Project Report

Sports Image Detection

Applications of Artificial Intelligence

Course Instructor

Dr. Shahela Saif

Mahad Sheikh **21I-1239**

Omer Hashmi **21I-1105**

Section

SE-P

Spring 2024

Department of Software Engineering

FAST – National University of Computer and Emerging Science

Table of Contents

Introduction:

We have chosen Image classification for this semester project moving forward. The model receives an image input and returns an output to classify the image into 1 of 7 distinct classes. This project has developed an image classification system using yolov8n-cls model to categorize digital images into predefined classes. The model classifies images of sports into 7 sports classes which are:

- Badminton
- Cricket
- Karate
- Soccer
- Swimming
- Tennis
- Wrestling

The focus will be on exploring the effectiveness of this architecture for accurate classification.

Data Source:

The Data source for training the model was selected from Kaggle. The link to the public data source is provided as follows:

https://www.kaggle.com/datasets/sidharkal/sports-image-classification

Technologies:

- Ultralytics YOLOv8
- Python 3.0
- React / JavaScript
- GitHub

Development Process:

Data Collection and Preparation for Image Classification:

We'll demonstrate how to prepare image data for a classification task using Python and Pandas. Specifically, we'll:

1. Load image data from CSV files.

2. Split the data into train, validation, and test sets.

3. Organize the data into directories suitable for training a machine learning model.

Let's get started by loading the necessary libraries and reading the CSV files containing image paths and corresponding labels. We'll then split the data and organize it into appropriate directories. Finally, we'll copy the images to their respective directories, readying them for model training. The dataset was arranged in the following hierarchy:

Dataset
|- train
  |- classes
    |- (*names of the classes*)

Code Details and Explanation:

Visualizing Training Results in Colab

We'll demonstrate how to visualize training results, specifically loss and validation accuracy, using matplotlib in Google Colab.

We first load the training results from a CSV file, typically generated during model training. Then, we plot the training loss and validation loss against epochs to understand how the model's performance evolves over time. Additionally, we plot the validation accuracy against epochs to observe the model's learning progress.

These visualizations provide insights into the training process, helping to assess the model's performance and identify potential areas for improvement. By analyzing these plots, researchers and practitioners can make informed decisions about model optimization and tuning strategies.

```
results_path = './runs/classify/train9/results.csv'

results = pd.read_csv(results_path)

plt.figure()
plt.plot(results['                epoch'], results['            train/loss'], label='train loss')
plt.plot(results['                epoch'], results['              val/loss'], label='val loss', c='red')
plt.grid()
plt.title('Loss vs epochs')
plt.ylabel('loss')
plt.xlabel('epochs')
plt.legend()


plt.figure()
plt.plot(results['                epoch'], results['  metrics/accuracy_top1'] * 100)
plt.grid()
plt.title('Validation accuracy vs epochs')
plt.ylabel('accuracy (%)')
plt.xlabel('epochs')

plt.show()
```

Object Detection API Backend with Flask and Ultralytics YOLO

This code sets up a Flask backend for an object detection API using the Ultralytics YOLO model. The purpose is to create a RESTful API endpoint (/predict) that accepts POST requests containing images, processes them using the YOLO model, and returns the predicted class based on the highest confidence score.

The backend is built using Flask, a lightweight web application framework for Python, and leverages the Ultralytics YOLO model for object detection tasks. Additionally, the Flask-CORS extension is used to enable cross-origin resource sharing, allowing requests from different origins.

The /predict endpoint handles incoming image files, checks their format, converts them to the required RGB format if necessary, and then performs object detection using the YOLO model. Finally, it extracts the prediction results and returns the predicted class along with its confidence score as a JSON response.

This backend provides a simple yet powerful solution for integrating object detection capabilities into web applications, allowing developers to easily deploy and serve machine learning models for real- world applications.

```python
app = Flask(__name__)
CORS(app)
model = YOLO('./runs/classify/train9/weights/last.pt')

@app.route('/predict', methods=['POST'])
def predict():
    if 'image' not in request.files:
        return jsonify({'error': 'No image provided'})

    file = request.files['image']
    image_bytes = file.read()

    try:
        # Open image using PIL to check format and convert to supported format
        image = Image.open(io.BytesIO(image_bytes))

        # Convert image to RGB format (if not already in RGB)
        if image.mode != 'RGB':
            image = image.convert('RGB')

        # Process the image with the Ultralytics model
        results = model(image)

        # Extract prediction results and return
        names_dict = results[0].names
        probs = results[0].probs.data.tolist()
        predicted_class = names_dict[probs.index(max(probs))]

        return jsonify({'predicted_class': predicted_class})
    except Exception as e:
        return jsonify({'error': str(e)})

if __name__ == '__main__':
    app.run(debug=True)
```

How to Setup:

Backend (Jupyter Notebook)

Clone this repository:

```
git clone https://github.com/your_username/sports-image-classification.git
```

Navigate to the project directory:

```
cd sports-image-classification
```

Open and run the Jupyter Notebook:

```
jupyter notebook Sport_Classifier.ipynb
```

Follow the instructions in the notebook to train and evaluate the classifier.

Frontend (Next.js)

Navigate to the ai-web-app directory:

```
cd ai-web-app
```

Install dependencies:

```
npm install
```

Start the Next.js project:

```
npm run dev
```

This will start the frontend application at http://localhost:3000.

Usage

Jupyter Notebook: Follow the instructions in Sport_Classifier.ipynb to train and evaluate the sports image classifier.

Frontend Application: Access the application by navigating to http://localhost:3000 in your web browser after starting the Next.js project.

Customization

If you want to customize the backend or frontend URLs:

Backend URL: Modify the backend URL in the frontend code located in ai-web-app/components/predictForm.js.

Frontend URL: Modify the port in the backend URL in the Jupyter Notebook if you're hosting the frontend on a different port.

Model Path: Modify the model path according to your model generation, in my case ./runs/classify/train9/weights/last.pt was the one, your code be some other train.

Challenges Faced:

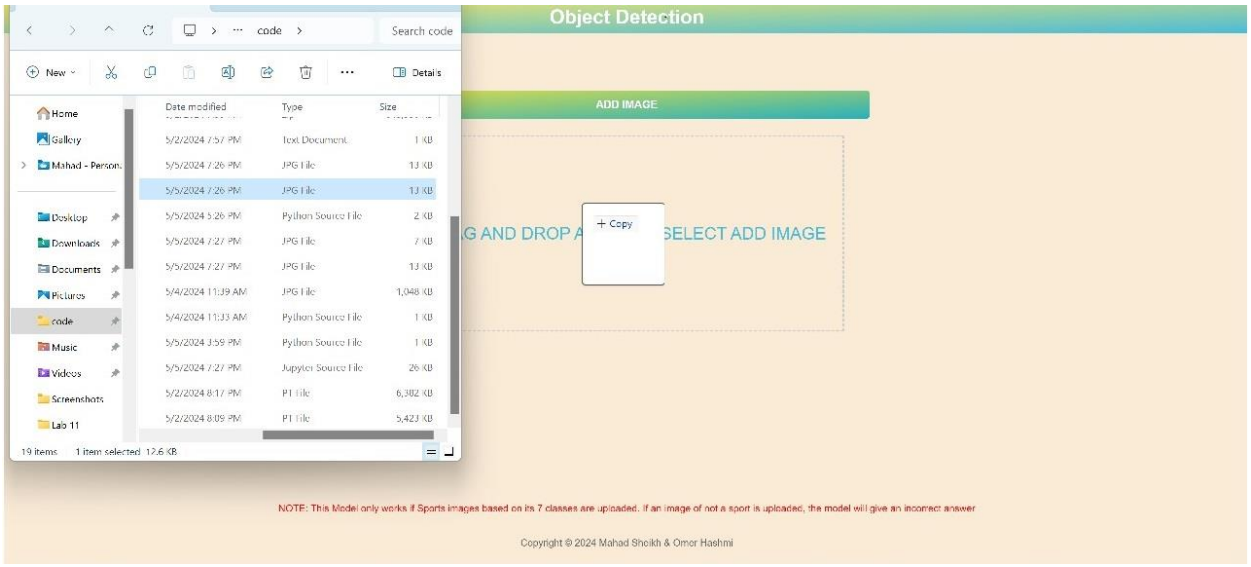
The integration of the model posed initial challenges, particularly in generating it in the .pt format, which was essential for compatibility with the chosen technology stack. However, this format proved difficult to integrate seamlessly with mobile applications, as most apps were designed to work with .tflite files. Despite investing significant time into developing a Flutter mobile app for deployment, we ultimately had to discard this approach due to compatibility issues. Subsequently, we shifted our focus towards implementing a web application, leveraging the flexibility of web technologies. Running the model within a Flask backend proved to be a more straightforward and effective solution, facilitating smoother integration with the frontend and streamlining the deployment process.

User-Interface Design:

The Main front page:

The Second Prediction Page allows the user an area to upload a file that the model shall predict and then display the proposed sport

An example of uploading a file:



An example of file uploaded and a response from the model