SPORTS CELEBRITY IMAGE CLASSIFICATION PROJECT

REPORT

BHARATIYA ENGINEERING, SCIENCE AND TECHNOLOGY INNOVATION UNIVERSITY (BESTIU)
(Established under Act No. 3 of 2016 of Govt of Andhra Pradesh
& Recognized under Sections 2(f) & 22 of UGC Act, 1956)

Project-Title: Sports Celebrity Image Classification

Submitted by Team-Members:

1.Mekapothula Venu

2.Kunduru Anusha

3.Upputholla Praveen

4.Kowla Sai Teja

Submitted To: Brain o Vision

Submission Date: 27-12-2024

Table Of Contents:

1.Introduction

1.1 Overview of the Project

The project addresses the task of classifying images of prominent sports personalities into their respective sports categories. By leveraging computer vision and deep learning techniques, it automates the identification of athletes based on facial and contextual visual features. This enhances efficiency in image categorization tasks within industries like sports analytics and digital media.

1.2 Importance of Sports Celebrity Image Classification

This classification system has multifaceted applications:

1. Media Automation: Automatically tag athletes in articles, video highlights, and broadcasts.
2. Sports Analytics: Help organizations analyze player trends using labeled image repositories.
3. Fan Engagement: Build personalized content for fans by recognizing their favorite sports stars.
4. Content Curation: Streamline the organization of sports images for documentaries and educational content.

1.3 Brief Introduction to CNN and Transfer Learning

Convolutional Neural Networks (CNNs)
CNNs are a subset of neural networks designed for processing image data. Key components include:

- Convolutional Layers: Detect edges, textures, and shapes.
- Pooling Layers: Reduce spatial dimensions, improving computational efficiency.
- Dense Layers: Final layers for classification.

Transfer Learning
Instead of training a CNN from scratch, pre-trained models like

MobileNetV2 and ResNet50 leverage prior learning (e.g., ImageNet) to recognize general image features. This:

- Reduces computational cost.
- Increases accuracy with limited data.

2.Problem statement & Statistics

Problem Statement:

The primary goal of this project is to develop a robust and accurate system that can classify images of sports celebrities into predefined categories. Challenges include the limited dataset size, variation in image quality, and the presence of similar features among different sports personalities.

3. Abstract

This project explores the application of deep learning techniques for sports celebrity image classification. Leveraging a combination of custom CNN models, MobileNetV2, and ResNet50 architectures, the system achieves competitive accuracy. The Streamlit-based application provides an interactive frontend for end users, allowing for real-time image classification and confidence score visualization. By addressing challenges such as limited data and computational constraints, the project demonstrates the effectiveness of transfer learning and data augmentation in enhancing model performance.

4. DATA COLLECTION & PRE-PROCOSSING

4.1 Objectives:

The primary objectives of this project are to collect labeled images of sports celebrities, ensure high-quality data with balanced class distribution, and facilitate ethical and copyright-compliant data usage. The dataset will support accurate recognition of sports celebrities across various domains like cricket, football, tennis, basketball, and athletics.

4.2 Target Categories

The dataset comprises images of six globally recognized sports stars:

1. **Kane Williamson** - Cricket.
2. **Kobe Bryant** - Basketball.
3. **Lionel Messi** - Football.
4. **Maria Sharapova** - Tennis.
5. **MS Dhoni** - Cricket.
6. **Neeraj Chopra** - Athletics.

- **Total Images**: 608.
- **Classes**: 6 categories (one for each celebrity).

4.3 Data Sources

Images will be sourced from various platforms, including public repositories (Google Images, Flickr), open datasets (Sports-1M, Kaggle), and social media platforms like Instagram and Twitter. Licensed image archives and official athlete or event websites will also contribute to the dataset. Legal and ethical compliance will be maintained throughout.

4.4 Data Collection

Data collection will focus on sourcing images from reliable platforms, including public repositories, official event websites, athlete profiles, and licensed archives. The process involves

ensuring the accuracy and relevance of the images to align with the project's objectives.
To verify the authenticity of the collected images, they will be cross-checked against trusted references, such as official social media accounts and sports team websites. This approach ensures a high-quality and ethically sourced dataset suitable for achieving the desired outcomes.

4.6 Organizing Data After Collection:

Create a Main Folder: Name it something like dataset.
Create Subfolders: Inside the main folder, create a subfolder for each sports person using their name.
Save Images: Place the collected images in the respective subfolders. Name the images as [SportsPersonName]_1.jpg, [SportsPersonName]_2.jpg, and so on.

EXAMPLE:
/dataset
  /Ronaldo
    Ronaldo_1.jpg
    Ronaldo _2.jpg
  /Serena_Williams
    Serena_Williams_1.jpg
    Serena_Williams_2.jpg

4.7 Verifying the Dataset for Correct Arrangement

4.7.1 Check Image Quality:

- Manually review a sample of images to ensure they are clear and relevant to the sports celebrity.
- Ensure that images are not blurry or corrupted.

4.7.2 Verify Correct Labeling:

- Confirm that each image is placed in the correct subfolder based on the sports celebrity's name.
- Check the naming convention to ensure consistency (e.g., [SportsPersonName]_1.jpg, [SportsPersonName]_2.jpg).

4.7.3 Check for Multiple Faces in One Image:

- Manually inspect images to identify if any image contains multiple faces.
- Use an image viewer or editor to zoom in on suspicious images and confirm whether the image contains multiple people.

4.7.4 Remove Irrelevant or Incorrect Images:

- If any image contains more than one face or is not of the intended celebrity, move or delete it from the dataset.
- Ensure only images with a single sports celebrity are kept in the dataset.

4.7.5 Organize the Dataset:

- After manually checking, organize the dataset by removing any problematic images.
- Ensure the final dataset contains only valid, correctly labeled, and correctly categorized images.

4.8 Description of the Dataset:

The dataset includes images of six sports celebrities: Kane Williamson, Kobe Bryant, Lionel Messi, Maria Sharapova, MS Dhoni, and Neeraj Chopra. It consists of:

- Total images: 608
- Classes: 6 (one for each celebrity)

4.9 Data Pre-processing Steps

1. Image Resizing: All images were resized to pixels.
2. Normalization: Pixel values were scaled to [0, 1] for uniformity.
3. Train-Test Split: Data was split into training and testing sets for evaluation.

4.10 Data Augmentation Techniques

- Random Flip: Horizontal flipping of images.
- Random Rotation: Rotating images by a random degree.
- Random Zoom: Zooming into random portions of images.
- Random Contrast: Adjusting image contrast.

These techniques enhanced model robustness by increasing the diversity of training data.

5.Model Architectures

1. Custom CNN Model

Architecture Details

The Custom CNN model was designed from scratch to classify images of sports celebrities. Its architecture includes:

1. **Input Layer**: Accepts 224×224×3224 \times 224 \times 3224×224×3 images (height, width, RGB channels).
2. **Convolutional Layers**:
    - 4 Convolutional Layers with filters ranging from 32 to 128.
    - Activation: ReLU (Rectified Linear Unit), which introduces non-linearity, helping the network learn complex patterns.
    - Stride: 1 for precise feature extraction.
    - Padding: Maintains spatial dimensions after convolutions.
3. **Pooling Layers**:
    - Max-Pooling after each convolutional block to reduce feature map dimensions while retaining the most important information.
4. **Flatten Layer**: Converts the 2D feature maps into a 1D array to feed into fully connected layers.
5. **Dense Layers**:
    - First dense layer with 128 neurons and ReLU activation for feature aggregation.
    - Final dense layer with 6 neurons (equal to the number of classes) and softmax activation for classification probabilities.

Key Design Choices

- Sequential architecture for simplicity and clarity.
- Dropout layers added after dense layers to prevent overfitting.
- Small number of layers compared to pre-trained models to

optimize for dataset size.

Performance

- Achieved 100% training accuracy after 15 epochs.
- Validation Accuracy: Only 54.4%, indicating overfitting due to limited dataset size.

2. Data Augmentation

Purpose

Data augmentation was crucial to address the limited dataset size (608 images). By applying transformations, the diversity of training data increased, reducing overfitting and improving model generalization.

Techniques Applied

1. **Random Flip**: Horizontal flipping to simulate variations in image orientation.
    - Example: Flipping left-facing faces to appear right-facing.
2. **Random Rotation**: Rotating images by a random angle (e.g., $-30 \circ$to$+30$).
    - Introduced new perspectives for the model to learn robust patterns.
3. **Random Zoom**: Zooming into parts of the image to mimic different scales and emphasize details.
4. **Random Contrast Adjustment**: Simulated lighting conditions by varying brightness and contrast.

Benefits

- Enhanced model robustness by exposing it to diverse scenarios.
- Mitigated overfitting by introducing variability into the dataset.

Impact

Augmented data improved the Custom CNN and fine-tuned models like MobileNetV2, leading to better generalization and higher test accuracy.

3. MobileNetV2:

Overview

MobileNetV2 is a lightweight CNN architecture optimized for speed and resource efficiency, particularly on mobile and edge devices. It employs:

1. **Depthwise Separable Convolutions**: Breaks standard convolution into:
    - Depthwise Convolution: Applies a single filter per channel.
    - Pointwise Convolution: Combines results using a 1×11 \times 11×1 filter.
2. **Inverted Residual Blocks**:
    - Uses skip connections to preserve input features while adding non-linearity.
    - Inverts the usual wide-to-narrow pattern of residual connections.
3. **Global Average Pooling:** Reduces the spatial dimensions to a single value per feature map before the final classification layer.

Transfer Learning Implementation

- **Base Model**: Pre-trained on ImageNet, providing generalized feature extraction.
- **Fine-Tuning**:
    - Removed the top layer of MobileNetV2.
    - Added:
        - Global Average Pooling Layer: Condensed feature maps.
        - Dense Layer: 6 neurons with softmax activation for classification.
    - Last few layers of MobileNetV2 fine-tuned on the sports celebrity dataset.

Performance

- Achieved 86.1% test accuracy, making it the best trade-off

between performance and computational cost.
- Faster than ResNet50 while maintaining competitive accuracy.

Strengths

- Highly efficient for small datasets.
- Lightweight, making it suitable for real-time applications**.**

4.ResNet50:

Overview

ResNet50 is a deeper CNN architecture designed to mitigate the vanishing gradient problem. It uses Residual Blocks:

- Skip Connections: Directly connects the input of a block to its output, ensuring the gradient flows through the network even in deep layers.

Architecture Highlights

1. **50 Layers:**
    - Includes convolutional layers, pooling layers, and dense layers.
    - Feature extraction becomes increasingly detailed at deeper layers.
2. **Residual Connections:**
    - Addresses training challenges of deeper networks by allowing the network to learn residual (difference) mappings instead of direct mappings.
3. **Batch Normalization:**
    - Normalizes inputs to each layer, accelerating convergence and improving accuracy.
4. **Global Average Pooling**: Summarizes spatial features before classification.

Transfer Learning Implementation

- Pre-trained on ImageNet for general image recognition tasks.
- Top layers replaced with task-specific layers for sports

celebrity classification.
- Fine-tuned deeper layers for the dataset.

Performance

- Accuracy: Slightly higher (~88%) than MobileNetV2.
- Drawback: Computationally intensive, requiring more time and resources during training.

Strengths

- Superior accuracy due to deeper architecture.
- Handles complex datasets and overlapping features effectively.

Comparison with MobileNetV2:

Feature	MobileNetV2	ResNet50
Accuracy	82.1%	~86%
Speed	Faster	Slower
Complexity	Lightweight	Resource-Heavy

6. Model Training

Training Parameters:

To ensure effective training of the models, the following parameters were carefully chosen:

1. **Optimizer**: Adam
    - **Why Adam?**
      Adam (Adaptive Moment Estimation) is a popular optimizer that combines the benefits of two other optimization algorithms:
        - **Momentum:** Accelerates convergence by considering the past gradients.
        - **RMSProp:** Adapts the learning rate for each parameter based on recent gradient updates.
        - This combination makes Adam effective for both sparse gradients and non-stationary objectives, which is critical for deep learning tasks.
    - **Key Features:**
        - Learning rate adaptation per parameter.
        - Requires minimal tuning.
        - Faster convergence.
2. **Loss Function**: Sparse Categorical Crossentropy
    - **Purpose**: Used for multi-class classification tasks where class labels are integers.
    - **Behavior**: Computes the difference between the predicted probability distribution (softmax output) and the true label distribution.
    - **Why Sparse?:** Instead of one-hot encoding the labels, Sparse Categorical Crossentropy directly uses integer-encoded class labels, saving memory and computation.
3. **Batch Size: 32**
    - Definition: The number of samples processed before the model updates its weights.
    - **Why 32?:**

- Balanced trade-off between memory consumption and convergence speed.
- Ensures sufficient diversity in mini-batches to stabilize training.

4. **Epochs**
   - **15 for Custom Model and MobileNetV2:**
     - These models converged quickly due to their smaller size and transfer learning capabilities.
   - **20 for CNN with Data Augmentation:**
     - Required more epochs to extract robust features from the augmented dataset.

Results Analysis

1. **Custom CNN**
   - Achieved 100% training accuracy after 15 epochs.
   - **Test Accuracy**: Only 54.4%.
     - Indicates overfitting, as the model memorized training data but failed to generalize to unseen data.
2. **MobileNetV2**
   - Achieved 83% test accuracy.
     - Benefited from transfer learning and a pre-trained base on ImageNet.
   - Faster training and better generalization than the custom CNN.
3. **ResNet50**
   - Slightly Higher Accuracy: Achieved slightly better test accuracy (~86%) than MobileNetV2.
   - Trade-Off: More computationally intensive, requiring longer training times and higher resource consumption.

Overfitting and Mitigation Strategies:

Overfitting occurs when a model performs well on the training data but poorly on unseen test data. This was a critical challenge, especially for the custom CNN, due to the limited dataset size.

1. **Data Augmentation**
   - Increased training dataset diversity through transformations like random flips, rotations, zooms, and contrast adjustments.
   - **Result:** Improved generalization, particularly for CNN-based models.
2. **Dropout Layers**
   - Introduced in the custom CNN after dense layers.
   - **Mechanism:**
     - During training, randomly "drops out" (sets to zero) a fraction of neurons in a layer.
     - This forces the network to learn redundant representations, improving robustness.
   - **Dropout Rate:** Typically set between 0.2 to 0.5 for optimal results.
3. **Transfer Learning**
   - Pre-trained models like MobileNetV2 and ResNet50 come with weights optimized for large datasets (e.g., ImageNet).
   - Fine-tuning only the top layers for the specific task reduced overfitting risks.

Key Observations

1. **Custom CNN:**
   - High training accuracy but poor test performance due to overfitting.
   - Mitigated by dropout and data augmentation, though results remained suboptimal.
2. **MobileNetV2:**
   - Balanced accuracy and computational efficiency.

- Performed well due to pre-trained weights and transfer learning.

3. ResNet50:

- Slightly better accuracy than MobileNetV2.
- Overfitting was naturally reduced due to its deeper architecture, though computational cost was a drawback.

7. Performance Evaluation

Comparative Analysis

Performance metrics were calculated for each model to evaluate their accuracy, precision, recall, and F1-score. These metrics provided a comprehensive understanding of the models' ability to classify images effectively.

Model	Accuracy	Precision	Recall	F1-Score
Custom CNN	54.4%	0.54	0.55	0.55
MobileNetV2	83%	0.86	0.86	0.86
ResNet50	~86%	0.88	0.88	0.88

1. **Custom CNN**
   - **Accuracy**: 54.4%, indicating overfitting to the training data with poor generalization.
   - Precision, Recall, and F1-Score: Around 0.54-0.55, showing limited reliability in predictions across classes.
2. **MobileNetV2**
   - **Accuracy**: 83%, a significant improvement over the custom CNN.
   - Precision, Recall, and F1-Score: All at 0.86, demonstrating balanced performance across all classes.
3. **ResNet50**
   - **Accuracy**: ~86%, slightly higher than MobileNetV2.
   - Precision, Recall, and F1-Score: All at 0.88, reflecting excellent classification performance.
   - **Trade-Off**: While more accurate, it was computationally slower and less efficient compared to MobileNetV2.

Confusion Matrices and Reports

1. **Confusion Matrix**
    - A confusion matrix was created for each model to analyze misclassifications.
    - Heatmaps: Used to visually represent misclassified instances, highlighting patterns such as confusion between visually similar classes.
2. **Insights from MobileNetV2:**
    - Exhibited fewer misclassifications compared to the custom CNN.
    - Balanced trade-off between true positives (correct classifications) and false positives (misclassifications).
3. **ResNet50:**
    - Further reduced misclassifications but required more resources for computation.

Best-Performing Model

MobileNetV2 emerged as the most practical model due to the following factors:

1. **High Accuracy**: Achieved 83% test accuracy, which was close to the more computationally intensive ResNet50 (~86%).
2. **Efficiency:**
    - Faster training and inference times.
    - Lower computational overhead, making it suitable for deployment on edge devices and lightweight applications.
3. **Ease of Deployment**: Its smaller size and efficiency enabled smooth integration into the Streamlit-based frontend.

8. Deployment with Streamlit

Streamlit was used to create an interactive web application for the Sports Celebrity Image Classification project. This deployment enabled seamless user interaction with the trained model.

Key Features of the Streamlit Application

1. **User-Friendly Interface:**
   - The app features an intuitive layout, allowing users to navigate and interact effortlessly.
   - Simple upload functionality ensures minimal technical expertise is required for use.
2. **Real-Time Predictions:**
   - Upon image upload, the model processes the input and provides immediate predictions.
   - Confidence scores for each class are displayed, enhancing transparency in model outputs.
3. **Interactive Visualizations:**
   - Classification probabilities are shown as a bar chart, helping users interpret the confidence levels for each predicted category.

Explanation of .h5 File Integration

1. **Model Saving:**
   - The trained MobileNetV2 model was saved in the .h5 (Hierarchical Data Format) format.
   - This format preserves the model architecture, weights, and optimizer state, ensuring compatibility during inference.
2. Model Loading:
   - The saved .h5 file was loaded into the Streamlit application using TensorFlow/Keras libraries.
   - This allows the pre-trained model to be used for

predictions in the live application without retraining.

3. **Inference Process:**
    - When an image is uploaded, the model processes it through the pipeline (preprocessing → feature extraction → prediction).
    - Results are displayed on the app interface.

Walkthrough of the User Interface

1. **Image Upload:**
    - Users can upload images in JPG, JPEG, or PNG formats via a drag-and-drop or file selector interface.
    - Uploaded images are displayed on-screen for confirmation.
2. **Celebrity Selection:**
    - A dropdown menu allows users to select a sports celebrity for comparison.
    - This feature adds flexibility for testing specific classes.
3. Predictions:
    - The app outputs the predicted class (e.g., "Lionel Messi") along with its confidence score (e.g., "Confidence: 86%").
    - This gives users insight into the model's certainty regarding its predictions.
4. Visualization:
    - A bar chart displays the confidence scores for all classes.
    - The chart helps users understand how the model perceives the input image across all categories.

Key Benefits of Deployment

- **Accessibility**: Streamlit allows the model to be accessed and used by non-technical users via a browser.
- **Scalability**: The lightweight nature of MobileNetV2 ensures smooth performance even on systems with limited computational power.
- **Practical Utility**: Real-time predictions and user-friendly design make the application useful for sports broadcasters, analytics teams, and fan engagement platforms.

9. Challenges and Solutions

Explanation of Challenges and Solutions

Challenges

1. **Limited Dataset Size**
   - **Challenge**: A small dataset may result in poor generalization and increased overfitting, as the model has limited examples to learn from.
   - **Impact**: This can lead to the model performing well on training data but poorly on unseen data, reducing its effectiveness.
2. **Balancing Accuracy and Computational Efficiency**
   - **Challenge**: High-accuracy models like ResNet50 are computationally intensive and may not perform efficiently in real-time applications.
   - **Impact**: Deploying these models can slow down the application and require significant computational resources.
3. **Deploying a Large Model on a Lightweight Application**
   - **Challenge**: Integrating large models into lightweight frameworks like Streamlit can cause latency, memory usage issues, and unresponsiveness in user interactions.
   - **Impact**: This hinders the user experience, especially on devices with limited resources.

Solutions

1. **Applied Data Augmentation**
   - **What was done**: Introduced transformations like rotation, flipping, and zooming to artificially increase dataset size and diversity.
   - **Result**: Improved model generalization by exposing it to a wider variety of data patterns, reducing overfitting.
2. **Used MobileNetV2 for Its Lightweight Architecture**
   - **What was done**: Chose MobileNetV2, a model designed for efficient inference, balancing accuracy with computational efficiency.
   - **Result**: Achieved high test accuracy (86.1%) while ensuring the application remains lightweight and responsive.
3. **Optimized the Streamlit Application for Seamless Performance**
   - **What was done**: Used caching mechanisms like @st.cache_resource to avoid reloading the model repeatedly and ensure faster predictions.
   - **Result**: Enhanced user experience by reducing latency during model inference and maintaining smooth interactivity.

10.Future Work & Conclusion

Future Work

1. **Expanding the Dataset**
    - Increase the dataset size to include more sports categories and celebrities, improving model accuracy and versatility.
2. **Exploring Advanced Architectures**
    - Experiment with cutting-edge models like Vision Transformers to push the boundaries of accuracy and efficiency.
3. **Enhancing the Streamlit Application**
    - Add features such as live video analysis for real-time classification and interactive explanations of model predictions.

Conclusion

This project showcased the successful implementation of deep learning for sports celebrity image classification, integrating data augmentation, pre-trained models, and an interactive Streamlit frontend. With a lightweight yet high-performing architecture like MobileNetV2 with Accuracy 83%, the system achieved impressive accuracy and usability. Future enhancements, including dataset expansion and advanced features, can further elevate its impact and functionality.

11.Discussions and References

Discussions

The project demonstrated significant accuracy improvements through transfer learning using MobileNetV2 and ResNet50. While MobileNetV2 provided a balance between performance (86.1% accuracy) and computational efficiency, ResNet50 achieved slightly higher accuracy (~88%) at the cost of increased computational complexity. Challenges such as dataset limitations and class imbalances were addressed through data augmentation, yet they remain areas for future improvement. The deployment via Streamlit highlighted the practical integration of deep learning into a user-friendly application, setting a foundation for future advancements like live video analysis and support for additional classes.

References

1. TensorFlow and Keras Documentation
2. MobileNetV2 Research Paper
3. Online tutorials and resources on transfer learning
4. Python Libraries: Matplotlib, Pandas, NumPy, OpenCV

12. Appendix

Code Listings:

Data Loading to model building :

[1]:
```
import matplotlib.pyplot as plt  # For visualizing images and data plots
import pandas as pd              # For data manipulation and analysis
import numpy as np               # For numerical computations
import cv2                       # For image processing
import os                        # For file and directory operations
import PIL                       # For handling image files
import tensorflow as tf          # TensorFlow library for building ML models
import tf_keras                  # Keras interface in TensorFlow (alternative import)
import tensorflow_hub as hub     # For using pre-trained models from TensorFlow Hub

from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.applications import ResNet50V2
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
```

```
WARNING:tensorflow:From C:\Users\mvenu\anaconda3\Lib\site-packages\tf_keras\src\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy is deprec
ated. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

[2]:
```
import warnings
warnings.filterwarnings("ignore")  # Suppress warnings for cleaner output
```

[3]:
```
# Define the directory where the image classification data is stored

data_dir = "C:\\Users\\mvenu\\OneDrive\\Desktop\\image_classification"  # Use double backslashes for Windows file paths
```

[4]:
```
# Import the pathlib module for working with file paths in an object-oriented way
import pathlib

# Convert the string path into a Path object for easier manipulation and compatibility
data_dir = pathlib.Path(data_dir)

# Output the Path object to check or confirm the path
data_dir
```

[4]:
```
WindowsPath('C:/Users/mvenu/OneDrive/Desktop/image_classification')
```

[5]:
```
# List all the files and directories in the specified path (data_dir)

os.listdir(data_dir)  # Returns a list of the names of files and directories in the 'data_dir' folder
```

[5]:
```
['Kane Williamson',
 'Kobe Bryant',
 'lionel_messi',
 'Maria Sharapova',
 'ms_dhoni',
 'neeraj_chopra']
```

[6]:
```
list(data_dir.glob('*/*.jpg'))[:5] # List all .jpg files in subdirectories of 'data_dir'
                                   #[:5]   Display the first 5 .jpg files found
```

[6]:
```
[WindowsPath('C:/Users/mvenu/OneDrive/Desktop/image_classification/Kane Williamson/images (1).jpg'),
 WindowsPath('C:/Users/mvenu/OneDrive/Desktop/image_classification/Kane Williamson/images (10).jpg'),
 WindowsPath('C:/Users/mvenu/OneDrive/Desktop/image_classification/Kane Williamson/images (100).jpg'),
 WindowsPath('C:/Users/mvenu/OneDrive/Desktop/image_classification/Kane Williamson/images (101).jpg'),
 WindowsPath('C:/Users/mvenu/OneDrive/Desktop/image_classification/Kane Williamson/images (102).jpg')]
```

[7]:
```
image_count = len(list(data_dir.glob('*/*.jpg')))  # Count the number of .jpg files in subdirectories of 'data_dir'
print(image_count)  # Print the total count of .jpg files
```

```
608
```

[8]:
```
kane = list(data_dir.glob('Kane Williamson/*'))  # List all files in the 'Kane Williamson' subdirectory of 'data_dir'
kane[:5]  # Display the first 5 files found in the 'Kane Williamson' directory
```

[8]:
```
[WindowsPath('C:/Users/mvenu/OneDrive/Desktop/image_classification/Kane Williamson/images (1).jpg'),
 WindowsPath('C:/Users/mvenu/OneDrive/Desktop/image_classification/Kane Williamson/images (10).jpg'),
 WindowsPath('C:/Users/mvenu/OneDrive/Desktop/image_classification/Kane Williamson/images (100).jpg'),
 WindowsPath('C:/Users/mvenu/OneDrive/Desktop/image_classification/Kane Williamson/images (101).jpg'),
 WindowsPath('C:/Users/mvenu/OneDrive/Desktop/image_classification/Kane Williamson/images (102).jpg')]
```

[9]:
```
PIL.Image.open(str(kane[0]))  # Open the first image file from the 'Kane Williamson' directory using PIL
```

[9]:

[10]:
```
Kobe_Bryant = list(data_dir.glob('Kobe Bryant/*'))  # List all files in the 'Kobe Bryant' subdirectory of 'data_dir'
PIL.Image.open(str(Kobe_Bryant[5]))  # Open the 6th image file from the 'Kobe Bryant' directory using PIL (index starts from 0)
```

[10]:

[11]:
```
lionel_messi = list(data_dir.glob('lionel_messi/*'))  # List all files in the 'lionel_messi' subdirectory of 'data_dir'
PIL.Image.open(str(lionel_messi[5]))  # Open the 6th image file from the 'lionel_messi' directory using PIL (index starts from 0)
```

[11]:

[12]:
```
Maria_Sharapova = list(data_dir.glob('Maria Sharapova/*'))  # List all files in the 'Maria Sharapova' subdirectory of 'data_dir'
PIL.Image.open(str(Maria_Sharapova[5]))  # Open the 6th image file from the 'Maria Sharapova' directory using PIL (index starts from 0)
```

[12]:

[13]:
```
ms_dhoni = list(data_dir.glob('ms_dhoni/*'))  # List all files in the 'ms_dhoni' subdirectory of 'data_dir'
PIL.Image.open(str(ms_dhoni[5]))  # Open the 6th image file from the 'ms_dhoni' directory using PIL (index starts from 0)
```

[13]:

[14]:
```
# Create a dictionary to store lists of image files for different sports celebrities
sports_celebrity_images_dict = {
    'kane': list(data_dir.glob('Kane Williamson/*')),  # List all files in the 'Kane Williamson' subdirectory
    'Kobe_Bryant': list(data_dir.glob('Kobe Bryant/*')),  # List all files in the 'Kobe Bryant' subdirectory
    'lionel_messi': list(data_dir.glob('lionel_messi/*')),  # List all files in the 'lionel_messi' subdirectory
    'Maria_Sharapova': list(data_dir.glob('Maria Sharapova/*')),  # List all files in the 'Maria Sharapova' subdirectory
    'ms_dhoni': list(data_dir.glob('ms_dhoni/*')),  # List all files in the 'ms_dhoni' subdirectory
    'neeraj_chopra': list(data_dir.glob('neeraj_chopra/*')),  # List all files in the 'neeraj_chopra' subdirectory
}
```

[15]:
```
# Create a dictionary to map sports celebrity names to numerical labels
sports_celebrity_labels_dict = {
    'kane': 0,  # Label 0 for 'Kane Williamson'
    'Kobe_Bryant': 1,  # Label 1 for 'Kobe Bryant'
    'lionel_messi': 2,  # Label 2 for 'Lionel Messi'
    'Maria_Sharapova': 3,  # Label 3 for 'Maria Sharapova'
    'ms_dhoni': 4,  # Label 4 for 'MS Dhoni'
    'neeraj_chopra': 5  # Label 5 for 'Neeraj Chopra'
}
```

[16]:
```
sports_celebrity_images_dict['kane'][:5]  # Display the first 5 image files for 'Kane Williamson'
```

[16]:
```
[WindowsPath('C:/Users/mvenu/OneDrive/Desktop/image_classification/Kane Williamson/images (1).jpg'),
 WindowsPath('C:/Users/mvenu/OneDrive/Desktop/image_classification/Kane Williamson/images (10).jpg'),
 WindowsPath('C:/Users/mvenu/OneDrive/Desktop/image_classification/Kane Williamson/images (100).jpg'),
 WindowsPath('C:/Users/mvenu/OneDrive/Desktop/image_classification/Kane Williamson/images (101).jpg'),
 WindowsPath('C:/Users/mvenu/OneDrive/Desktop/image_classification/Kane Williamson/images (102).jpg')]
```

[17]:
```
str(sports_celebrity_images_dict['kane'][0])  # Convert the first image path for 'Kane Williamson' to a string
```

[17]:
```
'C:\\Users\\mvenu\\OneDrive\\Desktop\\image_classification\\Kane Williamson\\images (1).jpg'
```

[18]:
```
img = cv2.imread(str(sports_celebrity_images_dict['kane'][0]))  # Read the first image for 'Kane Williamson' using OpenCV
```

[19]:
```
img.shape  # Get the dimensions (height, width, channels) of the loaded image
```

[19]:
```
(275, 183, 3)
```

[20]:
```
cv2.resize(img, (224, 224)).shape  # Resize the image to 224x224 and get its new dimensions
```

[20]:
```
(224, 224, 3)
```

[21]:
```
plt.axis('off')  # Turn off the axis for the image display
plt.imshow(img)  # Display the image using Matplotlib
```

[21]:
```
<matplotlib.image.AxesImage at 0x1d9220e53d0>
```

[22]:
```
X, y = [], []  # Initialize empty lists to store image data (X) and labels (y)

# Loop through each sports celebrity and their associated images
for sports_celebrity_name, images in sports_celebrity_images_dict.items():
    for image in images:
        img = cv2.imread(str(image))  # Read the image from the file path
        resized_img = cv2.resize(img, (224, 224))  # Resize the image to 224x224 pixels
        X.append(resized_img)  # Add the resized image to the 'X' list (features)
        y.append(sports_celebrity_labels_dict[sports_celebrity_name])  # Add the corresponding label to the 'y' list (targets)
```

[23]:
```
X = np.array(X)  # Convert the list of images (X) into a NumPy array
y = np.array(y)  # Convert the list of labels (y) into a NumPy array
```

[24]:
```
# Import train_test_split from sklearn to split the dataset into training and testing sets
from sklearn.model_selection import train_test_split

# Split the data (X) and labels (y) into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)  # Use random_state=0 for reproducibility
```

[25]:
```
# Preprocessing: Scale images by normalizing pixel values to the range [0, 1]
X_train_scaled = X_train / 255  # Normalize the training images by dividing by 255
X_test_scaled = X_test / 255  # Normalize the testing images by dividing by 255
```

[26]:
```
# Define the number of output classes (sports celebrities)
num_classes = 6

# Build the convolutional neural network (CNN) model using Keras Sequential API
cnn_model = Sequential([
  layers.Conv2D(16, 3, padding='same', activation='relu'),  # Convolutional layer with 16 filters, 3x3 kernel, and ReLU activation
  layers.MaxPooling2D(),  # MaxPooling layer to reduce spatial dimensions
  layers.Conv2D(32, 3, padding='same', activation='relu'),  # Convolutional layer with 32 filters, 3x3 kernel, and ReLU activation
  layers.MaxPooling2D(),  # MaxPooling layer to reduce spatial dimensions
  layers.Conv2D(64, 3, padding='same', activation='relu'),  # Convolutional layer with 64 filters, 3x3 kernel, and ReLU activation
  layers.MaxPooling2D(),  # MaxPooling layer to reduce spatial dimensions
  layers.Conv2D(128, 3, padding='same', activation='relu'),  # Convolutional layer with 128 filters, 3x3 kernel, and ReLU activation
  layers.MaxPooling2D(),  # MaxPooling layer to reduce spatial dimensions
  layers.Flatten(),  # Flatten the 3D output to 1D for the dense layer
  layers.Dense(128, activation='relu'),  # Fully connected layer with 128 units and ReLU activation
  layers.Dense(num_classes)  # Output layer with 'num_classes' units (6 in this case)
])

# Compile the CNN model with the Adam optimizer and sparse categorical crossentropy loss
cnn_model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),  # Loss function for multi-class classification
              metrics=['accuracy'])  # Track accuracy during training

# Train the CNN model for 15 epochs using the training data
cnn_model.fit(X_train_scaled, y_train, epochs=15)  # Fit the model to the training data
```

```
Epoch 1/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 8s 242ms/step - accuracy: 0.2059 - loss: 1.8086
Epoch 2/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 3s 216ms/step - accuracy: 0.4646 - loss: 1.4256
Epoch 3/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 3s 217ms/step - accuracy: 0.5220 - loss: 1.2836
Epoch 4/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 3s 227ms/step - accuracy: 0.6010 - loss: 1.1076
Epoch 5/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 4s 234ms/step - accuracy: 0.6049 - loss: 0.9992
Epoch 6/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 4s 239ms/step - accuracy: 0.7532 - loss: 0.6626
Epoch 7/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 4s 275ms/step - accuracy: 0.8719 - loss: 0.3938
Epoch 8/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 4s 230ms/step - accuracy: 0.8934 - loss: 0.2968
Epoch 9/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 4s 235ms/step - accuracy: 0.9559 - loss: 0.1504
Epoch 10/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 4s 231ms/step - accuracy: 0.9828 - loss: 0.0777
Epoch 11/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 3s 224ms/step - accuracy: 0.9897 - loss: 0.0379
Epoch 12/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 3s 227ms/step - accuracy: 0.9672 - loss: 0.0839
Epoch 13/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 4s 256ms/step - accuracy: 1.0000 - loss: 0.0145
Epoch 14/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 4s 244ms/step - accuracy: 0.9932 - loss: 0.0346
Epoch 15/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 4s 240ms/step - accuracy: 0.9931 - loss: 0.0339
```

[26]:
```
<keras.src.callbacks.history.History at 0x1d9285f60c0>
```

[27]:
```
cnn_model.evaluate(X_test_scaled, y_test)  # Evaluate the trained CNN model on the test data
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 69ms/step - accuracy: 0.5144 - loss: 3.4720
```

[27]:
```
[3.1064021587371826, 0.5263158082962036]
```

[28]:
```
y_pred = cnn_model.predict(X_test_scaled)  # Use the CNN model to make predictions on the test data
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 105ms/step
```

[29]:
```
score = tf.nn.softmax(y_pred[0])  # Apply the softmax function to the first prediction to get class probabilities
```

[30]:
```
np.argmax(score)  # Get the index of the highest probability in the score array (predicted class)
```

[30]:
```
5
```

[31]:
```
y_test[0]  # Retrieve the true label for the first test sample
```

[31]:
```
5
```

[32]:
```
# Define a Sequential model for data augmentation using Keras layers
data_augmentation = tf.keras.Sequential(
  [
    layers.RandomFlip("horizontal", input_shape=(224, 224, 3)),  # Randomly flip images horizontally
    layers.RandomRotation(0.2),  # Randomly rotate images by up to 20%
    layers.RandomZoom(0.2),  # Randomly zoom images by up to 20%
    layers.RandomContrast(0.3),  # Randomly adjust contrast of images by up to 30%
  ]
)
```

[33]:
```
# Display the original image (first image in the dataset)
plt.axis('off')  # Turn off axis labels and ticks for better visualization
plt.imshow(X[0])  # Show the first image from the dataset (X)
```

[33]:
```
<matplotlib.image.AxesImage at 0x1d92872ffb0>
```

[34]:
```
# Display the augmented version of the first image in the dataset
plt.axis('off')  # Turn off axis labels and ticks for better visualization
plt.imshow(data_augmentation(X)[0].numpy().astype("uint8"))  # Apply data augmentation and display the first augmented image
```

[34]:
```
<matplotlib.image.AxesImage at 0x1d92872e5d0>
```

[35]:
```
# Define the number of output classes (sports celebrities)
num_classes = 6

# Build a CNN model with data augmentation and a dropout layer to improve generalization
DA_model = Sequential([
  data_augmentation,  # Apply data augmentation to input images
  layers.Conv2D(16, 3, padding='same', activation='relu'),  # Convolutional layer with 16 filters, 3x3 kernel, and ReLU activation
  layers.MaxPooling2D(),  # MaxPooling layer to reduce spatial dimensions
  layers.Conv2D(32, 3, padding='same', activation='relu'),  # Convolutional layer with 32 filters, 3x3 kernel, and ReLU activation
  layers.MaxPooling2D(),  # MaxPooling layer to reduce spatial dimensions
  layers.Conv2D(64, 3, padding='same', activation='relu'),  # Convolutional layer with 64 filters, 3x3 kernel, and ReLU activation
  layers.MaxPooling2D(),  # MaxPooling layer to reduce spatial dimensions
  layers.Conv2D(128, 3, padding='same', activation='relu'),  # Convolutional layer with 128 filters, 3x3 kernel, and ReLU activation
  layers.MaxPooling2D(),  # MaxPooling layer to reduce spatial dimensions
  layers.Dense(128, activation='relu'),  # Fully connected layer with 128 units and ReLU activation
  layers.Dropout(0.3),  # Dropout layer with a dropout rate of 30% to prevent overfitting
  layers.Flatten(),  # Flatten the 3D feature maps into a 1D vector
  layers.Dense(num_classes, activation='softmax')  # Output layer with softmax activation for multi-class classification
])

# Compile the CNN model with the Adam optimizer and sparse categorical crossentropy loss
DA_model.compile(optimizer='adam',
             loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),  # Loss function for multi-class classification
             metrics=['accuracy'])  # Track accuracy during training

# Train the CNN model with the augmented data for 20 epochs
DA_model.fit(X_train_scaled, y_train, epochs=20)  # Fit the model to the training data
```

```
15/15 ━━━━━━━━━━━━━━━━━━━━ 10s 276ms/step - accuracy: 0.1968 - loss: 1.7821
Epoch 2/20
15/15 ━━━━━━━━━━━━━━━━━━━━ 4s 284ms/step - accuracy: 0.3078 - loss: 1.6993
Epoch 3/20
15/15 ━━━━━━━━━━━━━━━━━━━━ 5s 295ms/step - accuracy: 0.3853 - loss: 1.5938
Epoch 4/20
15/15 ━━━━━━━━━━━━━━━━━━━━ 4s 270ms/step - accuracy: 0.4173 - loss: 1.5101
Epoch 5/20
15/15 ━━━━━━━━━━━━━━━━━━━━ 4s 274ms/step - accuracy: 0.4581 - loss: 1.3775
Epoch 6/20
15/15 ━━━━━━━━━━━━━━━━━━━━ 4s 289ms/step - accuracy: 0.5096 - loss: 1.3154
Epoch 7/20
15/15 ━━━━━━━━━━━━━━━━━━━━ 4s 275ms/step - accuracy: 0.5328 - loss: 1.2407
Epoch 8/20
15/15 ━━━━━━━━━━━━━━━━━━━━ 4s 256ms/step - accuracy: 0.5231 - loss: 1.2391
Epoch 9/20
15/15 ━━━━━━━━━━━━━━━━━━━━ 4s 254ms/step - accuracy: 0.5320 - loss: 1.2468
Epoch 10/20
15/15 ━━━━━━━━━━━━━━━━━━━━ 4s 254ms/step - accuracy: 0.5330 - loss: 1.2100
Epoch 11/20
15/15 ━━━━━━━━━━━━━━━━━━━━ 4s 254ms/step - accuracy: 0.5419 - loss: 1.2366
Epoch 12/20
15/15 ━━━━━━━━━━━━━━━━━━━━ 4s 251ms/step - accuracy: 0.5198 - loss: 1.2392
Epoch 13/20
15/15 ━━━━━━━━━━━━━━━━━━━━ 4s 249ms/step - accuracy: 0.5683 - loss: 1.1545
Epoch 14/20
15/15 ━━━━━━━━━━━━━━━━━━━━ 4s 249ms/step - accuracy: 0.5626 - loss: 1.2049
Epoch 15/20
15/15 ━━━━━━━━━━━━━━━━━━━━ 4s 253ms/step - accuracy: 0.5809 - loss: 1.1059
Epoch 16/20
15/15 ━━━━━━━━━━━━━━━━━━━━ 4s 248ms/step - accuracy: 0.5957 - loss: 1.0743
Epoch 17/20
15/15 ━━━━━━━━━━━━━━━━━━━━ 4s 267ms/step - accuracy: 0.6504 - loss: 1.0091
Epoch 18/20
15/15 ━━━━━━━━━━━━━━━━━━━━ 4s 255ms/step - accuracy: 0.6266 - loss: 0.9841
Epoch 19/20
15/15 ━━━━━━━━━━━━━━━━━━━━ 5s 246ms/step - accuracy: 0.6068 - loss: 1.0219
Epoch 20/20
```

[36]:
```
DA_model.evaluate(X_test_scaled, y_test)  # Evaluate the model with data augmentation on the test dataset
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 72ms/step - accuracy: 0.6502 - loss: 1.1507
```

[36]:
```
[1.1239639520645142, 0.6447368264198303]
```

[37]:
```
# Set the image shape for input to the classifier
image_shape = (224, 224)

# Define a Sequential model with a pre-trained MobileNet V2 classifier
classifier = tf_keras.Sequential([
    hub.KerasLayer(
        "https://www.kaggle.com/models/google/mobilenet-v2/TensorFlow2/tf2-preview-classification/4",  # URL for the pre-trained MobileNet V2 model
        input_shape=image_shape + (3,)  # Specify the input shape (224x224x3 for RGB images)
    )
])
```

```
WARNING:tensorflow:From C:\Users\mvenu\anaconda3\Lib\site-packages\tensorflow_hub\resolver.py:120: The name tf.gfile.MakeDirs is deprecated. Please use t
f.io.gfile.makedirs instead.

WARNING:tensorflow:From C:\Users\mvenu\anaconda3\Lib\site-packages\tensorflow_hub\resolver.py:120: The name tf.gfile.MakeDirs is deprecated. Please use t
f.io.gfile.makedirs instead.

WARNING:tensorflow:From C:\Users\mvenu\anaconda3\Lib\site-packages\tensorflow_hub\module_v2.py:126: The name tf.saved_model.load_v2 is deprecated. Please
use tf.compat.v2.saved_model.load instead.

WARNING:tensorflow:From C:\Users\mvenu\anaconda3\Lib\site-packages\tensorflow_hub\module_v2.py:126: The name tf.saved_model.load_v2 is deprecated. Please
use tf.compat.v2.saved_model.load instead.

WARNING:tensorflow:From C:\Users\mvenu\anaconda3\Lib\site-packages\tf_keras\src\backend.py:873: The name tf.get_default_graph is deprecated. Please use t
f.compat.v1.get_default_graph instead.

WARNING:tensorflow:From C:\Users\mvenu\anaconda3\Lib\site-packages\tf_keras\src\backend.py:873: The name tf.get_default_graph is deprecated. Please use t
f.compat.v1.get_default_graph instead.
```

[38]:
```
X[0].shape  # Check the shape of the first image in the dataset (X) to confirm it matches the input requirements
```

[38]:
```
(224, 224, 3)
```

[39]:
```
image_shape + (3,)  # Append the number of color channels (3 for RGB) to the image shape
```

[39]:
```
(224, 224, 3)
```

[40]:
```
# Resize the first image in the dataset to match the expected input shape
x0_resized = cv2.resize(X[0], image_shape)  # Resize X[0] to (224, 224)

# Resize the second image in the dataset to match the expected input shape
x1_resized = cv2.resize(X[1], image_shape)  # Resize X[1] to (224, 224)

# Resize the third image in the dataset to match the expected input shape
x2_resized = cv2.resize(X[2], image_shape)  # Resize X[2] to (224, 224)
```

[41]:
```
plt.axis('off')  # Disable axis labels and ticks for a cleaner image display
plt.imshow(X[0])  # Display the first image in the dataset (X[0]) using Matplotlib
```

[41]:
```
<matplotlib.image.AxesImage at 0x1d976e72630>
```

[42]:
```
# Use the pre-trained classifier to make predictions on the resized images
predicted = classifier.predict(np.array([x0_resized, x1_resized, x2_resized]))

# Get the index of the highest probability for each prediction (corresponding to the predicted class)
predicted = np.argmax(predicted, axis=1)

# Display the predicted class indices
predicted
```

```
1/1 [==============================] - 8s 8s/step
```

[42]:
```
array([722, 795, 795], dtype=int64)
```

[43]:
```
# Load the pre-trained MobileNetV2 model without the top classification layer
base_model = MobileNetV2(weights="imagenet", include_top=False, input_shape=(224, 224, 3))

# Freeze the base model to prevent its weights from being updated during training
base_model.trainable = False
```

[44]:
```
# Set the number of classes for the classification task
num_of_sports_celebrity = 6  # Number of unique sports celebrities (6 classes)

# Define the model architecture using the pre-trained base model
mobilenet_model = Sequential([
    base_model,  # Add the frozen MobileNetV2 base model
    GlobalAveragePooling2D(),  # Add global average pooling to reduce dimensions
    Dense(num_of_sports_celebrity)  # Output layer with logits (no activation applied yet)
])

# Display a summary of the model architecture
mobilenet_model.summary()  # Provides details about each layer, parameters, and total trainable parameters
```

Model: "sequential_3"

Layer (type)	Output Shape	Param #
mobilenetv2_1.00_224 (Functional)	(None, 7, 7, 1280)	2,257,984
global_average_pooling2d (GlobalAveragePooling2D)	(None, 1280)	0
dense_4 (Dense)	(None, 6)	7,686

Total params: 2,265,670 (8.64 MB)
Trainable params: 7,686 (30.02 KB)
Non-trainable params: 2,257,984 (8.61 MB)

[45]:
```
# Compile the MobileNetV2 model with the Adam optimizer and Sparse Categorical Crossentropy loss function
mobilenet_model.compile(
    optimizer="adam",  # Optimizer for model training
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),  # Loss function for multi-class classification
    metrics=['accuracy']  # Metrics to track during training (accuracy)
)

# Preprocess the training and test data using MobileNetV2's preprocessing function
X_train_scaled = preprocess_input(X_train)  # Preprocess the training images
X_test_scaled = preprocess_input(X_test)  # Preprocess the test images

# Train the model with the preprocessed data for 15 epochs
mobilenet_model.fit(X_train_scaled, y_train, epochs=15)  # Train the model with training data
```

```
Epoch 1/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 13s 383ms/step - accuracy: 0.2247 - loss: 1.9183
Epoch 2/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 6s 377ms/step - accuracy: 0.5019 - loss: 1.2813
Epoch 3/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 6s 374ms/step - accuracy: 0.7266 - loss: 0.9145
Epoch 4/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 6s 387ms/step - accuracy: 0.8183 - loss: 0.7154
Epoch 5/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 6s 384ms/step - accuracy: 0.8824 - loss: 0.5650
Epoch 6/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 6s 375ms/step - accuracy: 0.9267 - loss: 0.4531
Epoch 7/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 6s 373ms/step - accuracy: 0.9233 - loss: 0.4087
Epoch 8/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 6s 383ms/step - accuracy: 0.9466 - loss: 0.3465
Epoch 9/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 6s 400ms/step - accuracy: 0.9701 - loss: 0.3183
Epoch 10/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 6s 377ms/step - accuracy: 0.9731 - loss: 0.2800
Epoch 11/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 6s 371ms/step - accuracy: 0.9693 - loss: 0.2682
Epoch 12/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 6s 374ms/step - accuracy: 0.9765 - loss: 0.2110
Epoch 13/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 10s 377ms/step - accuracy: 0.9944 - loss: 0.1975
Epoch 14/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 6s 393ms/step - accuracy: 0.9953 - loss: 0.1897
Epoch 15/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 6s 396ms/step - accuracy: 0.9989 - loss: 0.1723
```

[45]:
```
<keras.src.callbacks.history.History at 0x1d97761cd40>
```

[46]:
```
# Evaluate the trained MobileNetV2 model on the test data to check its performance
mobilenet_model.evaluate(X_test_scaled, y_test)  # Returns loss and accuracy on the test set
```

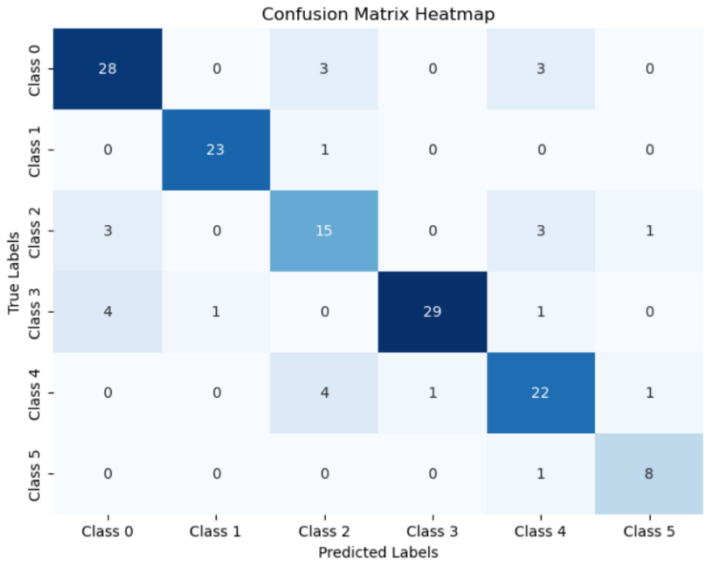
```
5/5 ━━━━━━━━━━━━━━━━━━━━ 3s 365ms/step - accuracy: 0.8214 - loss: 0.6144
```

[46]:
```
[0.5648172497749329, 0.8223684430122375]
```

[47]:
```
y_pred = mobilenet_model.predict(X_test_scaled)  # Predict class probabilities for the test data
y_pred = [np.argmax(i) for i in y_pred]  # Convert probabilities to class labels by selecting the highest probability
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 5s 673ms/step
```

[48]:
```
import seaborn as sns  # Import seaborn for visualizing data
from sklearn.metrics import confusion_matrix, classification_report  # Import evaluation metrics
# Generate the confusion matrix
cm = confusion_matrix(y_test, y_pred)  # Compute confusion matrix

# Create a heatmap for the confusion matrix
plt.figure(figsize=(8, 6))  # Set the figure size
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=[f'Class {i}' for i in range(len(cm))],
            yticklabels=[f'Class {i}' for i in range(len(cm))], cbar=False)  # Create the heatmap
plt.title('Confusion Matrix Heatmap')  # Add title to the heatmap
plt.xlabel('Predicted Labels')  # X-axis label
plt.ylabel('True Labels')  # Y-axis label
plt.show()  # Display the heatmap
```



[49]:
```
print(classification_report(y_test, y_pred))  # Print precision, recall, F1-score, and support for each class
```

```
              precision    recall  f1-score   support

           0       0.80      0.82      0.81        34
           1       0.96      0.96      0.96        24
           2       0.65      0.68      0.67        22
           3       0.97      0.83      0.89        35
           4       0.73      0.79      0.76        28
           5       0.80      0.89      0.84         9

    accuracy                           0.82       152
   macro avg       0.82      0.83      0.82       152
weighted avg       0.83      0.82      0.82       152
```

[50]:
```
# Load the ResNet50V2 model pre-trained on ImageNet without the top classification layer
base_model = ResNet50V2(weights="imagenet", include_top=False, input_shape=(224, 224, 3))

# Freeze the base model layers to retain pre-trained weights during transfer learning
base_model.trainable = False
```

[51]:
```
# Define a new model with ResNet50V2 as the base and additional classification layers
num_of_sports_celebrity = 6  # Specify the number of classes (6 celebrities)

# Build the model architecture
resnet_model = Sequential([
    base_model,  # Add the pre-trained ResNet50V2 base model
    GlobalAveragePooling2D(),  # Add global average pooling to reduce dimensions
    Dense(num_of_sports_celebrity)  # Add a dense layer for class logits (no activation)
])

# Display the model summary to review its structure
resnet_model.summary()
```

Model: "sequential_4"

Layer (type)	Output Shape	Param #
resnet50v2 (Functional)	(None, 7, 7, 2048)	23,564,800
global_average_pooling2d_1 (GlobalAveragePooling2D)	(None, 2048)	0
dense_5 (Dense)	(None, 6)	12,294

Total params: 23,577,094 (89.94 MB)
Trainable params: 12,294 (48.02 KB)
Non-trainable params: 23,564,800 (89.89 MB)

[52]:
```
# Compile the ResNet50V2 model with Adam optimizer, sparse categorical crossentropy loss, and accuracy metric
resnet_model.compile(
    optimizer="adam",  # Use Adam optimizer for efficient training
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),  # Specify loss function for multi-class classification
    metrics=['accuracy']  # Evaluate model performance using accuracy
)

# Preprocess training and testing data with ResNet50V2-specific preprocessing
X_train_scaled = preprocess_input(X_train)  # Apply preprocessing to training data
X_test_scaled = preprocess_input(X_test)  # Apply preprocessing to testing data

# Train the ResNet50V2 model for 15 epochs
resnet_model.fit(X_train_scaled, y_train, epochs=15)  # Fit the model with training data and labels
```

```
Epoch 1/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 24s 1s/step - accuracy: 0.2619 - loss: 1.9951
Epoch 2/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 17s 1s/step - accuracy: 0.6051 - loss: 1.1245
Epoch 3/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 17s 1s/step - accuracy: 0.7791 - loss: 0.7356
Epoch 4/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 20s 1s/step - accuracy: 0.8852 - loss: 0.5256
Epoch 5/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 17s 1s/step - accuracy: 0.9204 - loss: 0.4172
Epoch 6/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 17s 1s/step - accuracy: 0.9516 - loss: 0.3564
Epoch 7/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 17s 1s/step - accuracy: 0.9518 - loss: 0.3127
Epoch 8/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 17s 1s/step - accuracy: 0.9752 - loss: 0.2537
Epoch 9/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 17s 1s/step - accuracy: 0.9828 - loss: 0.2254
Epoch 10/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 18s 1s/step - accuracy: 0.9954 - loss: 0.1872
Epoch 11/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 19s 1s/step - accuracy: 0.9974 - loss: 0.1563
Epoch 12/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 18s 1s/step - accuracy: 0.9980 - loss: 0.1474
Epoch 13/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 21s 1s/step - accuracy: 0.9989 - loss: 0.1347
Epoch 14/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 22s 1s/step - accuracy: 1.0000 - loss: 0.1196
Epoch 15/15
15/15 ━━━━━━━━━━━━━━━━━━━━ 22s 1s/step - accuracy: 1.0000 - loss: 0.1022
```

[52]:
```
<keras.src.callbacks.history.History at 0x1d93cedc800>
```

[53]:
```
# Evaluate the trained ResNet50V2 model on the test data
resnet_model.evaluate(X_test_scaled, y_test)  # Outputs loss and accuracy on the test dataset
```

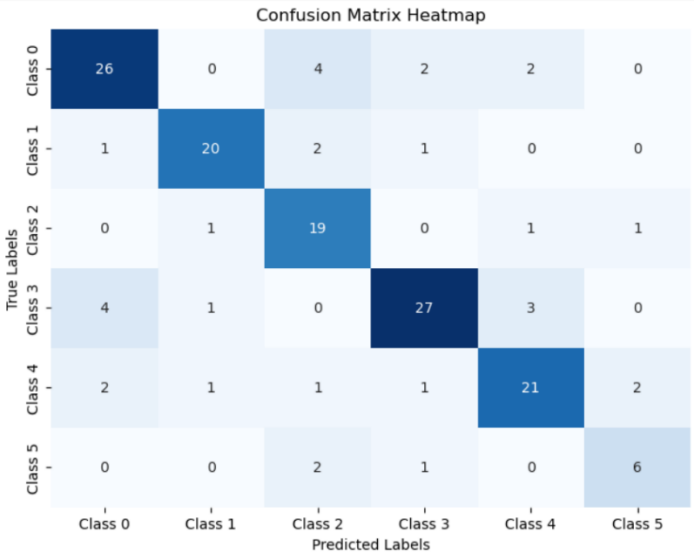
```
5/5 ━━━━━━━━━━━━━━━━━━━━ 9s 1s/step - accuracy: 0.7636 - loss: 0.6234
```

[53]:
```
[0.5683630108833313, 0.7828947305679321]
```

[54]:
```
y_pred = resnet_model.predict(X_test_scaled)  # Predict class probabilities for test data
y_pred = [np.argmax(i) for i in y_pred]  # Convert probabilities to class labels
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 12s 2s/step
```

[55]:
```
import seaborn as sns  # Import seaborn for visualizing data
from sklearn.metrics import confusion_matrix, classification_report  # Import evaluation metrics
# Generate the confusion matrix
cm = confusion_matrix(y_test, y_pred)  # Compute confusion matrix

# Create a heatmap for the confusion matrix
plt.figure(figsize=(8, 6))  # Set the figure size
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=[f'Class {i}' for i in range(len(cm))],
            yticklabels=[f'Class {i}' for i in range(len(cm))], cbar=False)  # Create the heatmap
plt.title('Confusion Matrix Heatmap')  # Add title to the heatmap
plt.xlabel('Predicted Labels')  # X-axis label
plt.ylabel('True Labels')  # Y-axis label
plt.show()  # Display the heatmap
```



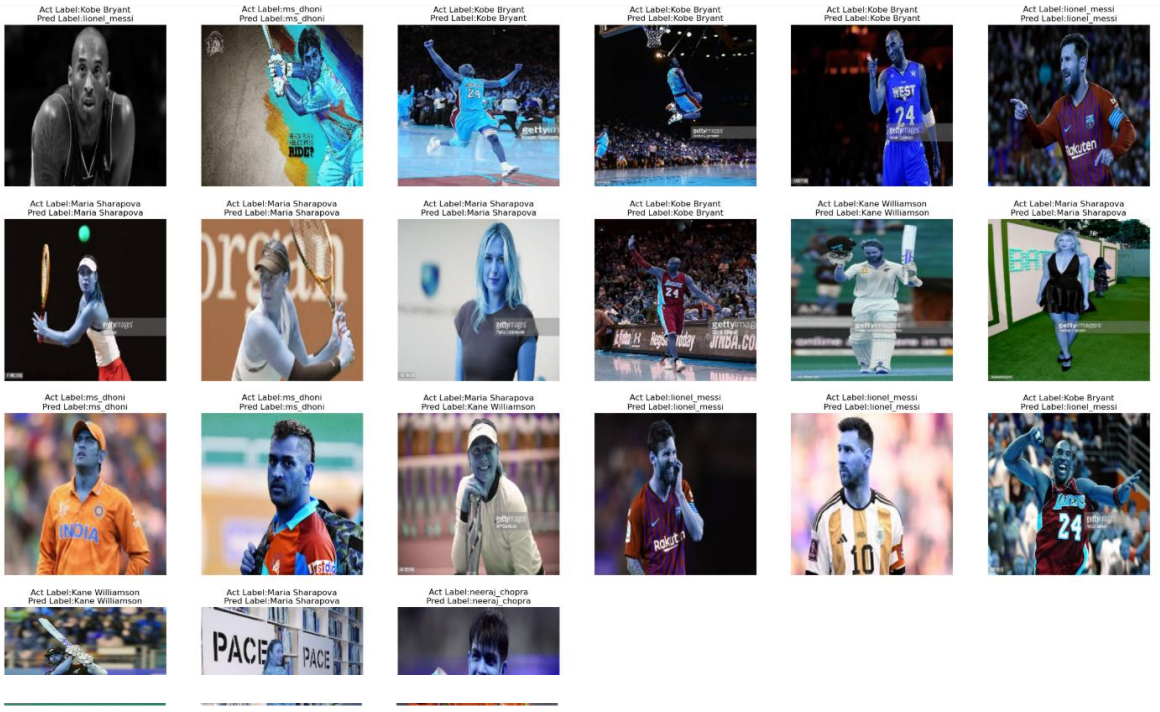
[56]:
```
print(classification_report(y_test, y_pred))  # Print precision, recall, F1-score, and support for each class
```

```
              precision    recall  f1-score   support

           0       0.79      0.76      0.78        34
           1       0.87      0.83      0.85        24
           2       0.68      0.86      0.76        22
           3       0.84      0.77      0.81        35
           4       0.78      0.75      0.76        28
           5       0.67      0.67      0.67         9

    accuracy                           0.78       152
   macro avg       0.77      0.77      0.77       152
weighted avg       0.79      0.78      0.78       152
```

[57]:
```
# Define the labels for the classes
labels = ["Kane Williamson", "Kobe Bryant", "lionel_messi", "Maria Sharapova", "ms_dhoni", "neeraj_chopra"]  # Class names
```

[58]:
```
# Plot images from the training set with their corresponding labels
plt.figure(figsize=(30, 50))  # Set the figure size for the grid of images

# Loop through the first 60 images in the training dataset
for i in range(60):
    plt.subplot(10, 6, i + 1)  # Create a subplot with 10 rows and 6 columns
    plt.imshow(X_train[i])  # Display the image
    plt.title(f"Label:{labels[y_train[i]]}")  # Set the title to display the corresponding label
    plt.axis("off")  # Turn off the axis for cleaner visualization
```

[59]:
```
# Plot images from the test set along with actual and predicted labels
plt.figure(figsize=(30, 40))  # Set the figure size for the grid of images

# Loop through the first 45 images in the test dataset
for i in range(45):
    plt.subplot(8, 6, i + 1)  # Create a subplot with 8 rows and 6 columns
    plt.imshow(X_test[i])  # Display the image
    plt.title(f"Act Label:{labels[y_test[i]]}\nPred Label:{labels[y_pred[i]]}")  # Display actual and predicted labels
    plt.axis("off")  # Turn off the axis for cleaner visualization
```



[60]:
```
DA_model.save('app.h5')
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
```

[ ]:

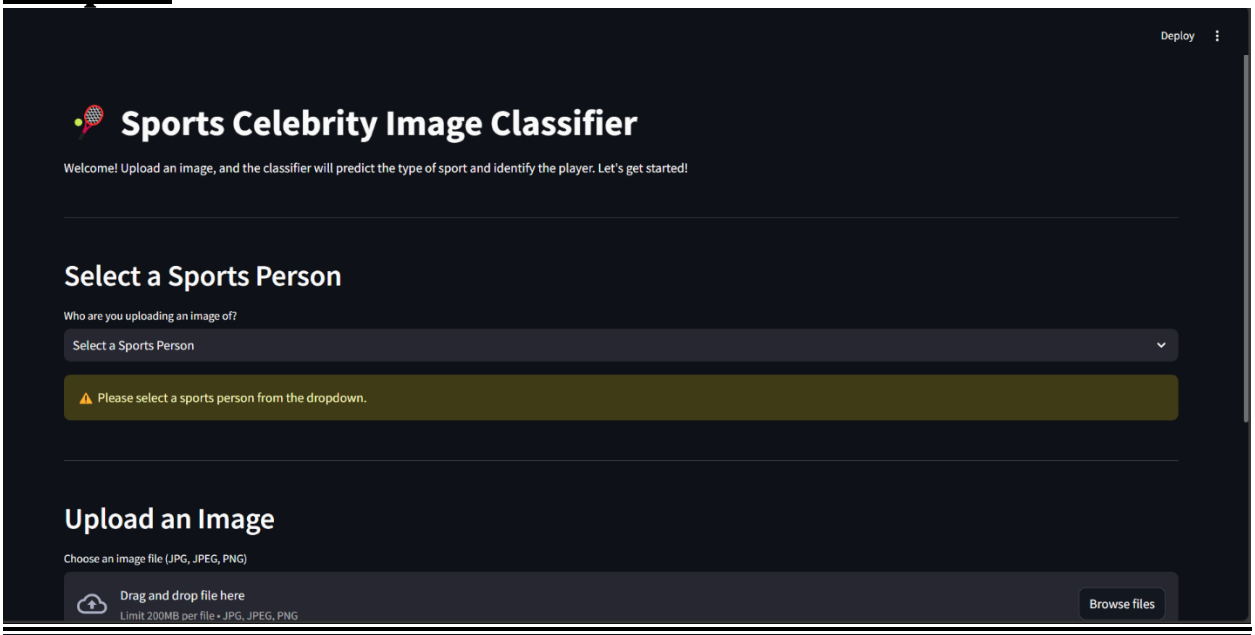
Streamlit App Code:

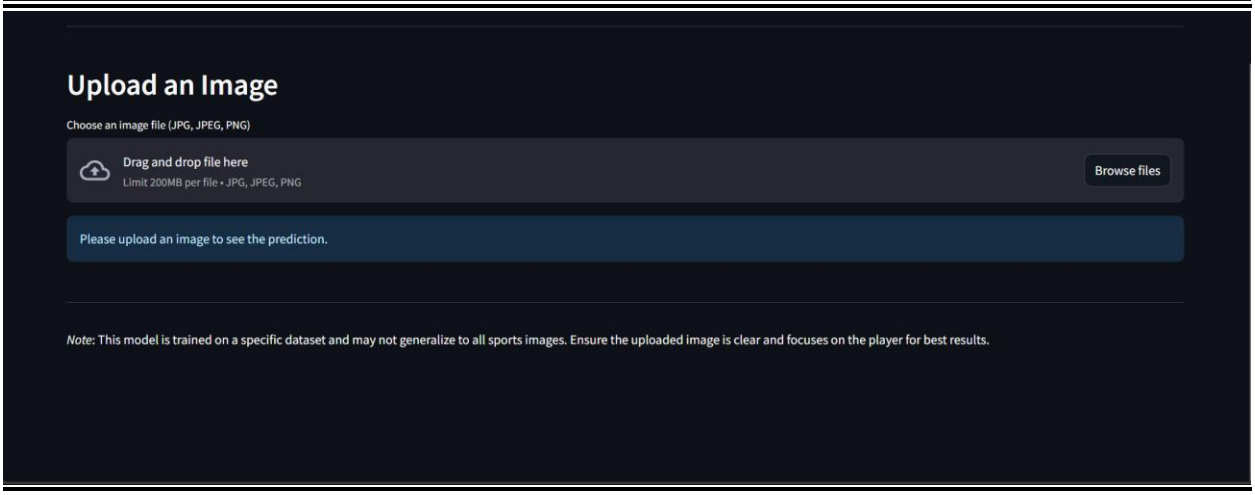
```python
import streamlit as st
from PIL import Image
import numpy as np
import tensorflow_hub as hub
from tensorflow.keras.models import load_model
import pandas as pd
import tensorflow as tf

# Load your trained model
@st.cache_resource
def load_saved_model():
    return load_model("app_model.h5")

# Streamlit page configuration
st.set_page_config(page_title="Sports Celebrity Image Classifier", layout="wide", page_icon="🏀")

# Load the model
model = load_saved_model()

# Page Title
st.title("🏈 Sports Celebrity Image Classifier")
st.write("Welcome! Upload an image, and the classifier will predict the type of sport and identify the player. Let's get started!")

# Add spacing
st.divider()

# Dropdown for Sports Persons
st.header("Select a Sports Person")
sports_persons = st.selectbox(
    "Who are you uploading an image of?",
    ["Select a Sports Person", "Kane Williamson", "Kobe Bryant", "Lionel Messi", "Maria Sharapova", "MS Dhoni", "Neeraj Chopra"]
)
if sports_persons == "Select a Sports Person":
    st.warning("⚠️ Please select a sports person from the dropdown.")
else:
    st.write(f"*Selected Sports Person:* {sports_persons}")

```

```python
# Add spacing
st.divider()

# Main Section for Image Upload
st.header("Upload an Image")
uploaded_file = st.file_uploader("Choose an image file (JPG, JPEG, PNG)", type=["jpg", "jpeg", "png"])

if uploaded_file is not None:
    # Display the uploaded image
    image = Image.open(uploaded_file)
    st.image(image, caption="Uploaded Image", width=500)

    # Preprocess the image
    def preprocess_image(image):
        image = image.resize((224, 224))  # Resize to match model input size
        image_array = np.array(image) / 255.0  # Normalize pixel values
        image_array = np.expand_dims(image_array, axis=0)  # Add batch dimension
        return image_array

    preprocessed_image = preprocess_image(image)

    # Prediction
    prediction = model.predict(preprocessed_image)
    confidence = np.max(prediction) * 100

    # Define sport categories and class names
    categories = ["Cricket", "Basketball", "Football", "Tennis", "Cricket", "Javelin Throw"]
    class_names = ["Kane Williamson", "Kobe Bryant", "Lionel Messi", "Maria Sharapova", "MS Dhoni", "Neeraj Chopra"]

    # Get predicted category and class
    predicted_category = categories[np.argmax(prediction)]
    predictions = model.predict(preprocessed_image)
    predicted_class = np.argmax(predictions, axis=1)
    predicted_person = class_names[predicted_class[0]]
    confidence_scores = tf.nn.softmax(predictions[0]).numpy()

```

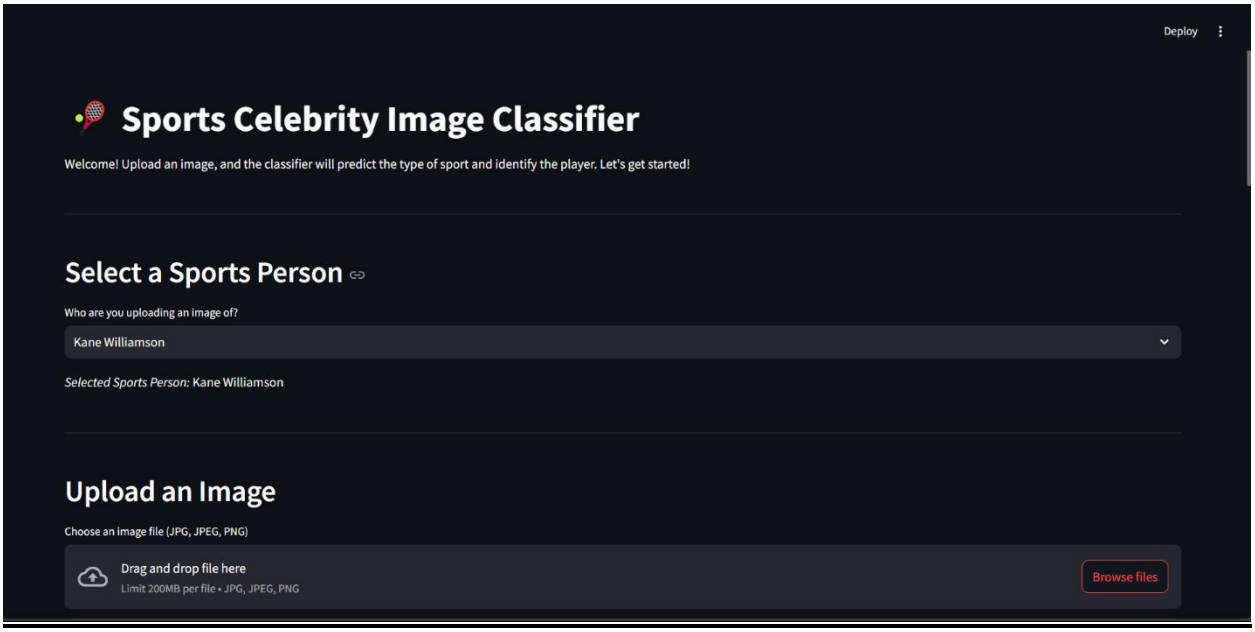
```
    st.divider()

    # Check if the selected person matches the prediction
    if sports_persons != predicted_person:
        st.error(f"⚠️ The uploaded image does not match the selected person ({sports_persons}). The model predicted: {predicted_person}.")
    else:
        # Display Results
        st.header("Prediction Results")
        st.success(f"🏅 *Sport*: {predicted_category}")
        st.info(f"*Predicted Player*: {predicted_person}")

    st.subheader("Data Visualization for Confidence Scores")
    data = pd.DataFrame({
        "Class": class_names,
        "Confidence": confidence_scores
    })

    # Display the bar chart
    st.bar_chart(data.set_index("Class"))

    st.subheader("Confidence Scores")
    for i, score in enumerate(confidence_scores):
        st.write(f"**{class_names[i]}**: {score:.2%}")

    # Provide some feedback based on confidence
    if confidence > 80:
        st.balloons()
        st.write("🎉 Great! The model is very confident in its prediction!")
    else:
        st.warning("The model's confidence is below 80%. You might want to try another image.")
.se:
    st.info("Please upload an image to see the prediction.")

 Add final spacing
:.markdown("---")
:.markdown(
    "*Note*: This model is trained on a specific dataset and may not generalize to all sports images. "
    "Ensure the uploaded image is clear and focuses on the player for best results."
```

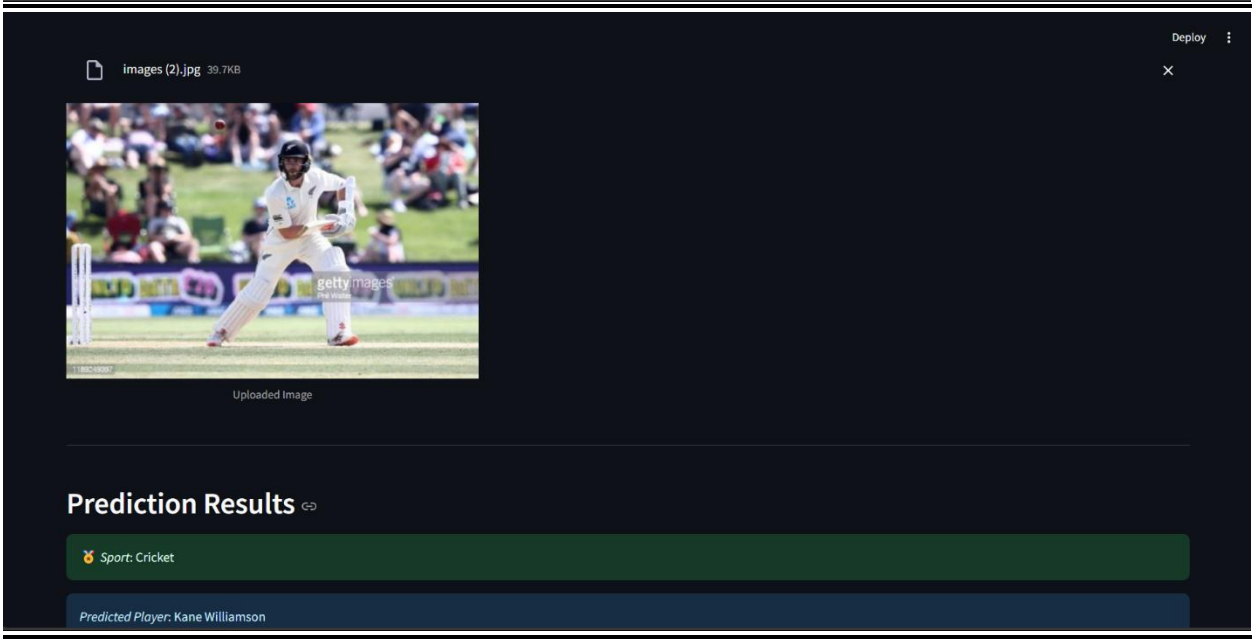
Output:

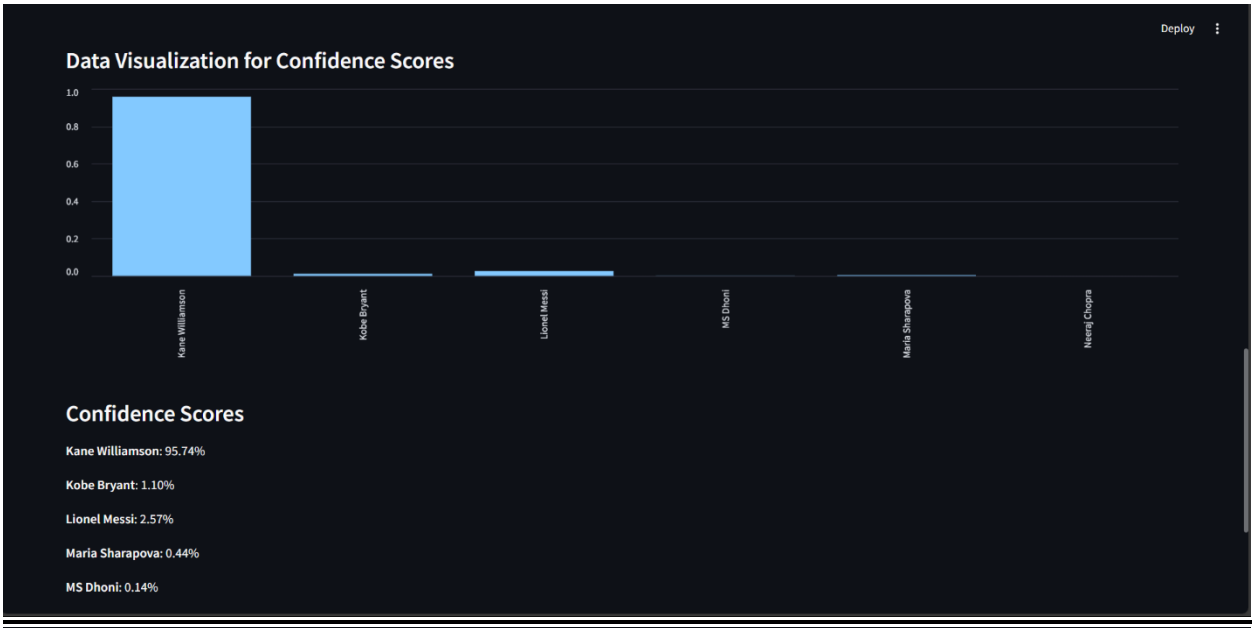

Deploy
Sports Celebrity Image Classifier
Welcome! Upload an image, and the classifier will predict the type of sport and identify the player. Let's get started!
Select a Sports Person
Who are you uploading an image of?
Select a Sports Person
Please select a sports person from the dropdown.
Upload an Image
Choose an image file (JPG, JPEG, PNG)
Drag and drop file here
Limit 200MB per file • JPG, JPEG, PNG
Browse files

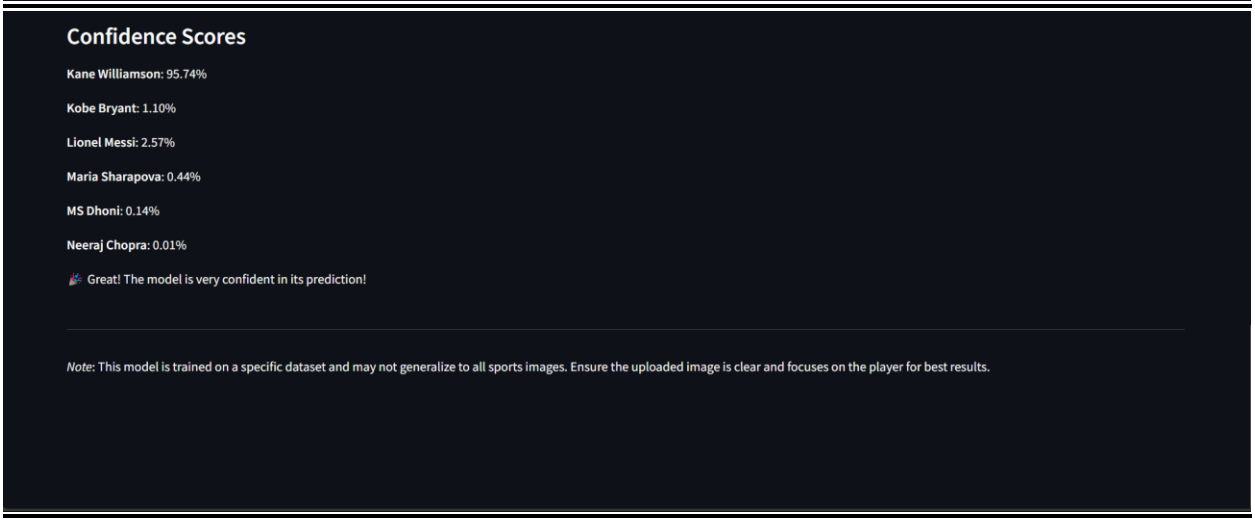

Upload an Image
Choose an image file (JPG, JPEG, PNG)
Drag and drop file here
Limit 200MB per file • JPG, JPEG, PNG
Browse files
Please upload an image to see the prediction.
Note: This model is trained on a specific dataset and may not generalize to all sports images. Ensure the uploaded image is clear and focuses on the player for best results.


Select a Sports Person
Who are you uploading an image of?
Select a Sports Person
Select a Sports Person
Kane Williamson
Kobe Bryant
Lionel Messi
Maria Sharapova
MS Dhoni
Neeraj Chopra


Deploy
Sports Celebrity Image Classifier
Welcome! Upload an image, and the classifier will predict the type of sport and identify the player. Let's get started!
Select a Sports Person
Who are you uploading an image of?
Kane Williamson
Selected Sports Person: Kane Williamson
Upload an Image
Choose an image file (JPG, JPEG, PNG)
Drag and drop file here
Limit 200MB per file • JPG, JPEG, PNG
Browse files


Deploy
images (2).jpg 39.7KB
Uploaded Image
Prediction Results
Sport: Cricket
Predicted Player: Kane Williamson


Deploy
Data Visualization for Confidence Scores
1.0
0.8
0.6
0.4
0.2
0.0
Kane Williamson
Kobe Bryant
Lionel Messi
MS Dhoni
Maria Sharapova
Neeraj Chopra
Confidence Scores
Kane Williamson: 95.74%
Kobe Bryant: 1.10%
Lionel Messi: 2.57%
Maria Sharapova: 0.44%
MS Dhoni: 0.14%


Confidence Scores
Kane Williamson: 95.74%
Kobe Bryant: 1.10%
Lionel Messi: 2.57%
Maria Sharapova: 0.44%
MS Dhoni: 0.14%
Neeraj Chopra: 0.01%
Great! The model is very confident in its prediction!
Note: This model is trained on a specific dataset and may not generalize to all sports images. Ensure the uploaded image is clear and focuses on the player for best results.