

Devices and circuits for Artificial Intelligence

-- 540438 - Maurizio La Rosa --

Final exam project

In this notebook I introduce the project for the final exam of the course *Devices and circuits for artificial intelligence* from the Data Analysis degree of the University of Messina. The project consists in building a machine learning model for image classification.

The dataset to be used is hosted at [kaggle](#), and is called [BIRDS 525 SPECIES- IMAGE CLASSIFICATION](#). The dataset currently contains images from 525 bird species to be classified by the model. I downloaded the dataset on April, 17th, 2023, and that version contains 515 bird species.

It is useful to note that images in the dataset should have all the exact same shape (224, 224, 3), while I found that all images of the 'PLUSH CRESTED JAY' species and one image from the 'DON'T REMEMBER THE SPECIES, FILL INFO WITH FUTURE COMMIT' species have variable shapes. Hence, in my code, I check for images' shapes and remove images that don't match the common shape. This is important because imported images have the shape of 3D Numpy (np, when imported) arrays and I need to transform the list of images into a 4D Numpy array. The function `np.array()` can do it automatically when fed a list of 3D Numpy arrays, but images must have all the same shape.

The following cell is for importing necessary modules in the file used for describing the data, the model and the results. I previously uploaded data for bird species classification on Colab, in the *content/kaggle_data* folder.

```

In [ ]: #####
      ### allow importing from Google Drive ###
      ### after uploading the needed files ###
      #####
      from google.colab import drive
      drive.mount('/content/drive', force_remount=True)
      import sys
      sys.path.insert(0, '/content/drive/MyDrive/Colab Notebooks/da_dcAI_project

      #####
      ### unzip entire dataset from Google ###
      ### Drive to colab's content folder ###
      #####
      !unzip '/content/drive/MyDrive/Colab Notebooks/da_dcAI_project/reshaped_d

      #####
      ### import necessary or useful modules ###
      #####
      import os
      import numpy as np
      #import tensorflow as tf
      from matplotlib import pyplot as plt
      from matplotlib import image as mpimg
      from tensorflow.keras.utils import to_categorical
      from keras.preprocessing.image import ImageDataGenerator
      from sklearn.metrics import classification_report, confusion
      from a_selectRandomFolders import selectRandomFolders
      from b_viewClasses import viewRandomClasses, viewRandomCla
      from c_selectData import selectData, classesToInt, countL
      from d_sequentialModel import seqModel
      from e_plotConfusionMatrix import plotCM

```

Retrieve data folders

The following code cell is used for retrieving the system folders where the data are located.

```
In [ ]: ### 1) set path to the directory of the dataset (this is the targetFolder
### and show how many bird classes there are in the path and their nam
trainPathWin = 'C:/Users/mzlarosa/OneDrive - unime.it/Learning/CdL Informa
testPathWin = 'C:/Users/mzlarosa/OneDrive - unime.it/Learning/CdL Informa
trainPathMac = '/Users/mau/OneDrive - unime.it/Learning/CdL Informatica/A
testPathMac = '/Users/mau/OneDrive - unime.it/Learning/CdL Informatica/An
validPathMac = '/Users/mau/OneDrive - unime.it/Learning/CdL Informatica/A
trainPath = '/content/reshaped/train'
testPath = '/content/reshaped/test'
validPath = '/content/reshaped/valid'
classes = os.listdir(trainPath)
if '.DS_Store' in classes:
    classes.remove('.DS_Store')
nOfClasses = len(classes)
print('\nThere are', nOfClasses, 'classes in the dataset.\n' +
      '\nHere is a list of the first 50 classes:')
print(classes[0 : 50], end = '')
print('[...]')
```

There are 515 classes in the dataset.

Here is a list of the first 50 classes:

```
['VERDIN', 'CANARY', 'DOUBLE BARRED FINCH', 'PARAKETT AKULET', 'CHARA D
E COLLAR', 'RED WISKERED BULBUL', 'BARN SWALLOW', 'NORTHERN MOCKINGBIRD'
, 'WHIMBREL', 'HELMET VANGA', 'TEAL DUCK', 'FAIRY PENGUIN', 'GOLDEN CHEE
KED WARBLER', 'BANDED PITA', 'BARN OWL', 'AMERICAN COOT', 'ASIAN DOLLARD
BIRD', 'SNOWY OWL', 'AMERICAN AVOCET', 'BLONDE CRESTED WOODPECKER', 'JAN
DAYA PARAKEET', 'PEACOCK', 'BROWN HEADED COWBIRD', 'TASMANIAN HEN', 'BLA
CK-THROATED SPARROW', 'UMBRELLA BIRD', 'OSPREY', 'GOLDEN PIPIT', 'GREAT
ARGUS', 'TRICOLORED BLACKBIRD', 'SNOWY PLOVER', 'VARIED THRUSH', 'LONG-E
ARED OWL', 'AVADAVAT', 'WHITE THROATED BEE EATER', 'BLACK THROATED HUET'
, 'HAWFINCH', 'ANDEAN GOOSE', 'WILLOW PTARMIGAN', 'TROPICAL KINGBIRD', '
RUFIOUS TREPE', 'REGENT BOWERBIRD', 'INDIAN ROLLER', 'GOLDEN EAGLE', 'LIM
PKIN', 'SURF SCOTER', 'ECUADORIAN HILLSTAR', 'DUSKY LORY', 'PURPLE GALLI
NULE', 'COMMON FIRECREST'] [...]
```

Introduce the data

The number of available pictures varies with the class.

I select a random sample of 15 bird species from the train data and show how many images are available for each sampled species.

```
In [ ]: ### 2) select a random sample of n (15) subfolders from the targetFolder
### and show their content (subfolders represent bird classes)
### modules: os, random, selectRandomFolders
targetClasses = selectRandomFolders(trainPath, 15)
```

There are 0 folders and 209 image files in /content/reshaped/train/MERLIN

There are 0 folders and 209 image files in /content/reshaped/train/PYRRHULOXIA

There are 0 folders and 160 image files in /content/reshaped/train/RED HEADED DUCK

There are 0 folders and 154 image files in /content/reshaped/train/NORTHERN GANNET

There are 0 folders and 160 image files in /content/reshaped/train/GRAY PARTRIDGE

There are 0 folders and 157 image files in /content/reshaped/train/BOBOLINK

There are 0 folders and 186 image files in /content/reshaped/train/INDIAN PITTA

There are 0 folders and 185 image files in /content/reshaped/train/NORTHERN BEARDLESS TYRANNULET

There are 0 folders and 159 image files in /content/reshaped/train/RED NAPED TROGON

There are 0 folders and 150 image files in /content/reshaped/train/BAIKAL TEAL

There are 0 folders and 155 image files in /content/reshaped/train/TEAL DUCK

There are 0 folders and 197 image files in /content/reshaped/train/RED BEARDED BEE EATER

There are 0 folders and 152 image files in /content/reshaped/train/EASTERN GOLDEN WEAVER

There are 0 folders and 134 image files in /content/reshaped/train/ANDEAN GOOSE

There are 0 folders and 139 image files in /content/reshaped/train/EMERALD TANAGER

Plot pictures from sample species

Plot one picture from 15 randomly selected species

Then I draw a random picture from each class and show their shape and size. A picture's shape shows typically three dimensions. The first two dimensions build a 2D matrix of n rows by m columns. The number of rows represents the image height in pixels, while the number of columns represents the image width in pixels. So each matrix coordinate point represents the intensity value of a single pixel. The third dimension refers to the number of color planes (or channels). There is one plane (2D matrix) for each RGB color, so the value of the third dimension is 3. A picture's size shows its total number of pixels, which results by multiplying the dimensions of the 2D matrices by themselves and by the number of color planes.

```
In [ ]: ### 3) plot one random image from each bird class
### modules: b_viewClasses
randomClasses = viewRandomClasses(trainPath, targetClasses[0])
plt.show()
```

Class: MERLIN
Image shape (rows, columns, channels): (56, 56, 3)
Image size (number of pixels): 9408

Class: PYRRHULOXIA
Image shape (rows, columns, channels): (56, 56, 3)
Image size (number of pixels): 9408

Class: RED HEADED DUCK
Image shape (rows, columns, channels): (56, 56, 3)
Image size (number of pixels): 9408

Class: NORTHERN GANNET
Image shape (rows, columns, channels): (56, 56, 3)
Image size (number of pixels): 9408

Class: GRAY PARTRIDGE
Image shape (rows, columns, channels): (56, 56, 3)
Image size (number of pixels): 9408

Class: BOBOLINK
Image shape (rows, columns, channels): (56, 56, 3)
Image size (number of pixels): 9408

