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 [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
validPathWin = 'C:/Users/mzlarosa/OneDrive - unime.it/Learning/CdL Inform
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 165 image files in /content/reshaped/train/HYACI
NTH MACAW
There are 0 folders and 152 image files in /content/reshaped/train/VENEZ
UELIAN TROUPIAL
There are 0 folders and 133 image files in /content/reshaped/train/AMERI
CAN GOLDFINCH
There are 0 folders and 150 image files in /content/reshaped/train/ANTBI
RD
There are 0 folders and 144 image files in /content/reshaped/train/ELEGA
NT TROGON
There are 0 folders and 157 image files in /content/reshaped/train/SCARL
ET MACAW
There are 0 folders and 172 image files in /content/reshaped/train/COLLA
RED CRESCENTCHEST
There are 0 folders and 163 image files in /content/reshaped/train/COCK
OF THE  ROCK
There are 0 folders and 208 image files in /content/reshaped/train/MILIT
ARY MACAW
There are 0 folders and 155 image files in /content/reshaped/train/AMERI
CAN KESTREL
There are 0 folders and 150 image files in /content/reshaped/train/HIMAL
AYAN BLUETAIL
There are 0 folders and 173 image files in /content/reshaped/train/BLUE
DACNIS
There are 0 folders and 144 image files in /content/reshaped/train/CHINE
SE POND HERON
There are 0 folders and 183 image files in /content/reshaped/train/INDIA
N ROLLER
There are 0 folders and 185 image files in /content/reshaped/train/SURF
SCOTER
```

Plot pictures from sample species

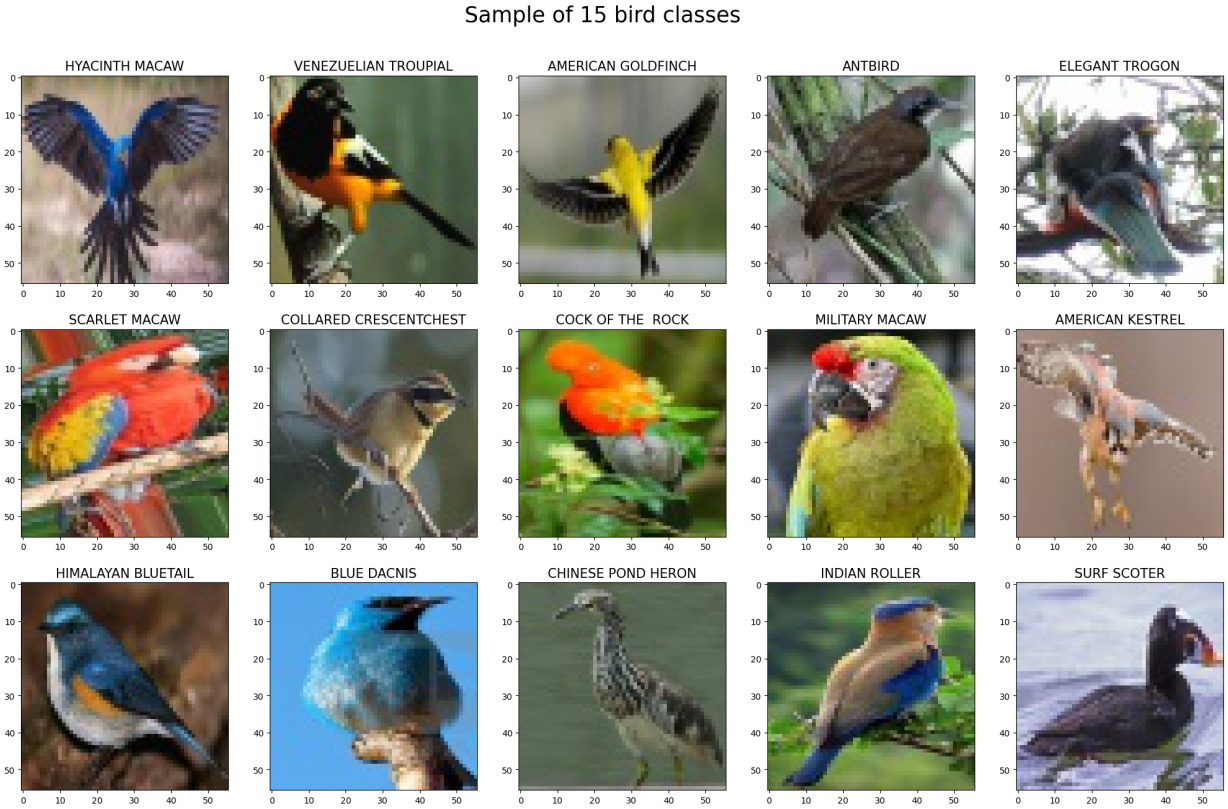
Plot one picture from 15 randomly selected species

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

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

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

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

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

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

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

Class: COCK OF THE  ROCK
Image shape (rows, columns, channels): (56, 56, 3)
Image size (number of pixels): 9408

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

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

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

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

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

Class: INDIAN ROLLER
```

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

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



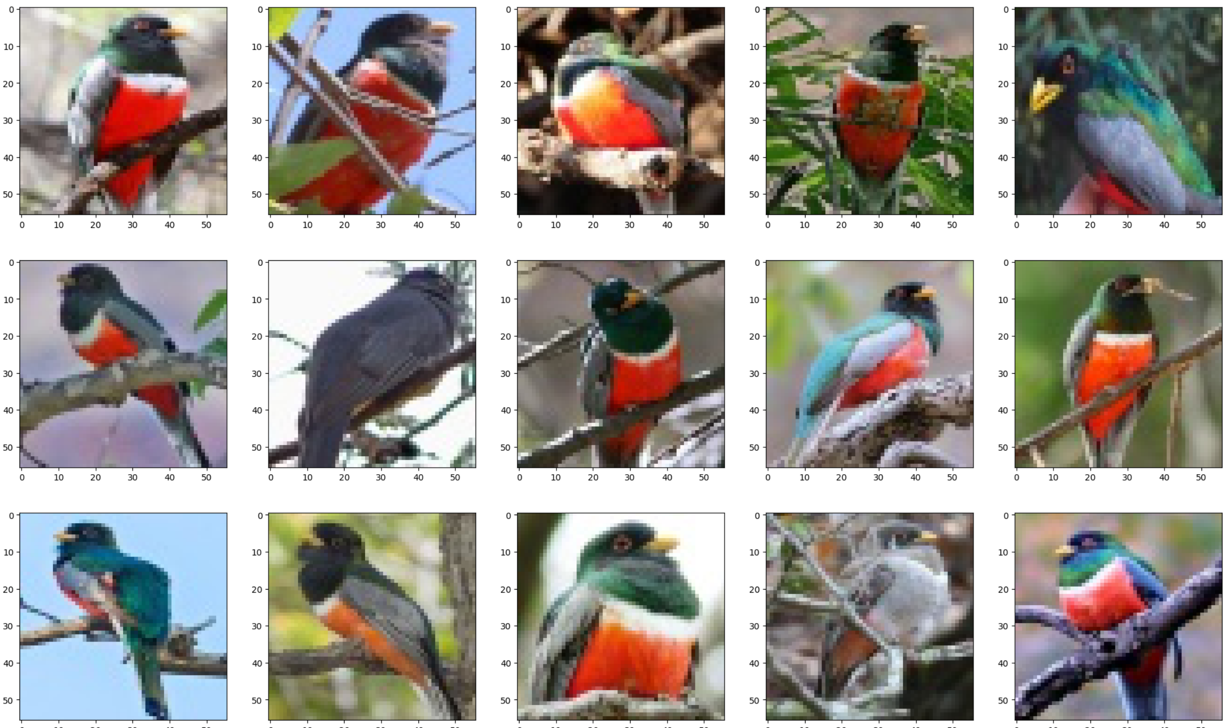
Plot 15 pictures from one of the previously selected random species

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

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

```
ELEGANT TROGON
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
```

ELEGANT TROGON



```
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 133 images, and a maxi
mum of 208 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)
validData, validClasses = selectData(validPath)
#train, test and validation data lists into numpy arrays
npTrainData = np.array(trainData)
npTestData = np.array(testData)
npValidData = np.array(validData)

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
print('Our validation data array is equivalent in number to the test data
      npValidData.ndim, 'dimensions, and a shape of', npValidData.shape,
      'for a total number of elements of', npValidData.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) f
or a total number of elements of 24225600 .
Our validation data array is equivalent in number to the test data array
. It has 4 dimensions, and a shape of (2575, 56, 56, 3) for a total numb
er 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, test and validation 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)
# validation labels
validClasses = classesToInt(validClasses)
npValidClasses = np.array(validClasses)
npValidClasses = np.expand_dims(npValidClasses, axis = 1) # add dimension
npValidLabels = to_categorical(npValidClasses)

trainCount = countLabels(npTrainClasses) # count train labels
testCount = countLabels(npTestClasses)   # count test labels
validCount = countLabels(npValidClasses) # count validation labels
if trainCount == testCount and trainCount == validCount:
    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, test labels count and validation la

# free up memory
del trainData, testData, validData, trainClasses, npTrainClasses, testCla
```

```
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)
npValidData = np.array(npValidData / npValidData.max(), dtype = np.float1
```

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
batchSize = 64 # batch size to be used in model.fit
testDatagen = ImageDataGenerator(rescale = 1. / 255)
nTestSamples = npTestData.shape[0] # number of images in test folder

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

myModel, Y_pred = seqModel(npTrainData, npTrainLabels, npTestData, npTest
                           npValidData, npValidLabels, batchSize, validGe
```

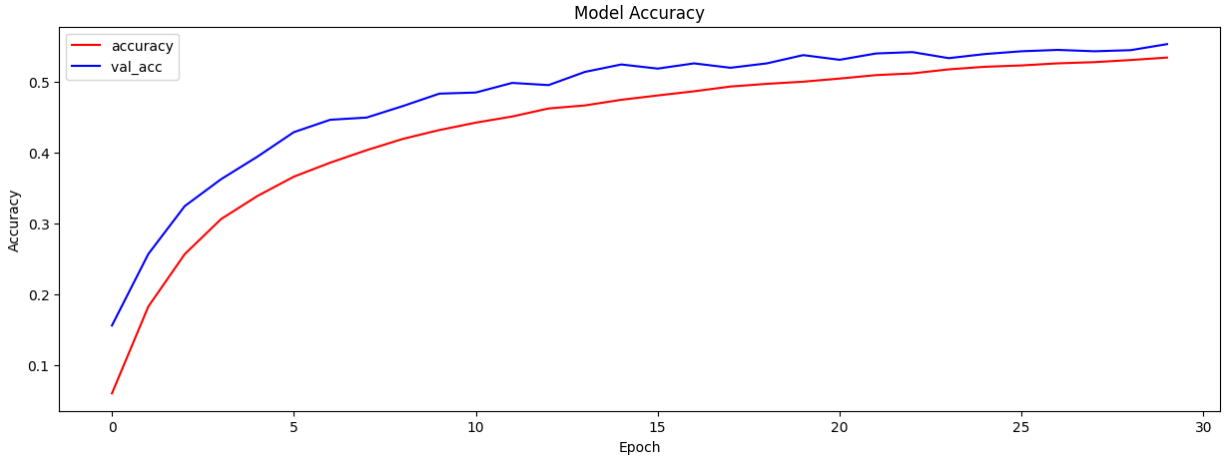
```
Found 2575 images belonging to 515 classes.
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 54, 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
1293/1293 [==============================] - 19s 7ms/step - loss: 5.2275
- accuracy: 0.0599 - val_loss: 4.2167 - val_accuracy: 0.1557
```

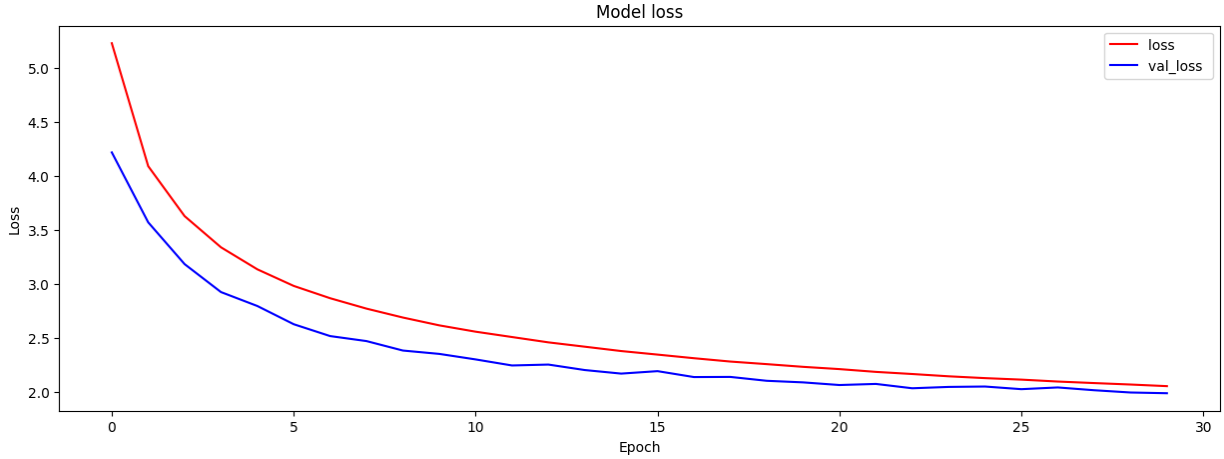
```
Epoch 2/30
1293/1293 [==============================] - 8s 6ms/step - loss: 4.0892
- accuracy: 0.1825 - val_loss: 3.5691 - val_accuracy: 0.2567
Epoch 3/30
1293/1293 [==============================] - 7s 5ms/step - loss: 3.6268
- accuracy: 0.2565 - val_loss: 3.1824 - val_accuracy: 0.3243
Epoch 4/30
1293/1293 [==============================] - 8s 6ms/step - loss: 3.3380
- accuracy: 0.3061 - val_loss: 2.9234 - val_accuracy: 0.3623
Epoch 5/30
1293/1293 [==============================] - 7s 5ms/step - loss: 3.1340
- accuracy: 0.3386 - val_loss: 2.7944 - val_accuracy: 0.3942
Epoch 6/30
1293/1293 [==============================] - 8s 6ms/step - loss: 2.9803
- accuracy: 0.3659 - val_loss: 2.6256 - val_accuracy: 0.4287
Epoch 7/30
1293/1293 [==============================] - 7s 5ms/step - loss: 2.8665
- accuracy: 0.3857 - val_loss: 2.5159 - val_accuracy: 0.4462
Epoch 8/30
1293/1293 [==============================] - 8s 6ms/step - loss: 2.7701
- accuracy: 0.4033 - val_loss: 2.4695 - val_accuracy: 0.4493
Epoch 9/30
1293/1293 [==============================] - 7s 5ms/step - loss: 2.6876
- accuracy: 0.4193 - val_loss: 2.3820 - val_accuracy: 0.4656
Epoch 10/30
1293/1293 [==============================] - 8s 6ms/step - loss: 2.6151
- accuracy: 0.4317 - val_loss: 2.3509 - val_accuracy: 0.4831
Epoch 11/30
1293/1293 [==============================] - 7s 6ms/step - loss: 2.5565
- accuracy: 0.4421 - val_loss: 2.2997 - val_accuracy: 0.4847
Epoch 12/30
1293/1293 [==============================] - 8s 6ms/step - loss: 2.5071
- accuracy: 0.4508 - val_loss: 2.2438 - val_accuracy: 0.4983
Epoch 13/30
1293/1293 [==============================] - 7s 6ms/step - loss: 2.4577
- accuracy: 0.4622 - val_loss: 2.2521 - val_accuracy: 0.4951
Epoch 14/30
1293/1293 [==============================] - 8s 6ms/step - loss: 2.4176
- accuracy: 0.4665 - val_loss: 2.2014 - val_accuracy: 0.5138
Epoch 15/30
1293/1293 [==============================] - 8s 6ms/step - loss: 2.3773
- accuracy: 0.4744 - val_loss: 2.1685 - val_accuracy: 0.5243
Epoch 16/30
1293/1293 [==============================] - 7s 5ms/step - loss: 2.3444
- accuracy: 0.4805 - val_loss: 2.1912 - val_accuracy: 0.5184
Epoch 17/30
1293/1293 [==============================] - 8s 6ms/step - loss: 2.3110
- accuracy: 0.4864 - val_loss: 2.1364 - val_accuracy: 0.5258
Epoch 18/30
1293/1293 [==============================] - 7s 5ms/step - loss: 2.2800
- accuracy: 0.4931 - val_loss: 2.1378 - val_accuracy: 0.5196
Epoch 19/30
1293/1293 [==============================] - 8s 6ms/step - loss: 2.2562
```

```
- accuracy: 0.4969 - val_loss: 2.1019 - val_accuracy: 0.5258
Epoch 20/30
1293/1293 [==============================] - 7s 5ms/step - loss: 2.2311
- accuracy: 0.5000 - val_loss: 2.0872 - val_accuracy: 0.5375
Epoch 21/30
1293/1293 [==============================] - 8s 6ms/step - loss: 2.2101
- accuracy: 0.5044 - val_loss: 2.0627 - val_accuracy: 0.5309
Epoch 22/30
1293/1293 [==============================] - 7s 5ms/step - loss: 2.1844
- accuracy: 0.5093 - val_loss: 2.0729 - val_accuracy: 0.5398
Epoch 23/30
1293/1293 [==============================] - 8s 6ms/step - loss: 2.1647
- accuracy: 0.5117 - val_loss: 2.0327 - val_accuracy: 0.5417
Epoch 24/30
1293/1293 [==============================] - 7s 6ms/step - loss: 2.1435
- accuracy: 0.5174 - val_loss: 2.0452 - val_accuracy: 0.5332
Epoch 25/30
1293/1293 [==============================] - 8s 6ms/step - loss: 2.1266
- accuracy: 0.5210 - val_loss: 2.0488 - val_accuracy: 0.5390
Epoch 26/30
1293/1293 [==============================] - 7s 6ms/step - loss: 2.1127
- accuracy: 0.5229 - val_loss: 2.0243 - val_accuracy: 0.5429
Epoch 27/30
1293/1293 [==============================] - 7s 6ms/step - loss: 2.0950
- accuracy: 0.5259 - val_loss: 2.0403 - val_accuracy: 0.5449
Epoch 28/30
1293/1293 [==============================] - 8s 6ms/step - loss: 2.0808
- accuracy: 0.5275 - val_loss: 2.0144 - val_accuracy: 0.5429
Epoch 29/30
1293/1293 [==============================] - 7s 6ms/step - loss: 2.0674
- accuracy: 0.5306 - val_loss: 1.9935 - val_accuracy: 0.5445
Epoch 30/30
1293/1293 [==============================] - 8s 6ms/step - loss: 2.0523
- accuracy: 0.5340 - val_loss: 1.9868 - val_accuracy: 0.5530

Test loss: 1.8440319299697876
Test accuracy: 0.568932056427002
41/41 [==============================] - 2s 58ms/step
```

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

In [10]:
```
### 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

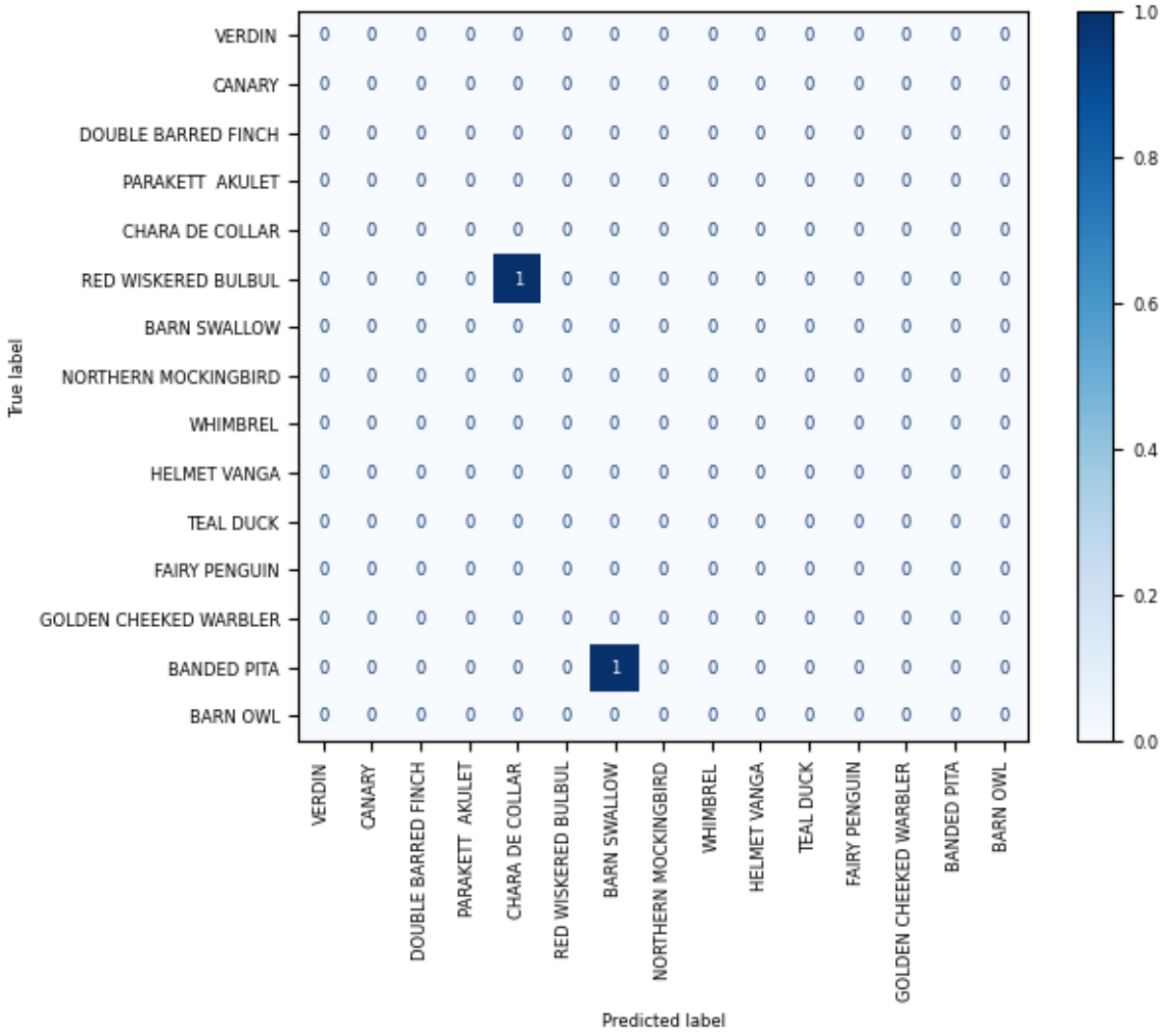
I finally plot the confusion matrix and the classification report for a small sample of bird species.

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


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 [ ]: