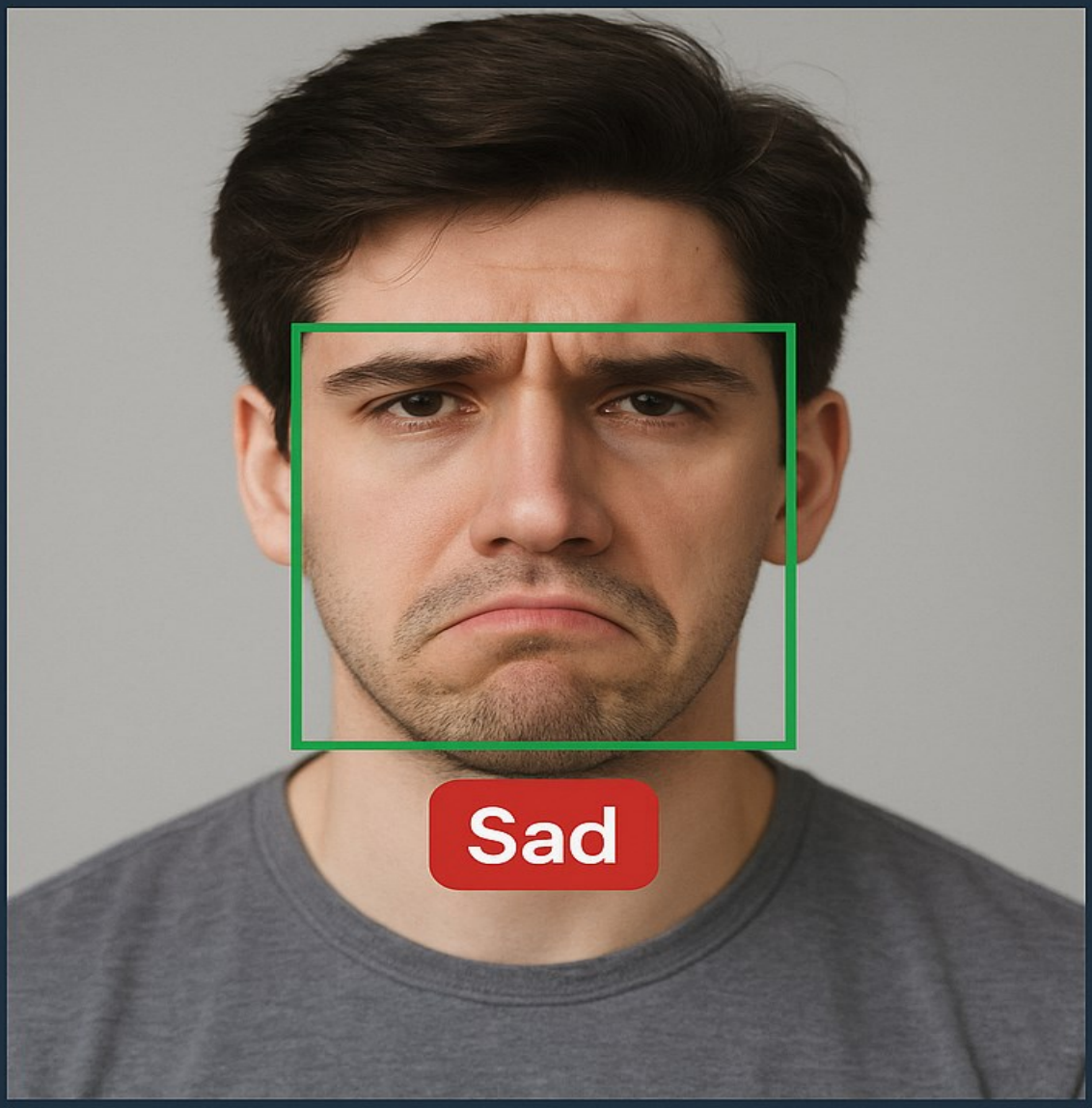
Real-Time Face Emotion Detection with YOLOv8

Complete Python Guide



AI Books
2025

Real-Time Face Emotion Detection with YOLOv8: Complete Python Guide

Table of Contents

Introduction

This guide teaches you to build a real-time face emotion detection system using YOLOv8 (You Only Look Once version 8) and Python. The application will capture video from your camera, detect faces in real-time, and classify emotions with high accuracy.

What You'll Build:

- Real-time face detection using YOLOv8
- Emotion classification (Happy, Sad, Angry, Surprised, Neutral, Fear, Disgust)
- Live camera feed processing
- Visual feedback with bounding boxes and emotion labels
- Performance metrics and confidence scores

Key Technologies:

- **YOLOv8**: State-of-the-art object detection model
- **OpenCV**: Computer vision library for camera handling
- **Ultralytics**: YOLOv8 implementation
- **NumPy**: Numerical computing
- **Pillow**: Image processing

Prerequisites & Setup

System Requirements

- Python 3.8 or higher
- Webcam or external camera
- 4GB+ RAM recommended
- CUDA-compatible GPU (optional, for better performance)

Required Knowledge

- Basic Python programming
- Understanding of computer vision concepts
- Familiarity with OpenCV (helpful but not required)

Understanding the Architecture

How YOLOv8 Works for Emotion Detection

1. **Input Processing**: Camera frames are captured and preprocessed
2. **Face Detection**: YOLOv8 identifies face regions in the frame
3. **Feature Extraction**: Facial features are extracted from detected regions
4. **Emotion Classification**: A trained model classifies emotions based on facial expressions
5. **Output Visualization**: Results are displayed with bounding boxes and labels

Data Flow

```
Camera → Frame Capture → YOLOv8 Processing → Face Detection →
Emotion Classification → Visualization → Display
```

Installation Guide

Step 1: Create Virtual Environment

```
# Create virtual environment
python -m venv emotion_detection_env

# Activate virtual environment
# On Windows:
emotion_detection_env\Scripts\activate
# On macOS/Linux:
source emotion_detection_env/bin/activate
```

Step 2: Install Required Packages

```
pip install ultralytics opencv-python numpy pillow torch torchvision
```

Step 3: Additional Dependencies

```
# For GPU support (optional)
pip install torch torchvision torchaudio --index-url
https://download.pytorch.org/whl/cu118

# For data visualization
pip install matplotlib seaborn
```

Building the Detection System

Core Components Overview

1. **Camera Handler**: Manages video capture and frame processing
2. **YOLOv8 Model**: Handles face detection and emotion classification
3. **Visualization Engine**: Draws bounding boxes and labels
4. **Performance Monitor**: Tracks FPS and accuracy metrics

Complete Application Code

Main Application (emotion_detector.py)

```
import cv2
import numpy as np
from ultralytics import YOLO
import torch
from PIL import Image
import time
from collections import defaultdict
import threading
from queue import Queue

class EmotionDetector:
    def __init__(self, model_path=None, confidence_threshold=0.5):
        """
        Initialize the Emotion Detection System

        Args:
            model_path (str): Path to custom YOLOv8 model (optional)
            confidence_threshold (float): Minimum confidence for detections
        """
        self.confidence_threshold = confidence_threshold
        self.emotion_labels = [
            'Angry', 'Disgust', 'Fear', 'Happy',
            'Neutral', 'Sad', 'Surprise'
        ]
        self.emotion_colors = {
            'Angry': (0, 0, 255),      # Red
            'Disgust': (0, 128, 0),    # Dark Green
            'Fear': (128, 0, 128),     # Purple
            'Happy': (0, 255, 0),      # Green
            'Neutral': (128, 128, 128), # Gray
            'Sad': (255, 0, 0),        # Blue
            'Surprise': (0, 255, 255)  # Yellow
        }

        # Initialize models
        self.face_model = self._load_face_model()
        self.emotion_model = self._load_emotion_model(model_path)

        # Performance tracking
        self.fps_counter = 0
        self.fps_start_time = time.time()
        self.current_fps = 0
        self.emotion_history = defaultdict(list)

        # Threading for better performance
        self.frame_queue = Queue(maxsize=2)
        self.result_queue = Queue(maxsize=2)

    def _load_face_model(self):
```

```
    """Load YOLOv8 model for face detection"""
    try:
        # Use pre-trained YOLOv8 model
        model = YOLO('yolov8n.pt')  # nano version for speed
        return model
    except Exception as e:
        print(f"Error loading face model: {e}")
        return None

def _load_emotion_model(self, model_path):
    """Load emotion classification model"""
    if model_path and os.path.exists(model_path):
        try:
            model = YOLO(model_path)
            return model
        except Exception as e:
            print(f"Error loading custom emotion model: {e}")

    # For demonstration, we'll use a simulated emotion classifier
    # In production, you'd use a trained emotion detection model
    return self._create_mock_emotion_classifier()

def _create_mock_emotion_classifier(self):
    """Create a mock emotion classifier for demonstration"""
    class MockEmotionClassifier:
        def predict(self, face_crop):
            # Simulate emotion prediction based on simple image features
            # In reality, this would be a trained deep learning model
            gray = cv2.cvtColor(face_crop, cv2.COLOR_BGR2GRAY)

            # Simple heuristic based on brightness and contrast
            brightness = np.mean(gray)
            contrast = np.std(gray)

            # Mock emotion prediction
            if brightness > 150 and contrast > 30:
                return 'Happy', 0.85
            elif brightness < 100:
                return 'Sad', 0.75
            elif contrast > 50:
                return 'Surprise', 0.70
            else:
                return 'Neutral', 0.60

    return MockEmotionClassifier()

def detect_faces(self, frame):
    """Detect faces in the frame using YOLOv8"""
    if self.face_model is None:
        return []

    try:
        # Run YOLOv8 inference
        results = self.face_model(frame, conf=self.confidence_threshold)
        faces = []

        for result in results:
            boxes = result.boxes
            if boxes is not None:
                for box in boxes:
                    # Get class ID and check if it's a person (class 0 in COCO)
```

```
                    class_id = int(box.cls[0])
                    if class_id == 0:  # Person class
                        # Extract bounding box coordinates
                        x1, y1, x2, y2 = map(int, box.xyxy[0])
                        confidence = float(box.conf[0])

                        # Focus on upper part of person detection (face region)
                        face_y1 = y1
                        face_y2 = y1 + int((y2 - y1) * 0.3)  # Top 30% for face

                        faces.append({
                            'bbox': (x1, face_y1, x2, face_y2),
                            'confidence': confidence
                        })

        return faces
    except Exception as e:
        print(f"Error in face detection: {e}")
        return []

def classify_emotion(self, face_crop):
    """Classify emotion from face crop"""
    try:
        emotion, confidence = self.emotion_model.predict(face_crop)
        return emotion, confidence
    except Exception as e:
        print(f"Error in emotion classification: {e}")
        return 'Neutral', 0.0

def draw_results(self, frame, faces, emotions):
    """Draw bounding boxes and emotion labels on frame"""
    for i, face in enumerate(faces):
        x1, y1, x2, y2 = face['bbox']
        face_conf = face['confidence']

        if i < len(emotions):
            emotion, emotion_conf = emotions[i]
            color = self.emotion_colors.get(emotion, (128, 128, 128))

            # Draw face bounding box
            cv2.rectangle(frame, (x1, y1), (x2, y2), color, 2)

            # Prepare label text
            label = f"{emotion}: {emotion_conf:.2f}"
            face_label = f"Face: {face_conf:.2f}"

            # Calculate text size for background
            (text_width, text_height), _ = cv2.getTextSize(
                label, cv2.FONT_HERSHEY_SIMPLEX, 0.6, 2
            )

            # Draw background rectangle for text
            cv2.rectangle(
                frame,
                (x1, y1 - text_height - 10),
                (x1 + text_width, y1),
                color,
                -1
            )

            # Draw emotion label
```

```
                cv2.putText(
                    frame, label, (x1, y1 - 5),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 255, 255), 2
                )

                # Draw face confidence
                cv2.putText(
                    frame, face_label, (x1, y2 + 20),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 1
                )

        return frame

    def draw_info_panel(self, frame):
        """Draw information panel with FPS and statistics"""
        height, width = frame.shape[:2]

        # Create semi-transparent overlay
        overlay = frame.copy()
        cv2.rectangle(overlay, (10, 10), (300, 120), (0, 0, 0), -1)
        frame = cv2.addWeighted(frame, 0.8, overlay, 0.2, 0)

        # Draw FPS
        cv2.putText(
            frame, f"FPS: {self.current_fps:.1f}", (20, 35),
            cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2
        )

        # Draw instructions
        cv2.putText(
            frame, "Press 'q' to quit", (20, 60),
            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 1
        )
        cv2.putText(
            frame, "Press 's' to save frame", (20, 80),
            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 1
        )
        cv2.putText(
            frame, "Press 'r' to reset stats", (20, 100),
            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 1
        )

        return frame

    def update_fps(self):
        """Update FPS counter"""
        self.fps_counter += 1
        current_time = time.time()

        if current_time - self.fps_start_time >= 1.0:
            self.current_fps = self.fps_counter / (current_time -
self.fps_start_time)
            self.fps_counter = 0
            self.fps_start_time = current_time

    def process_frame(self, frame):
        """Process a single frame for emotion detection"""
        # Detect faces
        faces = self.detect_faces(frame)
        emotions = []
```

```
        # Process each detected face
        for face in faces:
            x1, y1, x2, y2 = face['bbox']

            # Extract face region
            face_crop = frame[y1:y2, x1:x2]

            if face_crop.size > 0:
                # Classify emotion
                emotion, confidence = self.classify_emotion(face_crop)
                emotions.append((emotion, confidence))

                # Update emotion history
                self.emotion_history[emotion].append(confidence)
            else:
                emotions.append(('Unknown', 0.0))

        # Draw results
        frame_with_results = self.draw_results(frame, faces, emotions)
        frame_with_info = self.draw_info_panel(frame_with_results)

        return frame_with_info

    def save_frame(self, frame, timestamp=None):
        """Save current frame to file"""
        if timestamp is None:
            timestamp = int(time.time())

        filename = f"emotion_detection_{timestamp}.jpg"
        cv2.imwrite(filename, frame)
        print(f"Frame saved as {filename}")

    def run_camera(self, camera_index=0):
        """Main camera loop"""
        # Initialize camera
        cap = cv2.VideoCapture(camera_index)

        if not cap.isOpened():
            print("Error: Could not open camera")
            return

        # Set camera properties
        cap.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
        cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)
        cap.set(cv2.CAP_PROP_FPS, 30)

        print("Emotion Detection System Started")
        print("Controls:")
        print("  'q' - Quit")
        print("  's' - Save current frame")
        print("  'r' - Reset statistics")

        try:
            while True:
                ret, frame = cap.read()
                if not ret:
                    print("Error: Could not read frame")
                    break

                # Flip frame horizontally for mirror effect
                frame = cv2.flip(frame, 1)
```

```
                # Process frame
                processed_frame = self.process_frame(frame)

                # Update FPS
                self.update_fps()

                # Display frame
                cv2.imshow('Emotion Detection', processed_frame)

                # Handle key presses
                key = cv2.waitKey(1) & 0xFF
                if key == ord('q'):
                    break
                elif key == ord('s'):
                    self.save_frame(processed_frame)
                elif key == ord('r'):
                    self.emotion_history.clear()
                    print("Statistics reset")

        except KeyboardInterrupt:
            print("\nStopping emotion detection...")

        finally:
            # Cleanup
            cap.release()
            cv2.destroyAllWindows()
            self.print_statistics()

    def print_statistics(self):
        """Print emotion detection statistics"""
        print("\n--- Emotion Detection Statistics ---")
        for emotion, confidences in self.emotion_history.items():
            if confidences:
                avg_conf = np.mean(confidences)
                count = len(confidences)
                print(f"{emotion}: {count} detections, avg confidence:
{avg_conf:.3f}")

# Utility functions
def check_camera_availability():
    """Check available cameras"""
    available_cameras = []
    for i in range(5):  # Check first 5 camera indices
        cap = cv2.VideoCapture(i)
        if cap.isOpened():
            available_cameras.append(i)
            cap.release()
    return available_cameras

def main():
    """Main function to run the emotion detection system"""
    print("Initializing Emotion Detection System...")

    # Check available cameras
    cameras = check_camera_availability()
    if not cameras:
        print("Error: No cameras found")
        return

    print(f"Available cameras: {cameras}")
```

```
    # Initialize detector
    detector = EmotionDetector(confidence_threshold=0.3)

    # Start camera detection
    detector.run_camera(camera_index=cameras[0])

if __name__ == "__main__":
    main()
```

Configuration File (config.py)

```
"""
Configuration settings for emotion detection system
"""

# Model settings
MODEL_CONFIG = {
    'face_model': 'yolov8n.pt',  # YOLOv8 nano for face detection
    'emotion_model': None,       # Path to custom emotion model
    'confidence_threshold': 0.5,
    'device': 'auto'  # 'cpu', 'cuda', or 'auto'
}

# Camera settings
CAMERA_CONFIG = {
    'width': 640,
    'height': 480,
    'fps': 30,
    'mirror': True
}

# Emotion settings
EMOTION_CONFIG = {
    'labels': [
        'Angry', 'Disgust', 'Fear', 'Happy',
        'Neutral', 'Sad', 'Surprise'
    ],
    'colors': {
        'Angry': (0, 0, 255),
        'Disgust': (0, 128, 0),
        'Fear': (128, 0, 128),
        'Happy': (0, 255, 0),
        'Neutral': (128, 128, 128),
        'Sad': (255, 0, 0),
        'Surprise': (0, 255, 255)
    },
    'smoothing_window': 5  # Frames to average for smoothing
}

# Display settings
DISPLAY_CONFIG = {
    'show_fps': True,
    'show_confidence': True,
    'font_scale': 0.6,
    'font_thickness': 2,
    'box_thickness': 2
}
```

Advanced Features

1. Emotion Smoothing

```
def smooth_emotions(self, emotions, window_size=5):
    """Smooth emotion predictions over time"""
    # Implementation for temporal smoothing
    pass
```

2. Multi-face Tracking

```
def track_faces(self, faces, previous_faces):
    """Track multiple faces across frames"""
    # Implementation for face tracking
    pass
```

3. Performance Optimization

```
def optimize_processing(self, frame):
    """Optimize frame processing for better performance"""
    # Resize frame for faster processing
    small_frame = cv2.resize(frame, (320, 240))
    return small_frame
```

Troubleshooting

Common Issues and Solutions

1. Camera not opening:

```
# Check camera permissions and availability
cameras = check_camera_availability()
print(f"Available cameras: {cameras}")
```

2. Low FPS performance:

- Reduce frame resolution
- Use YOLOv8 nano model
- Enable GPU acceleration
- Process every nth frame

3. Inaccurate detections:

- Adjust confidence threshold
- Improve lighting conditions
- Use higher resolution model
- Train custom emotion model

4. Memory issues:

```
# Clear GPU memory
```

```
torch.cuda.empty_cache()
```

Performance Optimization

GPU Acceleration

```
# Enable CUDA if available
device = 'cuda' if torch.cuda.is_available() else 'cpu'
model = YOLO('yolov8n.pt').to(device)
```

Frame Processing Optimization

```
def process_every_nth_frame(self, frame, n=3):
    """Process every nth frame for better performance"""
    if self.frame_counter % n == 0:
        return self.process_frame(frame)
    return frame
```

Memory Management

```
def cleanup_memory(self):
    """Clean up memory usage"""
    import gc
    gc.collect()
    if torch.cuda.is_available():
        torch.cuda.empty_cache()
```

Deployment Considerations

Docker Deployment

```
FROM python:3.9-slim

WORKDIR /app
COPY requirements.txt .
RUN pip install -r requirements.txt

COPY . .
CMD ["python", "emotion_detector.py"]
```

Web Integration

```
# Flask web app integration
from flask import Flask, render_template, Response

app = Flask(__name__)

@app.route('/video_feed')
def video_feed():
    return Response(generate_frames(),
                    mimetype='multipart/x-mixed-replace; boundary=frame')
```

Mobile Deployment

- Use ONNX format for mobile optimization
- Implement TensorFlow Lite conversion
- Consider edge computing solutions

Conclusion

This guide provides a complete foundation for building real-time emotion detection systems using YOLOv8. The modular design allows for easy customization and extension based on specific requirements.

Next Steps:

1. Train custom emotion models on your data
2. Implement advanced tracking algorithms
3. Add data logging and analytics
4. Deploy to production environments

Resources:

- YOLOv8 Documentation
- OpenCV Python Tutorials
- PyTorch Documentation