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 reshaped dataset from Google ###
###  Drive to colab's content folder   ###
##########################################
!unzip '/content/drive/MyDrive/Colab Notebooks/da_dcAI_project/reshaped_dataset.zip

##########################################
### import necessary or useful modules ###
##########################################
import os
import numpy as np
#import tensorflow as tf
from    matplotlib                import pyplot as plt
```

```
from    tensorflow.keras.utils    import to_categorical
from    keras.preprocessing.image import ImageDataGenerator
from    sklearn.metrics           import classification_report, confusion_matrix
from    a_selectRandomFolders     import selectRandomFolders
from    b_viewClasses             import viewRandomClasses, viewRandomClass
from    c_selectData              import selectData, classesToInt, countLabels
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 names
trainPathWin = 'C:/Users/mzlarosa/OneDrive - unime.it/Learning/CdL Informatica/Annc
testPathWin = 'C:/Users/mzlarosa/OneDrive - unime.it/Learning/CdL Informatica/Anno
validPathWin = 'C:/Users/mzlarosa/OneDrive - unime.it/Learning/CdL Informatica/Annc
trainPathMac = '/Users/mau/OneDrive - unime.it/Learning/CdL Informatica/Anno II - D
testPathMac = '/Users/mau/OneDrive - unime.it/Learning/CdL Informatica/Anno II - De
validPathMac = '/Users/mau/OneDrive - unime.it/Learning/CdL Informatica/Anno II - D
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:
['AZURE BREASTED PITTA', 'HOUSE FINCH', 'WATTLED CURASSOW', 'GREATOR SAGE GROUSE',
'AVADAVAT', 'DUSKY ROBIN', 'STRIPPED SWALLOW', 'GRAY PARTRIDGE', 'YELLOW BELLIED F
LOWERPECKER', 'BELTED KINGFISHER', 'ENGGANO MYNA', 'HYACINTH MACAW', 'BOBOLINK',
'CALIFORNIA QUAIL', 'SNOWY EGRET', 'CUBAN TROGON', 'SHORT BILLED DOWITCHER', 'PUFF
IN', 'ROYAL FLYCATCHER', 'HORNED GUAN', 'MASKED BOOBY', 'EARED PITA', 'EASTERN MEA
DOWLARK', 'FLAME TANAGER', 'RED HEADED WOODPECKER', 'ABYSSINIAN GROUND HORNBILL',
'STRIATED CARACARA', 'INDIGO FLYCATCHER', 'BLUE HERON', 'D-ARNAUDS BARBET', 'GLOSS
Y IBIS', 'ASIAN OPENBILL STORK', 'PEREGRINE FALCON', 'JOCOTOCO ANTPITTA', 'GOLDEN
PIPIT', 'VIOLET CUCKOO', 'ROSE BREASTED COCKATOO', 'AMERICAN COOT', 'MALEO', 'PHAI
NOPEPLA', 'GREAT ARGUS', 'RED WINGED BLACKBIRD', 'ALPINE CHOUGH', 'TRICOLORED BLAC
KBIRD', 'HORNED SUNGEM', 'TURKEY VULTURE', 'CHINESE POND HERON', 'MASKED BOBWHIT
E', 'MALAGASY WHITE EYE', 'PATAGONIAN SIERRA FINCH'][...]
```

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 139 image files in /content/reshaped/train/NORTHERN FLICKE
R
There are 0 folders and 160 image files in /content/reshaped/train/BORNEAN LEAFBIR
D
There are 0 folders and 173 image files in /content/reshaped/train/HORNED LARK
There are 0 folders and 139 image files in /content/reshaped/train/TRICOLORED BLAC
KBIRD
There are 0 folders and 165 image files in /content/reshaped/train/TOWNSENDS WARBL
ER
There are 0 folders and 175 image files in /content/reshaped/train/HARPY EAGLE
There are 0 folders and 168 image files in /content/reshaped/train/BLACK-THROATED
SPARROW
There are 0 folders and 175 image files in /content/reshaped/train/KILLDEAR
There are 0 folders and 136 image files in /content/reshaped/train/BARRED PUFFBIRD
There are 0 folders and 187 image files in /content/reshaped/train/AUCKLAND SHAQ
There are 0 folders and 193 image files in /content/reshaped/train/VARIED THRUSH
There are 0 folders and 135 image files in /content/reshaped/train/GOLDEN CHLOROPH
ONIA
There are 0 folders and 156 image files in /content/reshaped/train/GREEN JAY
There are 0 folders and 134 image files in /content/reshaped/train/ANDEAN GOOSE
There are 0 folders and 159 image files in /content/reshaped/train/FRIGATE
```