Class: INDIAN PITTA
Image shape (rows, columns, channels): (56, 56, 3)
Image size (number of pixels): 9408

Class: NORTHERN BEARDLESS TYRANNULET
Image shape (rows, columns, channels): (56, 56, 3)
Image size (number of pixels): 9408

Class: RED NAPED TROGON
Image shape (rows, columns, channels): (56, 56, 3)
Image size (number of pixels): 9408

Class: BAIKAL TEAL
Image shape (rows, columns, channels): (56, 56, 3)
Image size (number of pixels): 9408

Class: TEAL DUCK
Image shape (rows, columns, channels): (56, 56, 3)
Image size (number of pixels): 9408

Class: RED BEARDED BEE EATER
Image shape (rows, columns, channels): (56, 56, 3)
Image size (number of pixels): 9408

Class: EASTERN GOLDEN WEAVER
Image shape (rows, columns, channels): (56, 56, 3)
Image size (number of pixels): 9408

Class: ANDEAN GOOSE

Image shape (rows, columns, channels): (56, 56, 3)

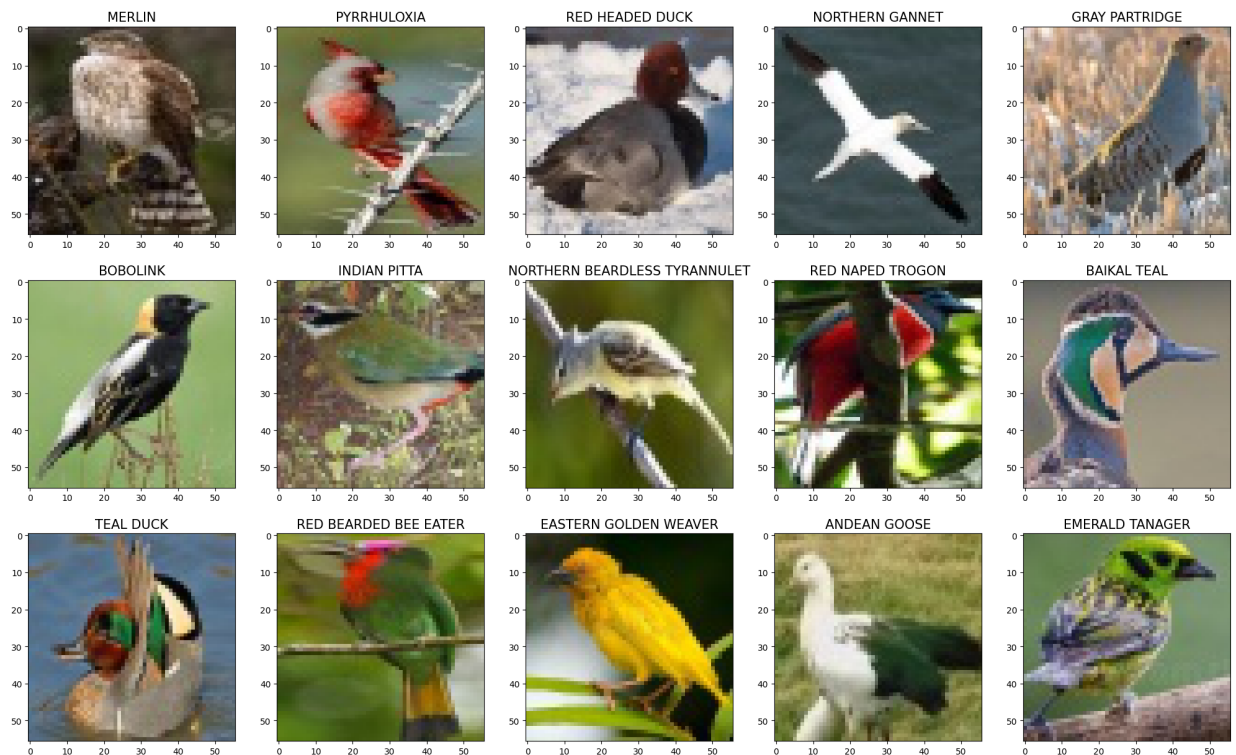
Image size (number of pixels): 9408

Class: EMERALD TANAGER

Image shape (rows, columns, channels): (56, 56, 3)

Image size (number of pixels): 9408

Sample of 15 bird classes



Plot 15 pictures from one of the previously selected random species

Finally, we print 15 random images from one of the previously chosen species and show their shape and size as before defined.

```
In [ ]: ### 4) plot 15 random images from one of the classes
### modules: b_viewClasses
randomClass = viewRandomClass(trainPath, targetClasses[0])
plt.show()
print('\nI conclude that, although there is a varying number of images',
      'for each bird class,')
print('in our sample of 15 classes there is a minimum',
      'of', min(targetClasses[2]), 'images, and a maximum of',
      max(targetClasses[2]), 'images.\n')
```

INDIAN PITTA

Image shape (rows, columns, channels): (56, 56, 3)

Image size (number of pixels): 9408

Image shape (rows, columns, channels): (56, 56, 3)

Image size (number of pixels): 9408

Image shape (rows, columns, channels): (56, 56, 3)

Image size (number of pixels): 9408

Image shape (rows, columns, channels): (56, 56, 3)

Image size (number of pixels): 9408

Image shape (rows, columns, channels): (56, 56, 3)

Image size (number of pixels): 9408

Image shape (rows, columns, channels): (56, 56, 3)

Image size (number of pixels): 9408

Image shape (rows, columns, channels): (56, 56, 3)

Image size (number of pixels): 9408

Image shape (rows, columns, channels): (56, 56, 3)

Image size (number of pixels): 9408

Image shape (rows, columns, channels): (56, 56, 3)

Image size (number of pixels): 9408

Image shape (rows, columns, channels): (56, 56, 3)

Image size (number of pixels): 9408

Image shape (rows, columns, channels): (56, 56, 3)

Image size (number of pixels): 9408

Image shape (rows, columns, channels): (56, 56, 3)

Image size (number of pixels): 9408

Image shape (rows, columns, channels): (56, 56, 3)

Image size (number of pixels): 9408

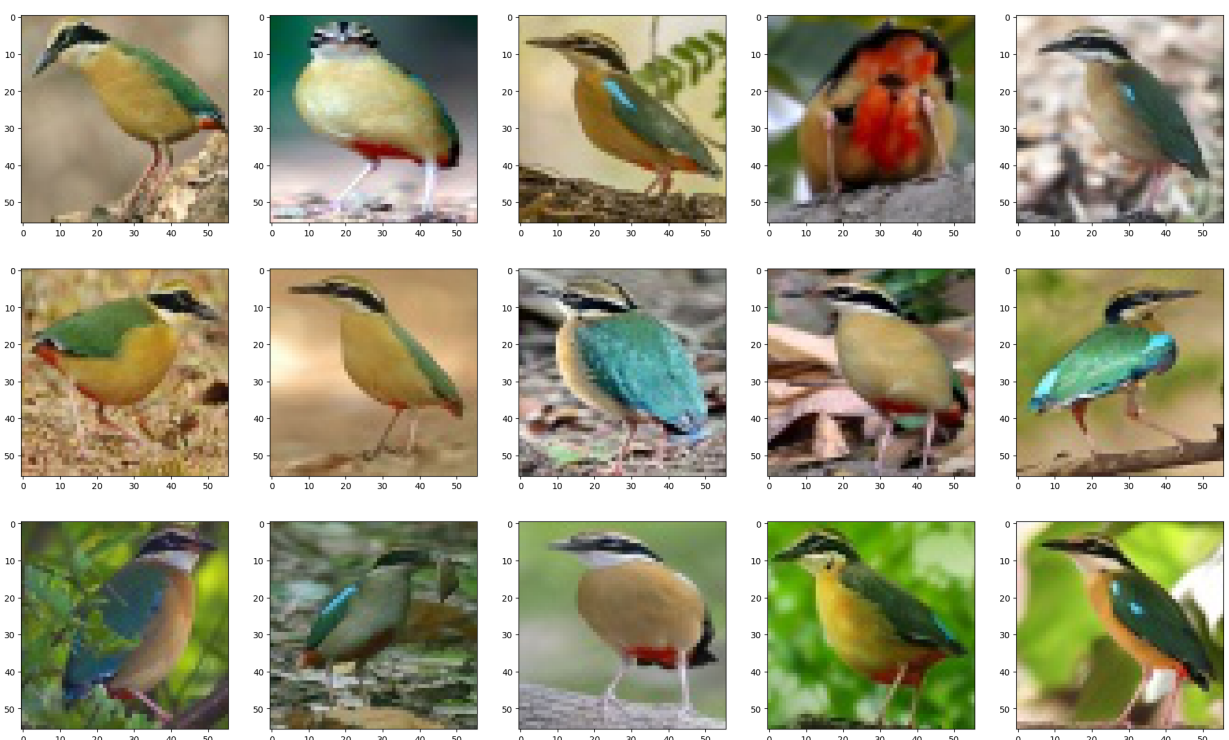
Image shape (rows, columns, channels): (56, 56, 3)

Image size (number of pixels): 9408

Image shape (rows, columns, channels): (56, 56, 3)

Image size (number of pixels): 9408

INDIAN PITTA



I conclude that, although there is a varying number of images for each bird class, in our sample of 15 classes there is a minimum of 134 images, and a maximum of 209 images.

Data preparation

The following code allows me to load each image of the train and test sets into Python lists. The lists containing the images are then turned into numerical arrays, with 4 dimensions: the first one represents the number of images, while the other three represent the images' shape, which has been rendered homogeneous (56, 56, 3) by removing those not matching the common shape within the execution of the custom *selectData* function.

```
In [ ]: ### 5) load the train and test data and labels into memory
trainData, trainClasses = selectData(trainPath)
testData, testClasses = selectData(testPath)
#train data list into numpy array
npTrainData = np.array(trainData)
#test data list into numpy array
npTestData = np.array(testData)

print('Our train data array has', npTrainData.ndim, 'dimensions, and a shape of',
      npTrainData.shape, 'for a total number of elements of', npTrainData.size)
print('Our test data array has', npTestData.ndim, 'dimensions, and a shape of',
      npTestData.shape, 'for a total number of elements of', npTestData.size)
```

Our train data array has 4 dimensions, and a shape of (82724, 56, 56, 3) for a total number of elements of 778267392 .
Our test data array has 4 dimensions, and a shape of (2575, 56, 56, 3) for a total number of elements of 24225600 .

The data labels represent the bird species to which the images belong. They are first converted into 2D arrays and finally turned into categorical data: bird classes are substituted by numerical categories.

Bird species should be homogeneous across train and test data, so I check if it's actually the case.


```
In [ ]: ### 6) turn the train and test labels into categorical arrays
# train labels

trainClasses = classesToInt(trainClasses)
npTrainClasses = np.array(trainClasses)
npTrainClasses = np.expand_dims(npTrainClasses, axis = 1) # add dimension
npTrainLabels = to_categorical(npTrainClasses)
# test labels
testClasses = classesToInt(testClasses)
npTestClasses = np.array(testClasses)
npTestClasses = np.expand_dims(npTestClasses, axis = 1) # add dimension t
npTestLabels = to_categorical(npTestClasses)

trainCount = countLabels(npTrainClasses) # count train labels
testCount = countLabels(npTestClasses) # count test labels
if trainCount == testCount:
    print('There are', npTrainClasses.size, 'categories (it means that ea
        'belongs to a category) for a set of', trainCount, 'categories.
else:
    print('ERROR: train labels count and test labels count don\'t match.')

# free up memory
del trainData, testData, trainClasses, npTrainClasses, testClasses, npTes
```

There are 82724 categories (it means that each image belongs to a category) for a set of 515 categories.

Data normalization

At this point I normalize the data by dividing them for the maximum value they can assume. In this way the data range from 0 to 1 and allow for better speed and prediction results.

```
In [ ]: ### 7) normalize the data
npTrainData = np.array(npTrainData / npTrainData.max(), dtype = np.float16)
npTestData = np.array(npTestData / npTestData.max(), dtype = np.float16)
```

Call the sequential model

I call the function executing the sequential model from the file d_sequentialModel. The function returns the model history, which is used to train the model and plot the loss function and the model accuracy.

```
In [ ]: ### 8) call the sequential model
batch_size = 64 # batch size to be used in model.fit
test_datagen = ImageDataGenerator(rescale = 1. / 255)
nTestSamples = npTestData.shape[0] # number of images in test folder

validGen = test_datagen.flow_from_directory(testPath,
                                             target_size=(56, 56),
                                             batch_size = batch_size,
                                             class_mode = 'categorical') #

myModel, Y_pred = seqModel(npTrainData, npTrainLabels, npTestData, npTest
```

Found 2575 images belonging to 515 classes.

Model: "sequential"

Layer (type)	Output Shape	Param #
=====		
conv2d (Conv2D)	(None, 54, 54, 16)	448
max_pooling2d (MaxPooling2D)	(None, 18, 18, 16)	0
ReLU (Activation)	(None, 18, 18, 16)	0
conv2d_1 (Conv2D)	(None, 16, 16, 32)	4640
ReLU2 (Activation)	(None, 16, 16, 32)	0
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 32)	0
conv2d_2 (Conv2D)	(None, 3, 3, 64)	18496
ReLU3 (Activation)	(None, 3, 3, 64)	0
max_pooling2d_2 (MaxPooling2D)	(None, 1, 1, 64)	0
flatten (Flatten)	(None, 64)	0
FC2 (Dense)	(None, 515)	33475
Softmax (Activation)	(None, 515)	0

```
=====
Total params: 57,059
Trainable params: 57,059
Non-trainable params: 0
```

```
Epoch 1/10
1035/1035 [=====] - 20s 9ms/step - loss: 5.3981
- accuracy: 0.0345 - val_loss: 24.6033 - val_accuracy: 0.0000e+00
Epoch 2/10
```

```

1035/1035 [=====] - 6s 6ms/step - loss: 4.2580
- accuracy: 0.1431 - val_loss: 27.5712 - val_accuracy: 6.0441e-05
Epoch 3/10
1035/1035 [=====] - 7s 7ms/step - loss: 3.7446
- accuracy: 0.2226 - val_loss: 27.5907 - val_accuracy: 6.0441e-05
Epoch 4/10
1035/1035 [=====] - 7s 7ms/step - loss: 3.4303
- accuracy: 0.2763 - val_loss: 30.8156 - val_accuracy: 6.0441e-05
Epoch 5/10
1035/1035 [=====] - 8s 7ms/step - loss: 3.2103
- accuracy: 0.3164 - val_loss: 31.7516 - val_accuracy: 6.0441e-05
Epoch 6/10
1035/1035 [=====] - 7s 6ms/step - loss: 3.0480
- accuracy: 0.3435 - val_loss: 32.9528 - val_accuracy: 6.0441e-05
Epoch 7/10
1035/1035 [=====] - 8s 7ms/step - loss: 2.9194
- accuracy: 0.3677 - val_loss: 33.0627 - val_accuracy: 6.0441e-05
Epoch 8/10
1035/1035 [=====] - 7s 7ms/step - loss: 2.8110
- accuracy: 0.3894 - val_loss: 33.2605 - val_accuracy: 6.0441e-05
Epoch 9/10
1035/1035 [=====] - 7s 7ms/step - loss: 2.7248
- accuracy: 0.4019 - val_loss: 33.8748 - val_accuracy: 6.0441e-05
Epoch 10/10
1035/1035 [=====] - 7s 7ms/step - loss: 2.6550
- accuracy: 0.4180 - val_loss: 34.5761 - val_accuracy: 6.0441e-05

Test loss: 9.046629905700684
Test accuracy: 0.3852427303791046
41/41 [=====] - 3s 59ms/step

```

I finally plot the loss function for the model and the model accuracy.

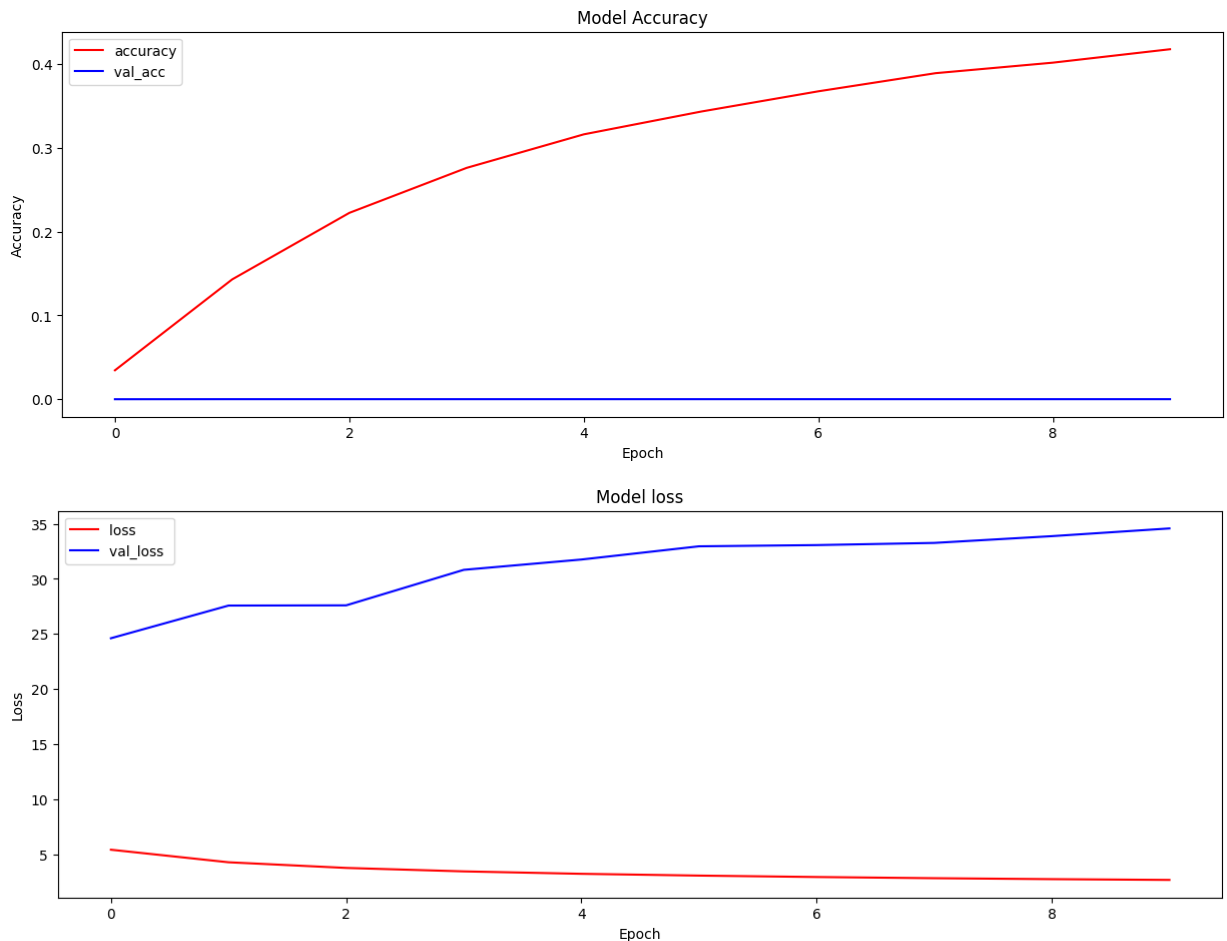
```

In [ ]: ### 9) plot model accuracy and loss function
# accuracy
plt.figure(figsize=(15, 5))
plt.plot(myModel.history['accuracy'], 'r', label='accuracy')
plt.plot(myModel.history['val_accuracy'], 'b', label='val_acc ')
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend()

# loss
plt.figure(figsize=(15, 5))
plt.plot(myModel.history['loss'], 'r', label='loss ')
plt.plot(myModel.history['val_loss'], 'b', label='val_loss ')
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend()

plt.show()

```



Plot confusion matrix and classification report

I finally plot the confusion matrix and the classification report for a small sample of bird species.

```
In [ ]: ### 10) Confution Matrix and Classification Report
y_pred = np.argmax(Y_pred, axis=1)
cm = confusion_matrix(validGen.classes, y_pred)
thresh = cm.max() / 2.
tick_marks = np.arange(len(classes))
target_names = classes
print('\nConfusion Matrix (small sample)\n')
print(cm[0 : 15, 0 : 15])
plotCM(cm[0 : 15, 0 : 15], classes[0 : 15])
print('\n\nClassification Report (small sample)\n')
class_report = classification_report(validGen.classes, y_pred, target_names)
print(class_report[0 : 1000], (' ' * 18) + '[...]\n') # print a few lines
```

Confusion Matrix (small sample)

```

[[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]]

```

Classification Report (small sample)

	precision	recall	f1-score	support
VERDIN	0.00	0.00	0.00	5
CANARY	0.00	0.00	0.00	5
DOUBLE BARRED FINCH	0.00	0.00	0.00	5
PARAKETT AKULET	0.00	0.00	0.00	5
CHARA DE COLLAR	0.00	0.00	0.00	5
RED WISKERED BULBUL	0.00	0.00	0.00	5
BARN SWALLOW	0.00	0.00	0.00	5
NORTHERN MOCKINGBIRD	0.00	0.00	0.00	5
WHIMBREL	0.00	0.00	0.00	5
HELMET VANGA	0.00	0.00	0.00	5
TEAL DUCK	0.00	0.00	0.00	5
FAIRY PENGUIN	0.00	0.00	0.00	5
GOLDEN CHEEKED WARBLER	0.00	0.00	0.00	5
[...]				

```

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.

```

```

_warn_prf(average, modifier, msg_start, len(result))

```

```

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.

```

```

_warn_prf(average, modifier, msg_start, len(result))

```

```

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.

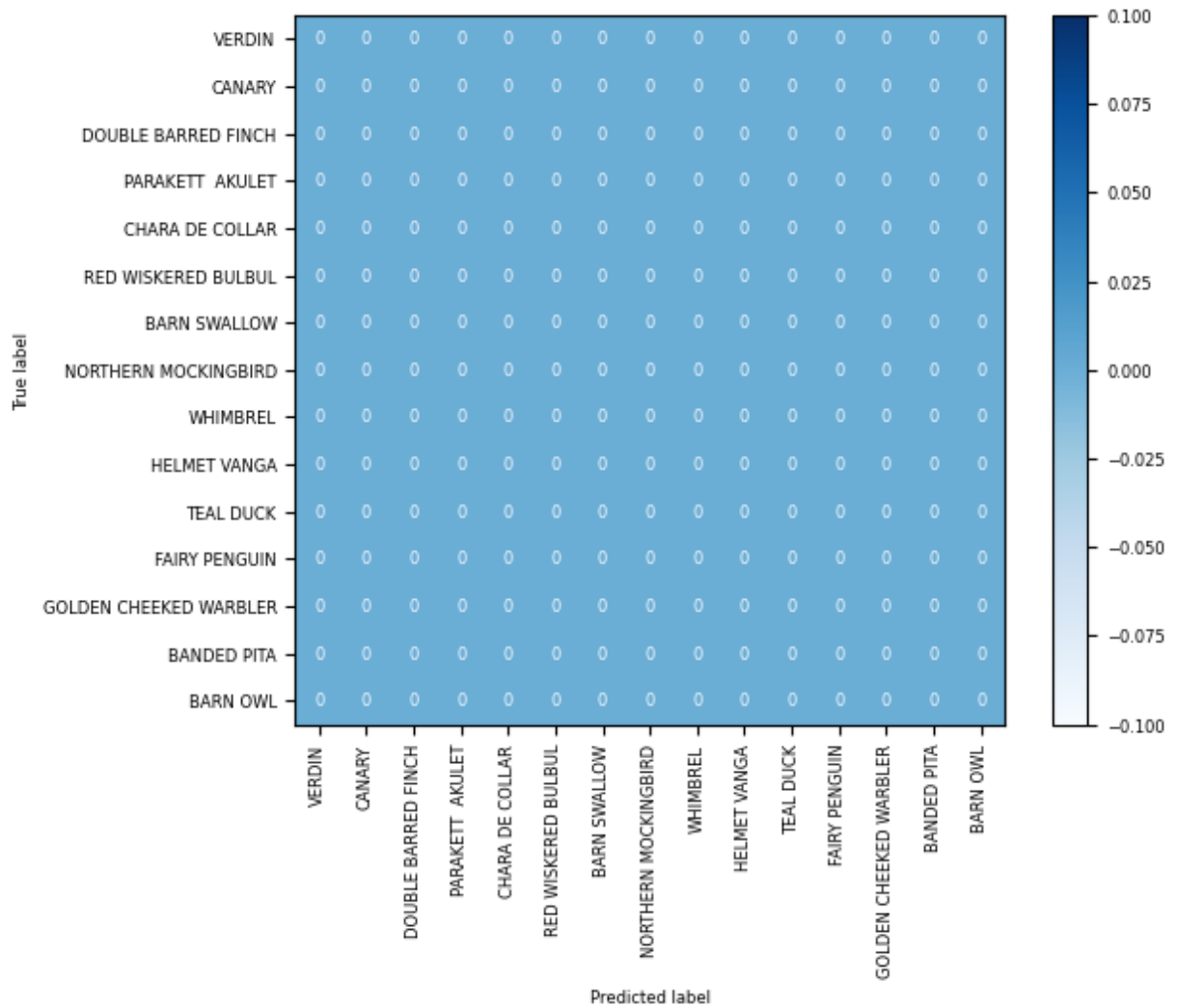
```

```

_warn_prf(average, modifier, msg_start, len(result))

```

<Figure size 600x600 with 0 Axes>



In []: