Comprehensive Report on Face Mask Detection Using CNN in PyTorch and YOLOV8 Model

Aarmaan Choudhary

June 24, 2025

Abstract

This report documents the complete process of building and training a Convolutional Neural Network (CNN) for the task of face mask detection. The implementation uses the PyTorch deep learning framework and is trained on a custom dataset comprising images of masked and unmasked individuals. The model employs advanced data augmentation, a custom CNN architecture with dropout and batch normalization, and is trained using binary cross-entropy loss with the Adam optimizer.

1 Introduction

Wearing face masks has become a public health necessity. Automating the detection of face masks can enhance safety protocols in public spaces. This project involves designing a deep learning model using PyTorch to classify images into:

- **With Mask**
- **Without Mask**

2 Dataset Preparation

The dataset used is extracted from a ZIP file containing two categories. The directory structure is compatible with `torchvision.datasets.ImageFolder`.

Data Augmentation and Transforms

We apply strong augmentation to the training set to improve generalization:

- Resize to 128x128
- Random Horizontal Flip
- Random Rotation ($\pm 15°$)

- Color Jitter (brightness, contrast, saturation, hue)
- RandomResizedCrop
- Normalization to [-1, 1] range

Validation set is resized and normalized similarly, but without augmentation.

Listing 1: Train Transform

```
train_transform = transforms.Compose([
    transforms.Resize((128, 128)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(15),
    transforms.ColorJitter(0.2, 0.2, 0.2, 0.1),
    transforms.RandomResizedCrop(128, scale=(0.8, 1.0)),
    transforms.ToTensor(),
    transforms.Normalize([0.5]*3, [0.5]*3)
])
```

3 Model Architecture

We define a custom CNN with three convolutional blocks, each followed by batch normalization, ReLU activation, max pooling, and dropout.

- **Block 1:** Conv2D(3→32), Conv2D(32→32), MaxPool, Dropout(0.25)
- **Block 2:** Conv2D(32→64), Conv2D(64→64), MaxPool, Dropout(0.25)
- **Block 3:** Conv2D(64→128), Conv2D(128→128), MaxPool, Dropout(0.25)
- **Fully Connected:** 128*16*16 → 512 → 1 (Sigmoid)

Listing 2: Model Architecture

```
class FaceMaskCNN(nn.Module):
    def __init__(self, num_classes=1):
        super(FaceMaskCNN, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 32, 3, padding=1), nn.BatchNorm2d(32), nn.ReLU(),
            nn.Conv2d(32, 32, 3, padding=1), nn.BatchNorm2d(32), nn.ReLU()
                ↪ ,
            nn.MaxPool2d(2), nn.Dropout(0.25),

            nn.Conv2d(32, 64, 3, padding=1), nn.BatchNorm2d(64), nn.ReLU()
                ↪ ,
            nn.Conv2d(64, 64, 3, padding=1), nn.BatchNorm2d(64), nn.ReLU()
                ↪ ,
            nn.MaxPool2d(2), nn.Dropout(0.25),

            nn.Conv2d(64, 128, 3, padding=1), nn.BatchNorm2d(128), nn.ReLU
                ↪ (),
            nn.Conv2d(128, 128, 3, padding=1), nn.BatchNorm2d(128), nn.
                ↪ ReLU(),
```

```
            nn.MaxPool2d(2), nn.Dropout(0.25),
        )
        self.classifier = nn.Sequential(
            nn.Flatten(),
            nn.Linear(128 * 16 * 16, 512),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(512, 1),
            nn.Sigmoid()
        )
    def forward(self, x):
        x = self.features(x)
        return self.classifier(x)
```

4 Training Setup

The model was trained using the following configuration:

- **Loss Function:** CrossEntropyLoss (suitable for multi-class logits)
- **Optimizer:** Adam
- **Learning Rate:** 0.0005
- **Epochs:** 3
- **Batch Size:** 32
- **Device:** CUDA if available, else CPU

Listing 3: Loss and Optimizer

```
model = FaceMaskCNN().to(device)
criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

5 Training Loop

The model is trained over multiple epochs with performance printed after each epoch.

- **train():** Loop over mini-batches, compute loss, backpropagate.
- **eval():** Evaluate on validation set.
- Accuracy and loss are tracked over epochs.

6 YOLOv8-Face Detection and Classification Pipeline

To improve the model's applicability in real-world settings, we integrate the face detection capability of the YOLOv8 architecture with the custom CNN-based face mask classifier. This two-stage approach ensures robustness in unconstrained environments, such as webcam footage, CCTV streams, or mobile camera input.

6.1 YOLOv8 for Face Detection

YOLOv8 is the latest version of the YOLO (You Only Look Once) object detection family. We use a pretrained YOLOv8-face model fine-tuned on facial detection datasets (e.g., WIDER FACE). YOLOv8 provides:

- Real-time face detection at high accuracy.
- Bounding boxes for each detected face.
- Light-weight architecture suitable for edge deployment.

The YOLOv8 model is loaded using the Ultralytics `yolov8-face` library and performs inference on each input frame to detect all faces.

Listing 4: Detecting Faces with YOLOv8

```
from ultralytics import YOLO
model = YOLO("yolov8n-face.pt")
results = model(frame)
for face in results:
    x1, y1, x2, y2 = face.boxes.xyxy
    face_crop = frame[y1:y2, x1:x2]
```

6.2 Face Cropping and Classification

Each detected face is cropped and resized to match the input requirements of the CNN model (`128x128` pixels). The face image is then normalized and passed through the trained CNN model to predict whether the person is:

- Wearing a mask
- Not wearing a mask

Listing 5: Classifying Face Mask Status

```
face_tensor = transform(face_crop).unsqueeze(0).to(device)
outputs = cnn_model(face_tensor)
_, predicted = torch.max(outputs, 1)
label = 'Mask' if predicted.item() == 0 else 'No␣Mask'
```

6.3 End-to-End Pipeline

The full pipeline can be summarized as follows:

1. Capture frame from video or webcam.
2. Detect faces using YOLOv8-face.
3. For each face:
    - Crop and preprocess image.
    - Pass through CNN model.
    - Annotate the frame with bounding box and predicted label.
4. Display or save the result.

6.4 Advantages of This Approach

- Modular: YOLOv8 and CNN can be upgraded independently.
- Accurate: Detection and classification are handled by specialized models.
- Real-Time: Optimized inference using CUDA and half-precision models.

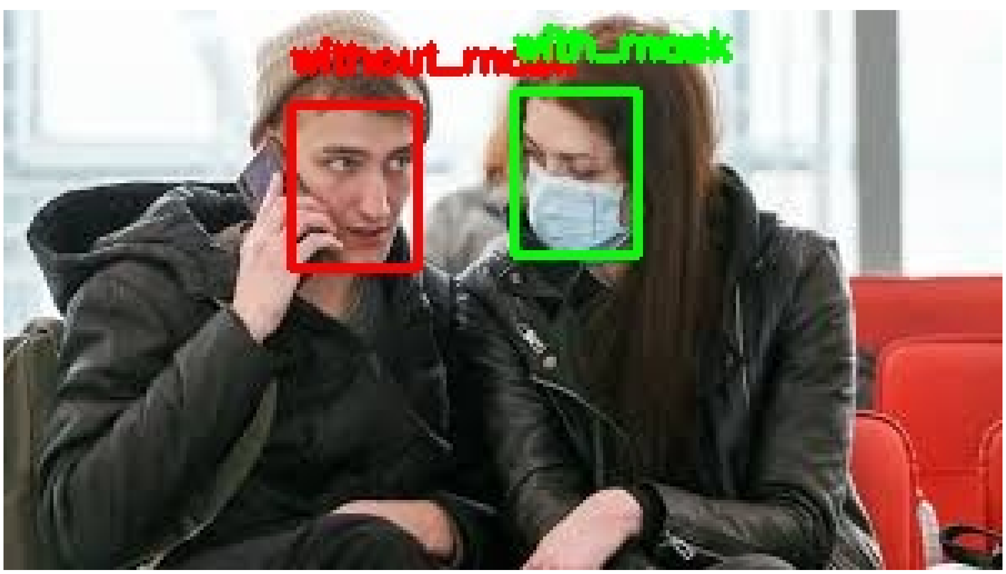

Figure 1: Example: Detected faces with mask classification using YOLOv8 + CNN

7 Conclusion

This CNN-based model shows promising results in binary classification of face mask usage. With relatively simple architecture and data augmentation, it generalizes well. The solution is deployable in real-time applications via webcam or security systems.