A blue and white logo

AI-generated content may be incorrect.

HIGH PERFORMANCE COMPUTING-07(SPRING 2025)

DATA SCIENCE APPLICATION: YOLOv8 OBJECT DETECTION

PROF.ADAM DOYLE

Group Members:

1.Haritha Thipparapu

2.Srija velumula

3.Snigdha Yalam

**YOLOv8 OBJECT DETECTION APPLICATION**

**CODE DOCUMENTATION**

**Overview**

This documentation provides a comprehensive explanation of the YOLOv8 Object Detection application, a Streamlit-based web application that leverages computer vision for real-time object detection. The application allows users to detect and classify objects either through a webcam feed or by uploading images.

**Application Structure**

The application is structured as a single Python script that integrates several key components:

1. **Setup and Initialization**: Configures the Streamlit interface and initializes dependencies
2. **Model Loading**: Loads the selected YOLOv8 model with caching for performance
3. **Object Detection Logic**: Processes images and extracts object information
4. **User Interface Components**: Organizes the app into image upload and webcam sections
5. **Results Visualization**: Displays detection results with bounding boxes and statistics

**Dependencies**

The application relies on several Python libraries:

import streamlit as st # Web application framework

import cv2 # OpenCV for image processing

import tempfile # For temporary file handling

import time # For time-related operations

import numpy as np # For numerical operations

from PIL import Image # For image handling

from ultralytics import YOLO # YOLOv8 implementation

from collections import Counter # For counting detected objects

import pandas as pd

# YOLOv8 Object Detection: Technical Implementation Guide

This technical guide explains the underlying principles, data science elements, and implementation details of the YOLOv8 Object Detection application.

## 1. YOLO Architecture Overview

YOLO (You Only Look Once) is a real-time object detection system that processes entire images in a single pass through a neural network, making it extremely efficient compared to multi-stage detection systems.

### Key Components

YOLOv8 consists of:

1. \*Backbone Network\*: CSPDarknet - A feature extraction network using Cross-Stage Partial Network architecture

2. \*Neck\*: PANet (Path Aggregation Network) - Enhances feature fusion across different scales

3. \*Head\*: Decoupled detection heads for classification and bounding box regression

### Detection Process

1. The input image is divided into a grid

2. For each grid cell, the model predicts:

- Bounding box coordinates (x, y, width, height)

- Confidence scores (objectness)

- Class probabilities

### Model Variants Used

| Model | Size (MB) | mAP | Inference Speed (ms) | Best For |

|-------|-----------|-----|----------------------|----------|

| YOLOv8n | 6.3 | 37.3 | 0.99 | Resource-constrained environments |

| YOLOv8s | 22.6 | 44.9 | 1.20 | Balanced performance |

| YOLOv8m | 52.2 | 50.2 | 1.83 | General usage |

| YOLOv8l | 86.1 | 52.9 | 2.39 | High accuracy needs |

| YOLOv8x | 121.1 | 53.9 | 3.53 | Maximum accuracy |

## 2. Data Science Elements

### 2.1 Computer Vision Fundamentals

The application utilizes several computer vision techniques:

- \*Object Detection\*: Identifying and localizing objects in images

- \*Bounding Box Regression\*: Predicting precise object boundaries

- \*Class Prediction\*: Identifying object categories from 80 COCO classes

- \*Non-Maximum Suppression (NMS)\*: Removing redundant overlapping detections

### 2.2 Pre-processing

python

# Convert PIL image to cv2 format

img\_bgr = cv2.cvtColor(np.array(image), cv2.COLOR\_RGB2BGR)

The pre-processing steps include:

- Converting from PIL Image format to NumPy array

- Converting color space from RGB to BGR (OpenCV's preferred format)

- The YOLOv8 library internally handles:

- Resizing to model input dimensions

- Normalization (scaling pixel values to [0,1])

- Tensor conversion

### 2.3 Inference Process

python

# Run detection

results = model(img\_bgr, conf=conf\_threshold)[0]

During inference:

1. The image passes through the backbone to extract features

2. Features are processed through the neck to enhance multi-scale representation

3. The head generates predictions (boxes, class probabilities)

4. Predictions are filtered by confidence threshold

5. Non-maximum suppression removes overlapping detections

### 2.4 Post-processing

After model inference, the application extracts useful information:

python

for box in results.boxes:

x1, y1, x2, y2 = map(int, box.xyxy[0]) # Bounding box coordinates

confidence = box.conf[0].item() # Detection confidence

class\_id = int(box.cls[0].item()) # Class ID

class\_name = model.names[class\_id] # Class name from COCO dataset

Post-processing includes:

- Converting

# For data manipulation

**Code Walkthrough**

**1. Page Configuration and Sidebar**

# Set Streamlit page config

st.set\_page\_config(page\_title="YOLOv8 Object Detection", layout="wide", page\_icon="🔍")

# Sidebar - Project info

with st.sidebar:

st.title("📦 YOLOv8 Showcase")

st.markdown("""

Welcome to the \*\*YOLOv8 Real-Time Object Detection App\*\*!

This demo is part of the \*\*Innovation Showcase 2025\*\*.

🚀 Powered by Ultralytics' YOLOv8.

🔍 Perform detection via \*\*webcam\*\* or \*\*image upload\*\*.

🛠️ Customize detection using pre-trained YOLOv8 models.

""")

st.markdown("---")

# Model selection

selected\_model = st.selectbox("Select YOLOv8 Model", ["yolov8n.pt", "yolov8s.pt",

"yolov8m.pt", "yolov8l.pt", "yolov8x.pt"])

# Confidence threshold slider

confidence\_threshold = st.slider("Confidence Threshold", 0.0, 1.0, 0.25, 0.05)

**Explanation:**

* st.set\_page\_config() sets up the application with a wide layout and custom page title and icon
* The sidebar contains:
  + Project introduction and description
  + A dropdown menu to select from five different YOLOv8 model variants (nano to extra large)
  + A slider to adjust the confidence threshold for object detection

**2. Model Loading with Caching**

# Load YOLOv8 model

@st.cache\_resource

def load\_model(model\_name):

return YOLO(model\_name)

model = load\_model(selected\_model)

**Explanation:**

* The @st.cache\_resource decorator ensures the model is cached in memory
* This improves performance by preventing model reloading when the app interface is refreshed
* The function loads the selected YOLOv8 model variant using the Ultralytics YOLO class

**3. Application Header**

# App header

st.markdown('<h1 style="text-align: center;">📸 Real-Time Object Detection using YOLOv8</h1>', unsafe\_allow\_html=True)

st.markdown("""

Use this interactive application to perform object detection using your webcam or by uploading an image.

Results will display detected objects with bounding boxes and class labels.

""")

**Explanation:**

* Creates a centered heading for the application
* Provides a brief description of the application's functionality

**4. Object Detection Function**

# Function to run object detection on image

def detect\_objects(image, model, conf\_threshold):

# Convert PIL image to cv2 format

img\_bgr = cv2.cvtColor(np.array(image), cv2.COLOR\_RGB2BGR)

# Run detection

results = model(img\_bgr, conf=conf\_threshold)[0]

# Prepare result image with annotations

annotated\_img = img\_bgr.copy()

# Dictionary to track detected objects

detected\_objects = []

# Process each detection

for box in results.boxes:

x1, y1, x2, y2 = map(int, box.xyxy[0])

confidence = box.conf[0].item()

class\_id = int(box.cls[0].item())

class\_name = model.names[class\_id]

# Add to detected objects list

detected\_objects.append(class\_name)

# Draw bounding box and label

label = f"{class\_name}: {confidence:.2f}"

cv2.rectangle(annotated\_img, (x1, y1), (x2, y2), (0, 255, 0), 2)

# Better text background for visibility

text\_size = cv2.getTextSize(label, cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, 2)[0]

cv2.rectangle(annotated\_img, (x1, y1 - text\_size[1] - 10), (x1 + text\_size[0], y1), (0, 0, 0), -1)

cv2.putText(annotated\_img, label, (x1, y1 - 5),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (255, 255, 255), 2)

# Count objects by class

object\_counts = dict(Counter(detected\_objects))

return cv2.cvtColor(annotated\_img, cv2.COLOR\_BGR2RGB), object\_counts

**Explanation:**

* This function processes an input image through the YOLOv8 model
* Key steps:
  1. Converts the PIL image to the BGR format needed by OpenCV
  2. Runs the YOLOv8 model with the specified confidence threshold
  3. Extracts bounding box coordinates, confidence scores, and class IDs for each detection
  4. Draws green bounding boxes around detected objects
  5. Adds text labels with class names and confidence scores
  6. Creates a black background for the text to improve visibility
  7. Counts the occurrences of each detected object class
  8. Returns the annotated image and object counts

**5. Image Upload Section**

# --- Image Upload Section ---

st.subheader("📁 Upload an Image for Detection")

uploaded\_image = st.file\_uploader("Choose an image", type=["jpg", "jpeg", "png"])

if uploaded\_image is not None:

image = Image.open(uploaded\_image)

col1, col2 = st.columns(2)

with col1:

st.image(image, caption="Uploaded Image", use\_container\_width=True)

if st.button("🔍 Detect Objects"):

with st.spinner("Processing image..."):

# Process image with selected confidence threshold

result\_image, detected\_objects = detect\_objects(image, model, confidence\_threshold)

with col2:

st.image(result\_image, caption="Detection Result", use\_container\_width=True)

# Display detection summary

if detected\_objects:

st.subheader("📊 Detection Summary")

# Create a formatted summary

summary\_text = "Detected objects:\n"

for obj\_name, count in detected\_objects.items():

summary\_text += f"- {obj\_name}: {count}\n"

# Display object counts

st.markdown(f"""

#### Objects Detected: {sum(detected\_objects.values())}

""")

# Display the counts as a table

object\_df = pd.DataFrame(

{"Object": list(detected\_objects.keys()),

"Count": list(detected\_objects.values())}

)

st.table(object\_df)

else:

st.info("No objects detected in this image.")

**Explanation:**

* Creates a file uploader that accepts common image formats
* Displays the uploaded image in the left column
* Provides a "Detect Objects" button that triggers the detection process
* Shows a loading spinner during processing
* Displays the annotated result image in the right column
* Provides a detection summary with:
  + Total count of detected objects
  + A table showing counts for each object type
* Shows an informational message if no objects are detected

**6. Webcam Detection Section**

# --- Webcam Section ---

st.subheader("📷 Try Real-Time Detection with Webcam")

webcam\_active = st.toggle("Enable Webcam Stream")

if webcam\_active:

# Create placeholder for webcam feed

FRAME\_WINDOW = st.empty()

# Create placeholder for object count display

object\_counter\_placeholder = st.empty()

cap = cv2.VideoCapture(0)

st.info("Streaming live from your webcam. Click the toggle again to stop.")

# Adjust camera settings if needed

cap.set(cv2.CAP\_PROP\_FRAME\_WIDTH, 640)

cap.set(cv2.CAP\_PROP\_FRAME\_HEIGHT, 480)

while webcam\_active:

ret, frame = cap.read()

if not ret:

st.error("Webcam not accessible. Try reloading.")

break

# Process each frame

results = model(frame, conf=confidence\_threshold)[0]

# Create a copy for drawing

annotated\_frame = frame.copy()

# Track objects in current frame

frame\_objects = []

for box in results.boxes:

x1, y1, x2, y2 = map(int, box.xyxy[0])

confidence = box.conf[0].item()

class\_id = int(box.cls[0].item())

class\_name = model.names[class\_id]

# Add to detected objects list

frame\_objects.append(class\_name)

label = f"{class\_name}: {confidence:.2f}"

# Draw bounding box

cv2.rectangle(annotated\_frame, (x1, y1), (x2, y2), (0, 255, 0), 2)

# Improved text rendering

text\_size = cv2.getTextSize(label, cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, 2)[0]

cv2.rectangle(annotated\_frame, (x1, y1 - text\_size[1] - 10), (x1 + text\_size[0], y1), (0, 0, 0), -1)

cv2.putText(annotated\_frame, label, (x1, y1 - 5),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (255, 255, 255), 2)

# Display annotated frame

FRAME\_WINDOW.image(cv2.cvtColor(annotated\_frame, cv2.COLOR\_BGR2RGB))

# Update object counter

if frame\_objects:

frame\_counts = dict(Counter(frame\_objects))

counter\_text = f"Detected: {sum(frame\_counts.values())} objects"

for obj, count in frame\_counts.items():

counter\_text += f" | {obj}: {count}"

object\_counter\_placeholder.text(counter\_text)

else:

object\_counter\_placeholder.text("No objects detected")

# Add a small delay to reduce CPU usage

time.sleep(0.1)

# Clean up resources when stopped

cap.release()

**Explanation:**

* Creates a toggle to enable/disable webcam streaming
* Sets up empty placeholders for the video feed and object counter
* Initializes the webcam capture with OpenCV
* Configures webcam resolution to 640x480 for better performance
* Continuously processes frames while the toggle is active:
  1. Reads frames from the webcam
  2. Processes each frame through the YOLOv8 model
  3. Draws bounding boxes and labels on detected objects
  4. Updates the video display with the annotated frame
  5. Updates the object counter text with counts for each detected class
  6. Adds a small delay to prevent excessive CPU usage
* Properly releases webcam resources when the toggle is turned off

**7. Footer**

# --- Footer ---

st.markdown("""

---

✅ \*\*Built for the Arch Data Network Innovation Showcase 2025\*\*

🧠 Powered by \*\*Ultralytics YOLOv8\*\* | 🌐 Interface by \*\*Streamlit\*\*

""")

**Explanation:**

* Adds a footer with project attribution and technology credits

**Technical Details**

**Model Architecture**

The application uses pre-trained YOLOv8 models from Ultralytics, which are state-of-the-art object detection models featuring:

* A CNN backbone for feature extraction
* Efficient object detection heads
* Training on the COCO dataset with 80 common object classes
* Multiple size variants balancing speed and accuracy:
  + yolov8n.pt: Nano (fastest, less accurate)
  + yolov8s.pt: Small (good balance)
  + yolov8m.pt: Medium (better accuracy)
  + yolov8l.pt: Large (high accuracy)
  + yolov8x.pt: Extra Large (highest accuracy, slowest)

**Performance Considerations**

* The application uses Streamlit's caching mechanism (@st.cache\_resource) to avoid reloading the model unnecessarily
* Frame rate is managed with a small delay (time.sleep(0.1)) to prevent excessive CPU usage
* Webcam resolution is set to 640x480 for a good balance between quality and performance
* The confidence threshold slider allows users to filter out low-confidence detections for better results

**Data Processing Pipeline**

1. **Image Input**:
   * Either from webcam frames or uploaded images
   * Converted to the appropriate format for processing
2. **Model Inference**:
   * YOLOv8 processes the image and returns detection results
   * Each detection includes coordinates, class ID, and confidence score
3. **Visualization**:
   * Bounding boxes are drawn around detected objects
   * Text labels show class names and confidence scores
   * Background rectangles improve text visibility
4. **Analysis**:
   * Detected objects are counted by class
   * Statistics are displayed in a table format

**Application Use Cases**

This application is well-suited for:

1. **Educational Purposes**: Demonstrating computer vision capabilities
2. **Security Systems**: Monitoring and identifying objects in the environment
3. **Inventory Management**: Counting and classifying objects
4. **Research**: Testing object detection models in real-world scenarios
5. **Interactive Demos**: Showcasing AI capabilities in a user-friendly interface

**Conclusion**

The YOLOv8 Object Detection application combines powerful deep learning models with an intuitive web interface to make computer vision accessible. Users can easily experiment with different models and settings, visualize detection results, and analyze object statistics through either uploaded images or real-time webcam feeds.

**References**

1.Ultralytics. YOLOv8 Docs. Ultralytics. <https://docs.ultralytics.com> Published 2023. Accessed May 5, 2025.

Redmon J, Divvala S, Girshick R, Farhadi A. You Only Look Once: Unified, Real-Time Object Detection. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016:779–788.

Bochkovskiy A, Wang CY, Liao HYM. YOLOv4: Optimal Speed and Accuracy of Object Detection. *arXiv*. <https://arxiv.org/abs/2004.10934>. Published April 23, 2020. Accessed May 5, 2025.

Streamlit. Streamlit Documentation. https://docs.streamlit.io. Published 2023. Accessed May 5, 2025.

Bradski G. The OpenCV Library. *Dr. Dobb’s Journal of Software Tools*. 2000.

van der Walt S, Colbert SC, Varoquaux G. The NumPy Array: A Structure for Efficient Numerical Computation. *Computing in Science & Engineering*. 2011;13(2):22-30. doi:10.1109/MCSE.2011.37

McKinney W. Data Structures for Statistical Computing in Python. In: *Proceedings of the 9th Python in Science Conference (SciPy 2010)*. 2010:51–56.

Pillow Developers. Pillow (PIL Fork) Documentation. <https://pillow.readthedocs.io>. Accessed May 5, 2025.

Python Software Foundation. Python Language Reference, version 3.11. <https://docs.python.org/3>. Accessed May 5, 2025.