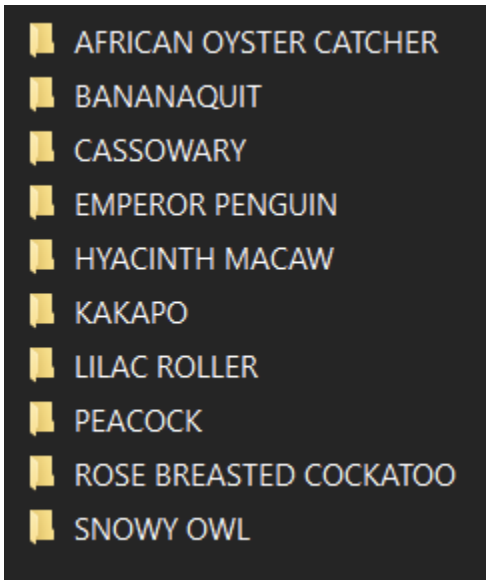
AAI 628 Final Project

The dataset for the project is called Bird Species Classification. It was previously found on kaggle but it has since been removed. The data modified dataset will be included in the submission of the report and source code. The original dataset is structured as follows.

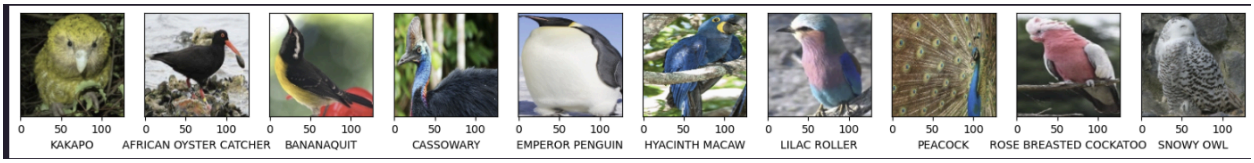
Each folder contains subfolders of each bird species. The original dataset contains 200 species of birds. For considerations of this project a subset of 10 birds was chosen.



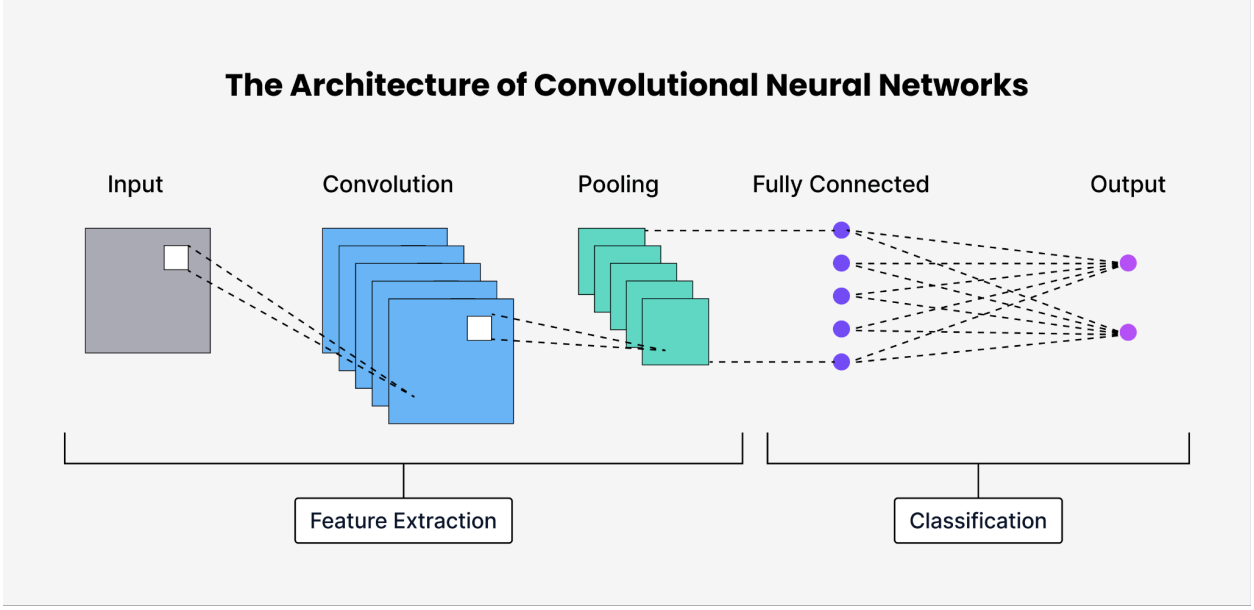
Each species contains ~150 train images, 5 test images and ~5 validation images. The data used for the project includes 1603 train images and 50 test images. The reason for selecting a subset of 10 from 200 is due to computational costs. The project was conducted on google colab and running cells often takes over 1 hour per cell with 10 species. Reducing the number of classifications allows for more simple model implementations which can result in more focused changes. Full 200 species would use a more complex and deeper model implementation that would result in very long train times. 10 should be satisfactory for the purpose of the project. The modified dataset is uploaded into a google colab runtime. Images are processed as follows

```
18 image_size = 128
19
20 for label in classes:
21   for image in os.listdir(image_path + label):
22       image_array = cv2.imread(os.path.join(image_path + label, image), cv2.IMREAD_COLOR)
23       image_array = cv2.cvtColor(image_array, cv2.COLOR_BGR2RGB)
24       resize_array = cv2.resize(image_array, (image_size, image_size))
25       train_images.append([resize_array])
26       train_labels.append(encoded_labels_dictionary[label])
27
28
29 train_images = np.array(train_images)
30 train_images = train_images.astype('float32') / 255.0
31 train_images = train_images.reshape((len(train_images), np.prod(train_images.shape[1:])))
32
```

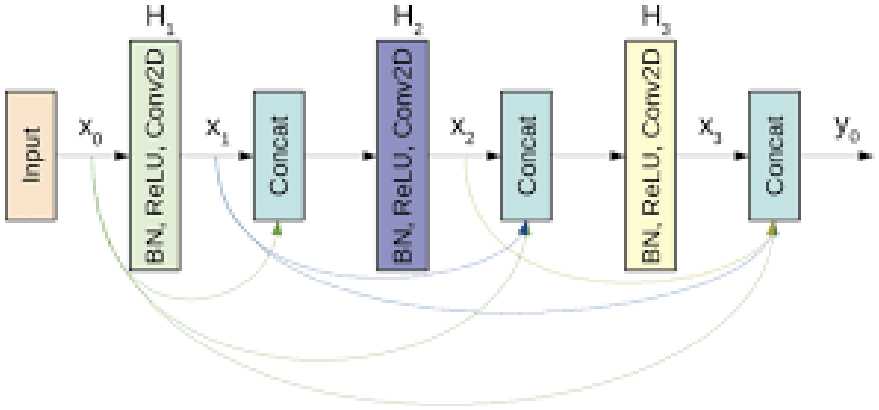
Images are read in using opencv methods and appended to an array after resizing. Resizing is used to make sure each image is uniform so it can be used in the same model. For previous assignments, image size of 28 x 28 was used. However, the size of 128 x 128 is chosen because the images of birds contain more complicated features so higher resolution is used. The procedure is the same for test and validation images. The labels are converted into a numerical value based on a dictionary that is used to convert string class name to an int class 0-9. It is also needed to convert into one-hot-encoded class labels. The following are samples from each class.



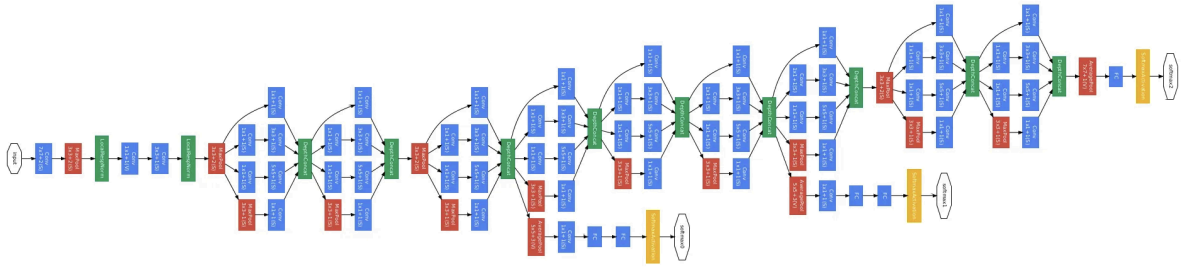
The bird species were picked manually by investigating the 200 bird species and looking for species that have distinct features. Kakapo is primarily green/yellow. African oyster catcher is primarily black with a distinct orange beak. Bananaquit is the only bird species in this group with bright yellow color. Cassowaries have distinct facial features. Emperor Penguins have a distinct body structure. Hyacinth Macaws are very blue. Lilac Rollers contain a mix of pinks and light blues. Peacocks have very distinct patterns in the tails. Rose Breasted Cockatoo has a distinct shade of red. Snowy owls are very white and gray and have a unique structure compared to the other species. These are some of the features of the classes detected by a human. The models will pick up on other features and not all pictures will express the same features. 2 Primary model architectures were studied for implementation of this project: DenseNet and InceptionNet Layers. DenseNet builds on top of CNNs that are discussed in class. In typical CNN architecture, A Convolution layer is often followed by a pooling layer which averages the features found through convolution and passes in down the network to another convolution layer or dense layer.



Dense Blocks found in dense net architecture differ by concatenating the layers of multiple different convolution layers into 1 output.



The theoretical benefit of such design is that the concatenation layers have access to all of the convolution layers rather than 1 at a time. This makes it so there are more features available at that layer which can be used to inform layers deeper in the network. Therefore it can lead to more accuracy on more complicated image classification datasets. However the increased complexity can lead to overfitting. Another architecture that was researched for this project is inception layers. Inception layers utilize convolution layers with different size filters such as 1x1, 3x3 and 5x5 along with a pooling layer concatenated together to form a block.



In this way more features can be extracted similar to densenet. With this knowledge different models were tested to see the effects of concatenating convolution blocks together on the bird classification dataset.

The base model architecture is as follows

Layer (type)	Output Shape	Param #	Connected to
input_layer_9 (InputLayer)	(None, 128, 128, 3)	0	-
conv2d_54 (Conv2D)	(None, 64, 64, 64)	9,472	input_layer_9[0][0]
conv2d_55 (Conv2D)	(None, 64, 64, 32)	2,080	conv2d_54[0][0]
conv2d_57 (Conv2D)	(None, 64, 64, 32)	2,080	conv2d_54[0][0]
conv2d_56 (Conv2D)	(None, 64, 64, 32)	9,248	conv2d_55[0][0]
conv2d_58 (Conv2D)	(None, 64, 64, 32)	25,632	conv2d_57[0][0]
concatenate_9 (Concatenate)	(None, 64, 64, 128)	0	conv2d_54[0][0], conv2d_56[0][0], conv2d_58[0][0]
conv2d_59 (Conv2D)	(None, 64, 64, 64)	8,256	concatenate_9[0][0]
conv2d_61 (Conv2D)	(None, 64, 64, 64)	8,256	concatenate_9[0][0]
conv2d_60 (Conv2D)	(None, 64, 64, 64)	36,928	conv2d_59[0][0]
conv2d_62 (Conv2D)	(None, 64, 64, 64)	102,464	conv2d_61[0][0]
concatenate_10 (Concatenate)	(None, 64, 64, 256)	0	concatenate_9[0][0], conv2d_60[0][0], conv2d_62[0][0]
global_average_pooling2d… (GlobalAveragePooling2D)	(None, 256)	0	concatenate_10[0][0]
dense_13 (Dense)	(None, 10)	2,570	global_average_poolin…

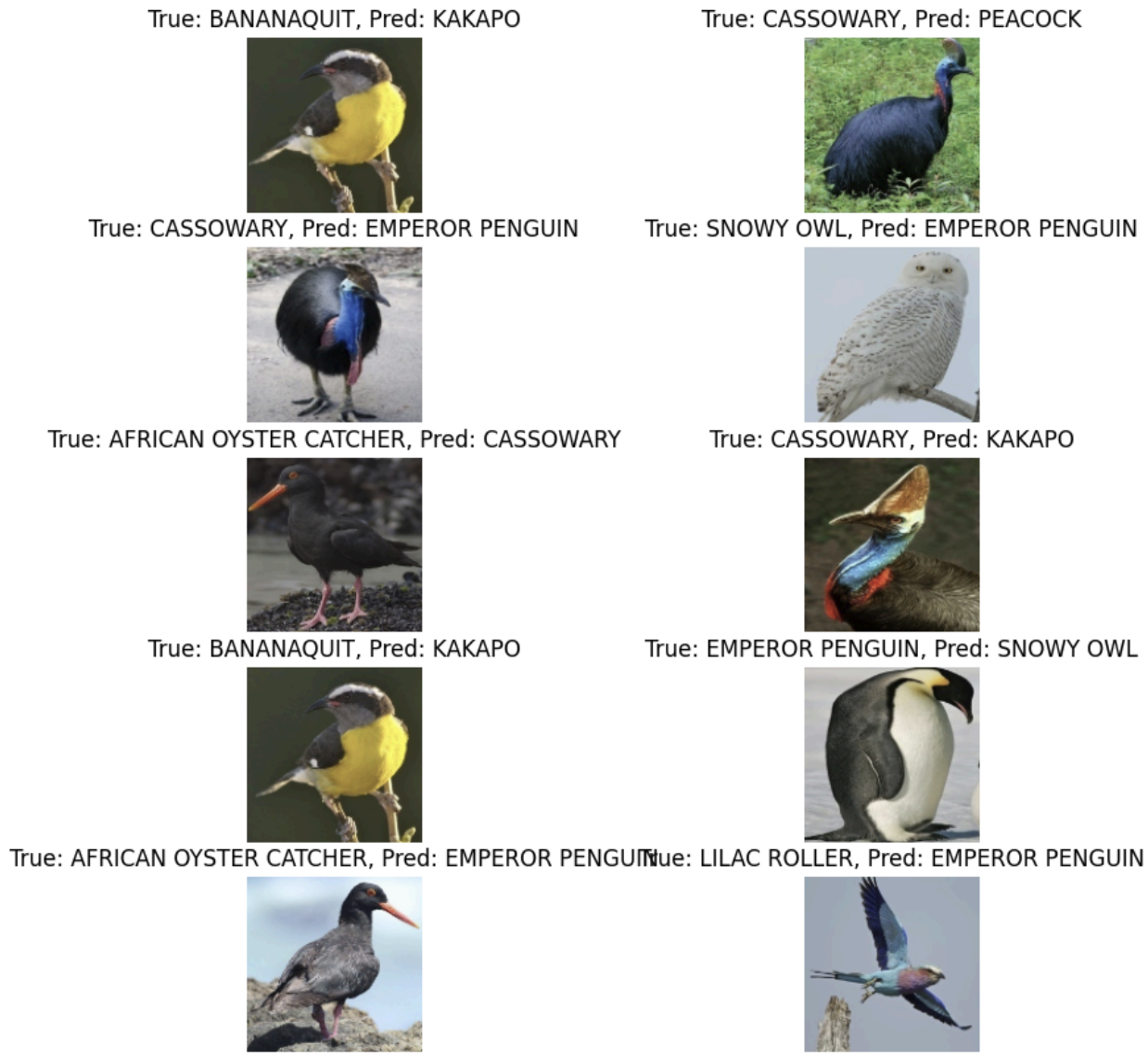
Total params: 206,986 (808.54 KB)
Trainable params: 206,986 (808.54 KB)
Non-trainable params: 0 (0.00 B)

It consists of 2 blocks of concatenated convolutions. In this case it is a Base 7x7 filter followed by 1x1 3x3 and 5x5 convolution filters. In this implementation there is no local pooling layer in

each block so the features are not averaged in each block. Instead there is a global pooling layer at the end to average all of the features collected and then fed to the output layer. The results are as follows.

```
Epoch 1/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 257s 5s/step - accuracy: 0.1857 - loss: 2.2164 - val_accuracy: 0.2600 - val_loss: 1.7577
Epoch 2/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 265s 5s/step - accuracy: 0.3071 - loss: 1.8401 - val_accuracy: 0.5400 - val_loss: 1.4334
Epoch 3/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 252s 5s/step - accuracy: 0.4264 - loss: 1.6082 - val_accuracy: 0.4000 - val_loss: 1.5946
Epoch 4/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 258s 5s/step - accuracy: 0.4718 - loss: 1.4834 - val_accuracy: 0.5000 - val_loss: 1.3894
Epoch 5/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 264s 5s/step - accuracy: 0.4798 - loss: 1.4662 - val_accuracy: 0.5400 - val_loss: 1.0804
Epoch 6/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 260s 5s/step - accuracy: 0.5635 - loss: 1.2830 - val_accuracy: 0.5600 - val_loss: 1.1931
Epoch 7/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 264s 5s/step - accuracy: 0.5831 - loss: 1.3017 - val_accuracy: 0.6600 - val_loss: 0.8963
Epoch 8/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 260s 5s/step - accuracy: 0.5859 - loss: 1.1979 - val_accuracy: 0.6000 - val_loss: 1.0138
Epoch 9/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 250s 5s/step - accuracy: 0.6331 - loss: 1.1246 - val_accuracy: 0.7200 - val_loss: 0.8902
Epoch 10/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 261s 5s/step - accuracy: 0.6259 - loss: 1.1555 - val_accuracy: 0.7200 - val_loss: 0.7700
2/2 ━━━━━━━━━━━━━━━━━━━━ 3s 1s/step - accuracy: 0.7092 - loss: 0.7893
Test accuracy:  0.7200000286102295
Train Time(h): 0.7233490757147472
Train Time(m): 43.40094454288483
Train Time(s): 2604.0566725730896
```

With some predictions



The model performed worse than expected with 72% test accuracy. By analyzing some of the incorrect predictions the model behavior may be made more clear. A bananaquit sample is predicted to be a kakapo. From a human perspective it would be expected that the model would pick up on the yellow color of the bananaquit and that it would be the strongest indicator. However it appears that color may not be the strongest indicator in the model. Maybe though the background color is identified as a similar shade of green found in kakapo and the primary color in the image. It still does seem that color is in fact a dominant feature picked up by the model just not as well as expected. Cassowary is predicted as a peacock likely due to the background color as well as the blue on the neck of the cassowary. It may be a similar proportion of green and blue to the train peacock images. Likewise cassowary to penguin could be due to the black body and the background may be similar to the arctic even though it is a beach. Snowy owl to penguin prediction may also be about color. They have distinct faces but the image may not be high enough resolution to distinguish from that angle. Oyster Catcher to Cassowary may be due to primarily black color. Cassowary to kakakapo could be because of

background. The Penguin to snowy owl and oyster catcher to penguin may again be due to color it seems. It is unclear what lilac roller to penguin prediction. Again maybe it is the background color. The model does appear to be strongly influenced by color but struggles with color proportions and locality. Background is given too much attention. Especially with lilac roller to penguin. If a significant proportion of light blue and pink is found it should be identified by the model.The next implementation is as follows

Model: "functional_30"

Layer (type)	Output Shape	Param #	Connected to
input_layer_6 (InputLayer)	(None, 128, 128, 3)	0	-
conv2d_27 (Conv2D)	(None, 64, 64, 64)	9,472	input_layer_6[0][0]
conv2d_28 (Conv2D)	(None, 64, 64, 32)	2,080	conv2d_27[0][0]
conv2d_30 (Conv2D)	(None, 64, 64, 32)	2,080	conv2d_27[0][0]
conv2d_29 (Conv2D)	(None, 64, 64, 32)	9,248	conv2d_28[0][0]
conv2d_31 (Conv2D)	(None, 64, 64, 32)	25,632	conv2d_30[0][0]
concatenate_3 (Concatenate)	(None, 64, 64, 128)	0	conv2d_27[0][0], conv2d_29[0][0], conv2d_31[0][0]
conv2d_32 (Conv2D)	(None, 64, 64, 64)	8,256	concatenate_3[0][0]
conv2d_34 (Conv2D)	(None, 64, 64, 64)	8,256	concatenate_3[0][0]
conv2d_33 (Conv2D)	(None, 64, 64, 64)	36,928	conv2d_32[0][0]
conv2d_35 (Conv2D)	(None, 64, 64, 64)	102,464	conv2d_34[0][0]
concatenate_4 (Concatenate)	(None, 64, 64, 256)	0	concatenate_3[0][0], conv2d_33[0][0], conv2d_35[0][0]
flatten_3 (Flatten)	(None, 1048576)	0	concatenate_4[0][0]
dense_9 (Dense)	(None, 256)	268,435,712	flatten_3[0][0]
dense_10 (Dense)	(None, 10)	2,570	dense_9[0][0]

