Transfer Learning for Identifying the Sports

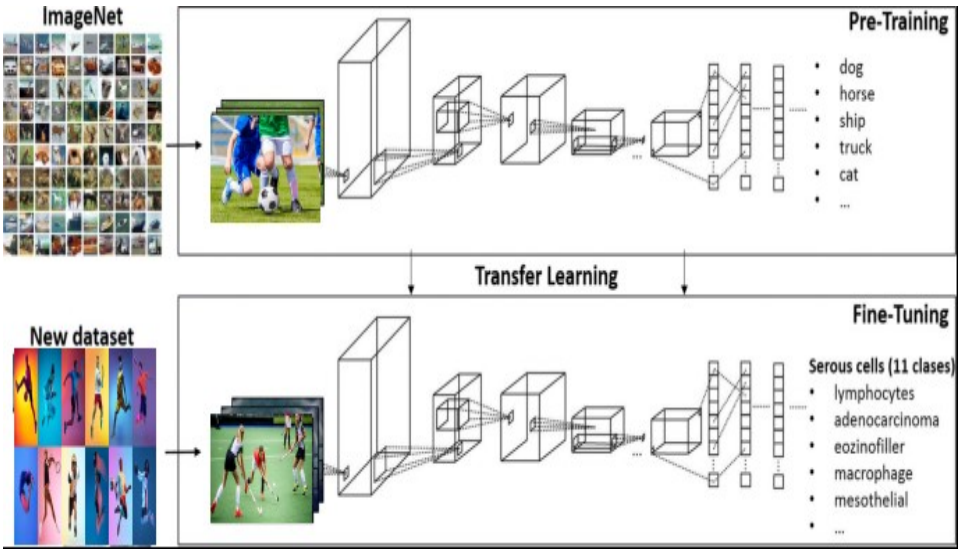
Project Description:

The objective of this study is to classify various types of sports on the basis of snapshots or images of those respective sports.

This model can be used to classify sports images or snapshots as per their respective categories.

Current sports competitions are mostly broadcast in the form of live video or video files, and information detection for athletes and sports economic processes can also be carried out through image detection technology. However, from the current situation, we can see that sports image detection technology is still immature. Therefore, this study uses sports images as a material to analyze the application of sports image detection technology. In this study, image detection technology edge detection, grayscale processing, object capture, target recognition, etc. are combined with the actual needs of sports image to achieve a variety of needs for sports image detection. Simultaneously, this study has realized the recognition of athletes, motion recognition, sports behavior judgment, etc. and built a test platform to verify the effectiveness of this research method. The results show that the research method has certain practicality and can provide a theoretical reference for subsequent related research.

Technical Architecture:



Pre requisites:

To complete this project, you must require the following software's, concepts, and packages

Anaconda Navigator is a free and open-source distribution of the Python and R programming languages for data science and machine learning related applications. It can be installed on Windows, Linux, and macOS.Conda is an open-source, cross-platform, package management system. Anaconda comes with very nice tools like JupyterLab, Jupyter Notebook, QtConsole, Spyder, Glueviz, Orange, Rstudio, Visual Studio Code. For this project, we will be using Jupyter notebook and Visual Studio Code.

To install Anaconda navigator and to know how to use Jupyter Notebook & Spyder using Anaconda watch the video

Link: Click here to watch the video

1. To build Machine learning models you must require the following packages

- **Numpy**:
  - It is an open-source numerical Python library. It contains a multidimensional array and matrix data structures and can be used to perform mathematical operations
- **Scikit-learn:**
  - It is a free machine learning library for Python. It features various algorithms like support vector machine, random forests, and k-neighbors, and it also supports Python numerical and scientific libraries like NumPy and SciPy
- **Flask:**

  Web framework used for building Web applications
- **Python packages:**
  - open anaconda prompt as administrator
  - Type "pip install numpy" and click enter.
  - Type "pip install pandas" and click enter.
  - Type "pip install scikit-learn" and click enter.
  - Type "pip install tensorflow==2.3.2" and click enter.
  - Type "pip install keras==2.3.1" and click enter.
  - Type "pip install Flask" and click enter.

- **Transfer Learning Concepts**
  - **EfficientNet-B0:** EfficientNet-b0 is a transfer learning method. A pre-trained model trained on
    More than a million images from the ImageNet database. The network can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals
  - **Xception**: Xception is a convolutional neural network that is 71 layers deep. The pretrained network can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals.

Flask: Flask is a popular Python web framework, meaning it is a third-party Python library used for developing web applications. **Flask Basics**

If you are using Pycharm IDE, you can install the packages through the command prompt and follow the same syntax as above.

Project Objectives:

By the end of this project you will:

- Know fundamental concepts and techniques of Convolutional Neural Network.
- Gain a broad understanding of image data.
- Know how to pre-process/clean the data using different data preprocessing techniques.
- Know how to build a web application using the Flask framework.

Project Flow:

- The user interacts with the UI (User Interface) to choose the image.
- The chosen image analyzed by the model which is integrated with flask application.
- Transfer learning Model analyzes the image, then prediction is showcased on the Flask UI.

To accomplish this, we have to complete all the activities and tasks listed below

- Data Collection.
  - Create Train and Test Folders.
- Model Building
  - Importing the Model Building Libraries
  - Loading the model
  - Adding Flatten Layers
  - Adding Output Layer
  - Creating a Model object:
  - Configure the Learning Process
  - Import the ImageDataGenerator library
  - Configure ImageDataGenerator class
  - Apply ImageDataGenerator functionality to Train Set and Test set

- Training
    - Train the Model
    - Save the Model
- Testing
    - Test the model
- Application Building
    - Create an HTML file
    - Build Python Code
    - Run the application
    - Final Output

Project Structure:

Create a Project folder which contains files as shown below

- The Dataset folder contains the training and testing images for training our model.
- We are building a Flask Application that needs HTML pages stored in the **templates** folder and a python script **app.py** for server side scripting.
- We need the model which is saved and the saved model in this content is a **sports_model_efficient_net.h5** .
- Templates folder contains about.html, index.html, nailhome.html, nailpred.html pages.
- An IPYNB file is a notebook document created by Jupyter Notebook. **sports-classification-efficientnet.ipynb** .
- We are building a flask application which needs HTML pages stored in the templates folder and a python script app.py for scripting.
- Model.pkl is our saved model. Further we will use this model for flask integration.
- Training folder contains model training files and the training_ibm folder contains IBM deployment files.

Milestone 1: Data Collection

Activity 1: Download the dataset

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.

Link: https://www.kaggle.com/datasets/gpiosenka/sports-classification

Milestone 2: Model Building

Now it's time to Build input and output layers for EfficientNet B0 model
Hidden layers freeze because they have trained sequence, so changing the input and output layers.

Activity 1: Importing the Model Building Libraries

Importing the necessary libraries

```
import warnings
from sklearn.exceptions import ConvergenceWarning
warnings.filterwarnings("ignore", category=ConvergenceWarning)
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.simplefilter(action='ignore', category=UserWarning)

import itertools
import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from PIL import Image
from sklearn.metrics import classification_report, f1_score , confusion_matrix


import tensorflow as tf
from tensorflow import keras
from keras.layers import Dense, Dropout , BatchNormalization
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import layers,models,Model
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers.experimental import preprocessing
from tensorflow.keras.callbacks import Callback, EarlyStopping, ModelCheckpoint, ReduceLROnPlateau
from tensorflow.keras import mixed_precision
mixed_precision.set_global_policy('mixed_float16')
```

Activity 2: Loading the model

```
efficient_net = tf.keras.applications.EfficientNetB0(
    input_shape=(224, 224, 3),
    include_top=False,
    weights='imagenet',
    pooling='max'
)

for i, layer in enumerate(efficient_net.layers):
    efficient_net.layers[i].trainable = False
```

```
Downloading data from https://storage.googleapis.com/keras-applications/efficientnetb0_notop.h5
16705208/16705208 [==============================] - 3s 0us/step
```

The EfficientNet B0 model needs to be loaded and we are storing that into a variable called efficient_net, we also set the hidden layer training as false.

Activity 3: Defining & layers augmentation

Next we are defining the Dense, Activation, Dropout layers for the EfficientNet model. Also we have to augment the layers using the variable augment.

```
augment = tf.keras.Sequential([
  layers.experimental.preprocessing.RandomFlip("horizontal"),
  layers.experimental.preprocessing.RandomRotation(0.1),
  layers.experimental.preprocessing.RandomZoom(0.1),
  layers.experimental.preprocessing.RandomContrast(0.1),
], name='AugmentationLayer')

inputs = layers.Input(shape = (224,224,3), name='inputLayer')
x = augment(inputs)
pretrain_out = efficient_net(x, training = False)
x = layers.Dense(350)(pretrain_out)
x = layers.Activation(activation="relu")(x)
x = BatchNormalization()(x)
x = layers.Dropout(0.25)(x)
x = layers.Dense(num_classes)(x)
```

Activity 4: Adding Output Layer

Our dataset has 100 classes, so the output layer needs to be changed as per the dataset 100 indicates no of classes, relu is the activation function we use for categorical output Adding fully connected layer.

Activity 5: Creating a Model object:

We have created inputs and outputs in the previous steps and we are creating a model and fitting it to the EfficientNet model, so that it will take inputs as per the given and displays the given no of classes.

```
outputs = layers.Activation(activation="softmax", dtype=tf.float32, name='activationLayer')(x)
model = Model(inputs=inputs, outputs=outputs)
```

Activity 6: Configure the Learning Process

- The compilation is the final step in creating a model. Once the compilation is done, we can move on to the training phase. The loss function is used to find errors or deviations in the learning process. Keras requires a loss function during the model compilation process.
- Optimization is an important process that optimizes the input weights by comparing the prediction and the loss function. Here we are using Adam optimizer.
- Metrics are used to evaluate the performance of your model. It is similar to the loss function, but not used in the training process.

```
model.compile(
    optimizer=Adam(0.0005),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

Activity 7: Import the ImageDataGenerator library

In this we will be improving the image data that suppresses unwilling distortions or enhances some image features important for further processing, although performing some geometric transformations of images like rotation, scaling, translation, etc.

Image data augmentation is a technique that can be used to artificially expand the size of a training dataset by creating modified versions of images in the dataset.

The Keras deep learning neural network library provides the capability to fit models using image data augmentation via the ImageDataGenerator class.
Let us import the ImageDataGenerator class from tensorflow Keras.

```
from keras.preprocessing.image import ImageDataGenerator
```

Activity 8: Configure ImageDataGenerator class

ImageDataGenerator class is instantiated and the configuration for the types of data augmentation
There are five main types of data augmentation techniques for image data; specifically:
Image shifts via the width_shift_range and height_shift_range arguments. The image flips via the horizontal_flip and vertical_flip arguments.
Image rotations via the rotation_range argument
Image brightness via the brightness_range argument.
Image zoom via the zoom_range argument.

An instance of the ImageDataGenerator class can be constructed for training set and testing

set.

```
generator = ImageDataGenerator(
    preprocessing_function = tf.keras.applications.efficientnet.preprocess_input,
)
```

Activity 9: Apply ImageDataGenerator functionality to Train Set and Test set

Let us apply ImageDataGenerator functionality to Train Set and Test Set by using the following code. For Training set using flow_from_directory function.

This function will return batches of images from the subdirectories
Arguments:

- Directory: Directory where the data is located. If labels are "inferred", it should contain subdirectories, each containing images for a class. Otherwise, the directory

structure is ignored.

- batch_size: Size of the batches of data which is 10.
- target_size: Size to resize images after they are read from disk.
- class_mode:
  - ‘int': means that the labels are encoded as integers (e.g. for sparse_categorical_crossentropy loss).
  - 'categorical' means that the labels are encoded as a categorical vector (e.g. for categorical_crossentropy loss).
  - 'binary' means that the labels (there can be only 2) are encoded as float32 scalars with values 0 or 1 (e.g. for binary_crossentropy).
  - None (no labels).

Loading our data and performing Data Augmentation.

```
generator = ImageDataGenerator(
    preprocessing_function = tf.keras.applications.efficientnet.preprocess_input,
)

train_images = generator.flow_from_dataframe(
    dataframe=train_df,
    x_col='imgpath',
    y_col='labels',
    target_size=IMAGE_SIZE,
    color_mode='rgb',
    class_mode='categorical',
    batch_size=BATCH_SIZE,
    shuffle=True,
    seed=42,
)

valid_images = generator.flow_from_dataframe(
    dataframe=valid_df,
    x_col='imgpath',
    y_col='labels',
    target_size=IMAGE_SIZE,
    color_mode='rgb',
    class_mode='categorical',
    batch_size=BATCH_SIZE,
    shuffle=False
)

test_images = generator.flow_from_dataframe(
    dataframe=test_df,
    x_col='imgpath',
    y_col='labels',
    target_size=IMAGE_SIZE,
    color_mode='rgb',
    class_mode='categorical',
    batch_size=BATCH_SIZE,
    shuffle=False
)
```

```
Found 13492 validated image filenames belonging to 100 classes.
Found 500 validated image filenames belonging to 100 classes.
Found 500 validated image filenames belonging to 100 classes.
```

We notice that 13492 images belong to 100 classes for training, 500 images belong to 100 classes for testing purposes and 500 images belong to 100 classes for validation purposes.

Milestone 3: Training

Activity 1: Train the Model

Now, let us train our model with our image dataset. The model is trained for 10 epochs first and then 25 epochs for fine tuning. After every epoch, the current model state is saved if the model has the least loss encountered till that time. We can see that the training loss decreases in almost every epoch till 10 epochs and till 25 epochs, probably there is still further scope to improve the model.

Fit_generator functions used to train a deep learning neural network

Arguments:

- steps_per_epoch: it specifies the total number of steps taken from the generator as soon as one epoch is finished and the next epoch has started. We can calculate the value of steps_per_epoch as the total number of samples in your dataset divided by the batch size.

- Epochs: an integer and number of epochs we want to train our model for.

- validation_data can be either:
  - an inputs and targets list
  - a generator
  - an inputs, targets, and sample_weights list which can be used to evaluate the loss and metrics for any model after any epoch has ended.

- validation_steps: only if the validation_data is a generator then only this argument can be used. It specifies the total number of steps taken from the generator before it is stopped at every epoch and its value is calculated as the total number of validation data points in your dataset divided by the validation batch size.

```
history = model.fit(
    train_images,
    steps_per_epoch=len(train_images),
    validation_data=valid_images,
    validation_steps=len(valid_images),
    epochs=10,
    callbacks=[
        EarlyStopping(monitor = "val_loss",
                      patience = 3,
                      restore_best_weights = True),
        ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=2, mode='min')
    ]
)
```

```
history = model.fit(
    train_images,
    steps_per_epoch=len(train_images),
    validation_data=valid_images,
    validation_steps=len(valid_images),
    epochs=25,
    callbacks=[
        EarlyStopping(monitor = "val_loss",
                      patience = 3,
                      restore_best_weights = True),
        ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=2, mode='min')
    ]
)
```

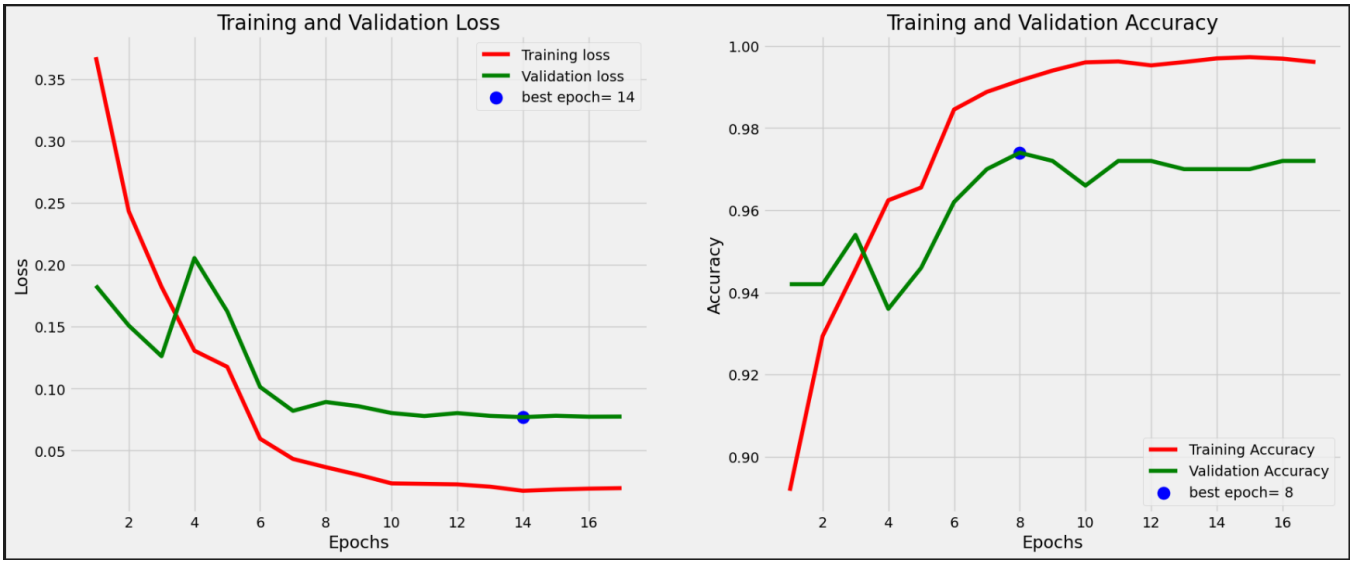
Note: for model, we need to fit the data

```
Epoch 1/10
1350/1350 [==============================] - 116s 71ms/step - loss: 1.7769 - accuracy: 0.5745 - val_loss: 0.4791 - val_accuracy: 0.8740 - lr: 5.0000e-04
Epoch 2/10
1350/1350 [==============================] - 43s 32ms/step - loss: 0.8104 - accuracy: 0.7868 - val_loss: 0.3759 - val_accuracy: 0.8820 - lr: 5.0000e-04
Epoch 3/10
1350/1350 [==============================] - 43s 32ms/step - loss: 0.6488 - accuracy: 0.8270 - val_loss: 0.3166 - val_accuracy: 0.9000 - lr: 5.0000e-04
Epoch 4/10
1350/1350 [==============================] - 43s 32ms/step - loss: 0.5908 - accuracy: 0.8414 - val_loss: 0.3276 - val_accuracy: 0.8980 - lr: 5.0000e-04
Epoch 5/10
1350/1350 [==============================] - 43s 32ms/step - loss: 0.5502 - accuracy: 0.8504 - val_loss: 0.3068 - val_accuracy: 0.9000 - lr: 5.0000e-04
Epoch 6/10
1350/1350 [==============================] - 43s 32ms/step - loss: 0.5176 - accuracy: 0.8529 - val_loss: 0.2448 - val_accuracy: 0.9300 - lr: 5.0000e-04
Epoch 7/10
1350/1350 [==============================] - 43s 32ms/step - loss: 0.4701 - accuracy: 0.8673 - val_loss: 0.2549 - val_accuracy: 0.9260 - lr: 5.0000e-04
Epoch 8/10
1350/1350 [==============================] - 44s 32ms/step - loss: 0.4520 - accuracy: 0.8726 - val_loss: 0.2432 - val_accuracy: 0.9240 - lr: 5.0000e-04
Epoch 9/10
1350/1350 [==============================] - 44s 32ms/step - loss: 0.4469 - accuracy: 0.8732 - val_loss: 0.2349 - val_accuracy: 0.9260 - lr: 5.0000e-04
Epoch 10/10
1350/1350 [==============================] - 43s 32ms/step - loss: 0.4387 - accuracy: 0.8756 - val_loss: 0.2287 - val_accuracy: 0.9240 - lr: 5.0000e-04
```

```
Epoch 1/25
1350/1350 [==============================] - 126s 64ms/step - loss: 0.3675 - accuracy: 0.8917 - val_loss: 0.1831 - val_accuracy: 0.9420 - lr: 1.0000e-04
Epoch 2/25
1350/1350 [==============================] - 83s 62ms/step - loss: 0.2433 - accuracy: 0.9294 - val_loss: 0.1510 - val_accuracy: 0.9420 - lr: 1.0000e-04
Epoch 3/25
1350/1350 [==============================] - 84s 62ms/step - loss: 0.1821 - accuracy: 0.9456 - val_loss: 0.1262 - val_accuracy: 0.9540 - lr: 1.0000e-04
Epoch 4/25
1350/1350 [==============================] - 83s 62ms/step - loss: 0.1306 - accuracy: 0.9624 - val_loss: 0.2054 - val_accuracy: 0.9360 - lr: 1.0000e-04
Epoch 5/25
1350/1350 [==============================] - 83s 62ms/step - loss: 0.1175 - accuracy: 0.9655 - val_loss: 0.1623 - val_accuracy: 0.9460 - lr: 1.0000e-04
Epoch 6/25
1350/1350 [==============================] - 84s 62ms/step - loss: 0.0595 - accuracy: 0.9845 - val_loss: 0.1013 - val_accuracy: 0.9620 - lr: 2.0000e-05
Epoch 7/25
1350/1350 [==============================] - 84s 62ms/step - loss: 0.0432 - accuracy: 0.9888 - val_loss: 0.0820 - val_accuracy: 0.9700 - lr: 2.0000e-05
Epoch 8/25
1350/1350 [==============================] - 83s 61ms/step - loss: 0.0366 - accuracy: 0.9916 - val_loss: 0.0892 - val_accuracy: 0.9740 - lr: 2.0000e-05
Epoch 9/25
1350/1350 [==============================] - 83s 62ms/step - loss: 0.0304 - accuracy: 0.9940 - val_loss: 0.0858 - val_accuracy: 0.9720 - lr: 2.0000e-05
Epoch 10/25
1350/1350 [==============================] - 83s 61ms/step - loss: 0.0235 - accuracy: 0.9960 - val_loss: 0.0803 - val_accuracy: 0.9660 - lr: 4.0000e-06
Epoch 11/25
1350/1350 [==============================] - 83s 61ms/step - loss: 0.0231 - accuracy: 0.9962 - val_loss: 0.0779 - val_accuracy: 0.9720 - lr: 4.0000e-06
Epoch 12/25
1350/1350 [==============================] - 83s 61ms/step - loss: 0.0227 - accuracy: 0.9953 - val_loss: 0.0802 - val_accuracy: 0.9720 - lr: 4.0000e-06
Epoch 13/25
1350/1350 [==============================] - 83s 61ms/step - loss: 0.0208 - accuracy: 0.9961 - val_loss: 0.0780 - val_accuracy: 0.9700 - lr: 4.0000e-06
Epoch 14/25
1350/1350 [==============================] - 83s 62ms/step - loss: 0.0175 - accuracy: 0.9970 - val_loss: 0.0770 - val_accuracy: 0.9700 - lr: 8.0000e-07
Epoch 15/25
1350/1350 [==============================] - 83s 61ms/step - loss: 0.0185 - accuracy: 0.9973 - val_loss: 0.0781 - val_accuracy: 0.9700 - lr: 8.0000e-07
Epoch 16/25
1350/1350 [==============================] - 83s 62ms/step - loss: 0.0192 - accuracy: 0.9969 - val_loss: 0.0773 - val_accuracy: 0.9720 - lr: 8.0000e-07
Epoch 17/25
1350/1350 [==============================] - 83s 62ms/step - loss: 0.0197 - accuracy: 0.9961 - val_loss: 0.0774 - val_accuracy: 0.9720 - lr: 1.6000e-07
```

Activity 2: Check Model Accuracy

Let’s find the accuracy of each model by plotting a graph.



Activity 3: Save the Model

The model is saved with .h5 extension as follows
An H5 file is a data file saved in the Hierarchical Data Format (HDF). It contains multidimensional arrays of scientific data.

```
model.save('sports_model_efficient_net.h5')
```

Milestone 4: Testing

Activity 1: Test the model

Evaluation is a process during the development of the model to check whether the model is the best fit for the given problem and corresponding data.

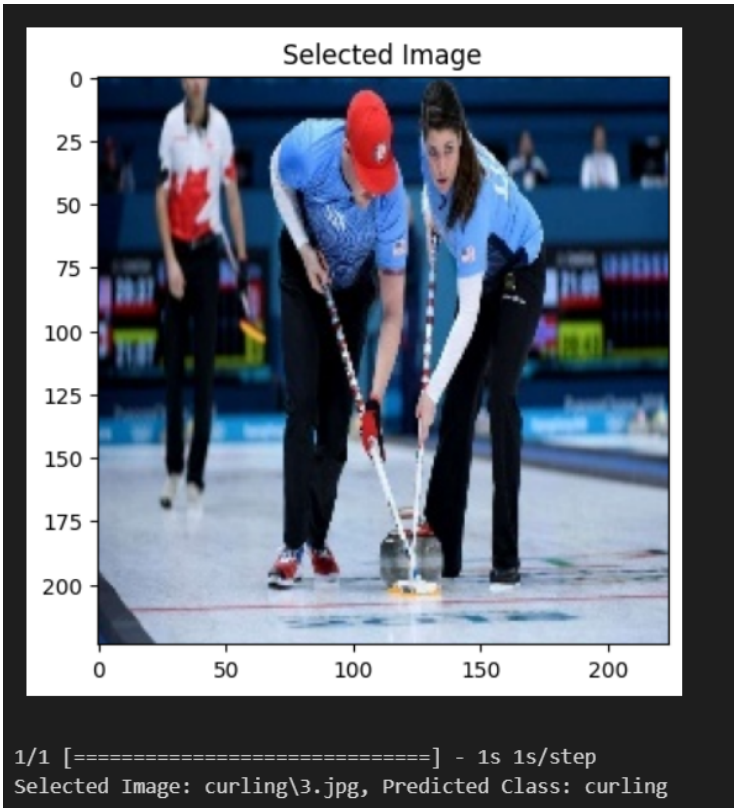
Load the saved model using load_model

```
import os
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import load_img, img_to_array

model = tf.keras.models.load_model('sports_model_efficient_net.h5')
```

Taking an image as input and checking the results
By using the model we are predicting the output for the given input image

```
test_folder = "C:\\Users\\chakr\\OneDrive\\Desktop\\SmartBridge\\Project\\model\\archive\\test"
test_image_rel_path = "curling\\3.jpg"

class_names = sorted(os.listdir(test_folder))

image_path = os.path.join(test_folder, test_image_rel_path)
img = load_img(image_path)
img_array = img_to_array(img)
img_array = img_array.reshape((1,) + img_array.shape)

plt.imshow(img)
plt.title("Test Image")
plt.show()

predictions = model.predict(img_array)

predicted_class_index = np.argmax(predictions, axis=1)
predicted_class_name = class_names[predicted_class_index[0]]
print(f"Selected Image: {test_image_rel_path}, Predicted Class: {predicted_class_name}")
```

The predicted class index name will be printed here.



Milestone 5: Application Building

Now that we have trained our model, let us build our flask application which will be running in our local browser with a user interface.

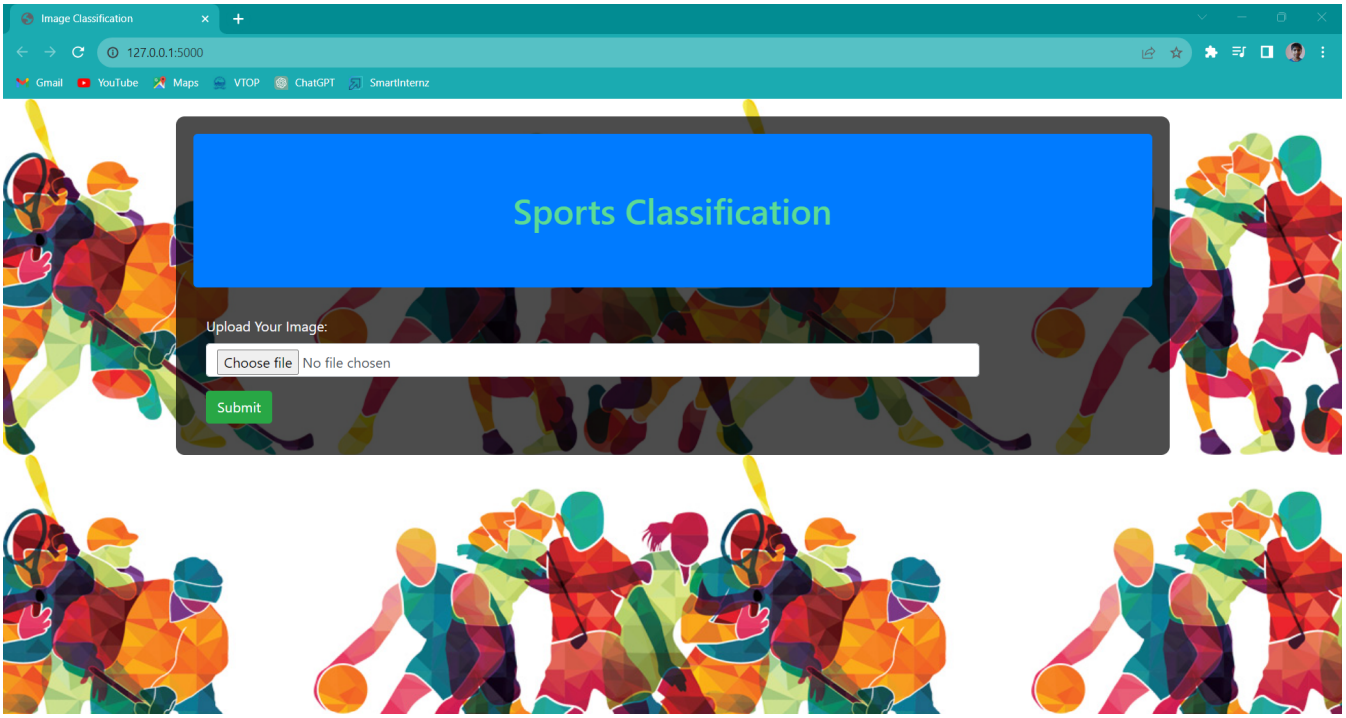
In the flask application, the input parameters are taken from the HTML page; these factors are then given to the model to predict the cost estimation for damage on the HTML page to notify the user. Whenever the user interacts with the UI and selects the “Image” button, the next page is opened where the user chooses the image and predicts the output.

Activity 1: Create HTML Pages

- We use HTML to create the front end part of the web page.
- Here, we have created 2 HTML pages- index.html and sportpred.html

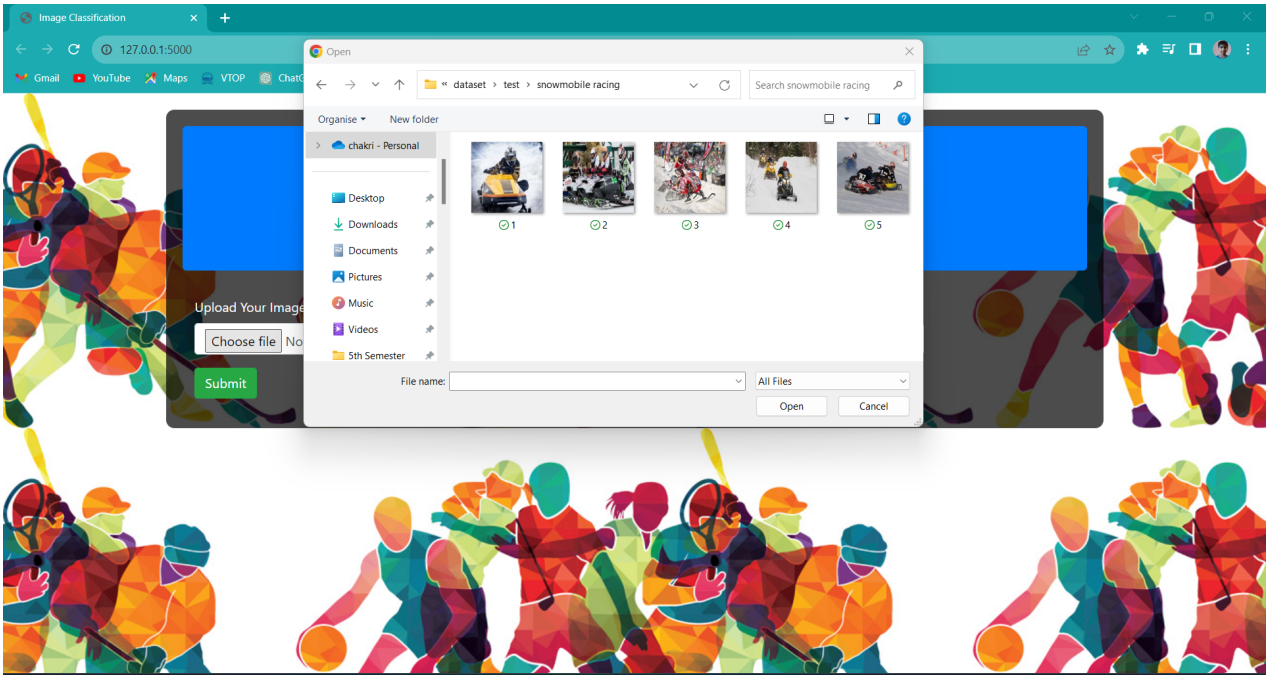
- index.html displays the home page.
- sportpred.html takes the input image and displays the identified result. For more information regarding HTML https://www.w3schools.com/html/
- We also use JavaScript-main.js and CSS-main.css to enhance our functionality and view of HTML pages.
  - **Link :**CSS , JS

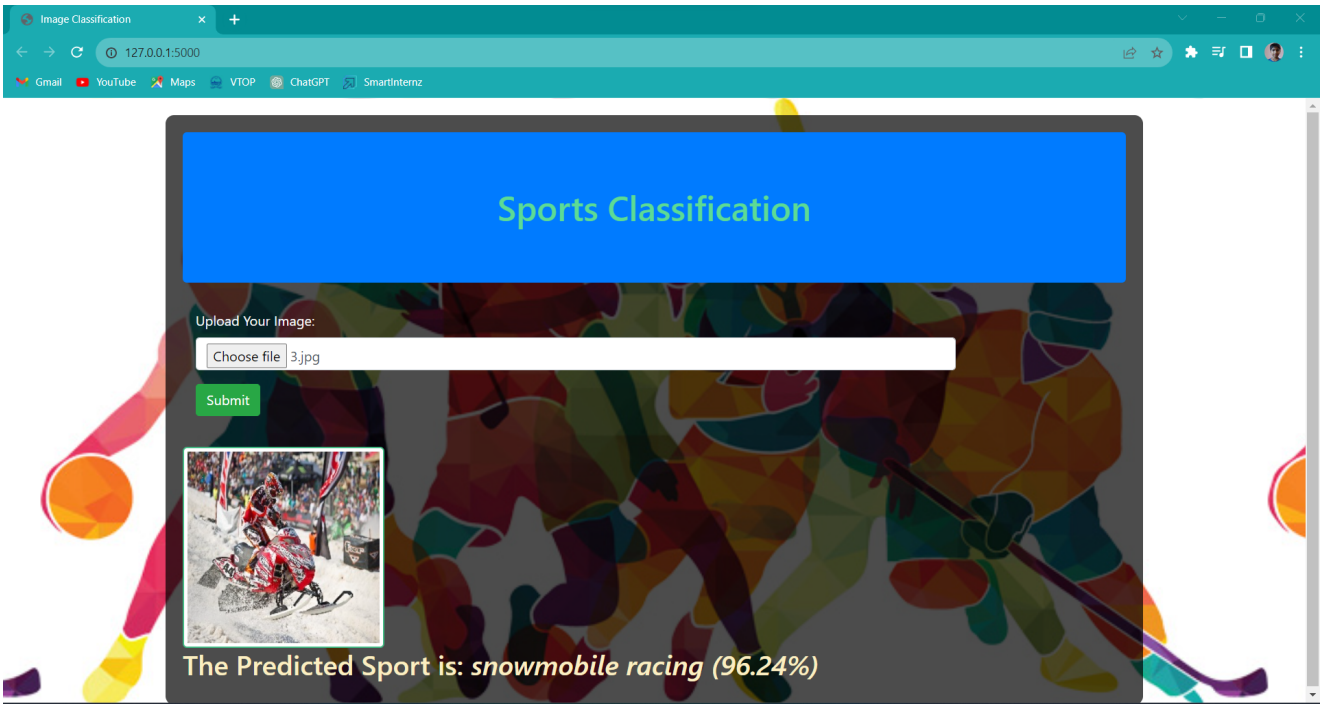
Index.html



Sports Identification Prediction Page:- Now when you click on the Submit button on the left you will get redirected to sportpred.html page.

Using the “Choose File” option users have to load the input image & then by clicking on the “Submit” button user can see the result.





Activity 2: Build python code

Task 1: Importing Libraries

The first step is usually importing the libraries that will be needed in the program.

```
from flask import Flask, render_template, request, jsonify
from PIL import Image
import os
import numpy as np
from keras.models import load_model
from keras.applications.efficientnet import preprocess_input
```

Importing the flask module in the project is mandatory. An object of Flask class is our WSGI application. Flask constructor takes the name of the current module (__name__) as argument.

Task 2: Creating our flask application and loading our model by using load_model method

```
model = load_model('sports_model_efficient_net.h5')
```

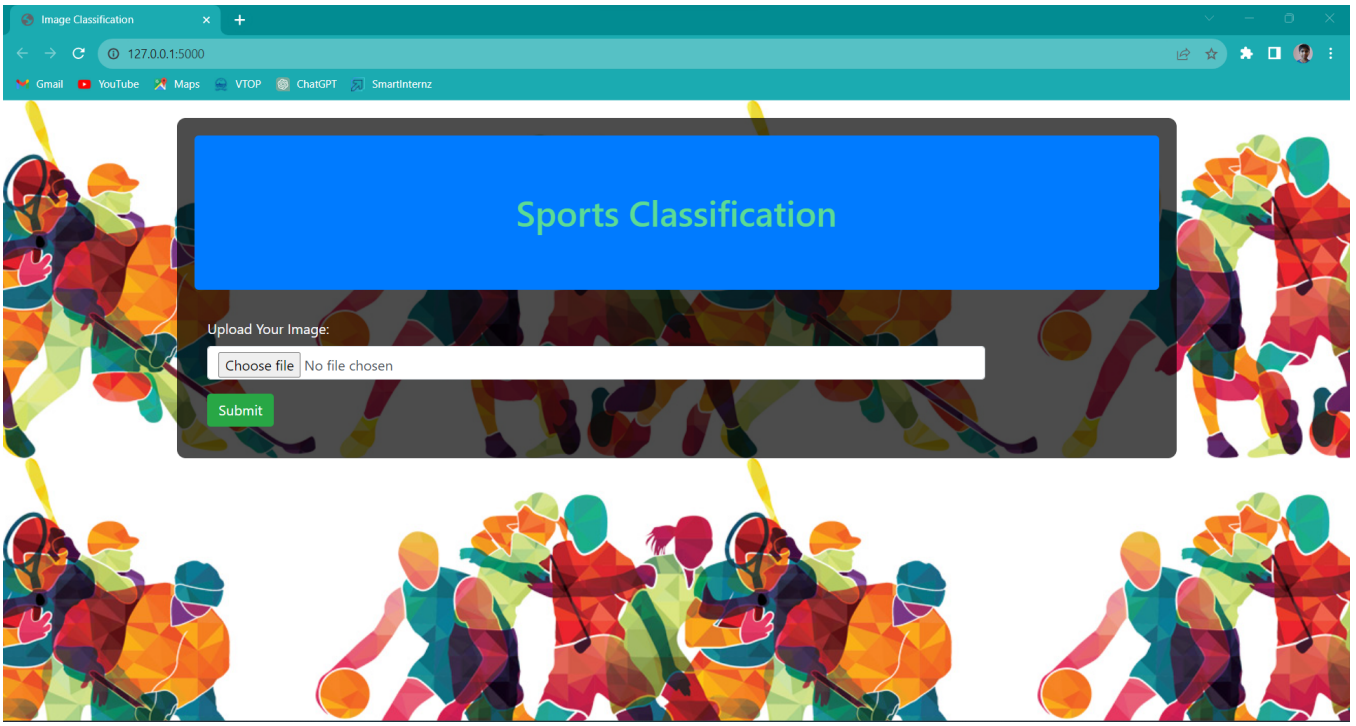
Default home page

Here we will be using a declared constructor to route to the HTML page which we have created earlier.

```python
@app.route('/', methods=['GET', 'POST'])
def predict():
    prediction = None
    imagefile = None
    image_path = None

    if request.method == 'POST' and 'imagefile' in request.files:
        imagefile = request.files['imagefile']

        if imagefile.filename == '':
            return jsonify({"error": "No selected file"})

        destination_directory = request.form.get('destination_directory', 'images')
        os.makedirs(destination_directory, exist_ok=True)

        image_path = os.path.join(destination_directory, imagefile.filename)
        imagefile.save(image_path)

        if is_valid_image(image_path):
            prediction = classify_image(image_path, model)
            return jsonify({"prediction": prediction})
        else:
            return jsonify({"error": "Invalid image file. Please upload a valid image."})

    return render_template('index.html', prediction=prediction)
```

In the above example, '/' URL is bound with the index.html function. Hence, when the home page of the web server is opened in the browser, the html page will be rendered. Whenever you enter the values from the html page the values can be retrieved using POST Method. When you click on about on the home page, it will redirect you to the about page When you click on sport, it will redirect you to the sports identification Home page

When you click on predict, it will redirect you to the sport predict page.



Showcasing prediction on UI:

Here we are defining a function which requests the browsed file from the html page using the post method. The requested picture file is then saved to the uploads folder in this same directory using the OS library. Using the load image class from Keras library we are retrieving the saved picture from the path declared. We are applying some image processing techniques and then sending that preprocessed image to the model for predicting the class. This returns the numerical value of a class (like 0, 1, 2, etc.) which lies in the 0th index of the variable preds. This numerical value is passed to the index variable declared. This returns the name of the class. This name is rendered to the predict variable used in the html page.

Finally, run the application

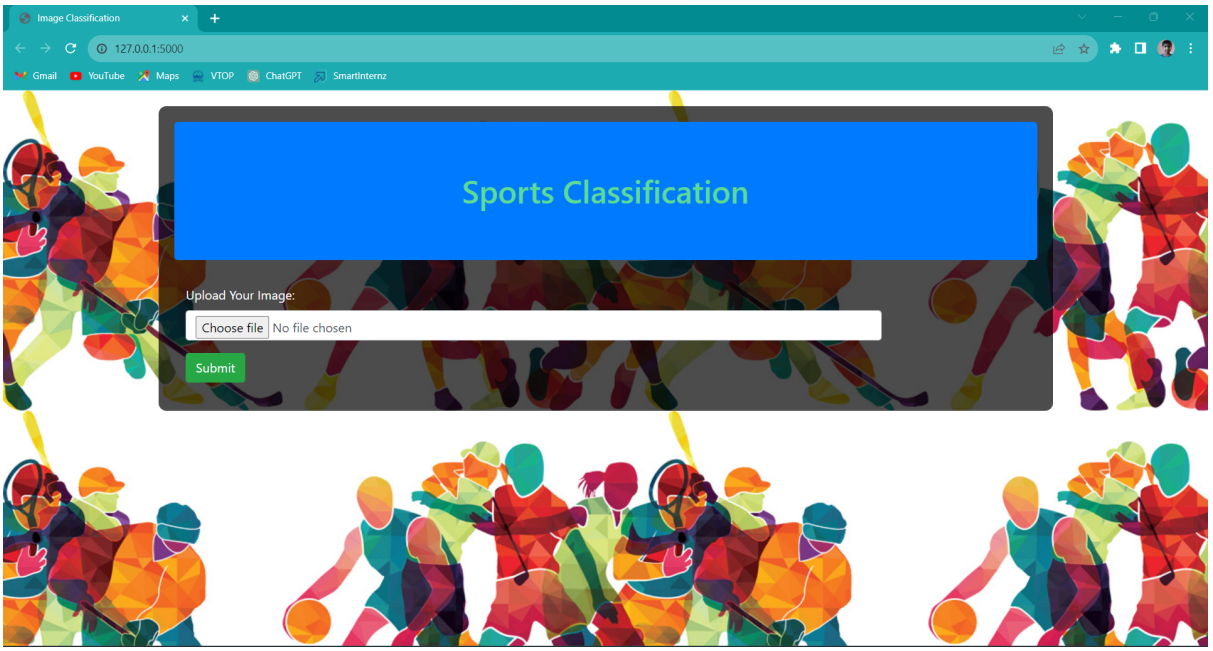
This is used to run the application in a localhost.

Activity 3: Run the application

- Open the anaconda prompt from the start menu.
- Navigate to the folder where your app.py resides.
- Now type “python app.py” command.
- It will show the local host where your app is running on **http://127.0.0.1.5000/** .
- Copy that local host URL and open that URL in the browser. It does navigate me to where you can view your web page.
- Enter the values, click on the predict button and see the result/prediction on the web page.

Then it will run on localhost: 5000
Navigate to the localhost (http://127.0.0.1:5000/) where you can view your web page.

FINAL OUTPUT:



Uploading input image

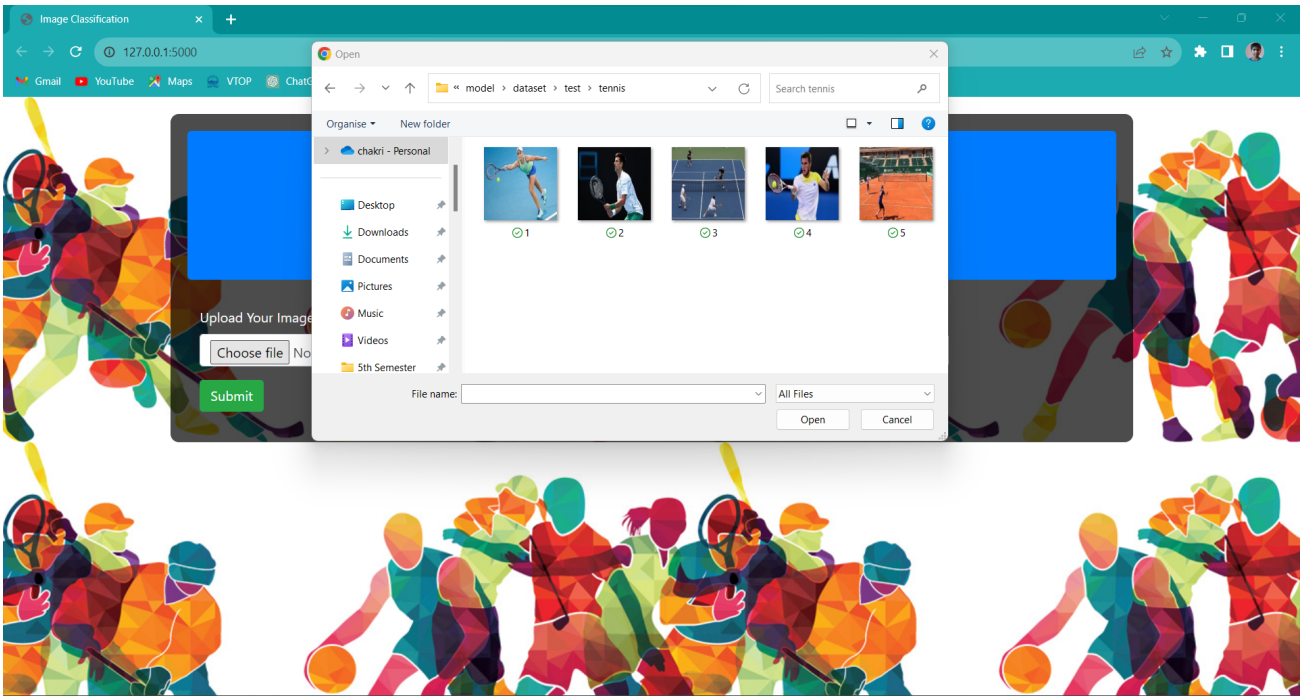
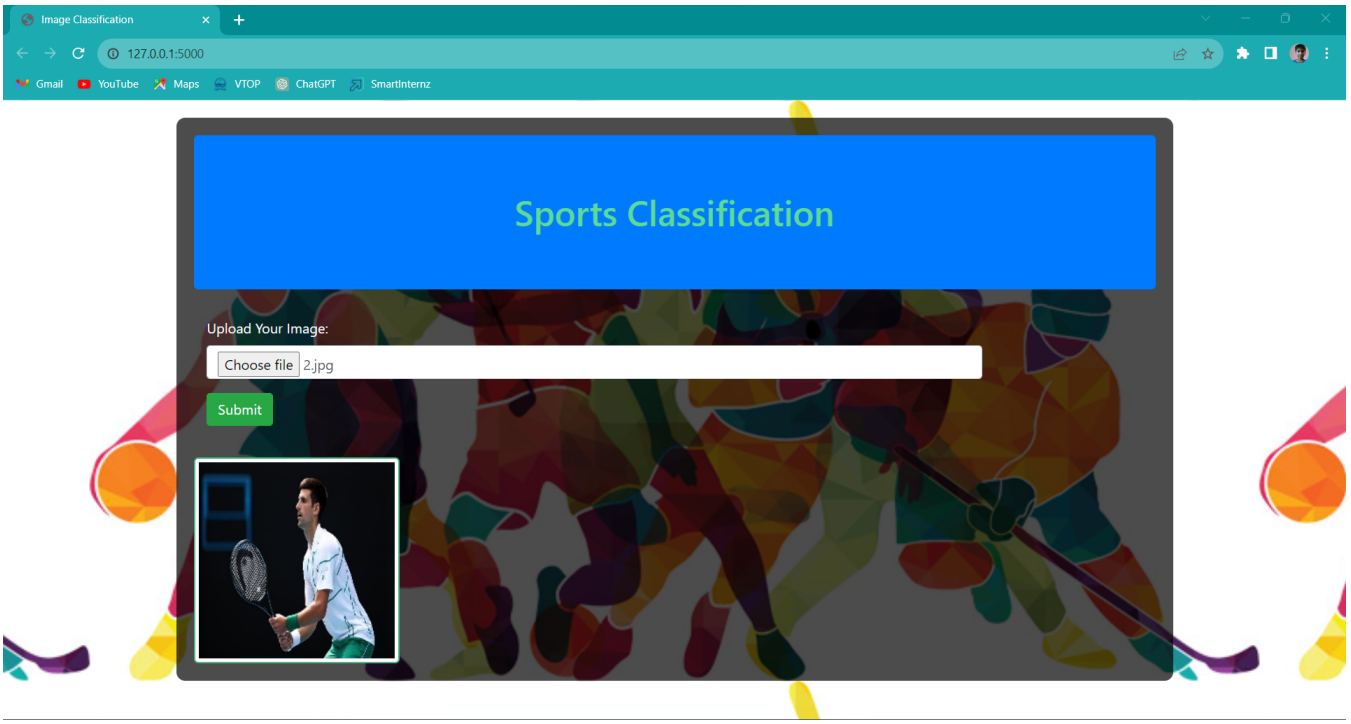
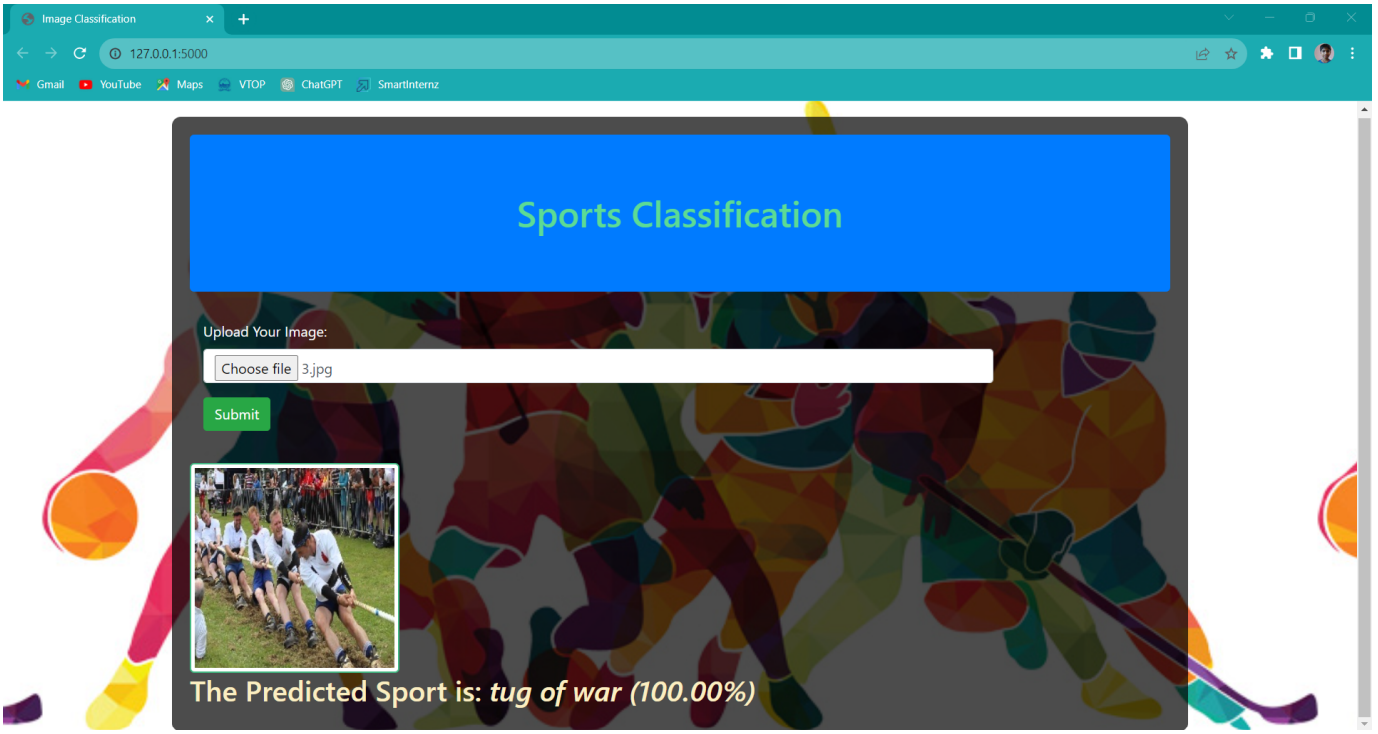


Image uploaded



After clicking on “Submit” button,



Other outputs: