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 [1]:
```
##########################################
### 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
```

```
Mounted at /content/drive
```

Retrieve data folders

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

In [2]:
```
### 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 Inform
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', '
RUFOUS 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 [3]:
```
### 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 141 image files in /content/reshaped/train/TAILO
RBIRD
There are 0 folders and 154 image files in /content/reshaped/train/FAIRY
PENGUIN
There are 0 folders and 150 image files in /content/reshaped/train/EASTE
RN WIP POOR WILL
There are 0 folders and 164 image files in /content/reshaped/train/JAVA
SPARROW
There are 0 folders and 153 image files in /content/reshaped/train/CHARA
DE COLLAR
There are 0 folders and 161 image files in /content/reshaped/train/CINNA
MON TEAL
There are 0 folders and 144 image files in /content/reshaped/train/IMPER
IAL SHAQ
There are 0 folders and 138 image files in /content/reshaped/train/WHIMB
REL
There are 0 folders and 168 image files in /content/reshaped/train/BAR-T
AILED GODWIT
There are 0 folders and 155 image files in /content/reshaped/train/BLUE
COAU
There are 0 folders and 155 image files in /content/reshaped/train/BULWE
RS PHEASANT
There are 0 folders and 213 image files in /content/reshaped/train/CASPI
AN TERN
There are 0 folders and 155 image files in /content/reshaped/train/BLACK
SKIMMER
There are 0 folders and 175 image files in /content/reshaped/train/LITTL
E AUK
There are 0 folders and 157 image files in /content/reshaped/train/ORIEN
TAL BAY OWL
```

Plot pictures from sample species

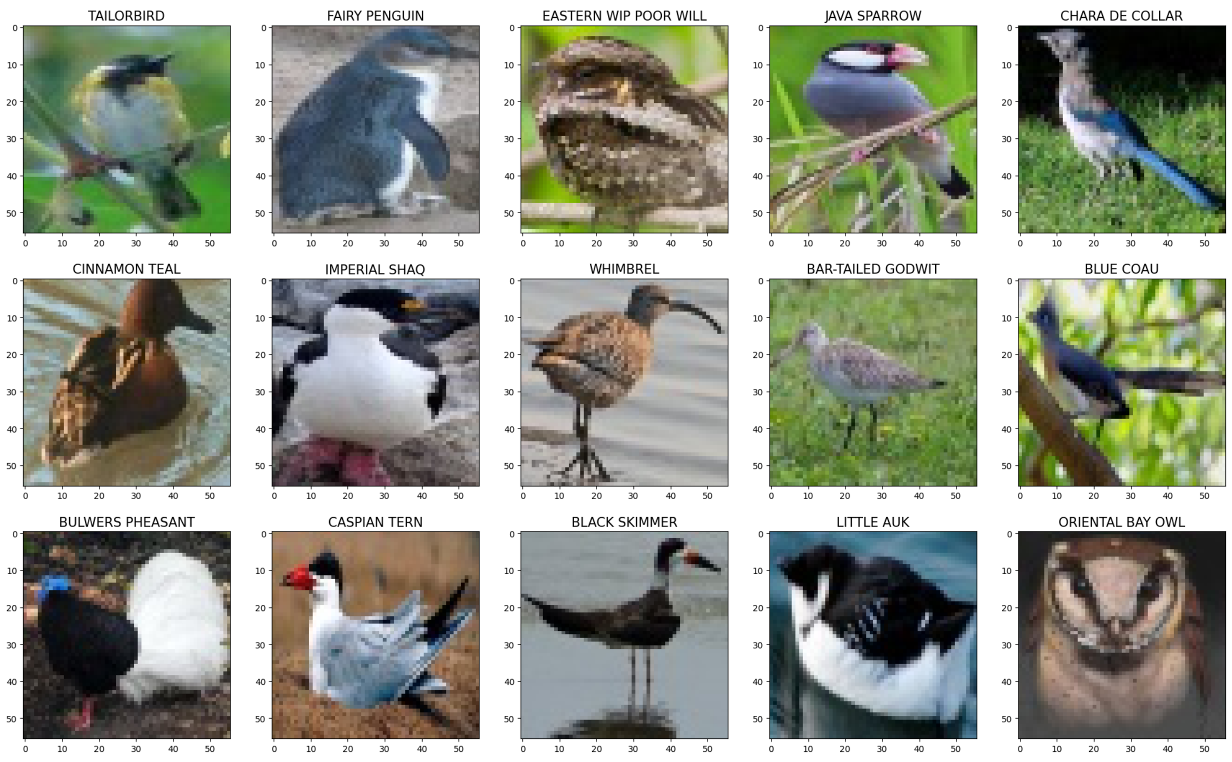
Plot one picture from 15 randomly selected species

Then I draw a random picture from each class and show their shape and size. A picture's shape shows tipically 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 [4]:
```
### 3) plot one random image from each bird class
###    modules: b_viewClasses
randomClasses = viewRandomClasses(trainPath, targetClasses[0])
plt.show()
```

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

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

Class: EASTERN WIP POOR WILL
Image shape (rows, columns, channels): (56, 56, 3)
Image size (number of pixels): 9408

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

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

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

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

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

Class: BAR-TAILED GODWIT
Image shape (rows, columns, channels): (56, 56, 3)
Image size (number of pixels): 9408

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

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

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

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

Class: LITTLE AUK
```