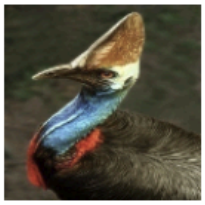
Total params: 268,642,698 (1.00 GB)
Trainable params: 268,642,698 (1.00 GB)
Non-trainable params: 0 (0.00 B)

The model is similar to the previous but instead of a global average pooling layer a flatten and dense layer of 256 neurons is used. Due to all of the feature maps generated from the various convolutions it is expected that a dense layer can capture and handle that information more effectively than a global pooling layer alone. It has results

```
Epoch 1/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 529s 10s/step - accuracy: 0.2205 - loss: 14.1825 - val_accuracy: 0.7400 - val_loss: 0.9738
Epoch 2/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 561s 10s/step - accuracy: 0.8124 - loss: 0.6266 - val_accuracy: 0.8000 - val_loss: 0.5874
Epoch 3/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 563s 10s/step - accuracy: 0.9457 - loss: 0.2231 - val_accuracy: 0.8600 - val_loss: 0.4766
Epoch 4/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 556s 10s/step - accuracy: 0.9938 - loss: 0.0388 - val_accuracy: 0.9000 - val_loss: 0.4079
Epoch 5/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 569s 10s/step - accuracy: 0.9994 - loss: 0.0095 - val_accuracy: 0.9000 - val_loss: 0.3652
Epoch 6/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 553s 10s/step - accuracy: 1.0000 - loss: 0.0037 - val_accuracy: 0.8800 - val_loss: 0.3658
Epoch 7/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 520s 10s/step - accuracy: 1.0000 - loss: 8.1232e-04 - val_accuracy: 0.8800 - val_loss: 0.4268
Epoch 8/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 566s 10s/step - accuracy: 1.0000 - loss: 9.2182e-04 - val_accuracy: 0.9000 - val_loss: 0.4497
Epoch 9/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 558s 10s/step - accuracy: 1.0000 - loss: 2.2897e-04 - val_accuracy: 0.9000 - val_loss: 0.4494
Epoch 10/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 518s 10s/step - accuracy: 1.0000 - loss: 1.4089e-04 - val_accuracy: 0.9000 - val_loss: 0.4930
2/2 ━━━━━━━━━━━━━━━━━━━━ 2s 882ms/step - accuracy: 0.9021 - loss: 0.5152
Test accuracy:  0.8999999761581421
Train Time(h): 1.538354638947381
Train Time(m): 92.30127833684286
Train Time(s): 5538.076700210571
```

And incorrect predictions

True: CASSOWARY, Pred: KAKAPO



True: CASSOWARY, Pred: AFRICAN OYSTER CATCHER

True: HYACINTH MACAW, Pred: LILAC ROLLER

True: PEACOCK, Pred: HYACINTH MACAW

True: PEACOCK, Pred: CASSOWARY

The accuracy has improved to ~90% from 72%, a significant increase. As shown by the flatten layer there are then 1048576 inputs which are then condensed to 256. The input is very large due to the resolution of the images and the fact that they are rgb not grayscale. Due to this information adding a simple dense layer is able to handle and process the information in a useful manner as shown by the predictions. However, a larger amount of neurons and extra neuron layers likely will not help much since it appears the model is overfitting as the train accuracy values per epoch are 1.00 but the test results are lower. The model still struggles with the 2 cassowary predictions as the previous model but does seem to solve color localization to

an extent with the dense layers. Even the second cassowary prediction to oyster catcher makes more sense then a penguin because the legs are more visually similar to an oyster catcher than a penguin which says that the dense layer is also handling other features other than color better than the previous model. Peacock to hyacinth macaw seems like a reasonable error since in that picture the most significant feature of the peacock is missing being its tail so the model must perceive the most important feature being its blue feathers which is the primary feature of the hyacinth macaw. It is very unclear why the model predicted the last peacock as a cassowary, however it is clearly not due to coloration so there is another feature that is more important to the model for that image. The next model implementation is as follows

Model: "functional"

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 128, 128, 3)	0	-
conv2d (Conv2D)	(None, 64, 64, 64)	9,472	input_layer[0][0]
conv2d_1 (Conv2D)	(None, 64, 64, 32)	2,080	conv2d[0][0]
conv2d_3 (Conv2D)	(None, 64, 64, 32)	2,080	conv2d[0][0]
max_pooling2d (MaxPooling2D)	(None, 64, 64, 64)	0	conv2d[0][0]
conv2d_2 (Conv2D)	(None, 64, 64, 32)	9,248	conv2d_1[0][0]
conv2d_4 (Conv2D)	(None, 64, 64, 32)	25,632	conv2d_3[0][0]
conv2d_5 (Conv2D)	(None, 64, 64, 32)	2,080	max_pooling2d[0][0]
concatenate (Concatenate)	(None, 64, 64, 160)	0	conv2d[0][0], conv2d_2[0][0], conv2d_4[0][0], conv2d_5[0][0]
conv2d_6 (Conv2D)	(None, 64, 64, 64)	10,304	concatenate[0][0]
conv2d_8 (Conv2D)	(None, 64, 64, 64)	10,304	concatenate[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 64, 64, 160)	0	concatenate[0][0]
conv2d_7 (Conv2D)	(None, 64, 64, 64)	36,928	conv2d_6[0][0]
conv2d_9 (Conv2D)	(None, 64, 64, 64)	102,464	conv2d_8[0][0]
conv2d_10 (Conv2D)	(None, 64, 64, 32)	5,152	max_pooling2d_1[0][0]
concatenate_1 (Concatenate)	(None, 64, 64, 320)	0	concatenate[0][0], conv2d_7[0][0], conv2d_9[0][0], conv2d_10[0][0]
flatten (Flatten)	(None, 1310720)	0	concatenate_1[0][0]
dense (Dense)	(None, 256)	335,544,576	flatten[0][0]
dense_1 (Dense)	(None, 10)	2,570	dense[0][0]

Total params: 335,762,890 (1.25 GB)
Trainable params: 335,762,890 (1.25 GB)
Non-trainable params: 0 (0.00 B)

Here Max pooling layers are added to each block of the model as done in injection net blocks. The model still uses 256 neuron dense layer as the previous model. This should help identify the prominent features in the image easier. It should result in a higher accuracy. The results are as follows.

```
Epoch 1/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 654s 13s/step - accuracy: 0.2137 - loss: 25.4478 - val_accuracy: 0.5600 - val_loss: 1.1619
Epoch 2/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 679s 13s/step - accuracy: 0.6501 - loss: 1.0020 - val_accuracy: 0.6800 - val_loss: 0.7378
Epoch 3/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 685s 13s/step - accuracy: 0.7813 - loss: 0.6538 - val_accuracy: 0.8200 - val_loss: 0.5230
Epoch 4/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 643s 13s/step - accuracy: 0.8901 - loss: 0.3086 - val_accuracy: 0.8200 - val_loss: 0.5837
Epoch 5/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 678s 13s/step - accuracy: 0.9404 - loss: 0.1846 - val_accuracy: 0.8400 - val_loss: 0.4669
Epoch 6/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 683s 13s/step - accuracy: 0.9579 - loss: 0.1790 - val_accuracy: 0.7800 - val_loss: 1.0403
Epoch 7/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 633s 12s/step - accuracy: 0.9849 - loss: 0.0593 - val_accuracy: 0.8800 - val_loss: 0.3952
Epoch 8/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 634s 12s/step - accuracy: 0.9955 - loss: 0.0161 - val_accuracy: 0.9600 - val_loss: 0.2094
Epoch 9/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 686s 13s/step - accuracy: 1.0000 - loss: 0.0046 - val_accuracy: 0.9000 - val_loss: 0.4121
Epoch 10/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 682s 12s/step - accuracy: 1.0000 - loss: 0.0011 - val_accuracy: 0.9000 - val_loss: 0.4029
2/2 ━━━━━━━━━━━━━━━━━━━━ 3s 1s/step - accuracy: 0.9021 - loss: 0.4069
Test accuracy:  0.8999999761581421
Train Time(h): 1.8613525515794753
Train Time(m): 111.68115309476852
Train Time(s): 6700.869185686111
```

With incorrect predictions

The accuracy has not improved from the previous model. However 3 of the predictions are different so it may help give insights on how the models behave. Both models have the same incorrect predictions of cassowary to oyster catcher and peacock to hyacinth macaw, likely for the same or similar reasons. First different prediction is cassowary to peacock for this model. This is different than the previous implementation of predicting kakapo. Even though the prediction is wrong it is possible that peacock is picked due to the blue coloring on the neck of the cassowary whereas the previous the color of the neck did not seem to be as important feature in previous models. Therefore it appears that the pooling layers extracted the importance of the blue coloring for the cassowary. However it may be that the train dataset does not have enough images of cassowaries with just their head. This image does not contain

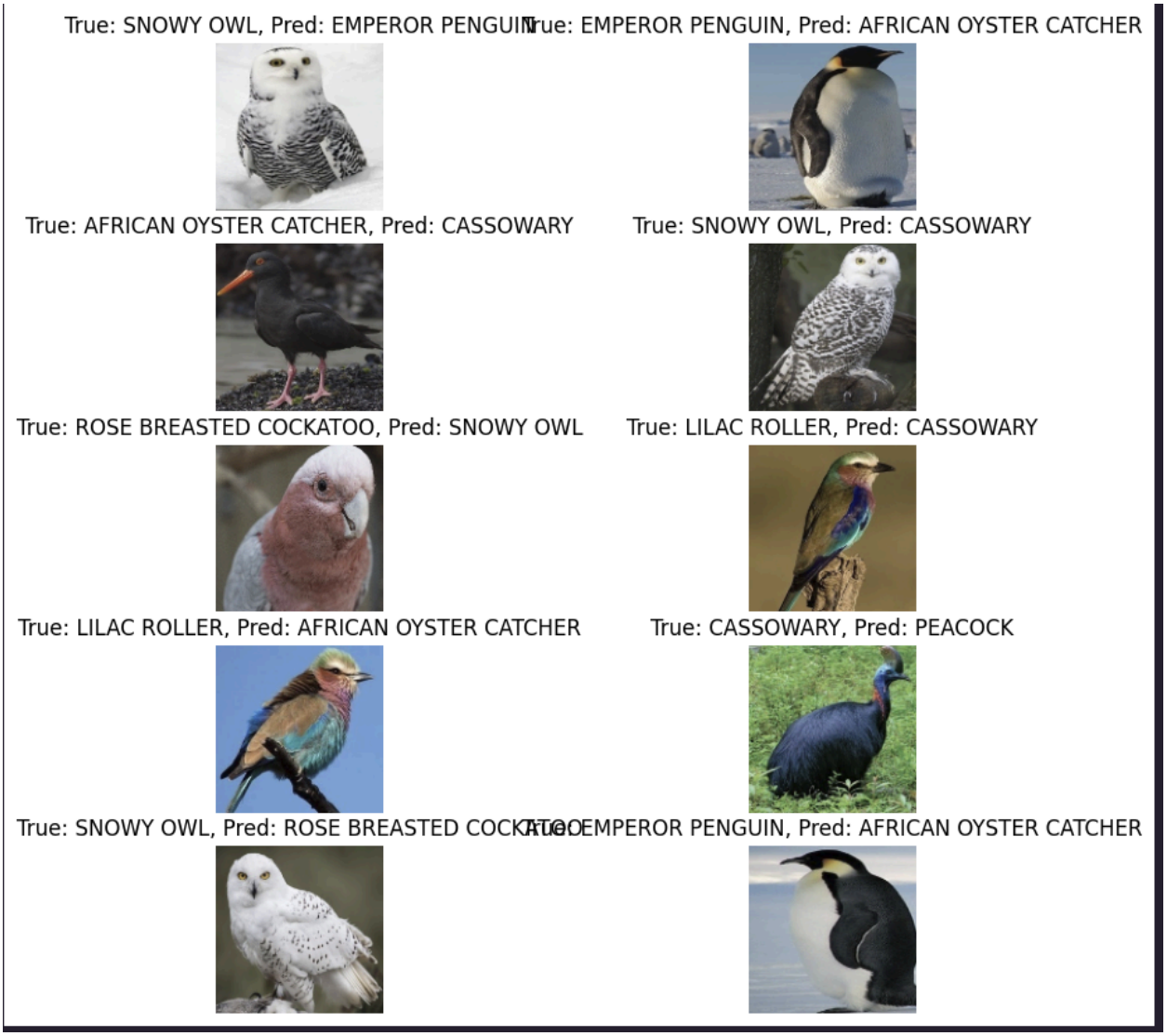
enough black for the model to correctly predict it. It does contain dark greens and some blue which would result in peacock prediction. It appears that none of the models can identify the beak as unique when compared to the other species. It is not clear the reason for penguin to oyster catcher due to the uniqueness of the penguin body structure. Snowy owl to rose breasted cockatoo is interesting because it may show that the model can detect the difference between back and front of a bird. Rose breasted cockatoo has a distinct color for chest but back is gray. Female snowy owls such as this one have gray feathers. The model may understand it is looking at the back of a bird and suspects due to the gray it is a rose breasted cockapoo because they are consistently gray in the back where snowy owls are usually pure white to some gray in most of the train images. No other model prediction so far has misclassified a bird as a rose breasted cockatoo. Overfitting may be occurring due to the train accuracy being 1.00 but it reached it in later epochs than the previous model without max pooling layer in the block which could show that the pooling layer does reduce overfitting. The next model is the same as the previous but it uses an imageDataGenerator as follows

```
43
44 gen = ImageDataGenerator(rotation_range=90, width_shift_range=0.08, shear_range=0.3, height_shift_range=0.2, zoom_range=0.2)
45
46 test_gen = ImageDataGenerator()
47
48 train_generator = gen.flow(train_images_cnn_input, train_labels_ohe, batch_size=64)
49 test_generator = test_gen.flow(test_images_cnn_input, test_labels_ohe, batch_size=64)
```

