StarAgile Assignment - Machine Learning - Level 1 - Assignment - 03

Submitted by VINAY.M on 12-09-2023

Image Classification - Sports Images Classification

- Problem Statement :
  - Apply various models to Sports Images and go with the best model that helps in classifying which sport that image belongs to.
  - Dataset Link : https://drive.google.com/drive/folders/168kbX_a3qSn2Q786RzTpUpXU3sZ-7GcP?usp=share_link (https://drive.google.com/drive/folders/168kbX_a3qSn2Q786RzTpUpXU3sZ-7GcP?usp=share_link)
- What Kind Of Approach You Could Follow for your Problem Statements :
  - Reading the Data, Understanding the Data, Exploratory Data Analysis, Data Visualization, Splitting the Data, Training and Testing, Modeling, Accuracy
  - Hints provided are only for your references or getting started. You're free to use your own methodology to work on your assignments.

In [ ]:

```
# Given Data contains images from 22 different sports type.
# Since Laptop is not capable of creating models and Modelling will be performed with G
oogle Colab
# StarAgile Given data extracted and re-uploaded to personal google drive
# Mypath
# /content/drive/MyDrive/DS_Datasets/Sports_Classification/SportsData_StarAgile
```

In [ ]:

```
# Mounting Google Drive for Getting access to uploaded Sports Image
from google.colab import drive
drive.mount('/content/drive/')
```

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

Import / Installing Important Libraries

In [ ]:

```
# Uncomment below line and run, if not installed
#!pip install tensorflow tensorflow-gpu opencv-python matplotlib
```

In [ ]:

```
import tensorflow as tf
import sys
import os
```

In [ ]:

```
print ('Running in colab:', 'google.colab' in sys.modules)
```

```
Running in colab: True
```

Remove Dodgy Images

In [ ]:

```
import cv2
import imghdr
```

In [ ]:

```
data_path = '/content/drive/MyDrive/DS_Datasets/Sports_Classification/SportsData_StarAg
ile/'
```

In [ ]:

```
image_ext_required = ['jpeg','jpg','bmp','png']
```

In [ ]:

```
count = 0
for image_class in os.listdir(data_path):
  for image in os.listdir(os.path.join(data_path, image_class)):
    img_path = os.path.join(data_path, image_class, image)

    try:
      img = cv2.imread(img_path)
      tip = imghdr.what(img_path)

      if tip not in image_ext_required:
        print("Image is not of desired extention", format(img_path))
        os.remove(img_path)
    except Exception as e:
      #print("Issue with the image")
      count = count + 1
      os.remove(img_path)
print(f"Total Images Removed: {count}")
```

```
Total Images Removed: 0
```

In [ ]:

```
folders = sorted(os.listdir(data_path))
folders
```

Out[ ]:

```
['badminton',
 'baseball',
 'basketball',
 'boxing',
 'chess',
 'cricket',
 'fencing',
 'football',
 'formula1',
 'gymnastics',
 'hockey',
 'ice_hockey',
 'kabaddi',
 'motogp',
 'shooting',
 'swimming',
 'table_tennis',
 'tennis',
 'volleyball',
 'weight_lifting',
 'wrestling',
 'wwe']
```

In [ ]:

```
# Create a Dictionary for folder names
folderLabels = {}
for index, value in enumerate(folders):
  folderLabels[value] = index
folderLabels
```

Out[ ]:

```
{'badminton': 0,
 'baseball': 1,
 'basketball': 2,
 'boxing': 3,
 'chess': 4,
 'cricket': 5,
 'fencing': 6,
 'football': 7,
 'formula1': 8,
 'gymnastics': 9,
 'hockey': 10,
 'ice_hockey': 11,
 'kabaddi': 12,
 'motogp': 13,
 'shooting': 14,
 'swimming': 15,
 'table_tennis': 16,
 'tennis': 17,
 'volleyball': 18,
 'weight_lifting': 19,
 'wrestling': 20,
 'wwe': 21}
```

In [ ]:

```
class_names = list(folderLabels.keys())
```

Loading of Training Data

In [ ]:

```
import numpy as np
from matplotlib import pyplot as plt
```

In [ ]:

```
datasets = tf.keras.utils.image_dataset_from_directory(data_path)
datasets.class_names
```

```
Found 14065 files belonging to 22 classes.
```

Out[ ]:

```
['badminton',
 'baseball',
 'basketball',
 'boxing',
 'chess',
 'cricket',
 'fencing',
 'football',
 'formula1',
 'gymnastics',
 'hockey',
 'ice_hockey',
 'kabaddi',
 'motogp',
 'shooting',
 'swimming',
 'table_tennis',
 'tennis',
 'volleyball',
 'weight_lifting',
 'wrestling',
 'wwe']
```

In [ ]:

```

```

In [ ]:

```
data_iterator = datasets.as_numpy_iterator()
```

In [ ]:

```
data_iterator
```

Out[ ]:

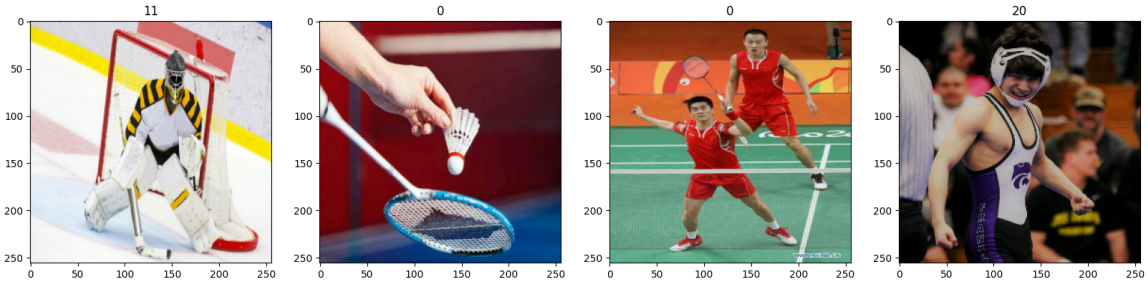
```
<tensorflow.python.data.ops.dataset_ops._NumpyIterator at 0x79afa932c280>
```

In [ ]:

```
batch = data_iterator.next()
```

In [ ]:

```
fig, ax = plt.subplots(ncols=4, figsize=(20,20))
for idx, img in enumerate(batch[0][:4]):
  ax[idx].imshow(img.astype(int))
  ax[idx].title.set_text(batch[1][idx])
```



In [ ]:

```
scaled_data = datasets.map(lambda x,y:(x/255,y))
```

In [ ]:

```
scaled_data
```

Out[ ]:

```
<_MapDataset element_spec=(TensorSpec(shape=(None, 256, 256, 3), dtype=tf.
float32, name=None), TensorSpec(shape=(None,), dtype=tf.int32, name=None))
>
```

In [ ]:

```
scaled_data.as_numpy_iterator().next()
```

Out[ ]:

```
(array([[[[0.33146447, 0.73164827, 0.80802697],
          [0.36473653, 0.7288603 , 0.8185049 ],
          [0.38072917, 0.7056373 , 0.80909926],
          ...,
          [0.35569853, 0.7523591 , 0.8136029 ],
          [0.4117647 , 0.74791664, 0.8465074 ],
          [0.40579045, 0.7352022 , 0.837163  ]],

         [[0.3523635 , 0.7449799 , 0.8251423 ],
          [0.3690523 , 0.73317605, 0.82282066],
          [0.37839392, 0.703302  , 0.806764  ],
          ...,
          [0.3845197 , 0.772726  , 0.83630514],
          [0.40529102, 0.7457588 , 0.83893996],
          [0.38291088, 0.7162837 , 0.816264  ]],

         [[0.36965153, 0.7528493 , 0.8368413 ],
          [0.37921777, 0.7338525 , 0.8256026 ],
          [0.37375176, 0.69865984, 0.8021219 ],
          ...,
          [0.3852526 , 0.7615275 , 0.83029675],
          [0.37552083, 0.7251157 , 0.814408  ],
          [0.34322688, 0.6841276 , 0.78034395]],

         ...,

         [[0.35745227, 0.69470716, 0.8005895 ],
          [0.37006226, 0.7073172 , 0.8131995 ],
          [0.4080047 , 0.7452596 , 0.8511419 ],
          ...,
          [0.31559926, 0.6643007 , 0.7676747 ],
          [0.3421723 , 0.69537556, 0.7972053 ],
          [0.34619415, 0.6995949 , 0.8013259 ]],

         [[0.3637735 , 0.7010284 , 0.80691075],
          [0.38281658, 0.7200715 , 0.82595384],
          [0.40946522, 0.74672014, 0.8526025 ],
          ...,
          [0.30334628, 0.65628743, 0.75824827],
          [0.2862459 , 0.6437786 , 0.7434436 ],
          [0.27077207, 0.6315564 , 0.7295956 ]],

         [[0.35836396, 0.69561887, 0.8015012 ],
          [0.35686275, 0.69411767, 0.8       ],
          [0.37830883, 0.7155637 , 0.82144606],
          ...,
          [0.31614584, 0.669087  , 0.7710478 ],
          [0.2846201 , 0.6454044 , 0.7434436 ],
          [0.27077207, 0.6315564 , 0.7295956 ]]],


        [[[0.18649131, 0.38256973, 0.62962854],
          [0.1882353 , 0.38431373, 0.6313726 ],
          [0.19215687, 0.3882353 , 0.63529414],
          ...,
          [0.69890505, 0.73753864, 0.67387444],
          [0.6534537 , 0.7085708 , 0.62342983],
          [0.6256534 , 0.6838871 , 0.59369105]],

         [[0.186466  , 0.38431373, 0.6313726 ],
          [0.18722624, 0.38507396, 0.63213277],
```

```
        [0.19106488, 0.38891262, 0.6359714 ],
        ...,
        [0.7540288 , 0.79333967, 0.72832084],
        [0.7283318 , 0.7734625 , 0.6910999 ],
        [0.7097343 , 0.75578976, 0.6691323 ]],

       [[0.18431373, 0.38431373, 0.6313726 ],
        [0.18768053, 0.38768053, 0.63473934],
        [0.1915565 , 0.3915565 , 0.6386153 ],
        ...,
        [0.7762426 , 0.8204551 , 0.74832845],
        [0.7731519 , 0.7991848 , 0.7231508 ],
        [0.7661269 , 0.78962564, 0.7102215 ]],

       ...,

       [[0.28641602, 0.58445525, 0.63543564],
        [0.2974571 , 0.58765316, 0.63863355],
        [0.26540288, 0.54775584, 0.6026578 ],
        ...,
        [0.2515625 , 0.54659927, 0.6422181 ],
        [0.2578508 , 0.55196846, 0.641055  ],
        [0.25490198, 0.54901963, 0.6372702 ]],

       [[0.28693387, 0.5849731 , 0.6359535 ],
        [0.29622963, 0.5864257 , 0.6374061 ],
        [0.27050397, 0.5528569 , 0.6077589 ],
        ...,
        [0.25289086, 0.5479276 , 0.6422181 ],
        [0.25490198, 0.54901963, 0.6347426 ],
        [0.25490198, 0.54901963, 0.6313726 ]],

       [[0.28609067, 0.5841299 , 0.6351103 ],
        [0.29332042, 0.5835165 , 0.63449687],
        [0.27688053, 0.5592335 , 0.61413544],
        ...,
        [0.25398284, 0.54901963, 0.6422181 ],
        [0.25157017, 0.5456878 , 0.63141084],
        [0.25157017, 0.5456878 , 0.62804073]]],


      [[[0.05143074, 0.06711701, 0.07888172],
        [0.03173768, 0.04742396, 0.05918866],
        [0.10706697, 0.12275325, 0.13451795],
        ...,
        [0.03137255, 0.04313726, 0.0627451 ],
        [0.06251173, 0.0664333 , 0.08604114],
        [0.1107295 , 0.11465107, 0.13425891]],

       [[0.0306883 , 0.04637457, 0.05813928],
        [0.04668501, 0.06237129, 0.074136  ],
        [0.02748934, 0.04317561, 0.05494032],
        ...,
        [0.03137255, 0.04313726, 0.0627451 ],
        [0.04757194, 0.05149351, 0.07110135],
        [0.04685406, 0.05077562, 0.07038347]],

       [[0.03798595, 0.0530365 , 0.06607265],
        [0.04366832, 0.05871887, 0.07175502],
        [0.04389379, 0.05894434, 0.0719805 ],
        ...,
```

```
       [0.03137255, 0.04313726, 0.0627451 ],
       [0.04705882, 0.05098039, 0.07058824],
       [0.031073  , 0.03499456, 0.05460241]],

      ...,

      [[0.8275438 , 0.60793597, 0.76087713],
       [0.8390088 , 0.61155784, 0.77234215],
       [0.81571347, 0.58826244, 0.7490468 ],
       ...,
       [0.8658029 , 0.68148917, 0.80697936],
       [0.889014  , 0.7047003 , 0.8301905 ],
       [0.8828048 , 0.6984911 , 0.8239813 ]],

      [[0.81005514, 0.58260417, 0.7433885 ],
       [0.8147473 , 0.5872963 , 0.7480806 ],
       [0.81792283, 0.57870716, 0.74341303],
       ...,
       [0.87554383, 0.6833869 , 0.80887717],
       [0.8707552 , 0.67859834, 0.80408853],
       [0.8623172 , 0.67016035, 0.79565054]],

      [[0.8083978 , 0.5719165 , 0.735711  ],
       [0.7936724 , 0.5571911 , 0.72098553],
       [0.8325521 , 0.5933364 , 0.7580423 ],
       ...,
       [0.88535196, 0.6931951 , 0.8186853 ],
       [0.88962364, 0.6974668 , 0.822957  ],
       [0.89199823, 0.6998414 , 0.82533157]]],


     ...,


     [[[0.49551165, 0.43252018, 0.6971588 ],
       [0.51567096, 0.45981157, 0.73957443],
       [0.49967623, 0.46646544, 0.75424117],
       ...,
       [0.13831955, 0.15008426, 0.1696921 ],
       [0.15323223, 0.1689185 , 0.18068321],
       [0.15323223, 0.1689185 , 0.18068321]],

      [[0.4864213 , 0.42201537, 0.68102664],
       [0.5071843 , 0.45491338, 0.7239429 ],
       [0.49401808, 0.4602036 , 0.7467344 ],
       ...,
       [0.14253217, 0.15429688, 0.17390472],
       [0.16165748, 0.17734376, 0.18910846],
       [0.16165748, 0.17734376, 0.18910846]],

      [[0.4712316 , 0.40921417, 0.65336245],
       [0.4968137 , 0.44346032, 0.69776213],
       [0.48862875, 0.4502737 , 0.7241268 ],
       ...,
       [0.14674479, 0.1585095 , 0.17811733],
       [0.16862746, 0.18431373, 0.19607843],
       [0.16862746, 0.18431373, 0.19607843]],

      ...,

      [[0.15062597, 0.09155177, 0.10113358],
```

```
       [0.12207552, 0.07838679, 0.08308961],
       [0.09179688, 0.05650276, 0.06681219],
       ...,
       [0.08314747, 0.07995354, 0.05816279],
       [0.07768429, 0.0622765 , 0.0444902 ],
       [0.07548673, 0.05733415, 0.04092024]],

      [[0.14733456, 0.08645833, 0.09822304],
       [0.12272506, 0.07307374, 0.08121936],
       [0.08909667, 0.05350306, 0.06608456],
       ...,
       [0.08477329, 0.08085172, 0.06472886],
       [0.08172876, 0.06249611, 0.04662224],
       [0.08191636, 0.06230852, 0.04662224]],

      [[0.14733456, 0.08645833, 0.09822304],
       [0.12322304, 0.07282475, 0.08121936],
       [0.08903186, 0.05340074, 0.06608456],
       ...,
       [0.08477329, 0.08085172, 0.06516544],
       [0.08235294, 0.0627451 , 0.04705882],
       [0.08235294, 0.0627451 , 0.04705882]]],


     [[[0.48356313, 0.18160233, 0.07179841],
       [0.4779412 , 0.17598039, 0.06617647],
       [0.4702589 , 0.1682981 , 0.05849418],
       ...,
       [0.12252917, 0.0875644 , 0.01488971],
       [0.12940446, 0.08825862, 0.01675092],
       [0.1336599 , 0.08959865, 0.01820863]],

      [[0.47591913, 0.17395833, 0.06415441],
       [0.4702972 , 0.16833639, 0.05853248],
       [0.46467525, 0.16271447, 0.05291054],
       ...,
       [0.43956974, 0.14869964, 0.10643899],
       [0.44200265, 0.1478084 , 0.10909264],
       [0.44839257, 0.1488894 , 0.11238477]],

      [[0.47512254, 0.17316176, 0.06335784],
       [0.4695006 , 0.16753983, 0.05773591],
       [0.4638787 , 0.1619179 , 0.05211397],
       ...,
       [0.5240273 , 0.16596232, 0.07545127],
       [0.529896  , 0.16319872, 0.07772499],
       [0.52894455, 0.16224724, 0.0774266 ]],

      ...,

      [[0.3899893 , 0.6919501 , 0.5664599 ],
       [0.38129595, 0.68325675, 0.55776656],
       [0.3817172 , 0.68367803, 0.5581878 ],
       ...,
       [0.3500153 , 0.6519761 , 0.5264859 ],
       [0.35667893, 0.6586397 , 0.5331495 ],
       [0.3690686 , 0.6710294 , 0.54553914]],

      [[0.39198837, 0.7063113 , 0.5785616 ],
       [0.3807445 , 0.6950674 , 0.5673177 ],
       [0.37246898, 0.6867919 , 0.55904216],
```

```
  ...,
  [0.36337316, 0.660346  , 0.5327359 ],
  [0.3731301 , 0.6669404 , 0.54027164],
  [0.37988594, 0.6729546 , 0.54650664]],

 [[0.38633057, 0.71574235, 0.582409  ],
  [0.37362897, 0.7030407 , 0.5697074 ],
  [0.36729473, 0.6967065 , 0.56337315],
  ...,
  [0.3728951 , 0.6630912 , 0.5336794 ],
  [0.38127297, 0.67146903, 0.5420573 ],
  [0.39121512, 0.6814112 , 0.55199945]]],


[[[0.24587695, 0.13159999, 0.14747009],
  [0.16240874, 0.10980458, 0.12212075],
  [0.09170461, 0.12332225, 0.14201097],
  ...,
  [0.15202206, 0.21326593, 0.28051472],
  [0.16744791, 0.21842831, 0.28509498],
  [0.16511416, 0.21609455, 0.28276122]],

 [[0.19489124, 0.08061428, 0.09648438],
  [0.12122025, 0.06861608, 0.08093226],
  [0.06854212, 0.10015977, 0.1188485 ],
  ...,
  [0.15269937, 0.21394324, 0.281192  ],
  [0.15648547, 0.20544775, 0.27312347],
  [0.15416732, 0.20160912, 0.27004507]],

 [[0.172572  , 0.05829504, 0.07416514],
  [0.10585172, 0.05324755, 0.06556372],
  [0.06383859, 0.09545624, 0.11414496],
  ...,
  [0.15421441, 0.20999834, 0.2790671 ],
  [0.14662553, 0.19059873, 0.260769  ],
  [0.14934495, 0.18826629, 0.26025984]],

 ...,

 [[0.96373314, 0.9813649 , 0.9843137 ],
  [0.9676547 , 0.9852865 , 0.9882353 ],
  [0.96915597, 0.9867877 , 0.9897365 ],
  ...,
  [0.95234084, 0.8601993 , 0.9337057 ],
  [0.9588388 , 0.8666973 , 0.9402037 ],
  [0.9588388 , 0.8666973 , 0.9402037 ]],

 [[0.95658064, 0.98318714, 0.98216146],
  [0.95855176, 0.98485553, 0.98393077],
  [0.9594308 , 0.98543197, 0.98460805],
  ...,
  [0.95184916, 0.84735316, 0.93185836],
  [0.948334  , 0.84752035, 0.9307981 ],
  [0.94853574, 0.84741944, 0.9307981 ]],

 [[0.9380538 , 0.9722656 , 0.9770604 ],
  [0.93826526, 0.9720818 , 0.9770604 ],
  [0.93847674, 0.97189796, 0.9770604 ],
  ...,
  [0.9529412 , 0.8398054 , 0.92608   ],
```

```
          [0.94311494, 0.8328884 , 0.9188266 ],
          [0.9433264 , 0.83286077, 0.9188266 ]]], dtype=float32),
 array([15, 18, 12,  6, 10, 21, 18,  0,  3, 16,  4, 15,  5, 13, 17, 19,
7,
         6, 16, 20,  7,  5, 17,  9, 12, 17,  7,  6, 15, 21,  0, 20],
       dtype=int32))
```

In [ ]:

```
print(len(scaled_data))
train_size = int(len(scaled_data)*0.7)
val_size = int(len(scaled_data)*0.2)
test_size = int(len(scaled_data)*0.1)
```

```
440
```

In [ ]:

```
train_size
```

Out[ ]:

```
308
```

In [ ]:

```
val_size
```

Out[ ]:

```
88
```

In [ ]:

```
test_size
```

Out[ ]:

```
44
```

In [ ]:

```
train = scaled_data.take(train_size)
test = scaled_data.skip(train_size).take(test_size)
val = scaled_data.skip(train_size+test_size).take(val_size)
```

In [ ]:

```
train
```

Out[ ]:

```
<_TakeDataset element_spec=(TensorSpec(shape=(None, 256, 256, 3), dtype=t
f.float32, name=None), TensorSpec(shape=(None,), dtype=tf.int32, name=Non
e))>
```

In [ ]:

```
# 256,256,3 , here 3 is channels (number of bands in an image)
```

In [ ]:

```
```

Train our CNN model on Training Dataset

In [ ]:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout
```

In [ ]:

```
labels = datasets.class_names
labels
```

Out[ ]:

```
['badminton',
 'baseball',
 'basketball',
 'boxing',
 'chess',
 'cricket',
 'fencing',
 'football',
 'formula1',
 'gymnastics',
 'hockey',
 'ice_hockey',
 'kabaddi',
 'motogp',
 'shooting',
 'swimming',
 'table_tennis',
 'tennis',
 'volleyball',
 'weight_lifting',
 'wrestling',
 'wwe']
```

In [ ]:

```
model = Sequential()

# Conv2D layer with 16 filters, 3x3 kernel, ReLU activation, and input shape
model.add(Conv2D(16, (3, 3), 1, activation='relu', input_shape=(256, 256, 3)))

# MaxPooling2D layer
model.add(MaxPooling2D(pool_size= 2))

# Conv2D layer with 32 filters, 3x3 kernel, ReLU activation
model.add(Conv2D(32, (3, 3), 1, activation='relu'))

# MaxPooling2D layer
model.add(MaxPooling2D(pool_size= 2))

# Conv2D layer with 16 filters, 3x3 kernel, ReLU activation
model.add(Conv2D(16, (3, 3), 1, activation='relu'))

# MaxPooling2D layer
model.add(MaxPooling2D(pool_size= 2))

# Flatten the output
model.add(Flatten())

# Dense layer with 128 units and ReLU activation
model.add(Dense(256, activation='relu'))

# 22 classes with softmax activation
model.add(Dense(len(datasets.class_names), activation='softmax'))
```

In [ ]:

```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accur
acy'])
```

In [ ]:

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 254, 254, 16)      448

 max_pooling2d (MaxPooling2D  (None, 127, 127, 16)     0
 )

 conv2d_1 (Conv2D)           (None, 125, 125, 32)      4640

 max_pooling2d_1 (MaxPooling  (None, 62, 62, 32)       0
 2D)

 conv2d_2 (Conv2D)           (None, 60, 60, 16)        4624

 max_pooling2d_2 (MaxPooling  (None, 30, 30, 16)       0
 2D)

 flatten (Flatten)           (None, 14400)             0

 dense (Dense)               (None, 256)               3686656

 dense_1 (Dense)             (None, 22)                5654

=================================================================
Total params: 3,702,022
Trainable params: 3,702,022
Non-trainable params: 0
_________________________________________________________________
```

Model Training

In [ ]:

```

```

In [ ]:

```
logdir='logs'
tensorboard_callback=tf.keras.callbacks.TensorBoard(log_dir=logdir)
```