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

Class: ORIENTAL BAY OWL
Image shape (rows, columns, channels): (56, 56, 3)
Image size (number of pixels): 9408
```



Sample of 15 bird classes

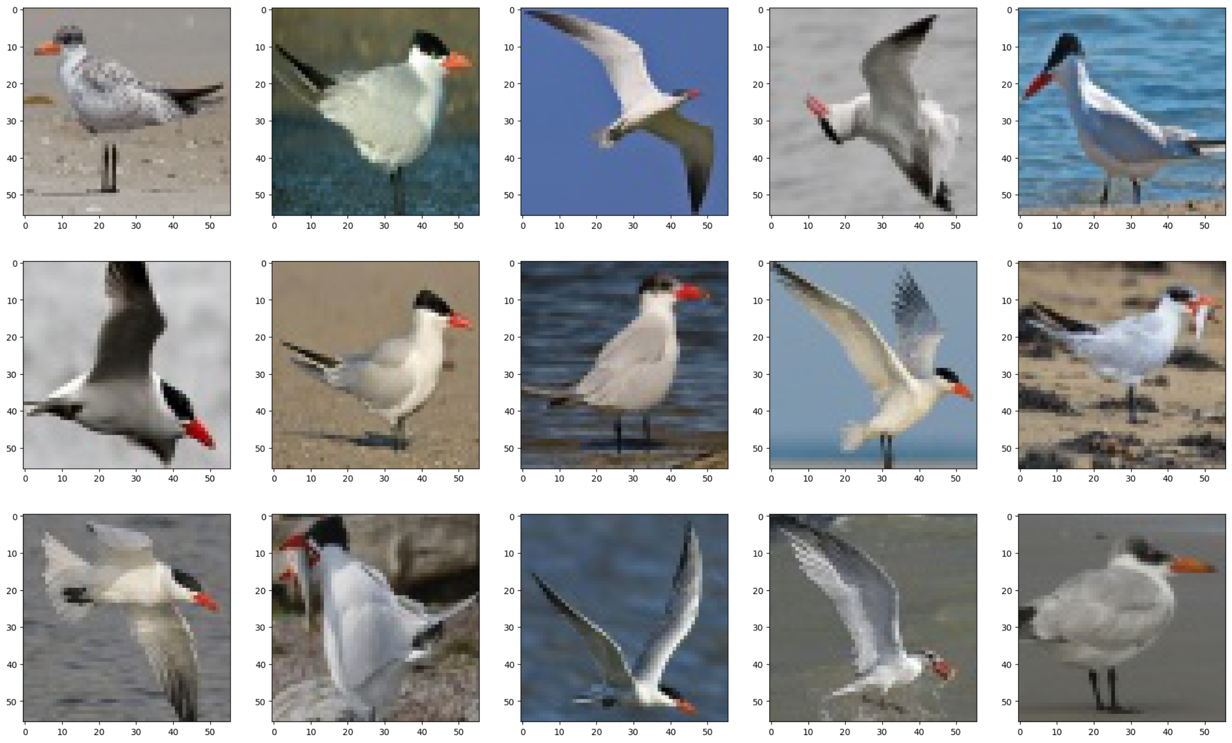
Plot 15 pictures from one of the previously selected random species

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

In [5]:
```python
### 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')
```

