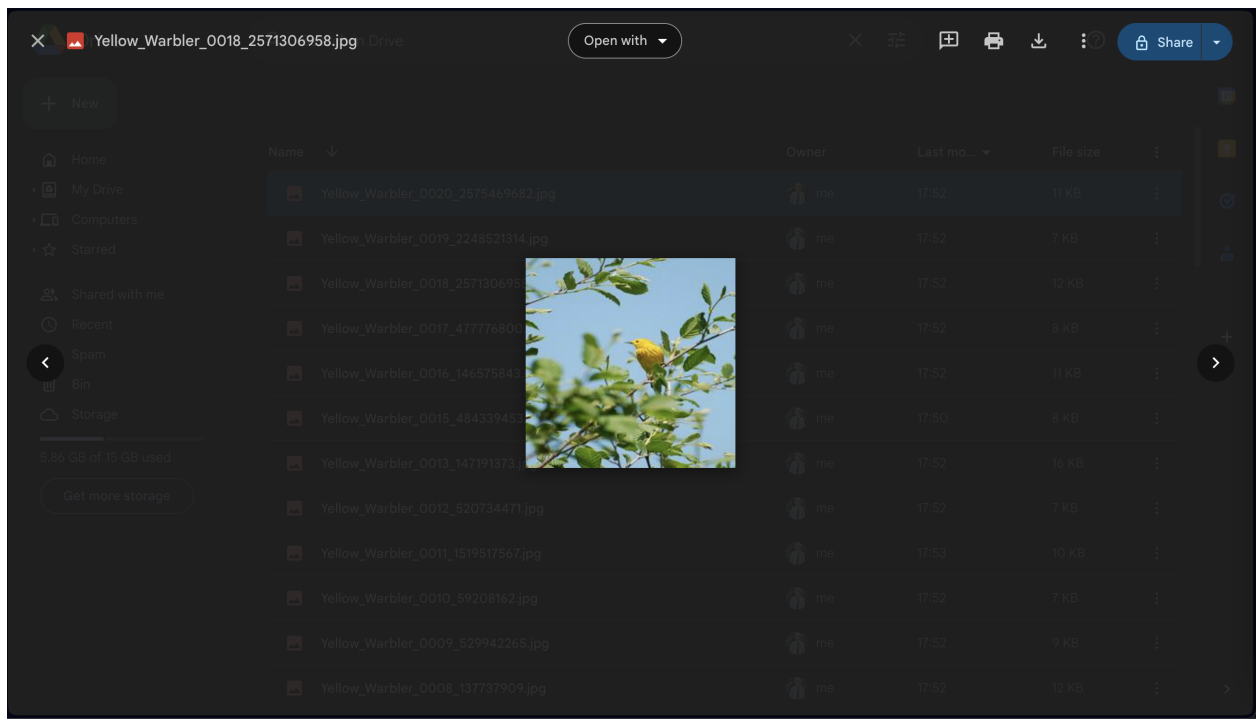


Figure 2: Sample Image 2 with the shape of (224, 224, 3)

## 1.2 Model Architecture

I decided to use the MobileNetV2 as a base mode. Then, I will add some fully connected layers and drop-out layers to perform fine-tuning technique. Below is the overview of the model.

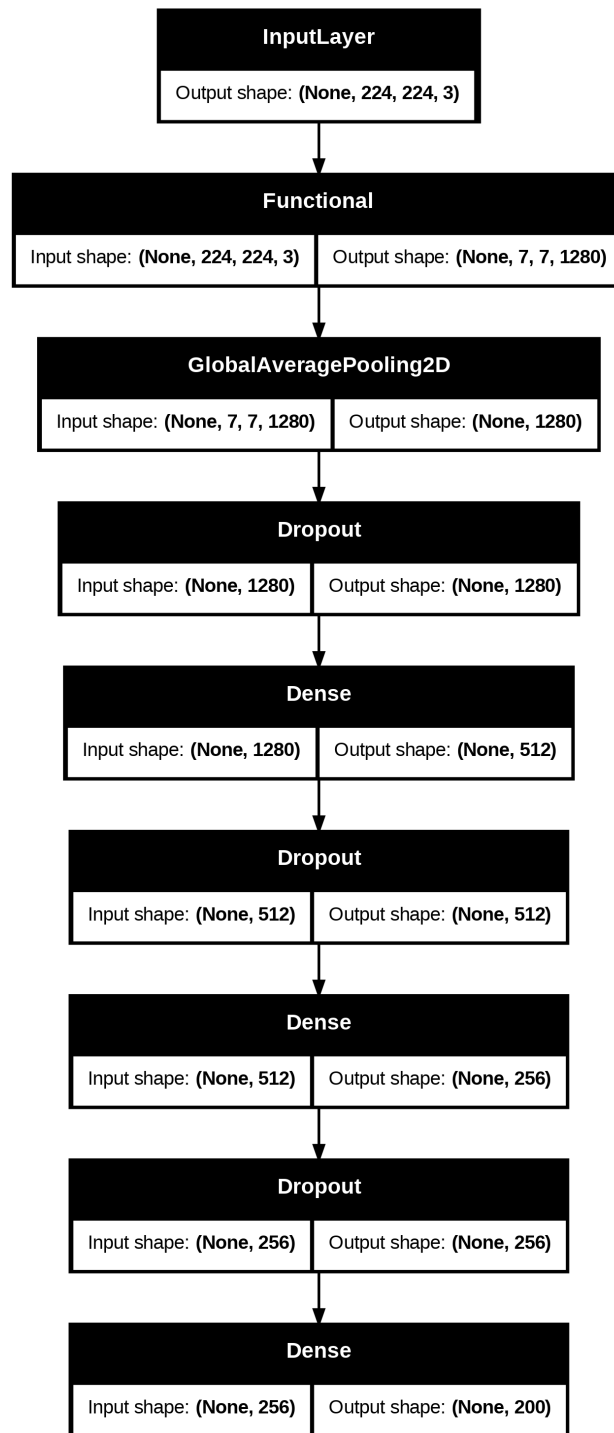
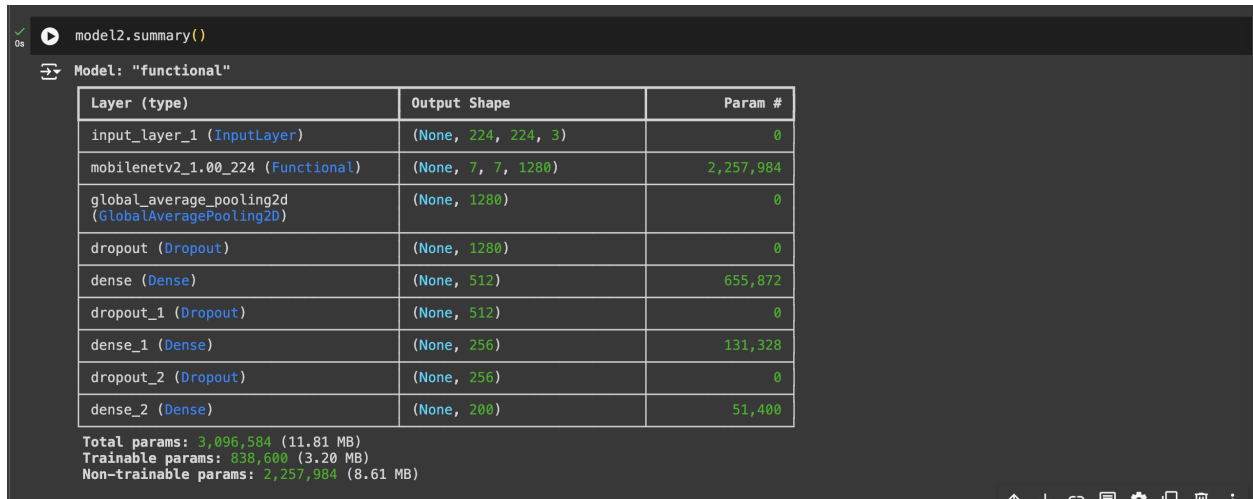


Figure 3: Architecture for this assignment

### 1.3 Model Discussion

If we use the function `summary()` from **Tensorflow**, we can see that there are total 3,096,584 parameters (including 838,600 trainable parameters) as Figure 4 below:



Layer (type)	Output Shape	Param #
input_layer_1 (InputLayer)	(None, 224, 224, 3)	0
mobilenetv2_1.00_224 (Functional)	(None, 7, 7, 1280)	2,257,984
global_average_pooling2d (GlobalAveragePooling2D)	(None, 1280)	0
dropout (Dropout)	(None, 1280)	0
dense (Dense)	(None, 512)	655,872
dropout_1 (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 256)	131,328
dropout_2 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 200)	51,400

Total params: 3,096,584 (11.81 MB)  
Trainable params: 838,600 (3.20 MB)  
Non-trainable params: 2,257,984 (8.61 MB)

Figure 4: Total parameters

### 1.4 Normalize Training Images

According to the requirements from Tensorflow, the model **MobileNetV2** only receive the value (pixel value) in range  $[-1, 1]$ . So before training, I need to scale the image as the script below:

```
1 normalized_images = training_images.astype('float32') / 255.0
```

### 1.5 Hyperparameters

For this experience, the hyperparameters for training this model is:

- **Number of Epochs:** 42
- **Loss Function:** Categorical Cross Entropy
- **Batch Size:** 64
- **Validation Split:** 0.3

## 2 Result

### 2.1 Loss and other Metrics

Within 42 epochs, the Figure 5 is the evaluation.

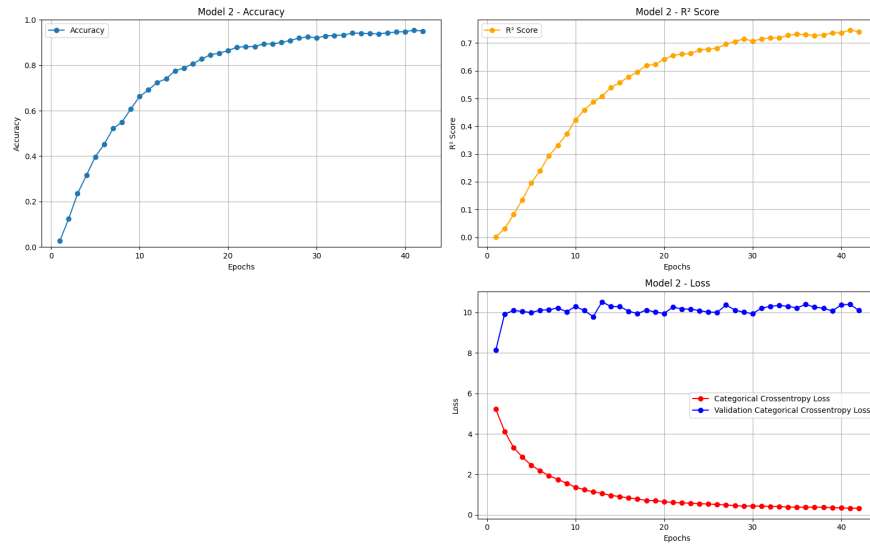


Figure 5: Model Evaluation

As Figure 5 below, the Accuracy and R2 Score tend to increase and get close to 1. However, in the **Loss** graph, the Validation Loss is not decreased as we expected. It is not a perfect result, but to compare with other results (as Figure 6), this model is the best performance in my experience.

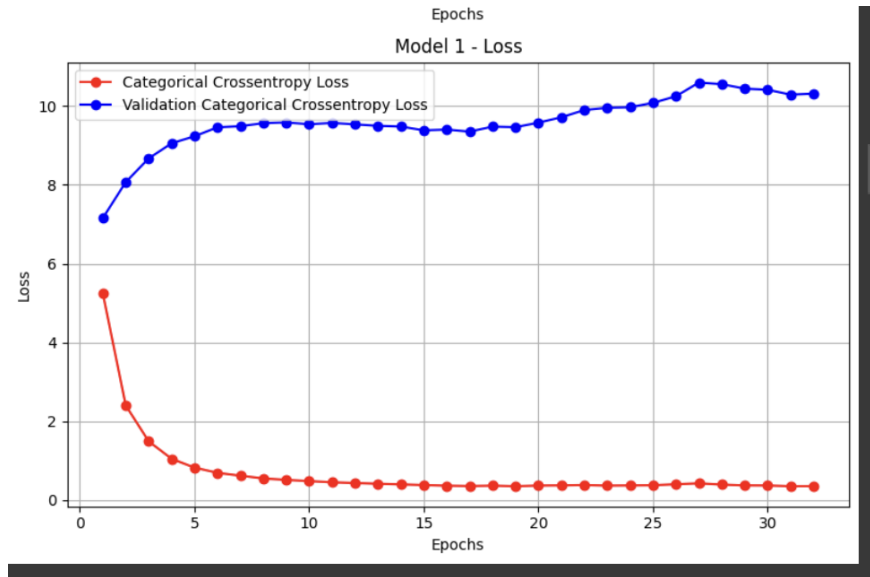
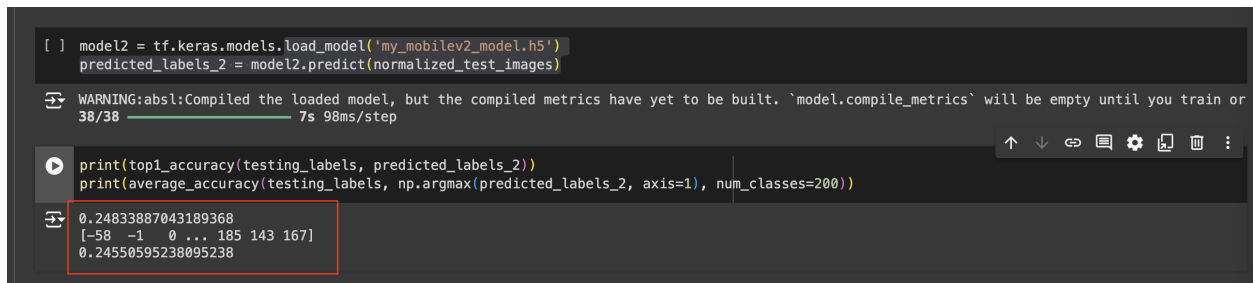


Figure 6: Other Model Evaluation

## 2.2 Evaluation Metric

The Figure 7 below is the **Top-1 accuracy** and **Average accuracy per class**, respectively. (please ignore the second line)

A screenshot of a Jupyter Notebook interface. The top cell contains two lines of Python code: `model2 = tf.keras.models.load_model('my_mobilenet2_model.h5')` and `predicted_labels_2 = model2.predict(normalized_test_images)`. Below the code is a warning message: `WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. 'model.compile_metrics' will be empty until you train or 38/38 7s 98ms/step`. The bottom cell contains two lines of code: `print(top1_accuracy(testing_labels, predicted_labels_2))` and `print(average_accuracy(testing_labels, np.argmax(predicted_labels_2, axis=1), num_classes=200))`. The output of the first line is `0.24833887043189368` and the output of the second line is `[-58 -1 0 ... 185 143 167]` and `0.24550595238095238`. The second line of the output is highlighted with a red box.

```
[ ] model2 = tf.keras.models.load_model('my_mobilenet2_model.h5')
predicted_labels_2 = model2.predict(normalized_test_images)

WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or
38/38 7s 98ms/step

print(top1_accuracy(testing_labels, predicted_labels_2))
print(average_accuracy(testing_labels, np.argmax(predicted_labels_2, axis=1), num_classes=200))

0.24833887043189368
[-58 -1 0 ... 185 143 167]
0.24550595238095238
```

Figure 7: Evaluation Metric

This result is not surprised. In Figure 5, the model does not perform well with unseen data. Hence, the accuracy is quite low and we are facing the situation of overfitting. But at least, comparing the final model (Figure 5) with other model (Figure 6), we minimized the problem of overfitting.