app.py

```
from flask import Flask, request, render_template, redirect, url_for
import cv2
import numpy as np
import os
app = Flask(__name__)
# Load the YOLO model
net = cv2.dnn.readNet("yolov3.weights", "yolov3.cfg")
# Define helmet-specific classes
classes = ["helmet","nohelmet"]
# Function to perform object detection for helmets
def detect_helmet(image_path):
    image = cv2.imread(image_path)
    height, width, _ = image.shape
        blob = cv2.dnn.blobFromImage(image, 1/255.0, (416, 416), swapRB=True,
crop=False)
    net.setInput(blob)
    output_layers_names = net.getUnconnectedOutLayersNames()
    outputs = net.forward(output_layers_names)
    boxes = []
    confidences = []
    class_ids = []
    for output in outputs:
        for detection in output:
            scores = detection[5:]
            class_id = np.argmax(scores)
```

```
            confidence = scores[class_id]
            if confidence > 0.5 and class_id < len(classes):
                center_x = int(detection[0] * width)
                center_y = int(detection[1] * height)
                w = int(detection[2] * width)
                h = int(detection[3] * height)
                x = int(center_x - w / 2)
                y = int(center_y - h / 2)
                boxes.append([x, y, w, h])
                confidences.append(float(confidence))
                class_ids.append(class_id)
        indexes = cv2.dnn.NMSBoxes(boxes, confidences, score_threshold=0.5, nms_threshold=0.4)
    # Draw bounding boxes and add labels
    for i in indexes.flatten():
        x, y, w, h = boxes[i]
        label = str(classes[class_ids[i]])
        confidence = confidences[i]
        color = (0, 255, 0) if label == "helmet" else (0, 0, 255)
        text = f"{label}: {confidence:.2f}"
        cv2.rectangle(image, (x, y), (x + w, y + h), color, 2)
        cv2.putText(image, text, (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)
    output_image_path = "static/detected_image.jpg"
    cv2.imwrite(output_image_path, image)
    return output_image_path
```

```
@app.route('/', methods=['GET', 'POST'])
def index():
    if request.method == 'POST':
        if 'file' not in request.files:
            return redirect(request.url)
        file = request.files['file']
        if file.filename == '':
            return redirect(request.url)
         file_path = os.path.join("static", file.filename)
        file.save(file_path)
        # Perform helmet detection
        output_image = detect_helmet(file_path)
        return render_template('index.html', output_image=output_image)
    return render_template('index.html', output_image=None)
if __name__ == "__main__":
    app.run(debug=True)
```

Desctiption For Code

1. Importing Necessary Libraries

The application imports libraries:

- Flask for web application functionality.
- cv2 (OpenCV) for image processing and object detection.
- numpy for numerical computations.
- os for file handling.

---

2. YOLO Model Initialization

```
net = cv2.dnn.readNet("yolov3.weights", "yolov3.cfg")
classes = ["helmet","nohelmet"]
```

- The YOLO model is loaded using pre-trained weights (yolov3.weights) and its configuration file (yolov3.cfg).
- The classes "helmet" and "nohelmet" are defined, indicating the model is trained to classify objects as either wearing a helmet or not.

---

3. Function for Helmet Detection

def detect_helmet(image_path):

This function performs the main task of helmet detection:

Load the Input Image:

```
image = cv2.imread(image_path)
height, width, _ = image.shape
```

The image is read and its dimensions are extracted.

Preprocessing for YOLO:

```
blob = cv2.dnn.blobFromImage(image, 1/255.0, (416, 416), swapRB=True, crop=False)
```

The image is converted into a blob to feed into YOLO:

- Normalized pixel values.
- Resized to 416x416, as required by YOLO.

Forward Pass Through the Network:

```
net.setInput(blob)
output_layers_names = net.getUnconnectedOutLayersNames()
outputs = net.forward(output_layers_names)
```

The blob is passed to the network, and predictions are extracted from the output layers.

Extract Bounding Boxes, Class IDs, and Confidence Scores:

```
for detection in output:
    scores = detection[5:]
    class_id = np.argmax(scores)
    confidence = scores[class_id]
    if confidence > 0.5:
        ...
```

- Scores for each class are evaluated.
- Only detections with confidence > 50% are considered.

- The bounding box dimensions (x, y, w, h) are calculated and stored.

Non-Maximum Suppression (NMS):

indexes = cv2.dnn.NMSBoxes(boxes, confidences, score_threshold=0.5, nms_threshold=0.4)

Overlapping boxes are filtered to retain only the most relevant detections.

Annotating the Image:

```
for i in indexes.flatten():
    x, y, w, h = boxes[i]
    label = str(classes[class_ids[i]])
    color = (0, 255, 0) if label == "helmet" else (0, 0, 255)
    cv2.rectangle(image, (x, y), (x + w, y + h), color, 2)
    cv2.putText(image, text, (x, y - 10), ...)
```

Detected objects are annotated with:

- Green rectangles for helmets.
- Red rectangles for "nohelmet".
- Confidence scores as text.

Save Annotated Image:

```
output_image_path = "static/detected_image.jpg"
cv2.imwrite(output_image_path, image)
```

The processed image is saved in the static folder.

4. Flask Application Routes

Home Route (/)

@app.route('/', methods=['GET', 'POST'])

- **GET Request:** Renders an HTML page (index.html) with an option to upload an image.
- **POST Request:**

1. Validates if a file is uploaded.
2. Saves the uploaded file in the static folder.
3. Calls detect_helmet to process the image.
4. Returns the processed image to the frontend.

Serving the App

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

Runs the Flask app in debug mode for development.

---

5. Expected Workflow

1. User uploads an image via the web interface.
2. The server processes the image using YOLO to detect helmets.
3. The processed image with detections is displayed on the webpage.

Key Files Required

- **YOLO Files:**
    - yolov3.weights (model weights).
    - yolov3.cfg (model configuration).
- **Frontend:**
    - index.html template to handle the upload form and display the results.