```
CASPIAN TERN
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
```

CASPIAN TERN



```
I conclude that, although there is a varying number of images for each b
ird class,
in our sample of 15 classes there is a minimum of 138 images, and a maxi
mum of 213 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 [6]:
```
### 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 sh
      npTrainData.shape, 'for a total number of elements of', npTrainData
print('Our test data array has', npTestData.ndim, 'dimensions, and a shap
      npTestData.shape, 'for a total number of elements of', npTestData.s
```

```
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) f
or 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 [7]:
```
### 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 catego
ry) 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 [8]:
```
### 7) normalize the data
npTrainData = np.array(npTrainData / npTrainData.max(), dtype = np.float1
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 [9]:
```
### 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
```

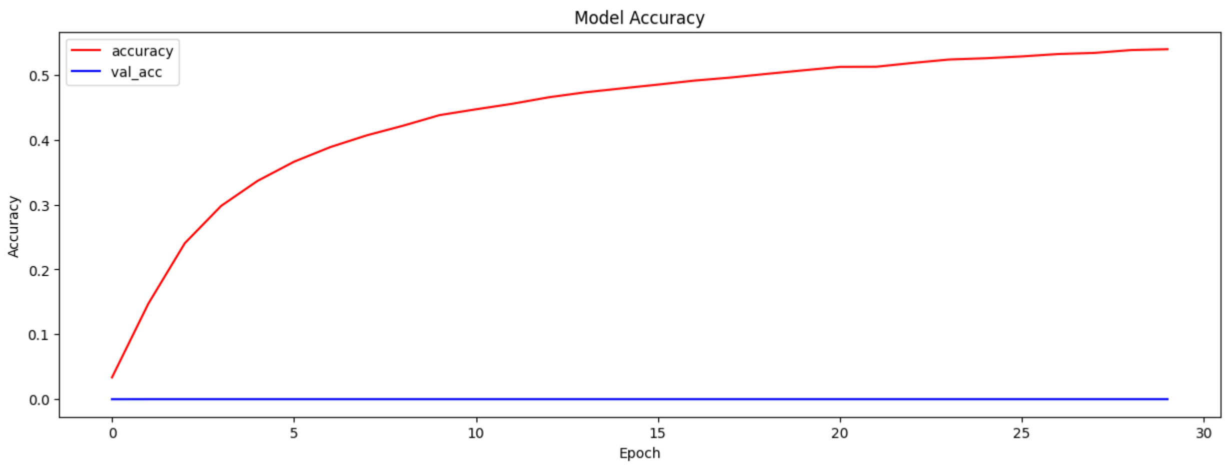
```
Found 2575 images belonging to 515 classes.
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 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 (MaxPooling  (None, 5, 5, 32)         0         
 2D)                                                             
                                                                 
 conv2d_2 (Conv2D)           (None, 3, 3, 64)          18496     
                                                                 
 ReLU3 (Activation)          (None, 3, 3, 64)          0         
                                                                 
 max_pooling2d_2 (MaxPooling  (None, 1, 1, 64)         0         
 2D)                                                             
                                                                 
 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 1/30
1035/1035 [==============================] - 21s 11ms/step - loss: 5.398
7 - accuracy: 0.0336 - val_loss: 19.8976 - val_accuracy: 0.0000e+00
Epoch 2/30
```

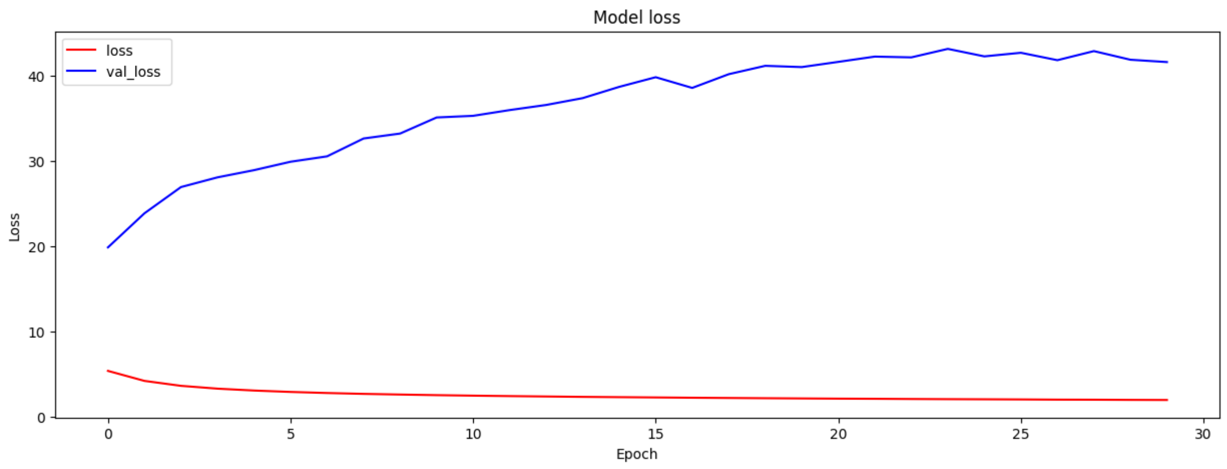
```
1035/1035 [==============================] - 8s 7ms/step - loss: 4.2244
- accuracy: 0.1474 - val_loss: 23.9008 - val_accuracy: 6.0441e-05
Epoch 3/30
1035/1035 [==============================] - 9s 9ms/step - loss: 3.6448
- accuracy: 0.2405 - val_loss: 26.9793 - val_accuracy: 6.0441e-05
Epoch 4/30
1035/1035 [==============================] - 8s 8ms/step - loss: 3.3141
- accuracy: 0.2981 - val_loss: 28.1184 - val_accuracy: 6.0441e-05
Epoch 5/30
1035/1035 [==============================] - 8s 8ms/step - loss: 3.0929
- accuracy: 0.3366 - val_loss: 28.9626 - val_accuracy: 6.0441e-05
Epoch 6/30
1035/1035 [==============================] - 8s 8ms/step - loss: 2.9290
- accuracy: 0.3661 - val_loss: 29.9495 - val_accuracy: 6.0441e-05
Epoch 7/30
1035/1035 [==============================] - 7s 7ms/step - loss: 2.8029
- accuracy: 0.3887 - val_loss: 30.5813 - val_accuracy: 6.0441e-05
Epoch 8/30
1035/1035 [==============================] - 8s 8ms/step - loss: 2.7009
- accuracy: 0.4068 - val_loss: 32.6764 - val_accuracy: 6.0441e-05
Epoch 9/30
1035/1035 [==============================] - 9s 8ms/step - loss: 2.6227
- accuracy: 0.4217 - val_loss: 33.2564 - val_accuracy: 6.0441e-05
Epoch 10/30
1035/1035 [==============================] - 8s 8ms/step - loss: 2.5516
- accuracy: 0.4380 - val_loss: 35.1444 - val_accuracy: 6.0441e-05
Epoch 11/30
1035/1035 [==============================] - 8s 7ms/step - loss: 2.4922
- accuracy: 0.4470 - val_loss: 35.3377 - val_accuracy: 6.0441e-05
Epoch 12/30
1035/1035 [==============================] - 7s 7ms/step - loss: 2.4398
- accuracy: 0.4555 - val_loss: 36.0121 - val_accuracy: 6.0441e-05
Epoch 13/30
1035/1035 [==============================] - 8s 7ms/step - loss: 2.3943
- accuracy: 0.4655 - val_loss: 36.6137 - val_accuracy: 6.0441e-05
Epoch 14/30
1035/1035 [==============================] - 8s 7ms/step - loss: 2.3495
- accuracy: 0.4732 - val_loss: 37.4161 - val_accuracy: 6.0441e-05
Epoch 15/30
1035/1035 [==============================] - 8s 8ms/step - loss: 2.3149
- accuracy: 0.4792 - val_loss: 38.7378 - val_accuracy: 6.0441e-05
Epoch 16/30
1035/1035 [==============================] - 8s 7ms/step - loss: 2.2821
- accuracy: 0.4850 - val_loss: 39.8723 - val_accuracy: 6.0441e-05
Epoch 17/30
1035/1035 [==============================] - 8s 8ms/step - loss: 2.2480
- accuracy: 0.4912 - val_loss: 38.6154 - val_accuracy: 6.0441e-05
Epoch 18/30
1035/1035 [==============================] - 7s 7ms/step - loss: 2.2193
- accuracy: 0.4959 - val_loss: 40.2263 - val_accuracy: 6.0441e-05
Epoch 19/30
1035/1035 [==============================] - 8s 8ms/step - loss: 2.1908
- accuracy: 0.5016 - val_loss: 41.2084 - val_accuracy: 6.0441e-05
```

```
Epoch 20/30
1035/1035 [==============================] - 7s 7ms/step - loss: 2.1649
- accuracy: 0.5071 - val_loss: 41.0609 - val_accuracy: 6.0441e-05
Epoch 21/30
1035/1035 [==============================] - 8s 8ms/step - loss: 2.1401
- accuracy: 0.5124 - val_loss: 41.6716 - val_accuracy: 6.0441e-05
Epoch 22/30
1035/1035 [==============================] - 7s 7ms/step - loss: 2.1242
- accuracy: 0.5126 - val_loss: 42.2913 - val_accuracy: 6.0441e-05
Epoch 23/30
1035/1035 [==============================] - 8s 7ms/step - loss: 2.0997
- accuracy: 0.5185 - val_loss: 42.2000 - val_accuracy: 6.0441e-05
Epoch 24/30
1035/1035 [==============================] - 7s 6ms/step - loss: 2.0803
- accuracy: 0.5237 - val_loss: 43.1978 - val_accuracy: 6.0441e-05
Epoch 25/30
1035/1035 [==============================] - 8s 7ms/step - loss: 2.0651
- accuracy: 0.5257 - val_loss: 42.3171 - val_accuracy: 6.0441e-05
Epoch 26/30
1035/1035 [==============================] - 7s 7ms/step - loss: 2.0480
- accuracy: 0.5285 - val_loss: 42.7404 - val_accuracy: 6.0441e-05
Epoch 27/30
1035/1035 [==============================] - 7s 7ms/step - loss: 2.0247
- accuracy: 0.5322 - val_loss: 41.8695 - val_accuracy: 6.0441e-05
Epoch 28/30
1035/1035 [==============================] - 7s 7ms/step - loss: 2.0156
- accuracy: 0.5339 - val_loss: 42.9363 - val_accuracy: 6.0441e-05
Epoch 29/30
1035/1035 [==============================] - 7s 7ms/step - loss: 1.9957
- accuracy: 0.5383 - val_loss: 41.9227 - val_accuracy: 6.0441e-05
Epoch 30/30
1035/1035 [==============================] - 7s 6ms/step - loss: 1.9834
- accuracy: 0.5396 - val_loss: 41.6505 - val_accuracy: 6.0441e-05

Test loss: 10.312976837158203
Test accuracy: 0.4563106894493103
41/41 [==============================] - 2s 42ms/step
```

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

In [10]:
```python
### 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

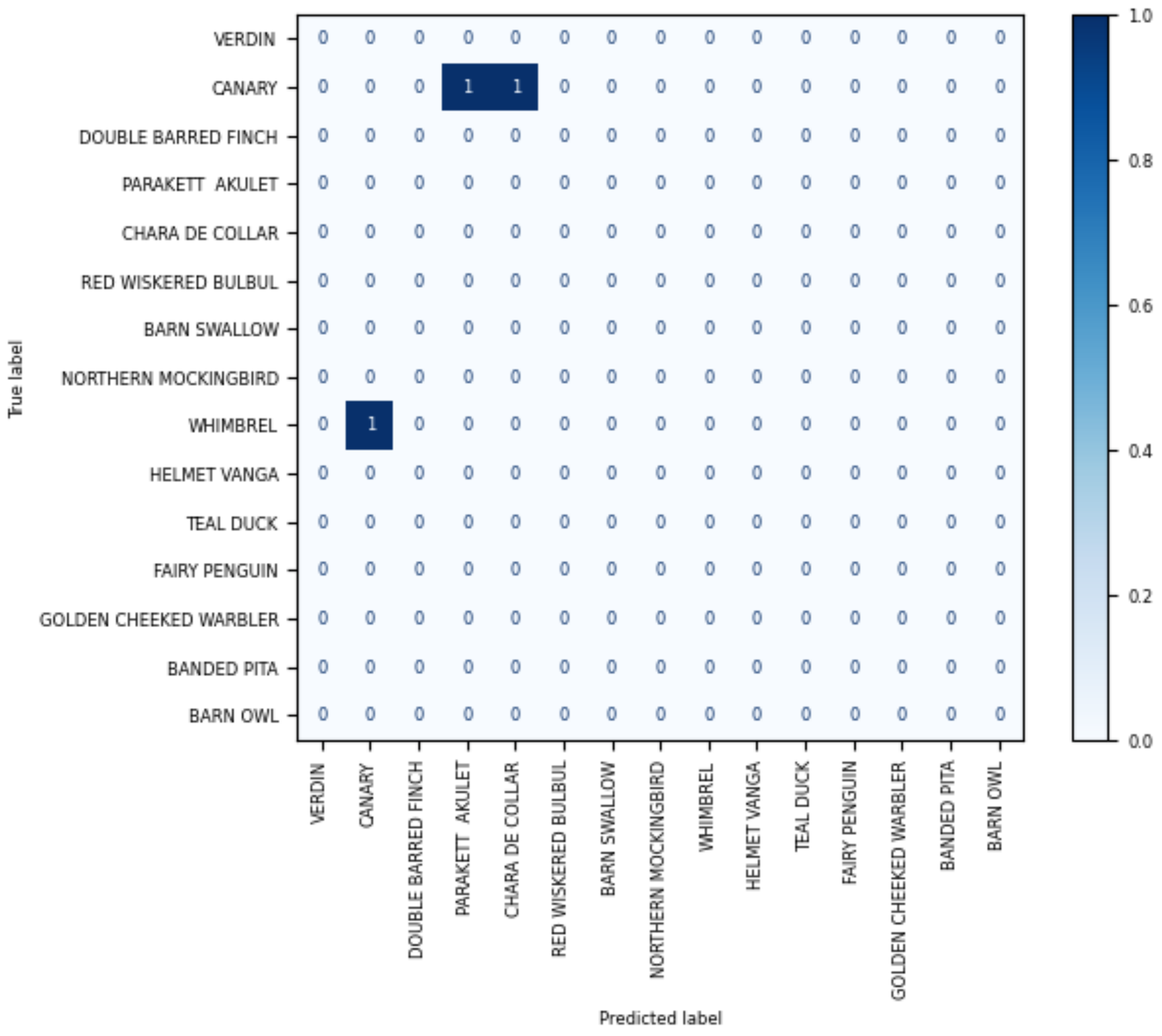
I finally plot the confusion matrix and the classification report for a small sample of bird species.

In [11]:
```
### 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_nam
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 1 1 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 1 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 a
nd being set to 0.0 in labels with no predicted samples. Use `zero_divis
ion` 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 a
nd being set to 0.0 in labels with no predicted samples. Use `zero_divis
ion` 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 a
nd being set to 0.0 in labels with no predicted samples. Use `zero_divis
ion` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
<Figure size 600x600 with 0 Axes>
```



	VERDIN	CANARY	DOUBLE BARRED FINCH	PARAKETT AKULET	CHARA DE COLLAR	RED WISKERED BULBUL	BARN SWALLOW	NORTHERN MOCKINGBIRD	WHIMBREL	HELMET VANGA	TEAL DUCK	FAIRY PENGUIN	GOLDEN CHEEKED WARBLER	BANDED PITA	BARN OWL
VERDIN	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CANARY	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0
DOUBLE BARRED FINCH	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PARAKETT AKULET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CHARA DE COLLAR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RED WISKERED BULBUL	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
BARN SWALLOW	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
NORTHERN MOCKINGBIRD	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
WHIMBREL	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
HELMET VANGA	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
TEAL DUCK	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FAIRY PENGUIN	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
GOLDEN CHEEKED WARBLER	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
BANDED PITA	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
BARN OWL	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

```
In [ ]:
```