Predictive Modelling Birds Classification

Ya Ting Huang
Alexandra Landry
Nastasia Kantsevitch
Julius Alessandro Xanthoudakis

Overview

- Introduction
  - Data Selection
  - Challenges
- Model 1 - Base
- Model 2 - Augmented
- Model 3 - Pre-Trained
- Lessons Learned
- Conclusion

Data Selection

- 15 most represented classes

[5]

```python
def create_data(df, type_data):
    processed_data_directory = 'data/'
    data = []
    top_15 = df['labels'].value_counts().head(15).index.tolist()
    for c in top_15:
        image_folder = os.path.join(processed_data_directory + f'{type_data}/', c)
        for filename in os.listdir(image_folder):
            image_path = os.path.join(image_folder, filename)
            img = imread(image_path, as_gray=False)
            img = img / 255
            img = resize(img, (64, 64))
            class_value = top_15.index(c)
            data.append([img, class_value])
    return data
```

[6]

```python
train_data = create_data(df, 'train')
valid_data = create_data(df, 'valid')
test_data = create_data(df, 'test')
```

[7]

```python
print("Training data length:   ", len(train_data))
print("Validation data length: ", len(valid_data))
print("Testing data length:    ", len(test_data))
```

```
Training data length:    3306
Validation data length:  75
Testing data length:     75
```

Data Selection cont’d

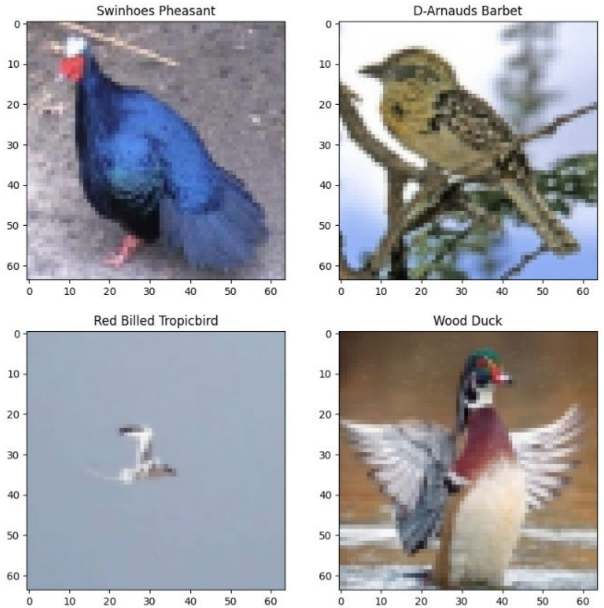
- Retaining Class information

```
[65]
def append_label(df, type_data):
    top_15 = df['labels'].value_counts().head(15).index.tolist()
    label_dict = {}
    for idx, c in enumerate(top_15):
        label_dict[idx] = c.title()
    return label_dict

labels = append_label(df, 'test')
```

```
labels
```

```
{0: 'Rufous Trepe',
 1: 'House Finch',
 2: 'D-Arnauds Barbet',
 3: 'Ovenbird',
 4: 'Asian Green Bee Eater',
 5: 'Swinhoes Pheasant',
 6: 'Wood Duck',
 7: 'Caspian Tern',
 8: 'Red Billed Tropicbird',
 9: 'Wood Thrush',
 10: 'Frill Back Pigeon',
 11: 'Pyrrhuloxia',
 12: 'Merlin',
 13: 'Ornate Hawk Eagle',
 14: 'Military Macaw'}
```

```
i = 23
plt.figure(figsize=(10,10))
plt.subplot(221), plt.imshow(X_train[i]), plt.title(labels[int(y_train[i])])
plt.subplot(222), plt.imshow(X_train[i+25]), plt.title(labels[int(y_train[i+25])])
plt.subplot(223), plt.imshow(X_train[i+50]), plt.title(labels[int(y_train[i+50])])
plt.subplot(224), plt.imshow(X_train[i+75]), plt.title(labels[int(y_train[i+75])])
plt.show()
```



Challenges

- Small Sample Size
- Computational Power
- Vast range of techniques to explore

Model 1
Base

```python
## Building the CNN model
model1 = keras.Sequential()

model1.add(Conv2D(32, (3, 3), input_shape=(224, 224, 3), activation='relu'))
model1.add(MaxPooling2D((2, 2)))

model1.add(Conv2D(64, (3, 3), activation='relu'))
model1.add(MaxPooling2D((2, 2)))

model1.add(Conv2D(64, (3, 3), activation='relu'))
model1.add(MaxPooling2D((2, 2)))

model1.add(Conv2D(128, (3, 3), activation='relu'))
model1.add(MaxPooling2D((2, 2)))

model1.add(Conv2D(128, (3, 3), activation='relu'))
model1.add(MaxPooling2D((2, 2)))

model1.add(Conv2D(128, (3, 3), activation='relu'))
model1.add(MaxPooling2D((2, 2)))

model1.add(Flatten())

model1.add(Dense(256, activation='relu'))

model1.add(Dense(15, activation='softmax'))

## Compiling the model
model1.compile(loss=SparseCategoricalCrossentropy(from_logits=True),optimizer='adam', metrics=['accuracy'])

## Printing the summary of the model
model1.summary()

## Fitting the model
epochs = 30

history1 = model1.fit(X_train, y_train
                      ,batch_size=32
                      ,validation_data=[X_valid, y_valid]
                      ,epochs=epochs
                      ,callbacks=[early_stopping])
```

```python
### Definition of the callbacks function when fitting the model
early_stopping = EarlyStopping(monitor="val_loss"
                               ,patience=10
                               ,verbose=1
                               ,restore_best_weights=True)
```

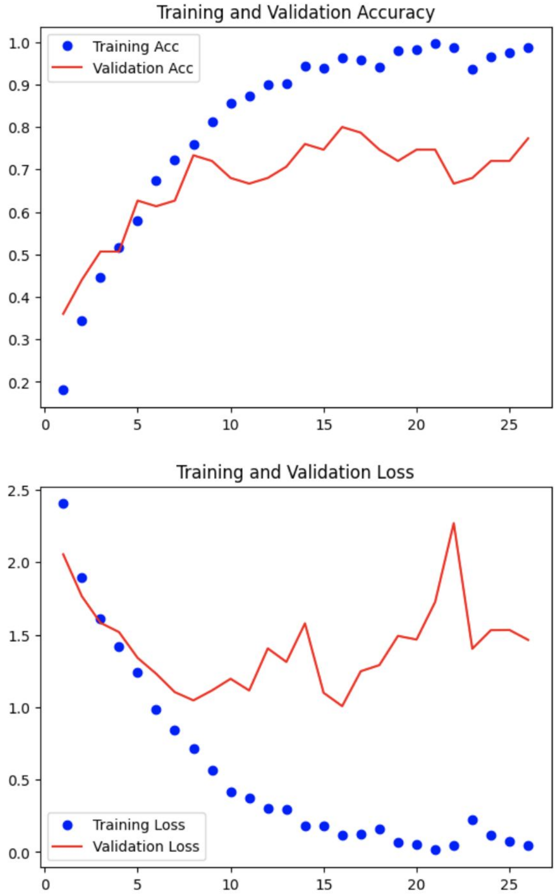
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 222, 222, 32)	896
max_pooling2d (MaxPooling2D)	(None, 111, 111, 32)	0
conv2d_1 (Conv2D)	(None, 109, 109, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 54, 54, 64)	0
conv2d_2 (Conv2D)	(None, 52, 52, 64)	36,928
max_pooling2d_2 (MaxPooling2D)	(None, 26, 26, 64)	0
conv2d_3 (Conv2D)	(None, 24, 24, 128)	73,856
max_pooling2d_3 (MaxPooling2D)	(None, 12, 12, 128)	0
conv2d_4 (Conv2D)	(None, 10, 10, 128)	147,584
max_pooling2d_4 (MaxPooling2D)	(None, 5, 5, 128)	0
conv2d_5 (Conv2D)	(None, 3, 3, 128)	147,584
max_pooling2d_5 (MaxPooling2D)	(None, 1, 1, 128)	0
flatten (Flatten)	(None, 128)	0
dense (Dense)	(None, 256)	33,024
dense_1 (Dense)	(None, 15)	3,855

Total params: 462,223 (1.76 MB)

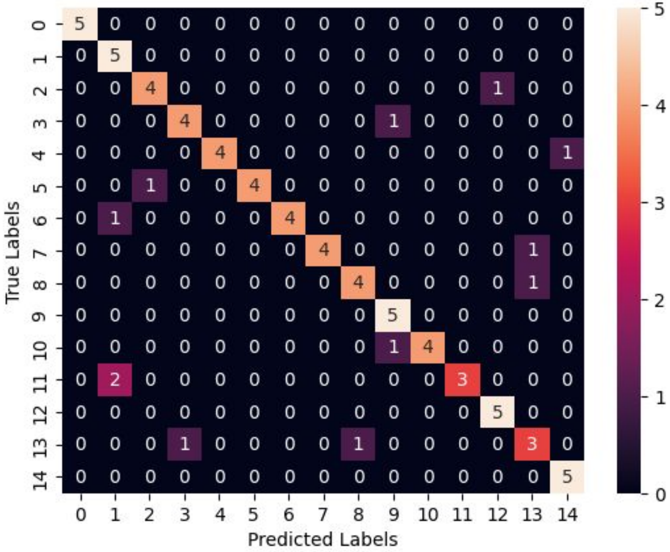
Trainable params: 462,223 (1.76 MB)

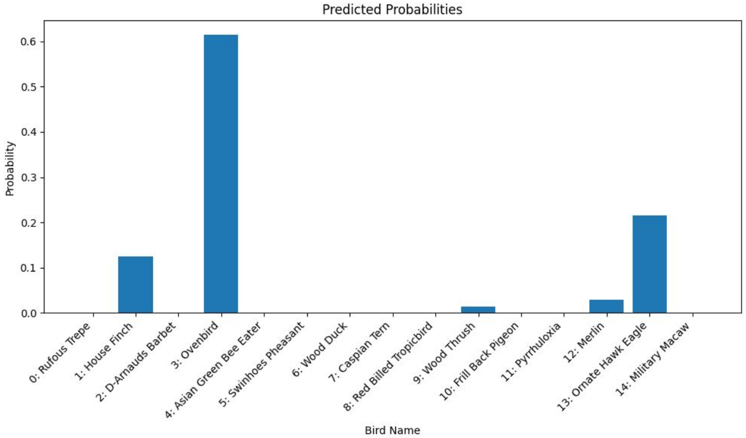
Non-trainable params: 0 (0.00 B)

```
Epoch 26: early stopping
Restoring model weights from the end of the best epoch: 16.
```




```
Test loss: 0.8631170392036438
Test accuracy: 0.8399999737739563
```





True label: Ornate Hawk Eagle, Predicted label: Ovenbird

Model 2 Augmented

Key Differences

- Image Preprocessing
- More Conv2D Layers
- Batch Normalization
- Regularization within Dense Layer
- Dropout
- Learning Rate

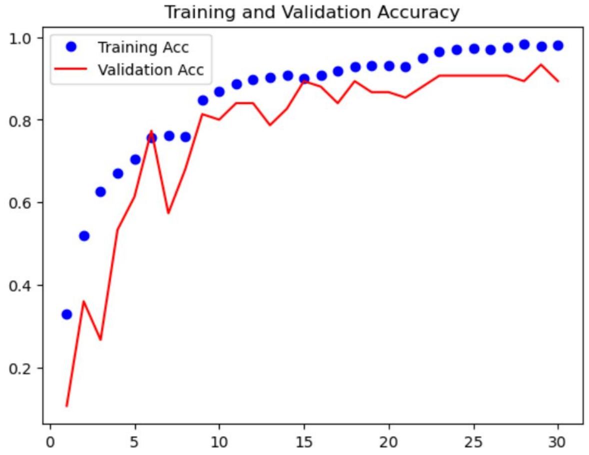
```python
### After testing numerous models, this is the best model we could come up with manually.
rlronp = tf.keras.callbacks.ReduceLROnPlateau(montitor='val_loss',factor=0.4,patience=2,verbose=1)

early_stopping2 = tf.keras.callbacks.EarlyStopping(monitor='val_loss',patience=4,verbose=1,
                                              restore_best_weights=True)

## Building the CNN model
model2 = keras.Sequential()

# Data augmentation
model2.add(layers.RandomFlip(mode='horizontal_and_vertical',input_shape=(224,224,3)))

model2.add(Conv2D(32,(3,3),activation='relu'))
model2.add(MaxPooling2D((2,2), padding='same'))

model2.add(Conv2D(64,(3,3),activation='relu'))
model2.add(MaxPooling2D((2,2), padding='same'))

model2.add(Conv2D(128,(3,3),activation='relu'))
model2.add(MaxPooling2D((2,2), padding='same'))

model2.add(Conv2D(256,(3,3),activation='relu'))
model2.add(MaxPooling2D((2,2), padding='same'))

model2.add(Conv2D(256,(3,3),activation='relu'))
model2.add(MaxPooling2D((2,2), padding='same'))

model2.add(layers.BatchNormalization())

model2.add(Flatten())

model2.add(layers.Dense(512, kernel_regularizer=regularizers.l2(0.016),
                        activity_regularizer=regularizers.l1(0.006),
                          bias_regularizer=regularizers.l1(0.006),activation='relu'))

model2.add(layers.Dropout(0.20))

model2.add(Dense(15,activation='softmax'))

## Compiling the model
model2.compile(loss=SparseCategoricalCrossentropy(from_logits=True),optimizer='adam',metrics=['accuracy'])

## Printing the summary of the model
model2.summary()

## Fitting the model
epochs=30
history2 = model2.fit(X_train, y_train
                    ,batch_size=32
                    ,validation_data=[X_valid, y_valid]
                    ,epochs=epochs
                    ,callbacks=[rlronp, early_stopping2])
```

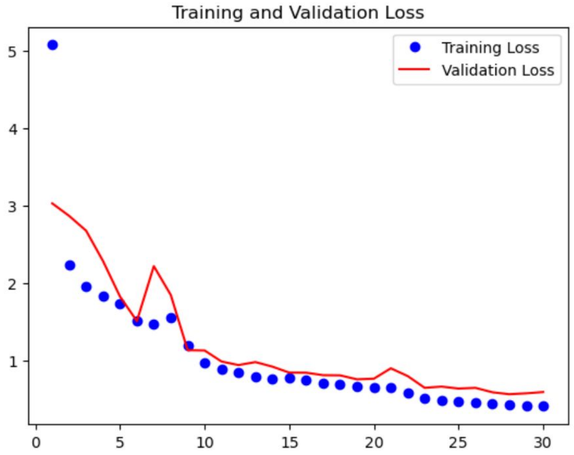
Layer (type)	Output Shape	Param #
random_flip_1 (RandomFlip)	(None, 224, 224, 3)	0
conv2d_11 (Conv2D)	(None, 222, 222, 32)	896
max_pooling2d_11 (MaxPooling2D)	(None, 111, 111, 32)	0
conv2d_12 (Conv2D)	(None, 109, 109, 64)	18,496
max_pooling2d_12 (MaxPooling2D)	(None, 55, 55, 64)	0
conv2d_13 (Conv2D)	(None, 53, 53, 128)	73,856
max_pooling2d_13 (MaxPooling2D)	(None, 27, 27, 128)	0
conv2d_14 (Conv2D)	(None, 25, 25, 256)	295,168
max_pooling2d_14 (MaxPooling2D)	(None, 13, 13, 256)	0
conv2d_15 (Conv2D)	(None, 11, 11, 256)	590,080
max_pooling2d_15 (MaxPooling2D)	(None, 6, 6, 256)	0
batch_normalization_1 (BatchNormalization)	(None, 6, 6, 256)	1,024
flatten_2 (Flatten)	(None, 9216)	0
dense_4 (Dense)	(None, 512)	4,719,104
dropout_1 (Dropout)	(None, 512)	0
dense_5 (Dense)	(None, 15)	7,695

Total params: 5,706,319 (21.77 MB)

Trainable params: 5,705,807 (21.77 MB)

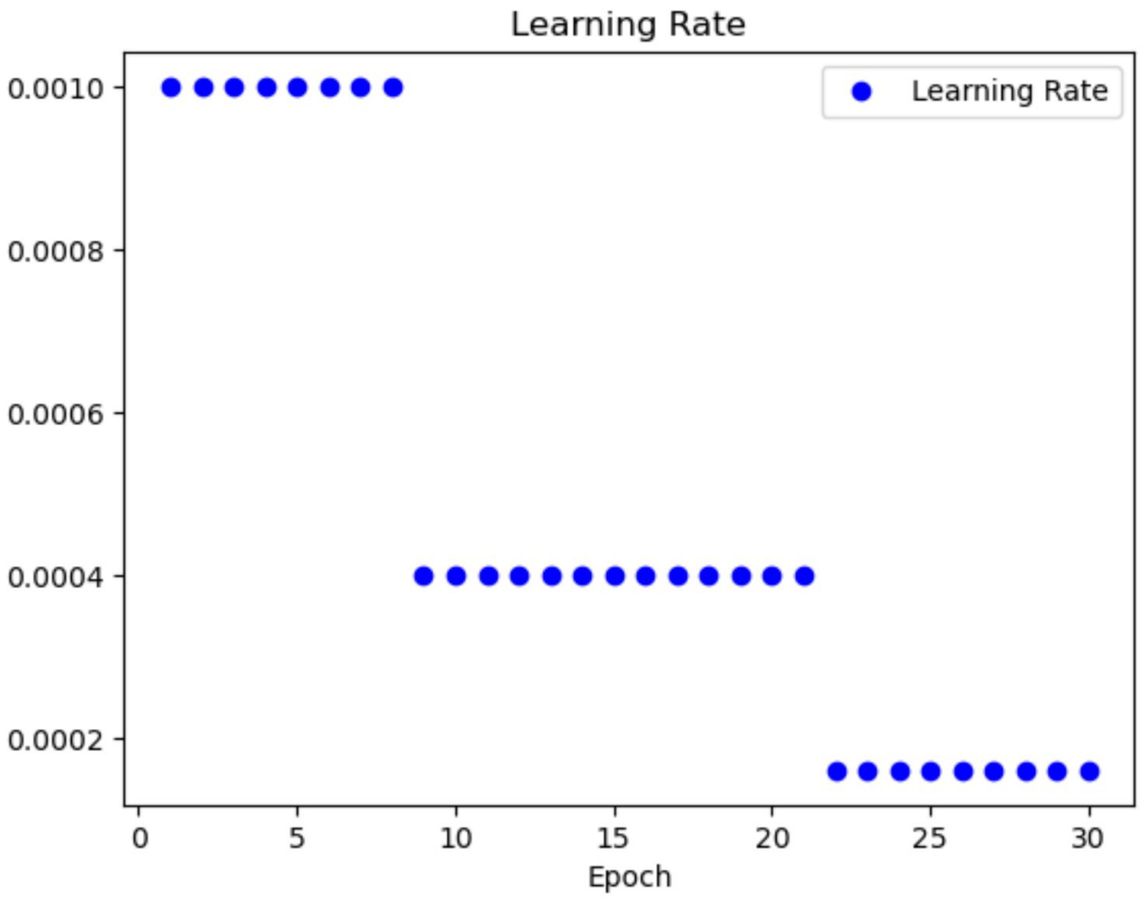
Non-trainable params: 512 (2.00 KB)

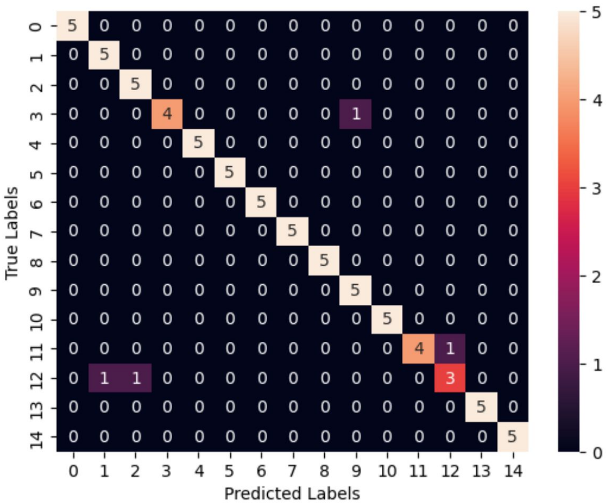




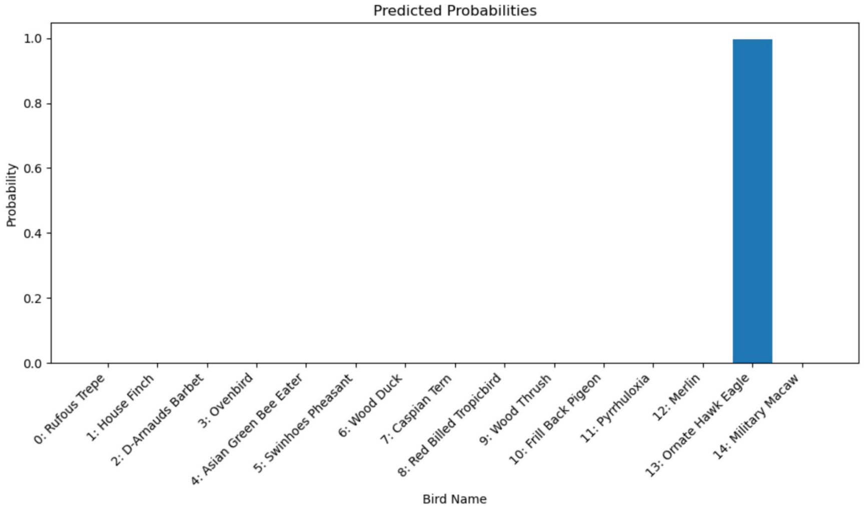
Test loss: 0.498855859041214
Test accuracy: 0.9466666579246521

The learning rate decreased in steps as the epochs progressed





5 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 5 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 5 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 4 0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 5 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 5 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 5 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 5 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 5 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 5 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 5 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 4 1 0 0
0 1 1 0 0 0 0 0 0 0 0 0 3 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 5 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 5
True Labels
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
Predicted Labels
0 1 2 3 4 5



Predicted Probabilities
Probability
0.0 0.2 0.4 0.6 0.8 1.0
0: Rufous Trepe
1: House Finch
2: D-Arnauds Barbet
3: Ovenbird
4: Asian Green Bee Eater
5: Swinhoes Pheasant
6: Wood Duck
7: Caspian Tern
8: Red Billed Tropicbird
9: Wood Thrush
10: Frill Back Pigeon
11: Pyrrhuloxia
12: Merlin
13: Ornate Hawk Eagle
14: Military Macaw
Bird Name

True label: Ornate Hawk Eagle, Predicted label: Ornate Hawk Eagle

Model 3
Pre-Trained

Key Differences

- VGG16 Pre-Trained Model
- Possibility to Freeze/Unfreeze
- No Data Augmentation
- Longer Training Time

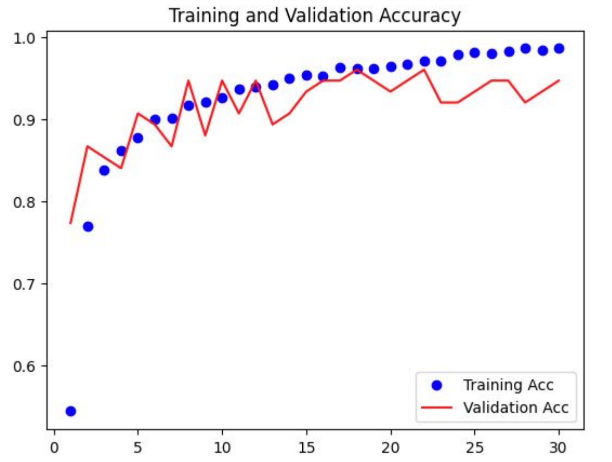
```python
### Testing pretrained model -- VGG16

## Building the CNN model - VGG16 pretrained model
base_model_vgg = VGG16(include_top = False, input_shape = (224, 224, 3))
base_model_vgg.trainable = False

# Define the hidden layers and the output
x = base_model_vgg.get_layer('block5_pool').output
x = layers.GlobalAveragePooling2D()(x)
outputs = layers.Dense(15, activation = 'softmax')(x)

# Create the model
model_vgg = models.Model(base_model_vgg.input, outputs)

## Compiling the model
model_vgg.compile(loss = 'sparse_categorical_crossentropy'
             ,optimizer = optimizers.Adam(learning_rate = 0.01)
             ,metrics = ['accuracy'])

## Printing the summary of the model
model_vgg.summary()

## Fitting the model
epochs=30
history_vgg = model_vgg.fit(X_train, y_train
         ,epochs = epochs
         ,validation_data = [X_valid, y_valid]
         ,batch_size=32
         ,callbacks=[early_stopping])

## Plot for the training results
plot_model_training(history_vgg)

## Evaluating the model on unseen data with the test data
score_vgg = model_vgg.evaluate(X_test, y_test, verbose=0)
print("Test loss:", score_vgg[0])
print("Test accuracy:", score_vgg[1])

# Part of the code from: https://medium.com/@bobbycxy/birds-classification-with-pre-trai
```

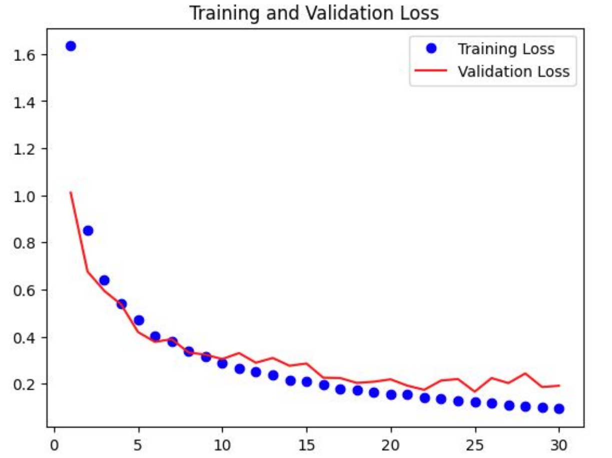
Layer (type)	Output Shape	Param #
input_layer_3 (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1,792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36,928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73,856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147,584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295,168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590,080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590,080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
global_average_pooling2d (GlobalAveragePooling2D)	(None, 512)	0
dense_6 (Dense)	(None, 15)	7,695

Total params: 14,722,383 (56.16 MB)

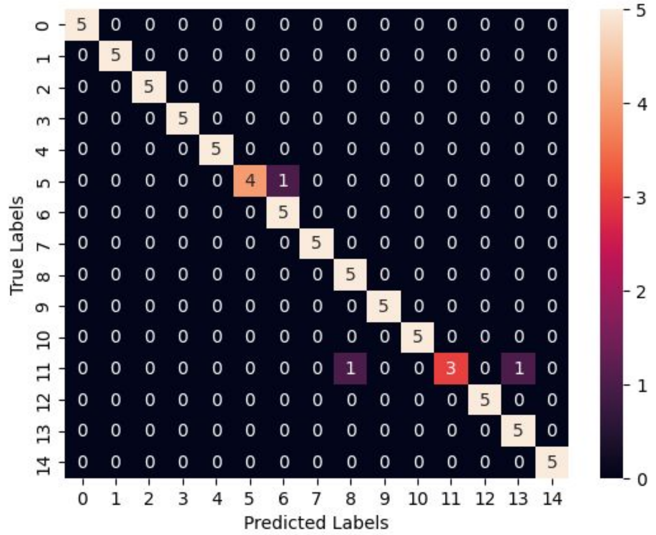
Trainable params: 7,695 (30.06 KB)

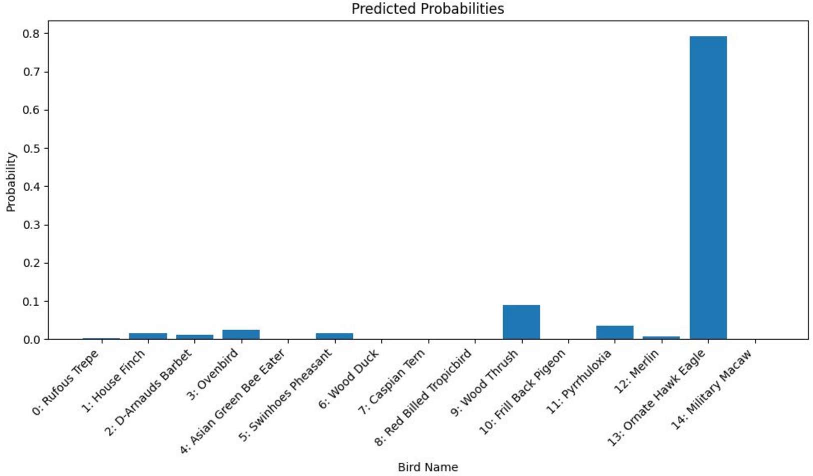
Non-trainable params: 14,714,688 (56.13 MB)






```
Test loss: 0.13346274197101593
Test accuracy: 0.9599999785423279
```



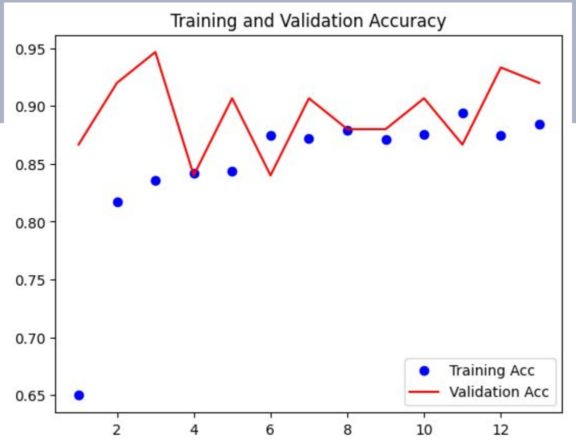


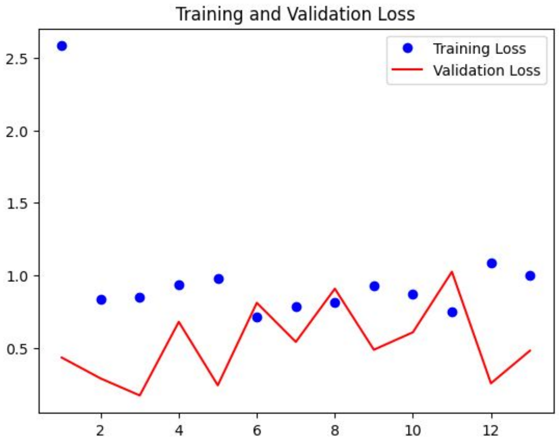
True label: Ornate Hawk Eagle, Predicted label: Ornate Hawk Eagle

InterceptV3

- Much quicker to train
- Similar Results to VGG16
- Used Data Augmentation (to counter overfitting)

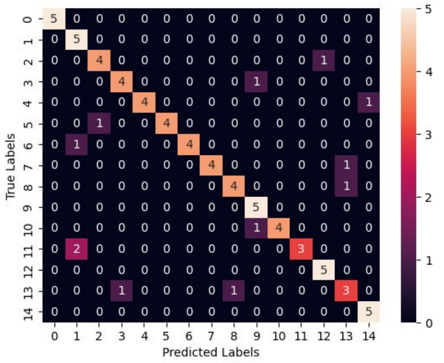
```
Test loss: 0.5461175441741943
Test accuracy: 0.8799999952316284
```



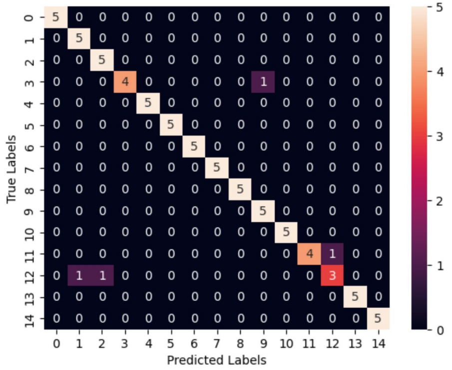


Lessons Learned

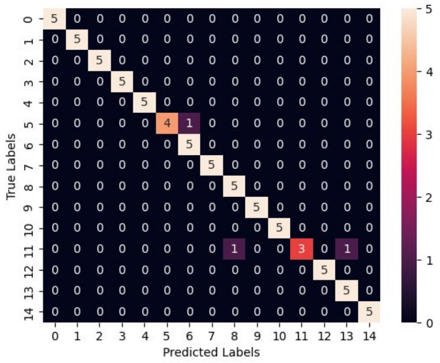
- Choose classes more carefully
- Version compatibility
- Utilizing GitHub to manage versions
- Use parameters that are tested and work
- Note on model selection:



Basic



Augmented



VGG16

Conclusion