Plot pictures from sample species

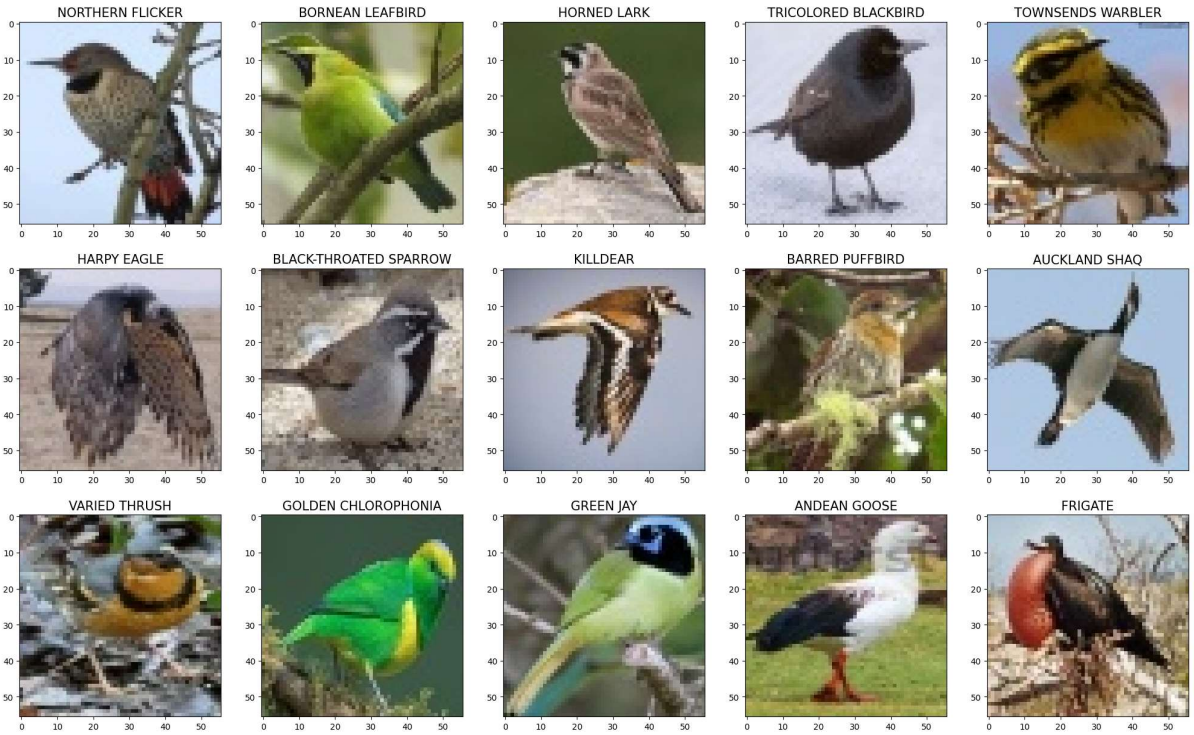
Plot one picture from 15 randomly selected species

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

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

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

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

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

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

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

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

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

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

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

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

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

Sample of 15 bird classes

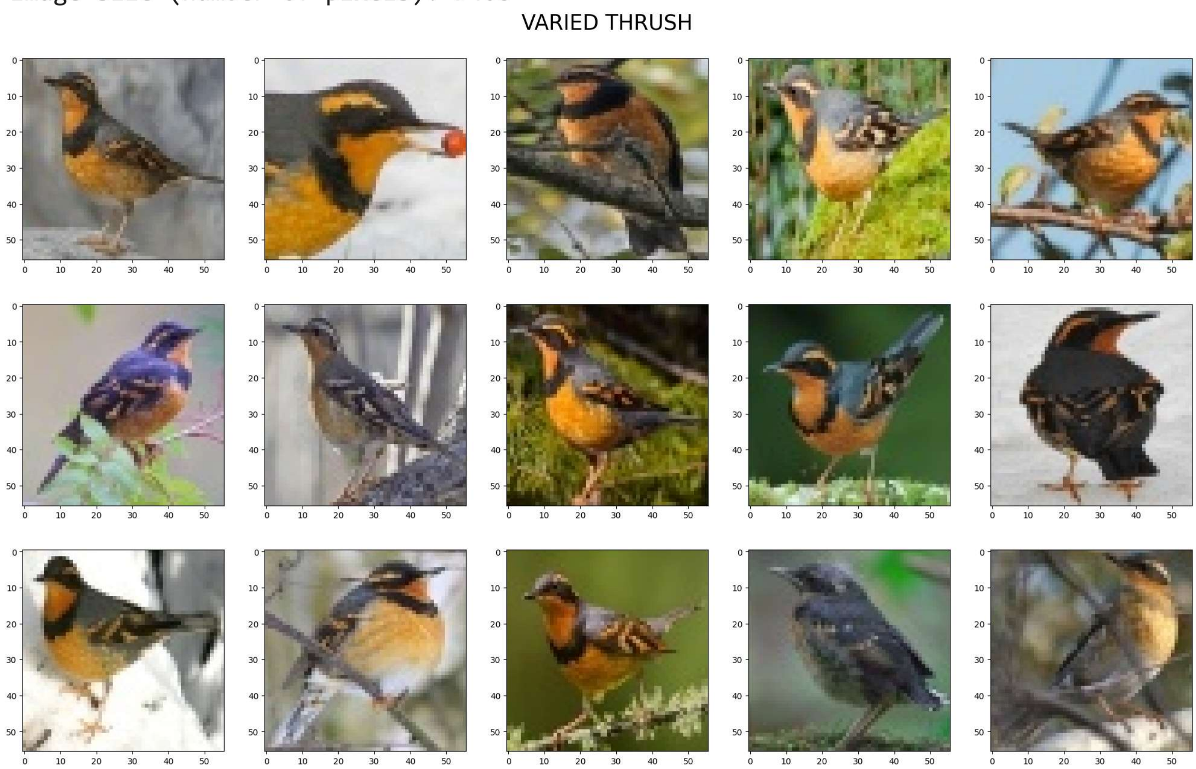


Plot 15 pictures from one of the previously selected random species

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

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

```
VARIED THRUSH
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
```




```
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 193
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, test and validation data and labels into memory
#trainData, trainClasses, testData, testClasses, validData, validClasses, cfClasses
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 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, '.')
print('Our validation data array is equivalent in number to the test data array. It
      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 tot
al 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 .
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 number of elements of 242
25600 .
```

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 to array
npTrainLabels = to_categorical(npTrainClasses)
# test labels
testClasses = classesToInt(testClasses)
npTestClasses = np.array(testClasses)
npTestClasses = np.expand_dims(npTestClasses, axis = 1) # add dimension to array
npTestLabels = to_categorical(npTestClasses)
# validation labels
validClasses = classesToInt(validClasses)
npValidClasses = np.array(validClasses)
npValidClasses = np.expand_dims(npValidClasses, axis = 1) # add dimension to array
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 each image',
          'belongs to a category) for a set of', trainCount, 'categories.')
else:
    print('ERROR: train labels count, test labels count and validation labels count
```

```
# free up memory
del trainData, testData, validData, trainClasses, testClasses, validClasses
```

```
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 [8]:
```
### 7) normalize the data
npTrainData = np.array(npTrainData / npTrainData.max(), dtype = np.float16)
npTestData = np.array(npTestData / npTestData.max(), dtype = np.float16)
npValidData = np.array(npValidData / npValidData.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
myModel, Y_pred = seqModel(npTrainData, npTrainLabels, npTestData, npTestLabels, np
```

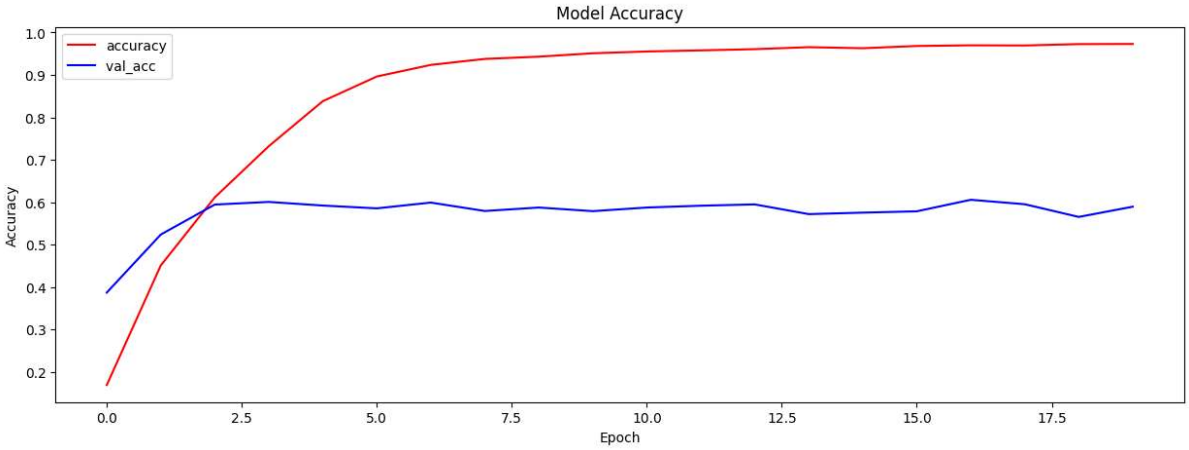
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 54, 54, 32)        896

 ReLU (Activation)           (None, 54, 54, 32)        0

 max_pooling2d (MaxPooling2  (None, 27, 27, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 25, 25, 64)        18496

 ReLU2 (Activation)          (None, 25, 25, 64)        0

 max_pooling2d_1 (MaxPoolin  (None, 12, 12, 64)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 10, 10, 128)       73856

 ReLU3 (Activation)          (None, 10, 10, 128)       0

 max_pooling2d_2 (MaxPoolin  (None, 5, 5, 128)         0
 g2D)

 flatten (Flatten)           (None, 3200)              0

 FC (Dense)                  (None, 3200)              10243200

 ReLU4 (Activation)          (None, 3200)              0

 FC2 (Dense)                 (None, 515)               1648515

 Softmax (Activation)        (None, 515)               0

=================================================================
Total params: 11984963 (45.72 MB)
Trainable params: 11984963 (45.72 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/20
1293/1293 [==============================] - 28s 13ms/step - loss: 4.3523 - accura
cy: 0.1692 - val_loss: 2.7803 - val_accuracy: 0.3872
Epoch 2/20
1293/1293 [==============================] - 16s 13ms/step - loss: 2.4934 - accura
cy: 0.4511 - val_loss: 2.1040 - val_accuracy: 0.5239
Epoch 3/20
1293/1293 [==============================] - 16s 13ms/step - loss: 1.6554 - accura
cy: 0.6115 - val_loss: 1.8278 - val_accuracy: 0.5946
Epoch 4/20
1293/1293 [==============================] - 16s 13ms/step - loss: 1.0511 - accura
cy: 0.7321 - val_loss: 1.8856 - val_accuracy: 0.6008
Epoch 5/20
1293/1293 [==============================] - 16s 12ms/step - loss: 0.5900 - accura
cy: 0.8382 - val_loss: 2.2669 - val_accuracy: 0.5922
Epoch 6/20
1293/1293 [==============================] - 16s 13ms/step - loss: 0.3595 - accura
cy: 0.8963 - val_loss: 2.7296 - val_accuracy: 0.5856
Epoch 7/20
1293/1293 [==============================] - 16s 13ms/step - loss: 0.2562 - accura
cy: 0.9237 - val_loss: 2.8615 - val_accuracy: 0.5992
Epoch 8/20
1293/1293 [==============================] - 16s 12ms/step - loss: 0.2093 - accura
cy: 0.9378 - val_loss: 3.2278 - val_accuracy: 0.5794
```

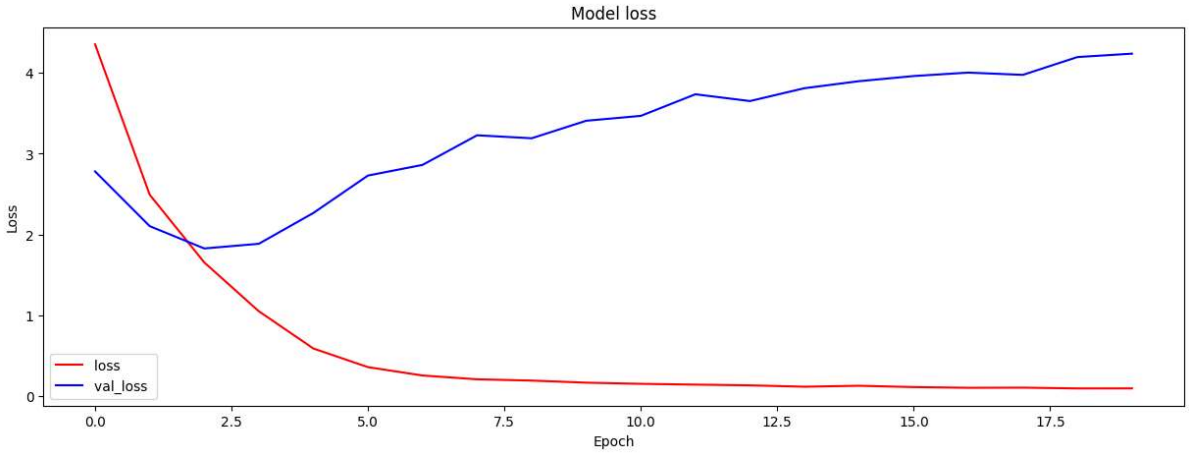
```
Epoch 9/20
1293/1293 [==============================] - 16s 13ms/step - loss: 0.1935 - accura
cy: 0.9431 - val_loss: 3.1909 - val_accuracy: 0.5876
Epoch 10/20
1293/1293 [==============================] - 16s 13ms/step - loss: 0.1676 - accura
cy: 0.9511 - val_loss: 3.4076 - val_accuracy: 0.5790
Epoch 11/20
1293/1293 [==============================] - 16s 12ms/step - loss: 0.1530 - accura
cy: 0.9553 - val_loss: 3.4679 - val_accuracy: 0.5876
Epoch 12/20
1293/1293 [==============================] - 16s 12ms/step - loss: 0.1435 - accura
cy: 0.9579 - val_loss: 3.7352 - val_accuracy: 0.5918
Epoch 13/20
1293/1293 [==============================] - 16s 12ms/step - loss: 0.1342 - accura
cy: 0.9608 - val_loss: 3.6513 - val_accuracy: 0.5950
Epoch 14/20
1293/1293 [==============================] - 16s 12ms/step - loss: 0.1175 - accura
cy: 0.9655 - val_loss: 3.8104 - val_accuracy: 0.5720
Epoch 15/20
1293/1293 [==============================] - 16s 12ms/step - loss: 0.1291 - accura
cy: 0.9631 - val_loss: 3.8973 - val_accuracy: 0.5755
Epoch 16/20
1293/1293 [==============================] - 16s 13ms/step - loss: 0.1131 - accura
cy: 0.9683 - val_loss: 3.9607 - val_accuracy: 0.5786
Epoch 17/20
1293/1293 [==============================] - 16s 12ms/step - loss: 0.1041 - accura
cy: 0.9699 - val_loss: 4.0032 - val_accuracy: 0.6058
Epoch 18/20
1293/1293 [==============================] - 16s 13ms/step - loss: 0.1059 - accura
cy: 0.9695 - val_loss: 3.9751 - val_accuracy: 0.5953
Epoch 19/20
1293/1293 [==============================] - 16s 12ms/step - loss: 0.0963 - accura
cy: 0.9727 - val_loss: 4.1953 - val_accuracy: 0.5654
Epoch 20/20
1293/1293 [==============================] - 17s 13ms/step - loss: 0.0967 - accura
cy: 0.9731 - val_loss: 4.2375 - val_accuracy: 0.5895

Test loss: 3.6642794609069824
Test accuracy: 0.6229126453399658
81/81 [==============================] - 0s 2ms/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()
```



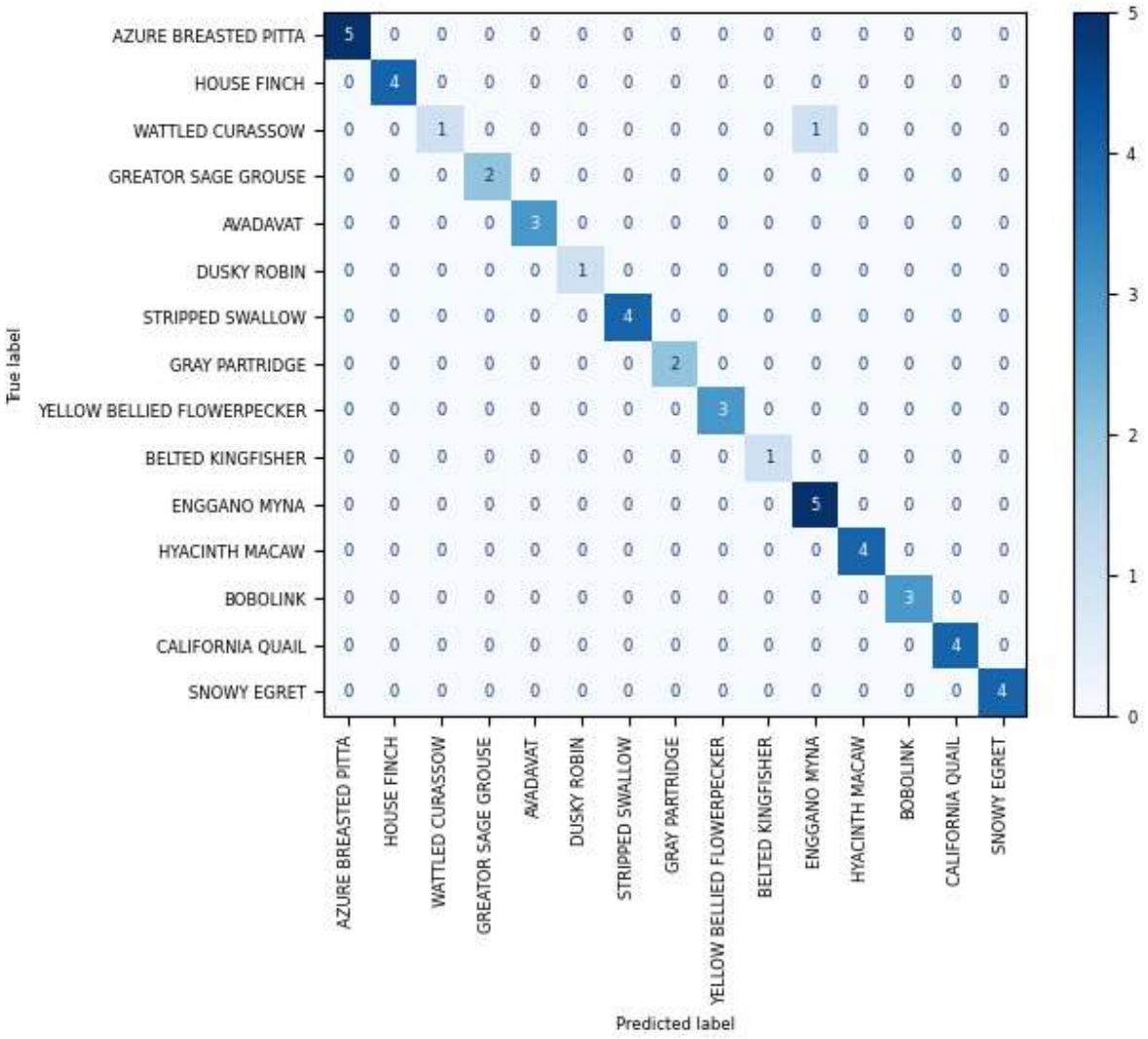


Plot confusion matrix and classification report

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(npTestClasses, 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], target_names[0 : 15])
print('\n\nClassification Report (small sample)\n')
class_report = classification_report(npTestClasses, y_pred, target_names = target_r
print(class_report[0 : 1142], (' ' * 18) + '[...]\n') # print a few lines of the cl
```

```
Confusion Matrix (small sample)

[[5 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 4 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 1 0 0 0 0 0 0 0 1 0 0 0 0]
 [0 0 0 2 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 3 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 1 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 4 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 2 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 3 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 1 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 5 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 4 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 3 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 4 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 4]]


Classification Report (small sample)

                               precision    recall  f1-score   support

         AZURE BREASTED PITTA       1.00      1.00      1.00         5
                  HOUSE FINCH       0.80      0.80      0.80         5
             WATTLED CURASSOW       0.17      0.20      0.18         5
           GREATOR SAGE GROUSE       0.50      0.40      0.44         5
                     AVADAVAT       0.50      0.60      0.55         5
                  DUSKY ROBIN       0.33      0.20      0.25         5
             STRIPPED SWALLOW       0.50      0.80      0.62         5
               GRAY PARTRIDGE       0.33      0.40      0.36         5
  YELLOW BELLIED FLOWERPECKER       1.00      0.60      0.75         5
            BELTED KINGFISHER       0.33      0.20      0.25         5
                 ENGGANO MYNA       0.45      1.00      0.62         5
               HYACINTH MACAW       1.00      0.80      0.89         5
                     BOBOLINK       0.43      0.60      0.50         5
             CALIFORNIA QUAIL       0.50      0.80      0.62         5
                  SNOWY EGRET       0.80      0.80      0.80         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>
```


AZURE BREASTED PITTA
HOUSE FINCH
WATTLED CURASSOW
GREATOR SAGE GROUSE
AVADAVAT
DUSKY ROBIN
STRIPPED SWALLOW
GRAY PARTRIDGE
YELLOW BELLIED FLOWERPECKER
BELTED KINGFISHER
ENGGANO MYNA
HYACINTH MACAW
BOBOLINK
CALIFORNIA QUAIL
SNOWY EGRET
True label
Predicted label

In [ ]: