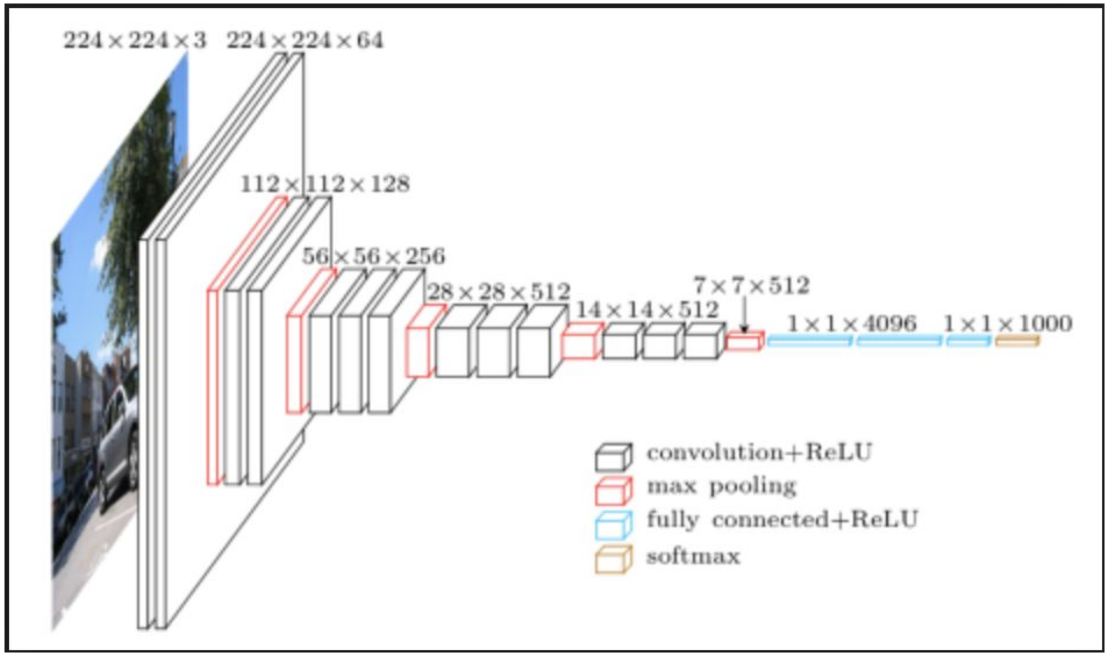
Image Classification based on Convolutional Neural Network

Presented by:

Veronica Naosekpam
PhD Scholar
Computer Science & Engineering
Indian Institute of Information Technology Guwahati, Assam, India

Introduction

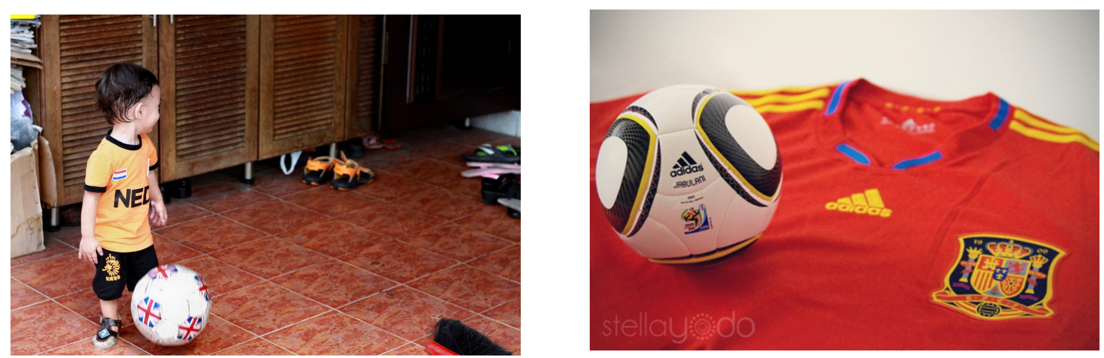
- Assigning an input image, one label from a fixed set of categories.
- Classify different types of sports (here, Soccer and Rugby) using convolution neural networks in two ways :
    - Training from scratch.
    - Transfer learning.
- Use Visual Geometry Group- VGG16- model.
    - A pre-trained model on ImageNet



Setting up custom Image Dataset

- Two popular sources : **ImageNet** and **Google OpenImages** via python scripts.
- Input : 3058
    - Train : 2448
        - Rugby : 1224
        - Soccer : 1224
    - Test: 610
        - Rugby : 305
        - Soccer : 305

Sample images from Rugby class



Sample images from Soccer class

- Import the required libraries.
- Matplotlib and Seaborn for visualizing our dataset
- Image data : OpenCV.

[1]:
```python
import matplotlib.pyplot as plt
import seaborn as sns
import keras
from keras.models import Sequential
from keras.layers import Dense, Conv2D , MaxPool2D , Flatten , Dropout
from keras.preprocessing.image import ImageDataGenerator
from keras.optimizers import Adam
from sklearn.metrics import classification_report,confusion_matrix
import tensorflow as tf
import cv2
import os
import numpy as np

os.environ["CUDA_DEVICE_ORDER"]="PCI_BUS_ID"
os.environ["CUDA_VISIBLE_DEVICES"]="1"
```

```
Using TensorFlow backend.
```

- Loading the data

```python
labels = ['rugby', 'soccer']
img_size = 224
def get_data(data_dir):
    data = []
    for label in labels:
        path = os.path.join(data_dir, label)
        class_num = labels.index(label)
        for img in os.listdir(path):
            try:
                img_arr = cv2.imread(os.path.join(path, img))[...,::-1] #convert BGR to RGB format
                resized_arr = cv2.resize(img_arr, (img_size, img_size)) # Reshaping images to preferred size
                data.append([resized_arr, class_num])
            except Exception as e:
                print(e)
    return np.array(data)
```

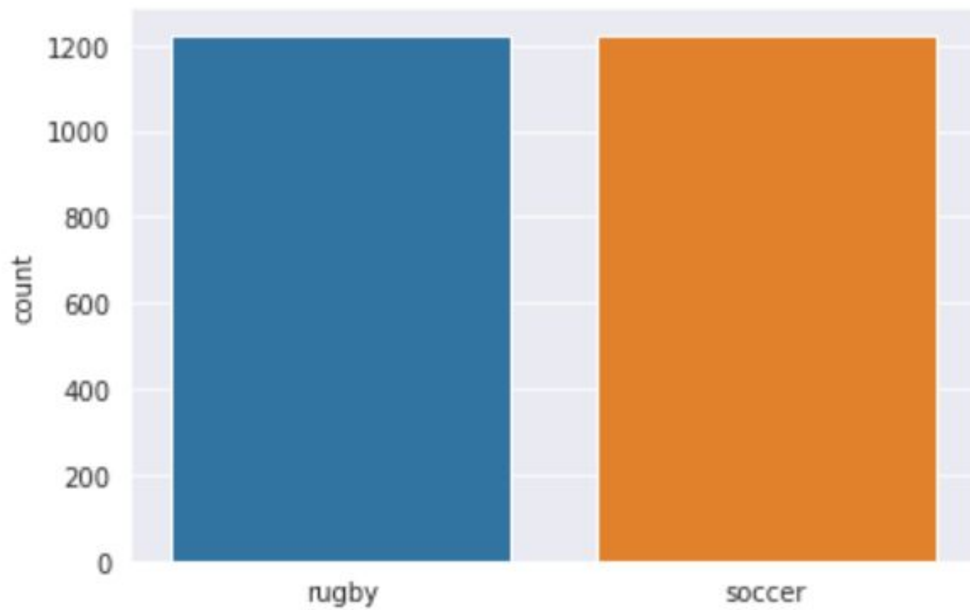
```python

train = get_data('/home/rs/veronica.naosekpam/Dataset/train')
val = get_data('/home/rs/veronica.naosekpam/Dataset/test')

```

```
l = []
for i in train:
    if(i[1] == 0):
        l.append("rugby")
    else:
        l.append("soccer")
sns.set_style('darkgrid')
sns.countplot(l)
```

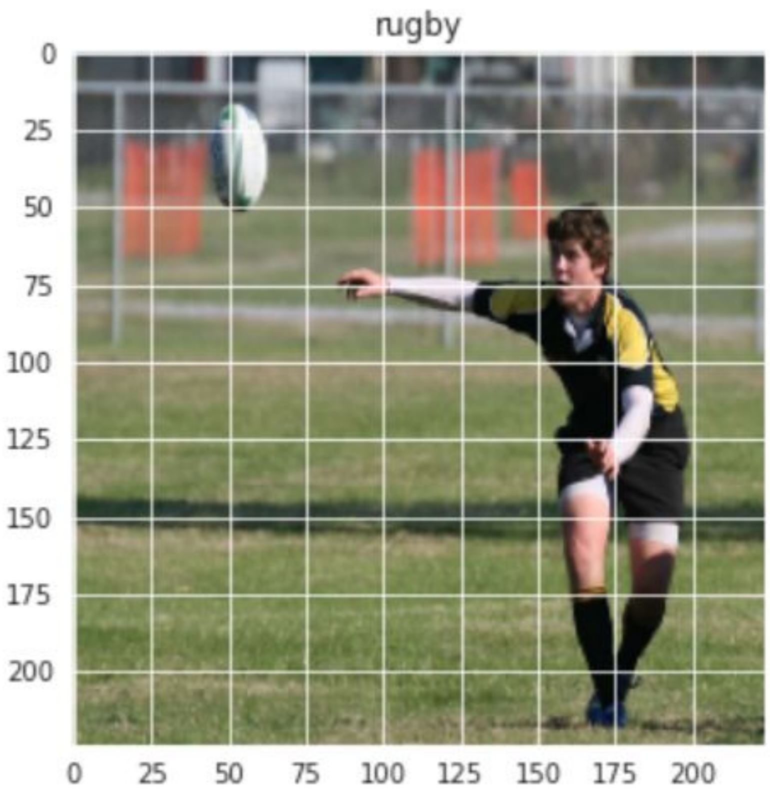
```
5]: <AxesSubplot:ylabel='count'>
```



- Visualizing random images

```
plt.figure(figsize = (5,5))
plt.imshow(train[1][0])
plt.title(labels[train[0][1]])
```

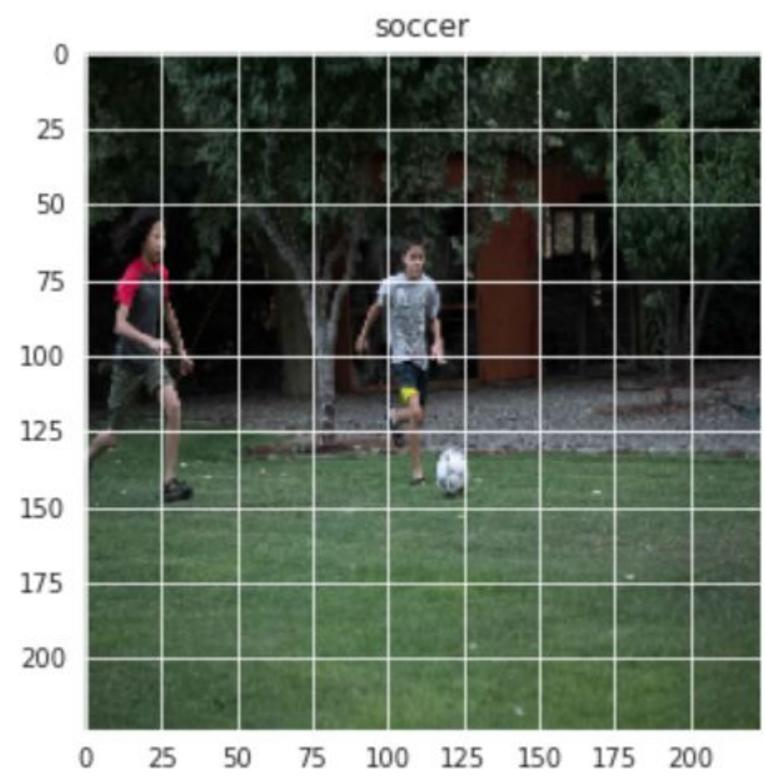
```
Text(0.5, 1.0, 'rugby')
```




```

plt.figure(figsize = (5,5))
plt.imshow(train[-4][0])
plt.title(labels[train[-1][1]])
```

```
Text(0.5, 1.0, 'soccer')
```



- Data Preprocessing and Data Augmentation.

```
x_train = []
y_train = []
x_val = []
y_val = []

for feature, label in train:
  x_train.append(feature)
  y_train.append(label)

for feature, label in val:
  x_val.append(feature)
  y_val.append(label)

# Normalize the data
x_train = np.array(x_train) / 255
x_val = np.array(x_val) / 255

x_train.reshape(-1, img_size, img_size, 1)
y_train = np.array(y_train)

x_val.reshape(-1, img_size, img_size, 1)
y_val = np.array(y_val)
```

```python
datagen = ImageDataGenerator(
        featurewise_center=False,  # set input mean to 0 over the dataset
        samplewise_center=False,  # set each sample mean to 0
        featurewise_std_normalization=False,  # divide inputs by std of the dataset
        samplewise_std_normalization=False,  # divide each input by its std
        zca_whitening=False,  # apply ZCA whitening
        rotation_range = 30,  # randomly rotate images in the range (degrees, 0 to 180)
        zoom_range = 0.2, # Randomly zoom image
        width_shift_range=0.1,  # randomly shift images horizontally (fraction of total width)
        height_shift_range=0.1,  # randomly shift images vertically (fraction of total height)
        horizontal_flip = True,  # randomly flip images
        vertical_flip=False)  # randomly flip images


datagen.fit(x_train)
```

- Define the Model

```
model = Sequential()
model.add(Conv2D(32,3,padding="same", activation="relu", input_shape=(224,224,3)))
model.add(MaxPool2D())

model.add(Conv2D(32, 3, padding="same", activation="relu"))
model.add(MaxPool2D())

model.add(Conv2D(64, 3, padding="same", activation="relu"))
model.add(MaxPool2D())
model.add(Dropout(0.4))

model.add(Flatten())
model.add(Dense(128,activation="relu"))
model.add(Dense(2, activation="softmax"))

model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 224, 224, 32)      896
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 112, 112, 32)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 112, 112, 32)      9248
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 56, 56, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 56, 56, 64)        18496
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 28, 28, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 28, 28, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 50176)             0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               6422656
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 258
=================================================================
Total params: 6,451,554
Trainable params: 6,451,554
Non-trainable params: 0
```

- Model compilation and training

```
opt = Adam(lr=0.000001)
model.compile(optimizer = opt , loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True) ,
              metrics = ['accuracy'])
```

```
history = model.fit(x_train,y_train,epochs = 500 , validation_data = (x_val, y_val))
```

```
ry libcublas.so.10
2022-11-05 15:26:33.581615: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic libra
ry libcudnn.so.7

2448/2448 [==============================] - 11s 4ms/step - loss: 0.6933 - accuracy: 0.3877 - val_loss: 0.6936 - val_accurac
y: 0.2547
Epoch 2/500
```

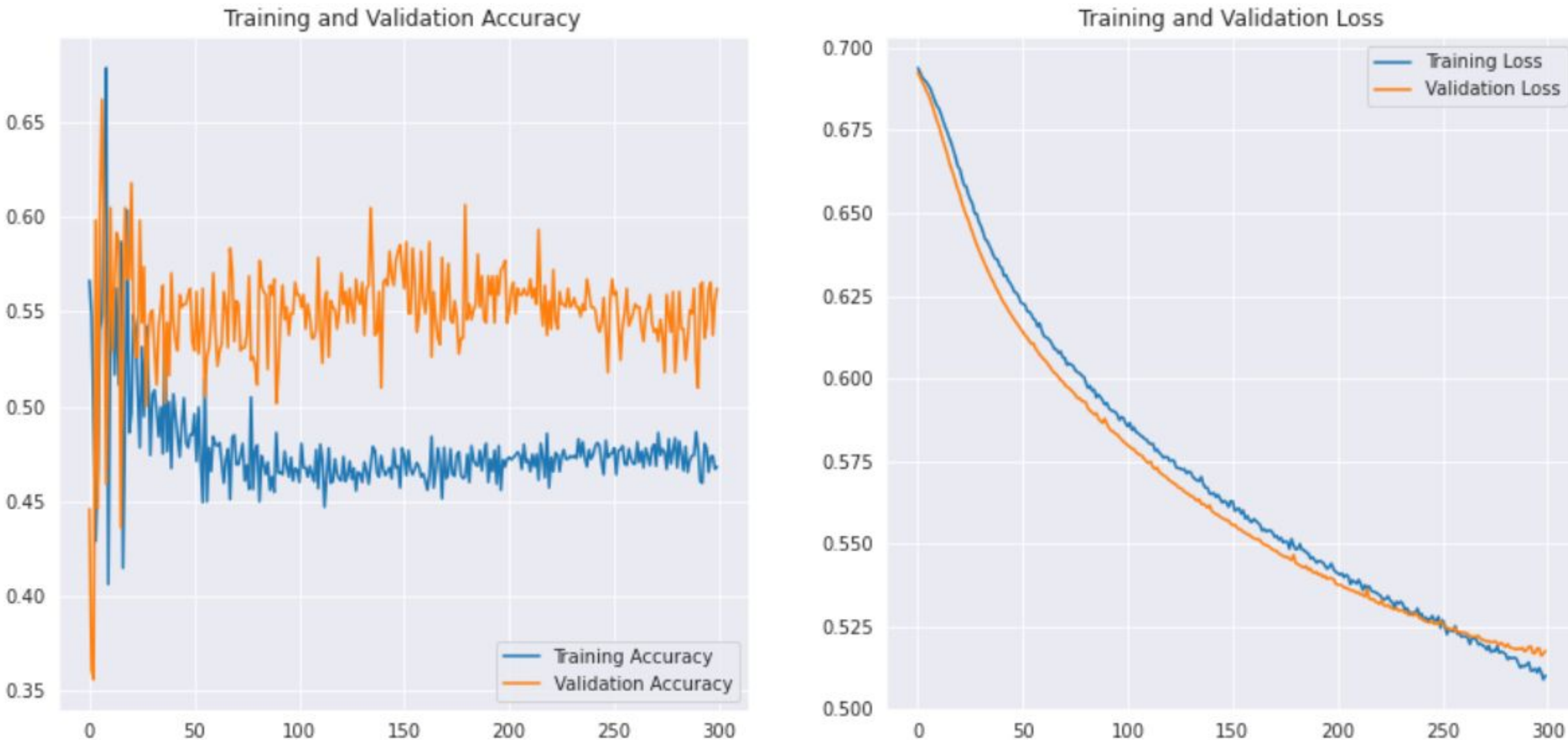
- Model evaluation after training.

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs_range = range(500)

plt.figure(figsize=(15, 15))
plt.subplot(2, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(2, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```

- Plot our training and validation accuracy along with training and validation loss



- Classification report.

```
]:  predictions = model.predict_classes(x_val)
    predictions = predictions.reshape(1,-1)[0]

    print(classification_report(y_val, predictions, target_names = ['Rugby (Class 0)','Soccer (Class 1)']))
```

```
                  precision    recall  f1-score   support

 Rugby (Class 0)       0.86      0.84      0.85       305
Soccer (Class 1)       0.73      0.76      0.74       170

        accuracy                           0.81       475
       macro avg       0.80      0.80      0.80       475
    weighted avg       0.81      0.81      0.81       475
```

```
16]:  model.evaluate(x_val,y_val)
```

```
610/610 [==============================] - 1s 2ms/step
```

```
16]: [0.5176380370484024, 0.5622950792312622]
```

Transfer Learning

- Technique where a model trained on one task is re-purposed on a second related task.
- When the dataset is small, by using a pre-trained model helps to achieve high performance.
- Import the base model : VGG16
- Pre-trained on the ImageNet dataset, a large dataset consisting of 1.4M images and 1000 classes.
- This base of knowledge will help us classify Rugby and Soccer from our specific dataset.
- include_top=False argument : load a network that doesn’t include the classification layers at the top.

```
base_model = tf.keras.applications.VGG16(input_shape = (224, 224, 3), include_top = False, weights = "imagenet")

```

17]:

```
base_model = tf.keras.applications.VGG16(input_shape = (224, 224, 3), include_top = False, weights = "imagenet")
```

18]:

```
base_model.trainable = False
```

19]:

```
model = tf.keras.Sequential([base_model,
                                 tf.keras.layers.GlobalAveragePooling2D(),
                                 tf.keras.layers.Dropout(0.2),
                                 tf.keras.layers.Dense(2, activation="softmax")
                                ])
```

*]:

```
base_learning_rate = 0.00001
model.compile(optimizer=tf.keras.optimizers.Adam(lr=base_learning_rate),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

history = model.fit(x_train,y_train,epochs = 300 , validation_data = (x_val, y_val))
```

```
Train on 2448 samples, validate on 610 samples
Epoch 1/300
2448/2448 [==============================] - 9s 4ms/sample - loss: 0.7029 - accuracy: 0.4726 - val_loss: 0.7071 - val_accurac
y: 0.4148
Epoch 2/300
2448/2448 [==============================] - 8s 3ms/sample - loss: 0.7020 - accuracy: 0.4808 - val_loss: 0.7046 - val_accurac
y: 0.4180
Epoch 3/300
```

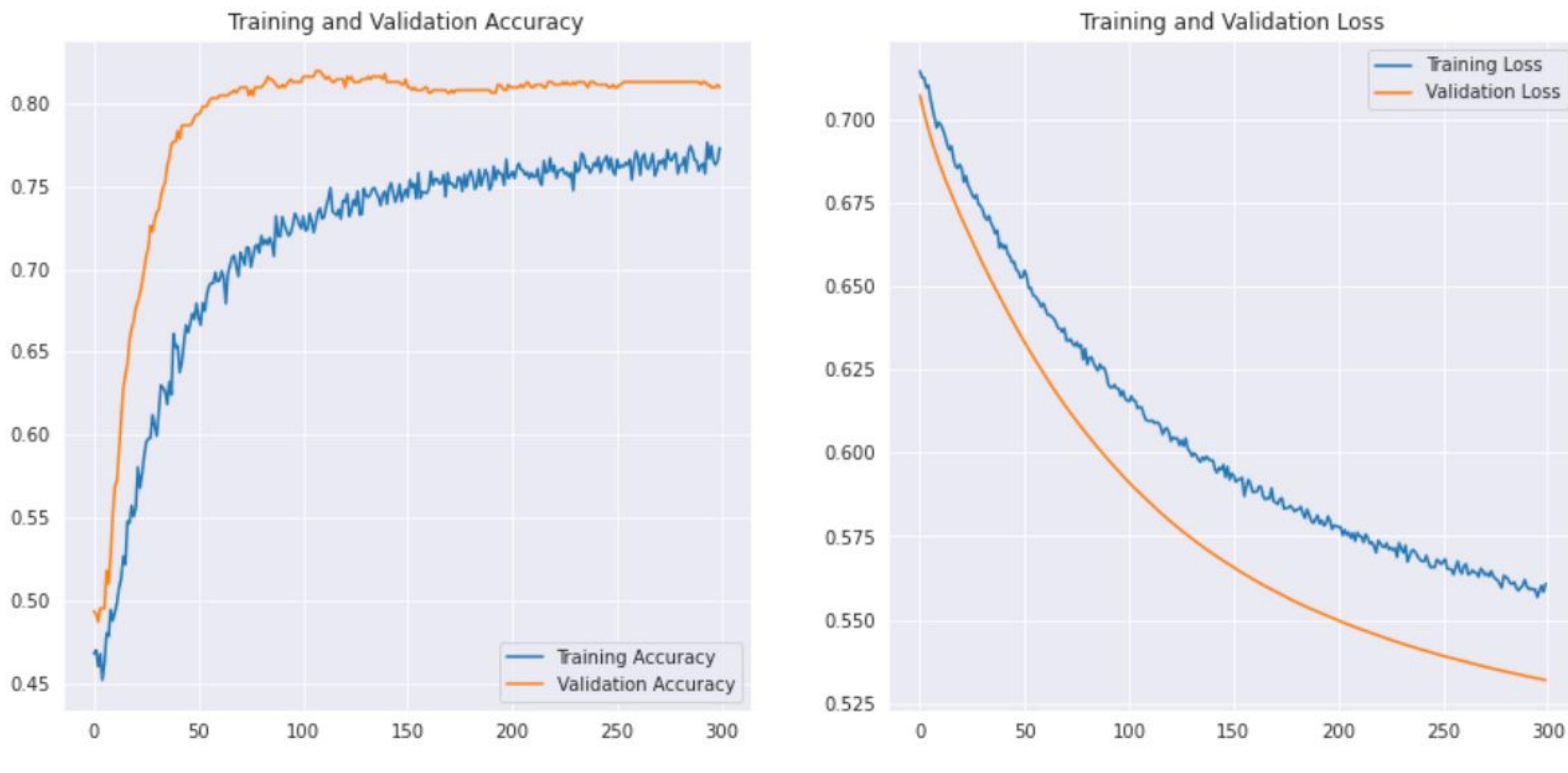
- Model evaluation after training.

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs_range = range(500)

plt.figure(figsize=(15, 15))
plt.subplot(2, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(2, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```

- Plot our training and validation accuracy along with training and validation loss



- Classification report.

```
predictions = model.predict_classes(x_val)
predictions = predictions.reshape(1,-1)[0]
print(classification_report(y_val, predictions, target_names = ['Rugby (Class 0)','Soccer (Class 1)']))
```

```
                  precision    recall  f1-score   support

Rugby (Class 0)        0.79      0.89      0.83       305
Soccer (Class 1)       0.87      0.76      0.81       305

       accuracy                            0.82       610
      macro avg        0.83      0.82      0.82       610
   weighted avg        0.83      0.82      0.82       610
```

```
[16]: model.evaluate(x_val,y_val)
```

```
610/610 [==============================] - 1s 2ms/sample - loss: 0.5319 - accuracy: 0.8098
```

```
[16]: [0.5318700773794143, 0.8098361]
```

Thank You