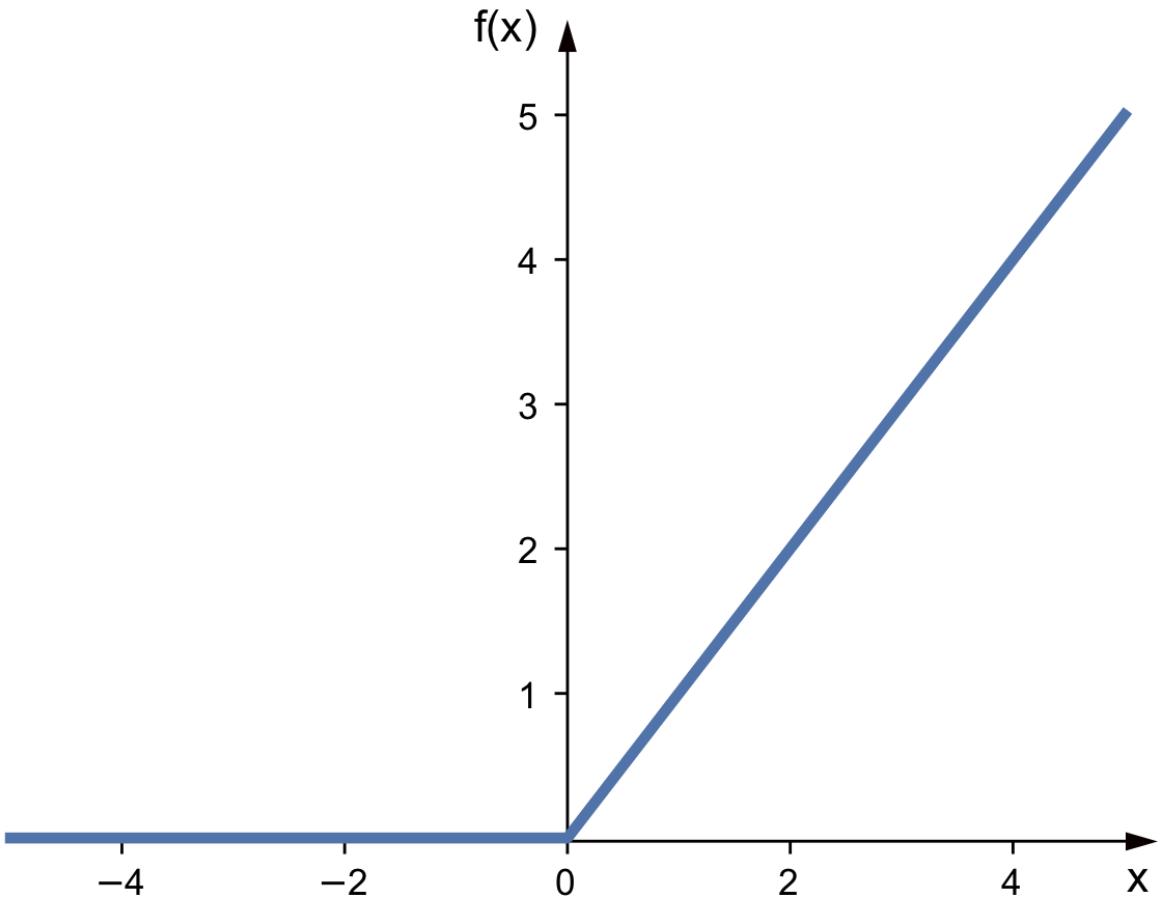
ImageDataGenerator is used for image augmentation. Additional images are generated with various modifications such as rotations, shifts, shears and zoom. The values chosen are arbitrary. The theory is that modifications in train data will make it more robust to outside data. With the modifications it will help the model learn other patterns less reliant on the orientation of image since that can be misleading if the bird is too small or too big. For example in other model predictions it appears the background was a result of some errors. Image data generator may alleviate some of these issues. The results are as follows.

```
 Non-trainable params: 0 (0.00 B)
Epoch 1/10
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:122: UserWarning: Your `PyDataset`
  self._warn_if_super_not_called()
6/6 ━━━━━━━━━━━━━━━━━━━━ 123s 19s/step - accuracy: 0.1136 - loss: 96.0022 - val_accuracy: 0.1600 - val_loss: 6.4427
Epoch 2/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 124s 21s/step - accuracy: 0.1863 - loss: 4.3045 - val_accuracy: 0.2600 - val_loss: 2.1836
Epoch 3/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 129s 21s/step - accuracy: 0.2412 - loss: 2.1199 - val_accuracy: 0.4000 - val_loss: 1.6525
Epoch 4/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 134s 23s/step - accuracy: 0.3652 - loss: 1.6811 - val_accuracy: 0.5600 - val_loss: 1.3023
Epoch 5/10
2/6 ━━━━━━━━━━━━━━━━━━━━ 1:25 21s/step - accuracy: 0.4570 - loss: 1.4513/usr/lib/python3.10/contextlib.py:153: UserWarning: Your i
  self.gen.throw(typ, value, traceback)
6/6 ━━━━━━━━━━━━━━━━━━━━ 46s 5s/step - accuracy: 0.4492 - loss: 1.4579 - val_accuracy: 0.5200 - val_loss: 1.2837
Epoch 6/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 129s 21s/step - accuracy: 0.4876 - loss: 1.3911 - val_accuracy: 0.5600 - val_loss: 1.1432
Epoch 7/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 112s 19s/step - accuracy: 0.4920 - loss: 1.4134 - val_accuracy: 0.6000 - val_loss: 1.0692
Epoch 8/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 125s 21s/step - accuracy: 0.4641 - loss: 1.3377 - val_accuracy: 0.5000 - val_loss: 1.1882
Epoch 9/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 132s 23s/step - accuracy: 0.4896 - loss: 1.4016 - val_accuracy: 0.6600 - val_loss: 0.8478
Epoch 10/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 44s 5s/step - accuracy: 0.6628 - loss: 1.1353 - val_accuracy: 0.6800 - val_loss: 0.8524
2/2 ━━━━━━━━━━━━━━━━━━━━ 3s 1s/step - accuracy: 0.6929 - loss: 0.8208
Test accuracy:  0.6800000071525574
Train Time(h): 0.31100158128473493
Train Time(m): 18.660094877084095
Train Time(s): 1119.6056926250458
```

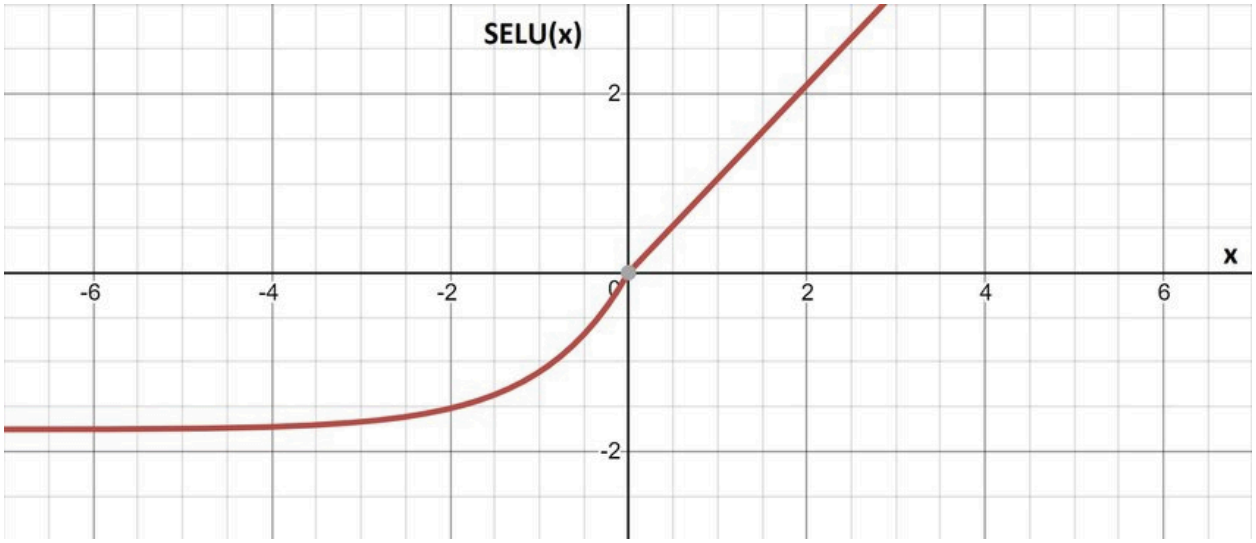
With some incorrect predictions



The accuracy score of 68% did not meet expectations. The incorrect predictions are also very different from other model incorrect predictions It is much weaker at predicting outside data when compared to the other models. It could be due to the values of the image data generator. The values may be too large in some cases that result in modified images that are too different and inefficient for training. It may be able to be optimized using some time of grid search for the values but that is too computationally complex for this project. It may be done for future work. It may also be more effective on larger datasets due to the lower variance in train images. Next the model is tested without image data generator but the activation functions of each layer is changed to Scaled Exponential Linear Unit (SELU) Instead of RELU. RELU activation function is as follows



With SELU as follows



An issue with RELU is the dying neuron problem. If a value is < 0, that neuron becomes “dead” as its value is mapped to 0 so it does not contribute to learning. A solution is Leaky RELU which instead maps negative values to very small negative values so learning still occurs. SELU however is a self normalizing activation function where each layer maintains the mean and

variance from previous layers. It has been shown in cases such as the MNIST and CIFAR-10 datasets to improve accuracy over RELU. The results for this bird classification dataset is as follows.

```
Epoch 1/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 599s 12s/step - accuracy: 0.1337 - loss: 313.7111 - val_accuracy: 0.1000 - val_loss: 3.8152
Epoch 2/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 593s 12s/step - accuracy: 0.1149 - loss: 3.3997 - val_accuracy: 0.1000 - val_loss: 2.7950
Epoch 3/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 622s 12s/step - accuracy: 0.1057 - loss: 2.5866 - val_accuracy: 0.1000 - val_loss: 2.3191
Epoch 4/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 617s 12s/step - accuracy: 0.1385 - loss: 2.3048 - val_accuracy: 0.1000 - val_loss: 2.3055
Epoch 5/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 628s 12s/step - accuracy: 0.1033 - loss: 2.3042 - val_accuracy: 0.1000 - val_loss: 2.3076
Epoch 6/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 616s 12s/step - accuracy: 0.1276 - loss: 2.3035 - val_accuracy: 0.1000 - val_loss: 2.3076
Epoch 7/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 619s 11s/step - accuracy: 0.1051 - loss: 2.3113 - val_accuracy: 0.1000 - val_loss: 2.3066
Epoch 8/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 619s 11s/step - accuracy: 0.1240 - loss: 2.3006 - val_accuracy: 0.1000 - val_loss: 2.3106
Epoch 9/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 627s 12s/step - accuracy: 0.1125 - loss: 2.3003 - val_accuracy: 0.1000 - val_loss: 2.3094
Epoch 10/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 622s 12s/step - accuracy: 0.1136 - loss: 2.3079 - val_accuracy: 0.1000 - val_loss: 2.3077
2/2 ━━━━━━━━━━━━━━━━━━━━ 3s 947ms/step - accuracy: 0.0667 - loss: 2.3161
Test accuracy:  0.10000000149011612
Train Time(h): 1.721233479115698
Train Time(m): 103.27400874694189
Train Time(s): 6196.440524816513
```

With sample incorrect predictions

This is very unexpected. With an accuracy of 10% this is by far the worst implementation. Clearly from the examples everything is being predicted as a rose breasted cockatoo hence the 10% accuracy since 5/50 of the test set is rose breasted cockatoo. This is an implementation error. It is not clear what the problem is but this is not the intended way to implement SELU activation. Another note is that it took a very long time to run the model with SELU activations. This is likely because each neuron contributes to learning due to the negative weight while with RELU activation those neurons have value of 0 and do not affect the learning processSince many of the models appear to be overfitting due to the high train accuracy compared to the test accuracy another model was tested with only 1 concatenated block as follows.

Model: "functional_1"

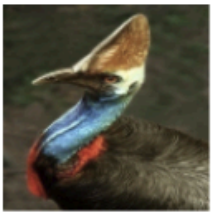
Layer (type)	Output Shape	Param #	Connected to
input_layer_1 (InputLayer)	(None, 128, 128, 3)	0	-
conv2d_11 (Conv2D)	(None, 64, 64, 64)	9,472	input_layer_1[0][0]
conv2d_12 (Conv2D)	(None, 64, 64, 32)	2,080	conv2d_11[0][0]
conv2d_14 (Conv2D)	(None, 64, 64, 32)	2,080	conv2d_11[0][0]
max_pooling2d_2 (MaxPooling2D)	(None, 64, 64, 64)	0	conv2d_11[0][0]
conv2d_13 (Conv2D)	(None, 64, 64, 32)	9,248	conv2d_12[0][0]
conv2d_15 (Conv2D)	(None, 64, 64, 32)	25,632	conv2d_14[0][0]
conv2d_16 (Conv2D)	(None, 64, 64, 32)	2,080	max_pooling2d_2[0][0]
concatenate_2 (Concatenate)	(None, 64, 64, 160)	0	conv2d_11[0][0], conv2d_13[0][0], conv2d_15[0][0], conv2d_16[0][0]
flatten_1 (Flatten)	(None, 655360)	0	concatenate_2[0][0]
dense_2 (Dense)	(None, 256)	167,772,416	flatten_1[0][0]
dense_3 (Dense)	(None, 10)	2,570	dense_2[0][0]

The reduced model produces the following results

```
Epoch 1/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 242s 5s/step - accuracy: 0.1738 - loss: 9.0523 - val_accuracy: 0.5400 - val_loss: 1.4413
Epoch 2/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 259s 5s/step - accuracy: 0.6077 - loss: 1.2004 - val_accuracy: 0.7400 - val_loss: 0.7728
Epoch 3/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 235s 5s/step - accuracy: 0.7758 - loss: 0.7379 - val_accuracy: 0.8400 - val_loss: 0.4633
Epoch 4/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 258s 5s/step - accuracy: 0.9142 - loss: 0.2962 - val_accuracy: 0.8800 - val_loss: 0.3224
Epoch 5/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 269s 5s/step - accuracy: 0.9787 - loss: 0.0822 - val_accuracy: 0.8800 - val_loss: 0.4315
Epoch 6/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 230s 4s/step - accuracy: 0.9937 - loss: 0.0337 - val_accuracy: 0.9200 - val_loss: 0.3299
Epoch 7/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 231s 5s/step - accuracy: 0.9914 - loss: 0.0414 - val_accuracy: 0.9200 - val_loss: 0.2989
Epoch 8/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 233s 5s/step - accuracy: 1.0000 - loss: 0.0077 - val_accuracy: 0.9200 - val_loss: 0.3225
Epoch 9/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 260s 5s/step - accuracy: 1.0000 - loss: 0.0015 - val_accuracy: 0.9200 - val_loss: 0.2846
Epoch 10/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 266s 5s/step - accuracy: 1.0000 - loss: 5.5087e-04 - val_accuracy: 0.9400 - val_loss: 0.2605
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 334ms/step - accuracy: 0.9392 - loss: 0.2776
Test accuracy:  0.9399999976158142
Train Time(h): 0.6978126169575585
Train Time(m): 41.86875701745351
Train Time(s): 2512.1254210472107
```

With incorrect predictions

True: CASSOWARY, Pred: LILAC ROLLER



True: EMPEROR PENGUIN, Pred: SNOWY OWL

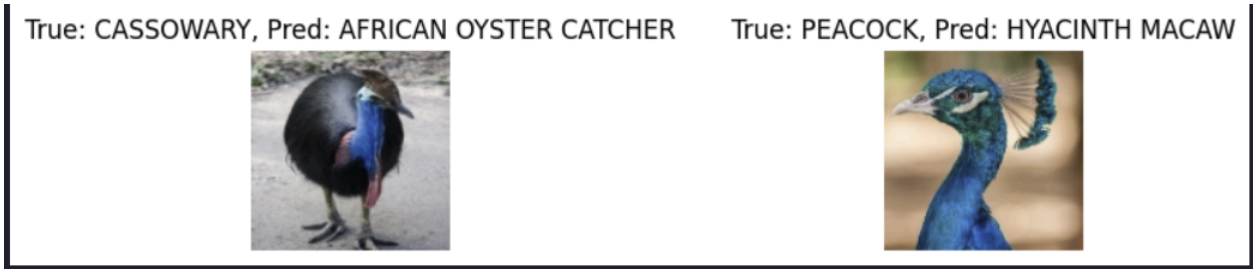
True: PEACOCK, Pred: HYACINTH MACAW

It appears that each model has had difficulty with this image of cassowary and peacock. Again it appears it is due to color. Cassowary train images often have its black feathers and unique body structure in frame but the primary color in this test image is dark green. Moreover the light blue that is similar to lilac roller is present which this specific model gives high importance to. Likewise the peacock to hyacinth macaw prediction is due to the peacock's tail not being in frame. Penguin to snowy owl is likely due to the high amount of white coloring and not the facial features which the model determined to be significant. In this case a way to solve this problem is to include more train data with cropped features of each species and with different color backgrounds. Because removing a layer resulted in higher performance with accuracy of 93% compared to 89% a very basic CNN was tested with the following results

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 126, 126, 32)	896
max_pooling2d (MaxPooling2D)	(None, 63, 63, 32)	0
conv2d_1 (Conv2D)	(None, 61, 61, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 30, 30, 64)	0
conv2d_2 (Conv2D)	(None, 28, 28, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 14, 14, 128)	0
flatten (Flatten)	(None, 25088)	0
dense (Dense)	(None, 128)	3,211,392
dense_1 (Dense)	(None, 10)	1,290

```
Epoch 1/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 55s 1s/step - accuracy: 0.2951 - loss: 2.0121 - val_accuracy: 0.8400 - val_loss: 0.6814
Epoch 2/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 82s 1s/step - accuracy: 0.7643 - loss: 0.7712 - val_accuracy: 0.8800 - val_loss: 0.3823
Epoch 3/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 81s 1s/step - accuracy: 0.8767 - loss: 0.3954 - val_accuracy: 0.8800 - val_loss: 0.3920
Epoch 4/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 82s 1s/step - accuracy: 0.8943 - loss: 0.3120 - val_accuracy: 0.9200 - val_loss: 0.2834
Epoch 5/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 82s 1s/step - accuracy: 0.9394 - loss: 0.1836 - val_accuracy: 0.9400 - val_loss: 0.2571
Epoch 6/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 80s 994ms/step - accuracy: 0.9752 - loss: 0.1077 - val_accuracy: 0.8600 - val_loss: 0.2800
Epoch 7/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 85s 1s/step - accuracy: 0.9666 - loss: 0.1006 - val_accuracy: 0.9600 - val_loss: 0.0945
Epoch 8/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 82s 1s/step - accuracy: 0.9717 - loss: 0.0894 - val_accuracy: 1.0000 - val_loss: 0.0374
Epoch 9/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 53s 1s/step - accuracy: 0.9913 - loss: 0.0340 - val_accuracy: 0.9600 - val_loss: 0.1007
Epoch 10/10
51/51 ━━━━━━━━━━━━━━━━━━━━ 82s 1s/step - accuracy: 0.9988 - loss: 0.0095 - val_accuracy: 0.9600 - val_loss: 0.1043
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 166ms/step - accuracy: 0.9629 - loss: 0.0895
Test accuracy:  0.9599999785423279
Train Time(h): 0.2120225793785519
Train Time(m): 12.721354762713114
Train Time(s): 763.2812857627869
```

And incorrect predictions



This model has performed the best with 96% accuracy. Still has issues with the specific peacock image but the cassowary is likely classified incorrectly because that image framing makes it look like it has the proportions of an oyster catcher which may explain the inaccuracy. Unfortunately it appears all of the other models are overfitting the data. While the other model can be further optimized by testing more layers, different filter amounts, normalization techniques, dense layers and other parameters for future work, it may be concluded that the provided dataset does not require much complexity and adding it results in overfitting of the data. This shows why it is important to test a variety of models and techniques both simple and complex since each dataset will respond differently. As another experiment this model was trained on shuffled train images rather than the sequential train images that are in order of how it was read from directories so each class was clustered together with each other. The result is a model with 98% accuracy however the difference is just 1 image

It is a different image from the non shuffled train data but this may just be due to randomness during training. Overall, concatenated convolutional layers benefit from dense neuron layers to reduce the information more effectively and also benefit from pooling layers in the blocks. However the model can overfit this dataset due to its relatively small size 1603 train images which leads to a simple CNN outperforming it. The benefits of InceptionNet and DenseNet like architectures and blocks are more easily observed in more complex datasets.