Introduction

This document details the steps involved in building a Face Mask Detection model using YOLOv8. The objective was to create a model that accurately distinguishes between images with and without masks. The development process involved five key steps: data filtering, data preparation, model creation, data ecaluation and model testing.

Step 1: Data Loading and Filtering

The initial dataset contained multiple categories, including "mask," "no mask," "colorful mask," and "surgical mask." To focus the model on distinguishing only between "Mask" and "No Mask," I filtered the dataset to retain only these two labels. While implementing this, I encountered structural inconsistencies, which I resolved by reorganizing the images and annotations to align with the binary labels. This step established a clear and well-defined scope for the model, allowing it to focus on mask detection.

```
filter_data = data[data['classname'].isin(['face_with_mask', 'face_no_mask'])]
filter_data = filter_data.reset_index(drop=True)
print(len(filter_data))
filter_data.head()
```

Step 2: Data Preparation

- With the filtered dataset ready, I examined the bounding box coordinates and identified labeling inconsistencies where:
- I corrected the labels to ensure proper bounding box placement:
  y1 to x2
  x2 to y1
- This correction was crucial for accurate object detection.
- Reviewed sample images to verify that the bounding boxes were positioned correctly.
- Confident that these adjustments would improve the model's ability to detect and classify faces with and without masks**.**

```
image_height, image_width = img.shape[:2]
x1 = row['x1']
y1 = row['x2']
x2 = row['y1']
y2 = row['y2']
x_center = (x1 + x2) / 2 / image_width
y_center = (y1 + y2) / 2 / image_height
width = (x2 - x1) / image_width
height = (y2 - y1) / image_height

# Generate YOLO annotation
yolo_annotation = f"{1 if row['classname'] == 'face_with_mask' else 0} {x_center} {y_center} {width} {height}"
```

STEP2: Data Processing Code Overview

In the data processing code:

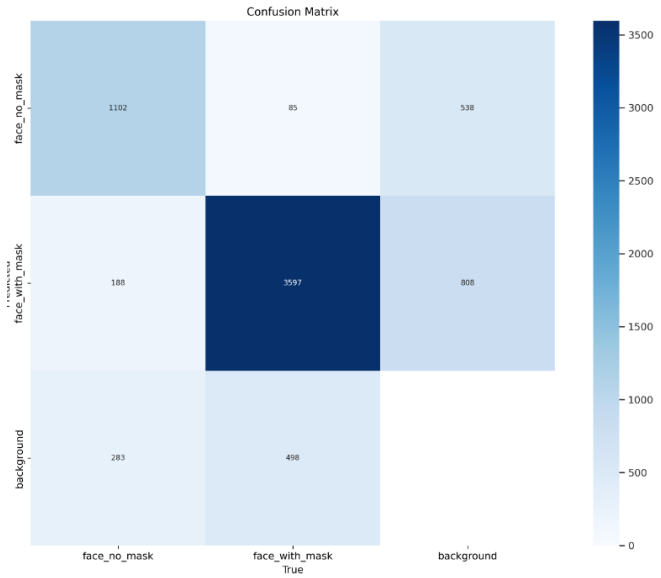
1. **Folder Setup**: The code initializes folders for saving processed images and YOLO annotations.
2. **Image Processing**: Each image is loaded, checked, and resaved to a new directory.
3. **Annotation Generation**: Bounding box coordinates are converted into YOLO format (normalized center coordinates and dimensions), and labels are assigned (1 for "face_with_mask" and 0 for "no_mask"). These annotations are saved as .txt files, formatted for YOLO training.

Step 3: Model Training

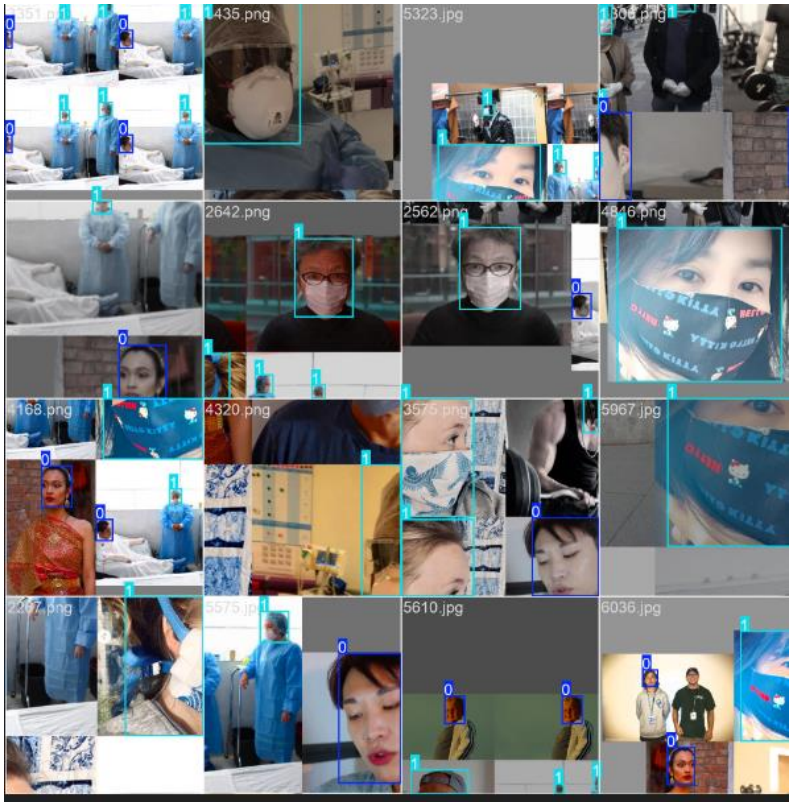
- Selected YOLOv8's lightweight variant, **YOLOv8n.yaml**, for its efficient balance of speed and accuracy, enabling faster training without compromising precision in face mask detection.
- After initial training, the model achieved **79% accuracy** at 5 epochs.
- Extended training to **7 epochs**, resulting in an improved accuracy of **81.5%**.

Step 4 : Data Evaluation:

Utilizing a confusion matrix alongside batch analysis offers a nuanced and thorough evaluation of the model's performance. These techniques not only highlight the strengths of the model but also illuminate areas for improvement, guiding further refinements to enhance accuracy in face mask detection.



Confusion Matrix



train_batch0.jpg

Step 5: Testing the Model on Video

To assess the performance of the trained face mask detection model in a practical setting, I implemented a video testing process. This section outlines the steps taken to test the model on a video file.

Implementation Steps

Video Capture:

- The video file is loaded using OpenCV's VideoCapture, enabling frame-by-frame processing.

Video Writer Initialization:

- A VideoWriter object is initialized to save the processed output video. The output file format and codec are specified, ensuring compatibility for playback.

Model Loading:

- The YOLOv8 model is loaded from a saved checkpoint file. This trained model contains the learned parameters necessary for detecting faces in the video.

Detection Loop:

- The code enters a loop to process each frame of the video:
- Frame Read: Each frame is captured in sequence.
- Model Inference: The model runs inference on the current frame to identify detected objects.

Result Processing: For each detected object with a confidence score above a predefined threshold:

- A bounding box is drawn around the detected face.
- The corresponding class label is displayed above the bounding box.

Output Handling:

- Each processed frame is written to the output video file.
- The frame is displayed in a window, allowing real-time visualization of detections.
- The loop continues until the end of the video or until interrupted by the user.

Resource Cleanup:

- Upon completion, resources such as the video capture and writer are released, ensuring no memory leaks occur.