In [ ]:

```
from keras import callbacks
hist = model.fit(train, epochs=10, validation_data=val, callbacks=[tensorboard_callbac
k])
```

```
Epoch 1/10
308/308 [==============================] - 936s 3s/step - loss: 2.4397 - a
ccuracy: 0.2699 - val_loss: 2.1066 - val_accuracy: 0.3781
Epoch 2/10
308/308 [==============================] - 919s 3s/step - loss: 1.8231 - a
ccuracy: 0.4616 - val_loss: 1.8830 - val_accuracy: 0.4470
Epoch 3/10
308/308 [==============================] - 913s 3s/step - loss: 1.3671 - a
ccuracy: 0.5899 - val_loss: 1.9816 - val_accuracy: 0.4466
Epoch 4/10
308/308 [==============================] - 908s 3s/step - loss: 0.8201 - a
ccuracy: 0.7577 - val_loss: 2.3099 - val_accuracy: 0.4206
Epoch 5/10
308/308 [==============================] - 908s 3s/step - loss: 0.4175 - a
ccuracy: 0.8790 - val_loss: 3.0096 - val_accuracy: 0.4084
Epoch 6/10
308/308 [==============================] - 908s 3s/step - loss: 0.2064 - a
ccuracy: 0.9409 - val_loss: 3.5436 - val_accuracy: 0.4038
Epoch 7/10
308/308 [==============================] - 913s 3s/step - loss: 0.1373 - a
ccuracy: 0.9619 - val_loss: 3.9733 - val_accuracy: 0.4009
Epoch 8/10
308/308 [==============================] - 878s 3s/step - loss: 0.1168 - a
ccuracy: 0.9668 - val_loss: 4.0629 - val_accuracy: 0.4177
Epoch 9/10
308/308 [==============================] - 1447s 5s/step - loss: 0.0747 -
accuracy: 0.9787 - val_loss: 4.7888 - val_accuracy: 0.3945
Epoch 10/10
308/308 [==============================] - 911s 3s/step - loss: 0.0599 - a
ccuracy: 0.9841 - val_loss: 4.3880 - val_accuracy: 0.4077
```

In [ ]:

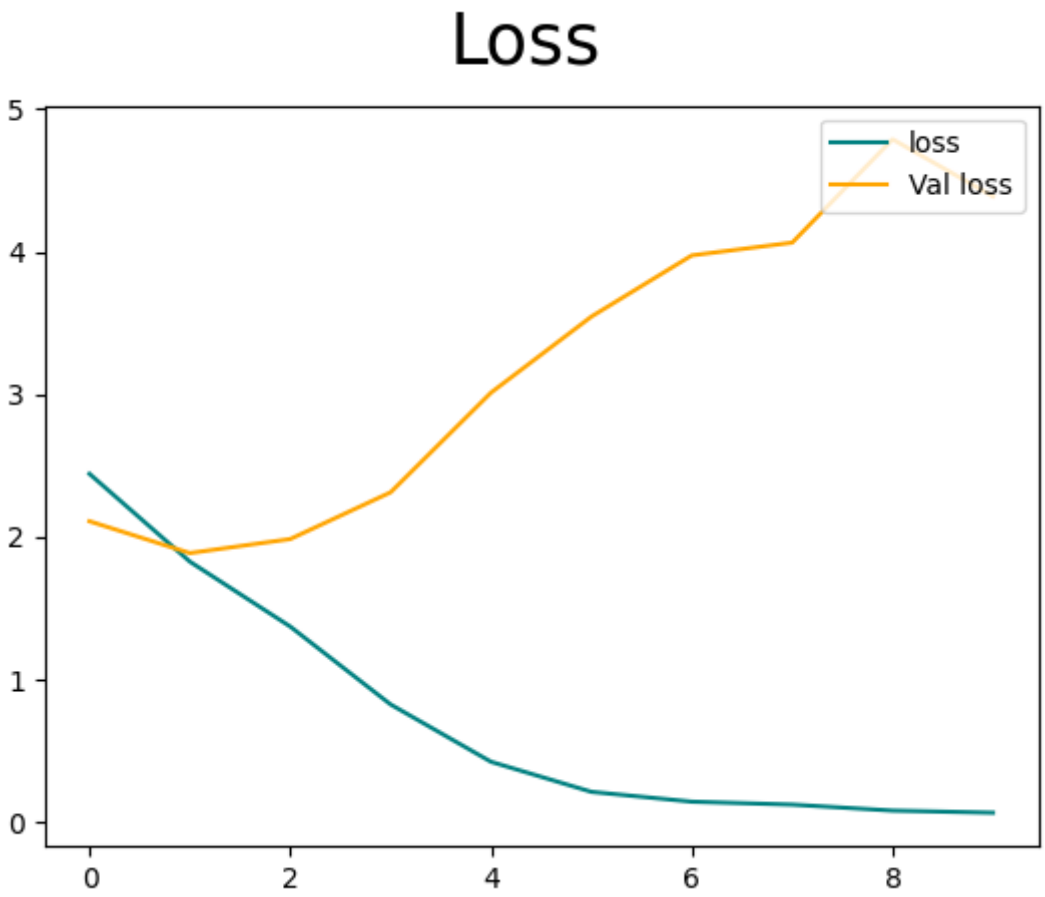
```
#Model Training Ends
```

In [ ]:

```
#Plotting the performance - Training Accuracy and Validation Accuracy
```

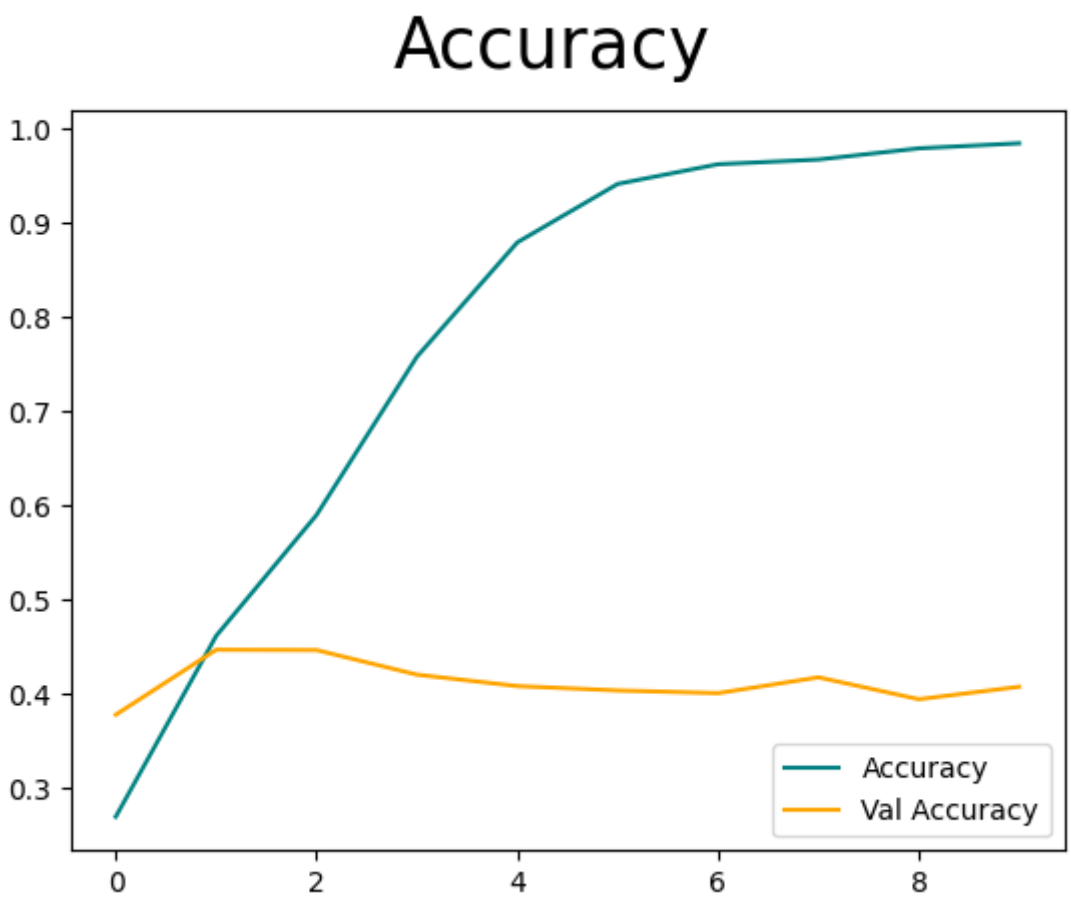
In [ ]:

```
fig = plt.figure()
plt.plot(hist.history['loss'],color='teal',label='loss')
plt.plot(hist.history['val_loss'],color='orange',label='Val loss')
fig.suptitle("Loss", fontsize=25)
plt.legend(loc='upper right')
plt.show()
```



In [ ]:

```
fig = plt.figure()
plt.plot(hist.history['accuracy'],color='teal',label='Accuracy')
plt.plot(hist.history['val_accuracy'],color='orange',label='Val Accuracy')
fig.suptitle("Accuracy", fontsize=25)
plt.legend(loc='lower right')
plt.show()
```



In [ ]:

```
#Evaluate
```

In [ ]:

```
from tensorflow.keras.metrics import Precision, Recall, SparseCategoricalAccuracy
```

In [ ]:

```
# Create an instance of the SparseCategoricalAccuracy metric.
metric = tf.keras.metrics.SparseCategoricalAccuracy()

# Call the `update_state` method for each input/label pair.
for X, y in test:
  metric.update_state(y, model.predict(X))

# Call the `result` method to get the final accuracy score.
accuracy = metric.result()

# Print the accuracy score.
print('Accuracy:', accuracy)
```

```
1/1 [==============================] - 1s 1s/step
1/1 [==============================] - 1s 1s/step
1/1 [==============================] - 1s 788ms/step
1/1 [==============================] - 1s 654ms/step
1/1 [==============================] - 1s 654ms/step
1/1 [==============================] - 1s 611ms/step
1/1 [==============================] - 1s 634ms/step
1/1 [==============================] - 1s 662ms/step
1/1 [==============================] - 1s 616ms/step
1/1 [==============================] - 1s 607ms/step
1/1 [==============================] - 1s 592ms/step
1/1 [==============================] - 1s 597ms/step
1/1 [==============================] - 1s 611ms/step
1/1 [==============================] - 1s 571ms/step
1/1 [==============================] - 1s 605ms/step
1/1 [==============================] - 1s 599ms/step
1/1 [==============================] - 1s 1s/step
1/1 [==============================] - 1s 1s/step
1/1 [==============================] - 1s 1s/step
1/1 [==============================] - 1s 984ms/step
1/1 [==============================] - 1s 1s/step
1/1 [==============================] - 1s 948ms/step
1/1 [==============================] - 1s 918ms/step
1/1 [==============================] - 1s 1s/step
1/1 [==============================] - 1s 964ms/step
1/1 [==============================] - 1s 993ms/step
1/1 [==============================] - 1s 614ms/step
1/1 [==============================] - 1s 634ms/step
1/1 [==============================] - 1s 591ms/step
1/1 [==============================] - 1s 636ms/step
1/1 [==============================] - 1s 584ms/step
1/1 [==============================] - 1s 559ms/step
1/1 [==============================] - 1s 621ms/step
1/1 [==============================] - 1s 606ms/step
1/1 [==============================] - 1s 600ms/step
1/1 [==============================] - 1s 616ms/step
1/1 [==============================] - 1s 611ms/step
1/1 [==============================] - 1s 628ms/step
1/1 [==============================] - 1s 610ms/step
1/1 [==============================] - 1s 608ms/step
1/1 [==============================] - 1s 1s/step
1/1 [==============================] - 1s 1s/step
1/1 [==============================] - 1s 1s/step
1/1 [==============================] - 1s 1s/step
Accuracy: tf.Tensor(0.51846594, shape=(), dtype=float32)
```

In [ ]:

```
print(accuracy)
```

```
tf.Tensor(0.51846594, shape=(), dtype=float32)
```

In [ ]:

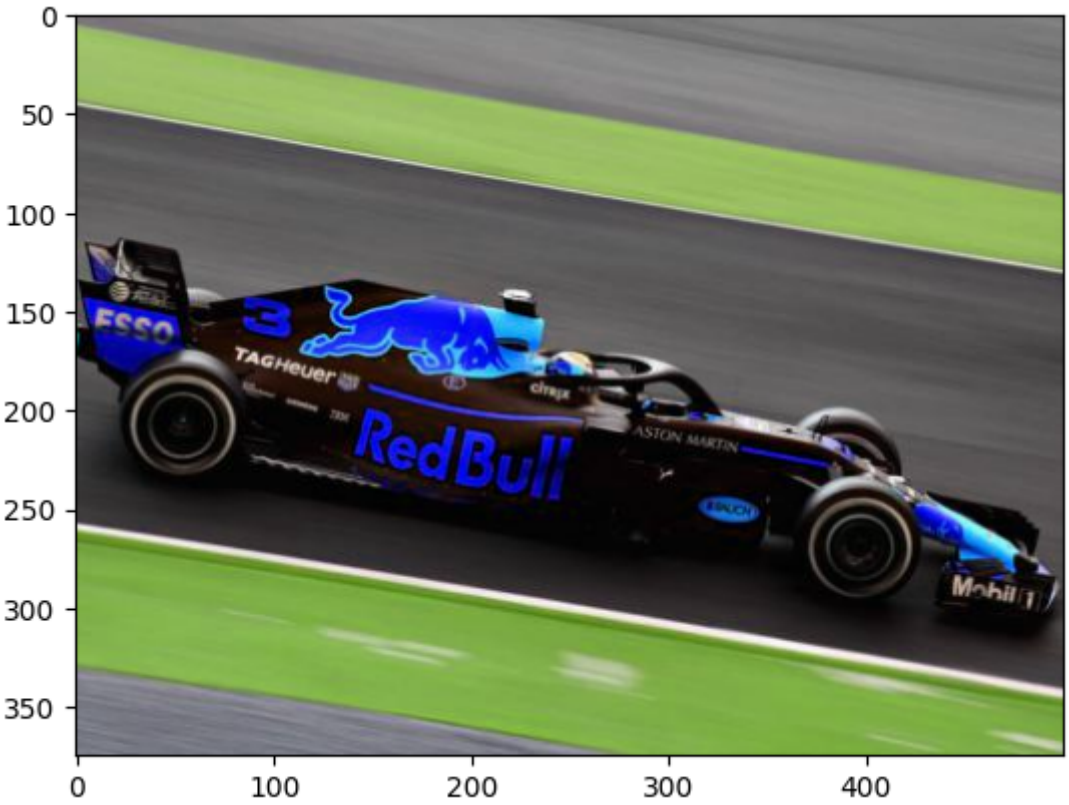
```
#Testing the model
```

In [ ]:

```
img = cv2.imread('/content/drive/MyDrive/DS_Datasets/Sports_Classification/test.jpg')
```

In [ ]:

```
plt.imshow(img)
plt.show()
```



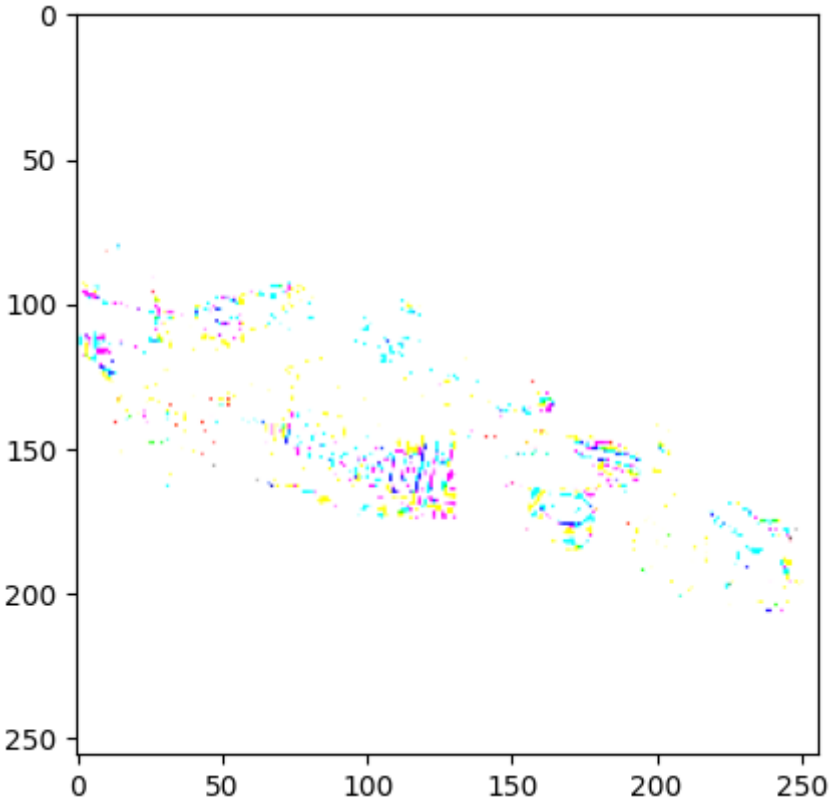
In [ ]:

```
#Read as RBG
```

In [ ]:

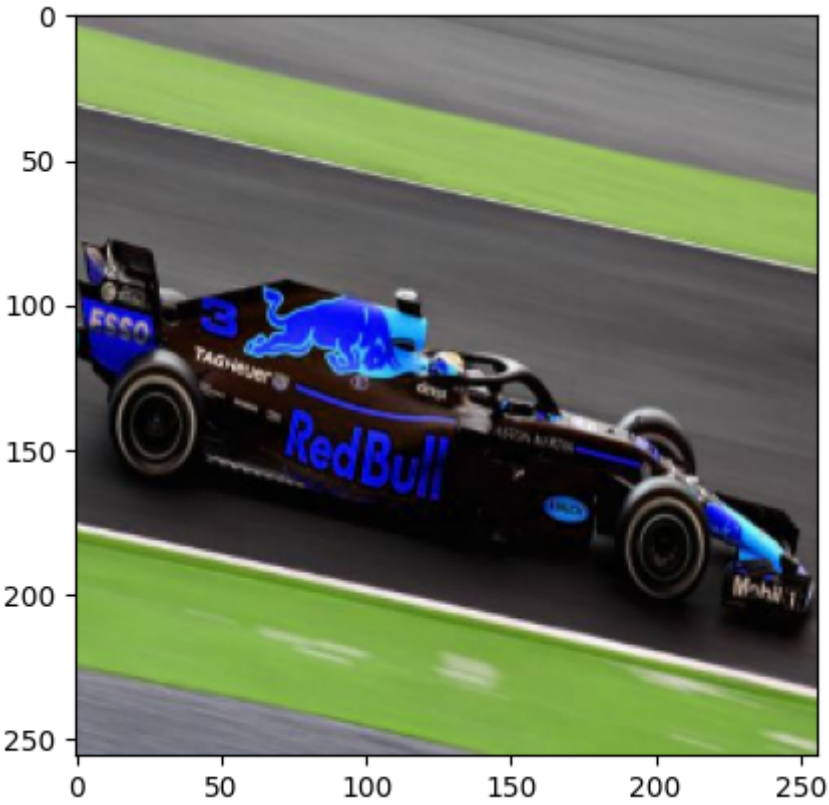
```
#resize
resized = tf.image.resize(img, (256,256))
plt.imshow(resized)
plt.show()
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow
with RGB data ([0..1] for floats or [0..255] for integers).
```



In [ ]:

```
#resize
resized = tf.image.resize(img, (256,256))
plt.imshow(resized.numpy().astype(int))
plt.show()
```



In [ ]:

```
predictions = model.predict(np.expand_dims(resized/255,0))
```

```
1/1 [==============================] - 0s 217ms/step
```

In [ ]:

```
predictions
```

Out[ ]:

```
array([[2.65907060e-04, 2.22427752e-05, 3.56513263e-09, 2.16710168e-06,
        6.79050790e-05, 3.46566109e-10, 3.32177078e-05, 4.08718675e-01,
        5.06906807e-01, 1.40236343e-05, 4.74949848e-05, 4.19803037e-09,
        8.12861833e-10, 5.36027364e-03, 1.01295045e-05, 1.11075497e-05,
        6.71933815e-02, 1.09981168e-02, 5.75902048e-10, 3.48480447e-04,
        4.49890653e-10, 2.48681058e-08]], dtype=float32)
```

In [ ]:

```
folders
```

Out[ ]:

```
['badminton',
 'baseball',
 'basketball',
 'boxing',
 'chess',
 'cricket',
 'fencing',
 'football',
 'formula1',
 'gymnastics',
 'hockey',
 'ice_hockey',
 'kabaddi',
 'motogp',
 'shooting',
 'swimming',
 'table_tennis',
 'tennis',
 'volleyball',
 'weight_lifting',
 'wrestling',
 'wwe']
```

In [ ]:

```
class_labels = {}
for index, value in enumerate(folders):
  class_labels[index] = value
class_labels
```

Out[ ]:

```
{0: 'badminton',
 1: 'baseball',
 2: 'basketball',
 3: 'boxing',
 4: 'chess',
 5: 'cricket',
 6: 'fencing',
 7: 'football',
 8: 'formula1',
 9: 'gymnastics',
 10: 'hockey',
 11: 'ice_hockey',
 12: 'kabaddi',
 13: 'motogp',
 14: 'shooting',
 15: 'swimming',
 16: 'table_tennis',
 17: 'tennis',
 18: 'volleyball',
 19: 'weight_lifting',
 20: 'wrestling',
 21: 'wwe'}
```

In [ ]:

```
for class_index, probability in enumerate(predictions[0]):
    class_label = class_labels.get(class_index, 'Unknown')
    print("Class: {}, Probability: {:.2f}%".format(class_label, probability * 100))
```

```
Class: badminton, Probability: 0.03%
Class: baseball, Probability: 0.00%
Class: basketball, Probability: 0.00%
Class: boxing, Probability: 0.00%
Class: chess, Probability: 0.01%
Class: cricket, Probability: 0.00%
Class: fencing, Probability: 0.00%
Class: football, Probability: 40.87%
Class: formula1, Probability: 50.69%
Class: gymnastics, Probability: 0.00%
Class: hockey, Probability: 0.00%
Class: ice_hockey, Probability: 0.00%
Class: kabaddi, Probability: 0.00%
Class: motogp, Probability: 0.54%
Class: shooting, Probability: 0.00%
Class: swimming, Probability: 0.00%
Class: table_tennis, Probability: 6.72%
Class: tennis, Probability: 1.10%
Class: volleyball, Probability: 0.00%
Class: weight_lifting, Probability: 0.03%
Class: wrestling, Probability: 0.00%
Class: wwe, Probability: 0.00%
```

In [ ]:

```
for class_index, probability in sorted(enumerate(predictions[0]), key=lambda x: x[1], r
everse=True):
  class_label = class_labels.get(class_index, 'Unknown')
  print("Class: {}, Probability: {:.2f}%".format(class_label, probability * 100))
```

```
Class: formula1, Probability: 50.69%
Class: football, Probability: 40.87%
Class: table_tennis, Probability: 6.72%
Class: tennis, Probability: 1.10%
Class: motogp, Probability: 0.54%
Class: weight_lifting, Probability: 0.03%
Class: badminton, Probability: 0.03%
Class: chess, Probability: 0.01%
Class: hockey, Probability: 0.00%
Class: fencing, Probability: 0.00%
Class: baseball, Probability: 0.00%
Class: gymnastics, Probability: 0.00%
Class: swimming, Probability: 0.00%
Class: shooting, Probability: 0.00%
Class: boxing, Probability: 0.00%
Class: wwe, Probability: 0.00%
Class: ice_hockey, Probability: 0.00%
Class: basketball, Probability: 0.00%
Class: kabaddi, Probability: 0.00%
Class: volleyball, Probability: 0.00%
Class: wrestling, Probability: 0.00%
Class: cricket, Probability: 0.00%
```

In [ ]:

```
for class_index, probability in sorted(enumerate(predictions[0]), key=lambda x: x[1], r
everse=True):
  if probability > 0.1:
    class_label = class_labels.get(class_index, 'Unknown')
    print("Top Class: {}, Probability: {:.2f}%".format(class_label, probability * 100))
```

```
Top Class: formula1, Probability: 50.69%
Top Class: football, Probability: 40.87%
```

In [ ]:

```
from tensorflow.keras.models import load_model
```

In [ ]:

```
model.save(os.path.join("/content/drive/MyDrive/MyDrive/DS_Datasets/Sports_Classificati
on/",'image22classifier.h5'))
```

In [ ]:

```
new_model = load_model('/content/drive/MyDrive/MyDrive/DS_Datasets/Sports_Classificatio
n/image22classifier.h5')
```