Name: Emmanuel Asante
Assg: Image Classification in Python
DSet: Sports image classification

1: Prerequisites (Data Cleaning and Organization)

Step 1: Installation of dependecies

In [1]:
```
#pip install tensorflow opencv-python matplotlib
```

Step 2: Import Dependencies

- Tensorflow is a Google-Backed open source library that contains several machine learning models, written in python, buit immplemented in c++
- OS provides us with a os independent way to traverse file directories
- cv2 allows us to import images as matrices
- imghdr is an image processor
- matplotlib allows us to plot various data structures in python

In [2]:
```
import tensorflow as tf
import os
import cv2
import imghdr
import numpy as np
from matplotlib import pyplot as plt
```

```
2023-04-22 15:47:16.829655: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized
to use available CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compile
r flags.
```

Step 3: Limit the VRAM Consumption of Tensor Flow

In [3]:
```
gpus = tf.config.experimental.list_physical_devices('GPU')
for gpu in gpus:
  tf.config.experimental.set_memory_growth(gpu, True)
```

Step 4: Data Cleaning and Sorting

- Data Cleaning involves removing data that is too ambimgous or unidentifiable or not part of the classificaiton i.e. a motorbike in a set of cars
- In this walkthrough, the data set has already been cleaned and classified and sorted into different categories but here are the steps one would take to prepare their data

In [4]:
```
# Preconditions:
# 1. Preparing a variable to store our direcctory information
data_dir = 'data'

# 2. Validating that python can access the directory using the os library
os.listdir(data_dir)

# Process:
# 1. Ensuring that only files with valid file extensions are present
valid_ext = ['jpg', 'jpeg', 'png', 'bmp']

# 2. Loop through data set and remove broken/unuusable files
num_valid = 0
for image_class in os.listdir(data_dir):
    if os.path.isdir(os.path.join(data_dir,image_class)):# If statement useful so that we don'
        for image in os.listdir(os.path.join(data_dir, image_class)):
            image_path = os.path.join(data_dir, image_class,image)
            try:
```

```
                img = cv2.imread(image_path)# tries to open the image file
                tip = imghdr.what(image_path)# returns the type of file present at a file
                if tip not in valid_ext:
                    print("invalid image. Removing:", image_path )
                    os.remove(image_path)
                else:
                    num_valid+=1
            except Exception as e:
                print ('issue with image: {}'.format(image_path))


print (num_valid)
```

```
13572
```

2: Loading Data into Python

To process our data with minimal effort, the *keras API* is integrated into tensorflow provides developers with easy data processing. It provides inbuilt methods to partition,label refactor and shuffle our data sets

Step 1: Loading our images into the keras API. Note: 'data' is the name of the directory that images are located

In [5]:
```
unmodified_data_set = tf.keras.utils.image_dataset_from_directory(data_dir)
class_names = unmodified_data_set.class_names
print (class_names)
```

```
Found 13572 files belonging to 100 classes.
['air hockey', 'ampute football', 'archery', 'arm wrestling', 'axe throwing', 'balance beam', 'barell racing', 'base
ball', 'basketball', 'baton twirling', 'bike polo', 'billiards', 'bmx', 'bobsled', 'bowling', 'boxing', 'bull ridin
g', 'bungee jumping', 'canoe slamon', 'cheerleading', 'chuckwagon racing', 'cricket', 'croquet', 'curling', 'disc go
lf', 'fencing', 'field hockey', 'figure skating men', 'figure skating pairs', 'figure skating women', 'fly fishing',
'football', 'formula 1 racing', 'frisbee', 'gaga', 'giant slalom', 'golf', 'hammer throw', 'hang gliding', 'harness
racing', 'high jump', 'hockey', 'horse jumping', 'horse racing', 'horseshoe pitching', 'hurdles', 'hydroplane racin
g', 'ice climbing', 'ice yachting', 'jai alai', 'javelin', 'jousting', 'judo', 'lacrosse', 'log rolling', 'luge', 'm
otorcycle racing', 'mushing', 'nascar racing', 'olympic wrestling', 'parallel bar', 'pole climbing', 'pole dancing',
'pole vault', 'polo', 'pommel horse', 'rings', 'rock climbing', 'roller derby', 'rollerblade racing', 'rowing', 'rug
by', 'sailboat racing', 'shot put', 'shuffleboard', 'sidecar racing', 'ski jumping', 'sky surfing', 'skydiving', 'sn
ow boarding', 'snowmobile racing', 'speed skating', 'steer wrestling', 'sumo wrestling', 'surfing', 'swimming', 'tab
le tennis', 'tennis', 'track bicycle', 'trapeze', 'tug of war', 'ultimate', 'uneven bars', 'volleyball', 'water cycl
ing', 'water polo', 'weightlifting', 'wheelchair basketball', 'wheelchair racing', 'wingsuit flying']
```

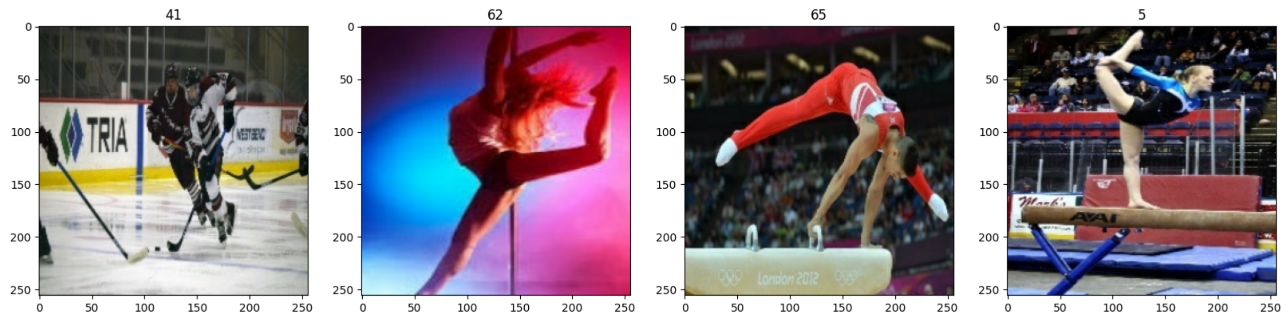
In [6]:
```
# Exploring the data
data_iterator = unmodified_data_set.as_numpy_iterator()

# Batch retrieves a set of 32 images from the data set
batch = data_iterator.next()
print(batch[0].shape)
```

```
(32, 256, 256, 3)
2023-04-22 15:47:32.487743: I tensorflow/core/common_runtime/executor.cc:1197] [/device:CPU:0] (DEBUG INFO) Executor
start aborting (this does not indicate an error and you can ignore this message): INVALID_ARGUMENT: You must feed a
value for placeholder tensor 'Placeholder/_4' with dtype int32 and shape [13572]
         [[{{node Placeholder/_4}}]]
2023-04-22 15:47:32.488023: I tensorflow/core/common_runtime/executor.cc:1197] [/device:CPU:0] (DEBUG INFO) Executor
start aborting (this does not indicate an error and you can ignore this message): INVALID_ARGUMENT: You must feed a
value for placeholder tensor 'Placeholder/_4' with dtype int32 and shape [13572]
         [[{{node Placeholder/_4}}]]
```

Step 4: Testing the classifiers (Ensuring that the labels are actually mapped to some classification)

In [7]:
```
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])
```



3: Preprocessing Data

Step 1: Partition Data

In [8]:
```
batch_size = 32
img_height = 180
img_width = 180

train_set = tf.keras.utils.image_dataset_from_directory(
    data_dir,
    validation_split = 0.2,
    subset = 'training',
    seed = 123,
    image_size = (img_height, img_width),
    batch_size = batch_size)

val_set = tf.keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="validation",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

```
Found 13572 files belonging to 100 classes.
Using 10858 files for training.
Found 13572 files belonging to 100 classes.
Using 2714 files for validation.
```

Step 2: Optimizing set for performance

In [9]:
```
# Performance Script

AUTOTUNE = tf.data.AUTOTUNE

train_set = train_set.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_set = val_set.cache().prefetch(buffer_size=AUTOTUNE)
```

Step 3: Standardize Data

- To reduces the complexity of classification, ideally, our data must be represented in values between 0 and 1
- A simple way to achieve this is by dividing the each element in our batch by 255

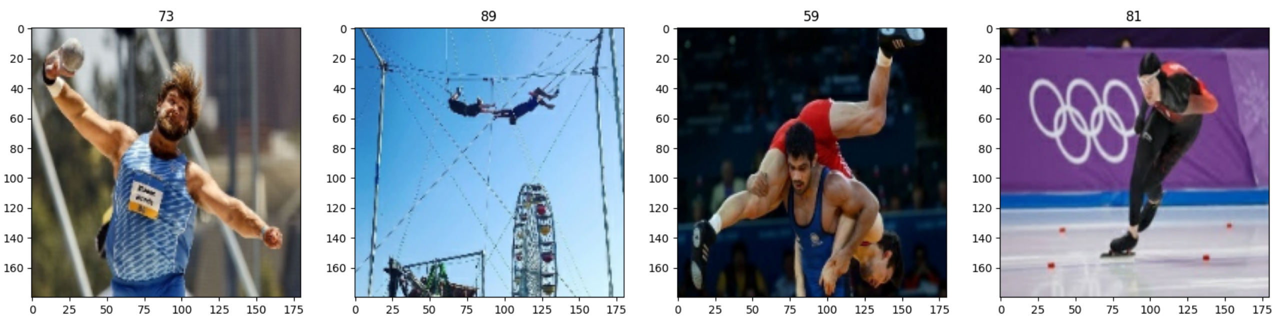
In [10]:
```
normalization_layer = tf.keras.layers.Rescaling(1./255)
normalized_train = train_set.map(lambda x, y: (normalization_layer(x), y))
image_batch, labels_batch = next(iter(normalized_train))
first_image = image_batch[0]
print(np.min(first_image), np.max(first_image))
```

```
2023-04-22 15:47:35.387147: I tensorflow/core/common_runtime/executor.cc:1197] [/device:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an error and you can ignore this message): INVALID_ARGUMENT: You must feed a value for placeholder tensor 'Placeholder/_4' with dtype int32 and shape [10858]
	 [[{{node Placeholder/_4}}]]
2023-04-22 15:47:35.387521: I tensorflow/core/common_runtime/executor.cc:1197] [/device:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an error and you can ignore this message): INVALID_ARGUMENT: You must feed a value for placeholder tensor 'Placeholder/_4' with dtype int32 and shape [10858]
	 [[{{node Placeholder/_4}}]]
0.0 1.0
```

In [11]:

```python
# Ensuring that files are not corrupted after normalization
it_2 = normalized_train.as_numpy_iterator()
batch = it_2.next()

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



Step 3: Configuring Data Set for performance

4: Bulding Neural Network Models

Step 1: Importing relevant libraries.

- We're using the sequential model because it's best used for single input/output modelling
- Conv2D is a convolutional neural network built into tensorflow
- Maxpooling is a condensing function for batches
- Dense, flatten, and dropout are additional libraries that help minimize our input data

In [12]:

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

Step 2: Creating the model sequetially

Layers Explained:

- Input Layer:
  - Retrieves an input of size 256 * 256 with 3bit color depth and applies a 16 filter convolution which scans an image and tries to condense/extract relevant information to make an output classification
  - The filter is 3 * 3 pixels in size and the mask has a stride of 1 pixel
- (MaxPooling):
  - Max cooling is an optimization that traverses and image in 2*2 block and assigns all neuros in that block with the value of the local maximum. This can reduce and images complexity/size by half

In [13]:

```python
# Creating the interal layers of the neural network

# Layer 1:
num_classes = len(class_names)

model = Sequential([
```

```
    Rescaling(1./255, input_shape=(img_height, img_width, 3)),
    Conv2D(16, 3, padding='same', activation='relu'),
    MaxPooling2D(),
    Conv2D(32, 3, padding='same', activation='relu'),
    MaxPooling2D(),
    Conv2D(64, 3, padding='same', activation='relu'),
    MaxPooling2D(),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(num_classes)
])
```

Step 3: Compile the Model

In [14]:
```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

In [15]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_1 (Rescaling)     (None, 180, 180, 3)       0

 conv2d (Conv2D)             (None, 180, 180, 16)      448

 max_pooling2d (MaxPooling2D  (None, 90, 90, 16)       0
 )

 conv2d_1 (Conv2D)           (None, 90, 90, 32)        4640

 max_pooling2d_1 (MaxPooling  (None, 45, 45, 32)       0
 2D)

 conv2d_2 (Conv2D)           (None, 45, 45, 64)        18496

 max_pooling2d_2 (MaxPooling  (None, 22, 22, 64)       0
 2D)

 flatten (Flatten)           (None, 30976)             0

 dense (Dense)               (None, 128)               3965056

 dense_1 (Dense)             (None, 100)               12900

=================================================================
Total params: 4,001,540
Trainable params: 4,001,540
Non-trainable params: 0
_________________________________________________________________
```

Step 4: Train

- When dealing with complex neural networks, maintaining logs is essential to determine the accuracy of one's model, and record any errors if present
- The fit functions allows us to train our model
    - the first argument is the source of the data
    - The second argument is the number of times to pass through the data
    - The third argument is the source of the validation
    - The fourth arguement is the location of the log folder

In [17]:
```
epochs=10
history = model.fit(
  train_set,
  validation_data=val_set,
```

```
    epochs=epochs
)
```

```
Epoch 1/10
2023-04-22 15:48:27.864026: I tensorflow/core/common_runtime/executor.cc:1197] [/device:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an error and you can ignore this message): INVALID_ARGUMENT: You must feed a value for placeholder tensor 'Placeholder/_4' with dtype int32 and shape [10858]
	 [[{{node Placeholder/_4}}]]
2023-04-22 15:48:27.864469: I tensorflow/core/common_runtime/executor.cc:1197] [/device:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an error and you can ignore this message): INVALID_ARGUMENT: You must feed a value for placeholder tensor 'Placeholder/_4' with dtype int32 and shape [10858]
	 [[{{node Placeholder/_4}}]]
340/340 [==============================] - ETA: 0s - loss: 4.1418 - accuracy: 0.0657
2023-04-22 15:50:19.102439: I tensorflow/core/common_runtime/executor.cc:1197] [/device:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an error and you can ignore this message): INVALID_ARGUMENT: You must feed a value for placeholder tensor 'Placeholder/_4' with dtype int32 and shape [2714]
	 [[{{node Placeholder/_4}}]]
2023-04-22 15:50:19.102900: I tensorflow/core/common_runtime/executor.cc:1197] [/device:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an error and you can ignore this message): INVALID_ARGUMENT: You must feed a value for placeholder tensor 'Placeholder/_4' with dtype int32 and shape [2714]
	 [[{{node Placeholder/_4}}]]
340/340 [==============================] - 125s 366ms/step - loss: 4.1418 - accuracy: 0.0657 - val_loss: 3.6310 - val_accuracy: 0.1444
Epoch 2/10
340/340 [==============================] - 134s 394ms/step - loss: 3.0117 - accuracy: 0.2620 - val_loss: 3.0386 - val_accuracy: 0.2653
Epoch 3/10
340/340 [==============================] - 147s 433ms/step - loss: 2.0372 - accuracy: 0.4787 - val_loss: 2.9655 - val_accuracy: 0.3007
Epoch 4/10
340/340 [==============================] - 132s 387ms/step - loss: 1.1065 - accuracy: 0.7003 - val_loss: 3.6025 - val_accuracy: 0.2933
Epoch 5/10
340/340 [==============================] - 128s 375ms/step - loss: 0.4722 - accuracy: 0.8725 - val_loss: 4.4981 - val_accuracy: 0.2653
Epoch 6/10
340/340 [==============================] - 131s 387ms/step - loss: 0.2267 - accuracy: 0.9428 - val_loss: 5.3131 - val_accuracy: 0.2642
Epoch 7/10
340/340 [==============================] - 130s 384ms/step - loss: 0.1284 - accuracy: 0.9692 - val_loss: 6.1570 - val_accuracy: 0.2867
Epoch 8/10
340/340 [==============================] - 129s 379ms/step - loss: 0.0959 - accuracy: 0.9761 - val_loss: 6.4640 - val_accuracy: 0.2760
Epoch 9/10
340/340 [==============================] - 131s 385ms/step - loss: 0.0979 - accuracy: 0.9752 - val_loss: 6.5527 - val_accuracy: 0.2594
Epoch 10/10
340/340 [==============================] - 150s 442ms/step - loss: 0.0750 - accuracy: 0.9806 - val_loss: 7.6832 - val_accuracy: 0.2535
```

Step 5: Visualizing Training Results

In [18]:

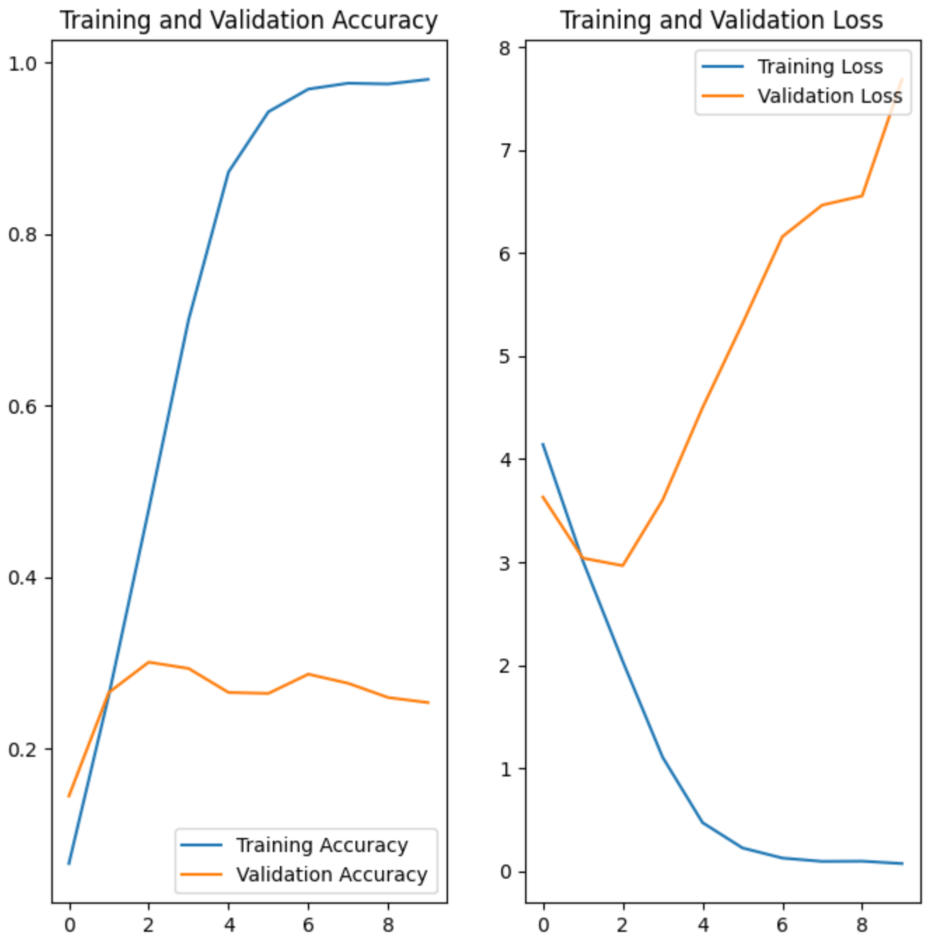
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 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(1, 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()
```



Step 5: Analysis

- Based on the graphs above, one can see that training accuracy increases drastically over each run but validation accuracy is stuck around 20%
- This is a common sign for data overfitting
- We can somewhat remedy through data augmentation

In [ ]: