Real-Time Face Mask Detection Using PyTorch, OpenCV, and MTCNN: Advancing Public Safety and Mask Compliance

Short Summary:

The research report titled "Real-Time Face Mask Detection Using PyTorch, OpenCV, and MTCNN: Advancing Public Safety and Mask Compliance" proposes a real-time face mask detection system using Python, PyTorch, OpenCV, and the MTCNN model. The system accurately detects faces and classifies them as wearing masks or not. The report discusses challenges in face mask detection and presents a systematic methodology for real-time detection. It includes code snippets and implementation details. The report emphasises the importance of such systems in promoting public safety during the COVID-19 pandemic and suggests potential future research directions. Additionally, the resources section provides links to the GitHub repository, YouTube channel, and dataset used in the research, offering readers access to the project's code and resources.

Introduction:

The COVID-19 pandemic has made wearing face masks crucial for preventing the spread of the virus. However, it can be difficult to ensure that people follow mask-wearing policies in different places. To address this issue, face mask detection systems have emerged. These systems use computer vision and deep learning techniques to automatically identify whether individuals are wearing masks or not. They are valuable in public spaces, workplaces, and healthcare facilities as they help monitor and enforce mask-wearing protocols. This report focuses on creating a real-time face mask detection system using Python, PyTorch, OpenCV, and the MTCNN model. The goal is to accurately identify people wearing masks and those without masks, contributing to public safety and efforts to contain COVID-19. By combining PyTorch, OpenCV, and MTCNN, this solution enables efficient face detection and mask classification, facilitating effective enforcement of mask-wearing guidelines in crowded settings.

Challenges and Research Gaps:

Although progress has been made in face mask detection, there are still important research gaps to be filled. One gap is the lack of diverse and well-annotated datasets specifically designed for face mask detection. These datasets are crucial for training and evaluating accurate models that can handle various scenarios, demographics, and mask types.

Another gap is in dealing with occlusions and partial face visibility. In real-world situations, faces can be partially covered by objects or masks may be worn incorrectly, making it challenging for existing detection systems to accurately identify faces. More research is needed to improve the performance of these systems in such cases.

Real-time face mask detection is complex due to variations in face sizes, orientations, and different mask designs. Achieving a balance between accuracy and efficiency while handling unseen scenarios remains a challenge. Although traditional computer vision techniques and deep learning models have been explored, there is ongoing work to refine face mask detection systems.

In summary, filling the research gaps related to diverse datasets, handling occlusions, and optimising accuracy and efficiency will enhance the reliability and effectiveness of real-time face mask detection systems in various practical applications.

Existing Works and Problems:

Previous studies have investigated different approaches to face mask detection. Traditional computer vision methods such as Viola-Jones, Haar cascades, and HOG have been used for face detection, but they struggle with accurate mask classification due to their complex visual patterns. Deep learning methods, particularly convolutional neural networks (CNNs) implemented using a framework such as PyTorch, have shown promising results in accurately classifying masked and unmasked faces. The MTCNN model is widely used for its robust performance in face mask detection.

Despite these advances, there are problems with current work. One problem is the limited generalisation of models to unseen scenarios and changes. Models trained on a particular data set may perform well on new data or have difficulty identifying faces from different demographics and lighting conditions. Improving the generalisation capabilities of face mask detection models remains an important challenge. Another

problem is the trade-off between accuracy and efficiency. Deep learning models, especially complex CNN architectures, require large computing resources and processing time. Achieving a balance between accurate detection and real-time performance is particularly important in applications such as public space monitoring. Developing lightweight and optimised models that provide high accuracy while providing efficient inference is an ongoing challenge.

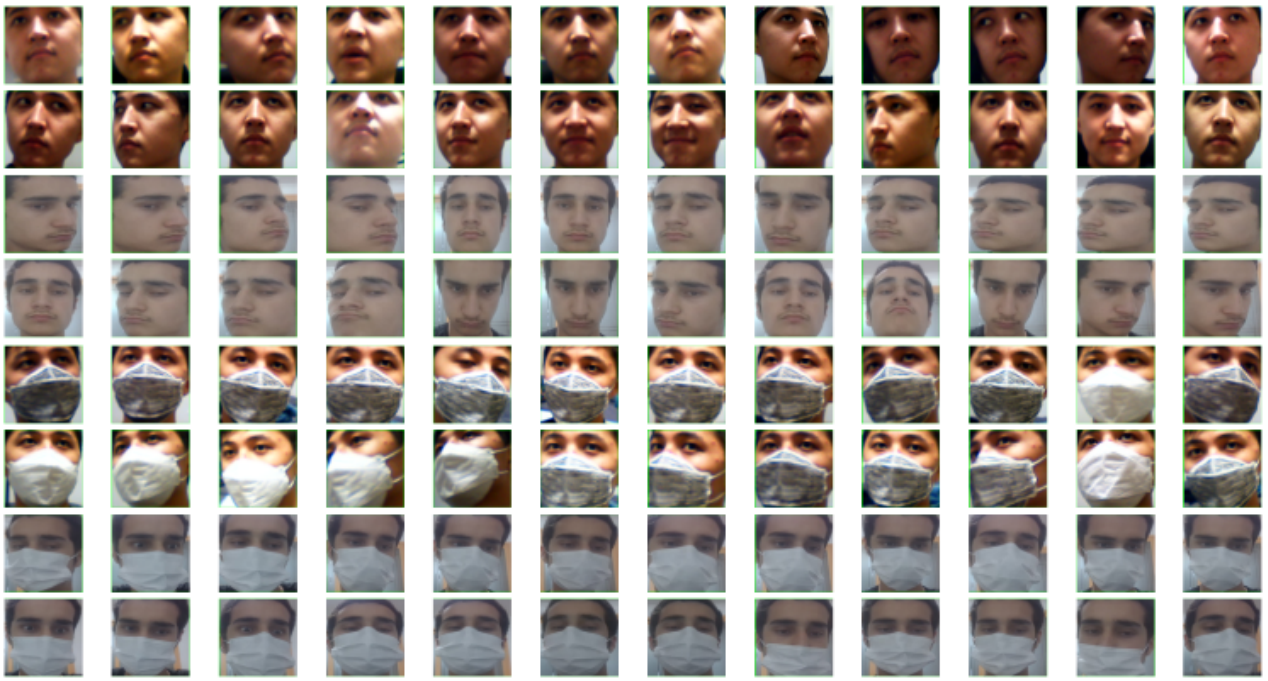
In summary, face mask detection research faces challenges related to face variations, different mask types, and real-time processing requirements. While previous work has explored traditional computer vision techniques and deep learning models, CNN-based approaches show promise.

Proposed Methodology/Solution:

The proposed solution adopts a systematic approach to achieve real-time face mask detection. On top of that the MTCNN model is employed for **face detection** due to its superior accuracy and robustness compared to traditional approaches such as Haar cascades. MTCNN utilises a cascaded architecture consisting of three stages: **face detection**, **facial landmark localization**, and **face alignment**. This model can accurately detect faces in various conditions, including variations in pose, lighting, and occlusions.

For the face mask classification task, a deep learning approach is employed. A pre-trained ResNet-50 model is used as the backbone, and the final fully connected layer is modified to have two output neurons representing the mask and no-mask classes. Transfer learning is leveraged to utilise the knowledge gained from pre-training on a large-scale dataset to improve the performance of the face mask detection model.

I used a dataset consisting of 200 images that included pictures of myself and my friends. Each image was labelled with either the "**mask**" or "**no-mask**" class to indicate whether the person in the image was wearing a mask or not. The purpose of this dataset was to train a machine learning model to classify images based on the presence or absence of a mask.



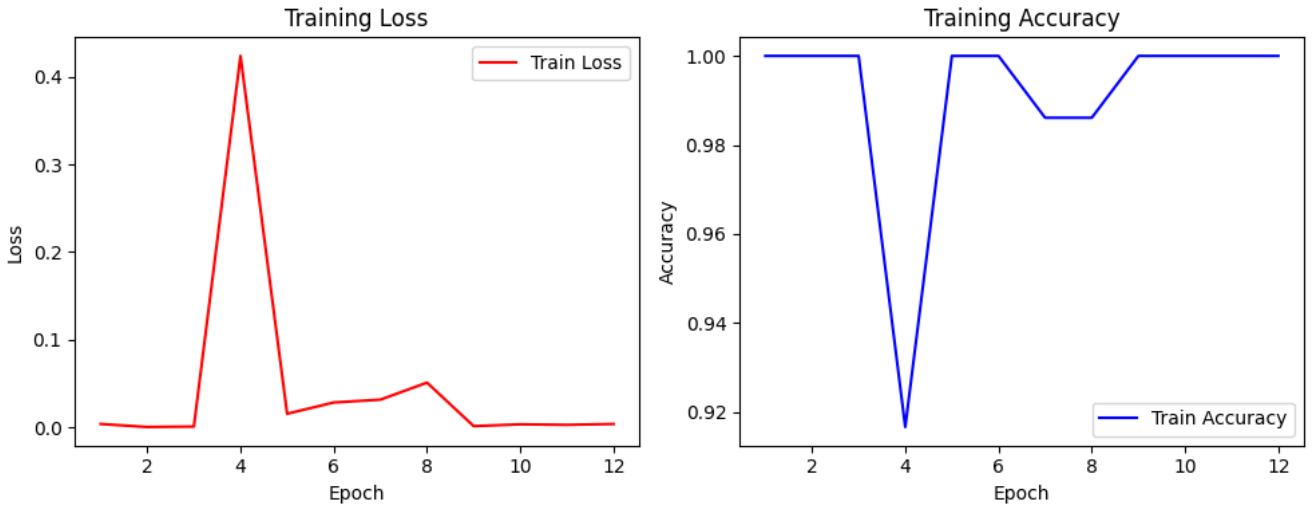
The images are preprocessed by resizing them to a fixed size, converting them to RGB colour space, and applying normalisation by using the following code on Pytorch framework:

```
# Transformation for input image
transform = transforms.Compose([
  transforms.Resize((224, 224)),
  transforms.ToTensor(),
  transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))
])
```

The model is trained using the dataset, optimising the model parameters with the Adam optimizer and minimising the binary cross-entropy loss.

```
criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
scheduler=      optim.lr_scheduler.StepLR(optimizer,      step_size=3,
gamma=0.1)
model = model.to(device) # run on GPU
```

After training the model for 12 epochs, we have achieved significant improvements in its performance. With higher accuracy and lower loss, the model is now more reliable for real-time face mask detection.

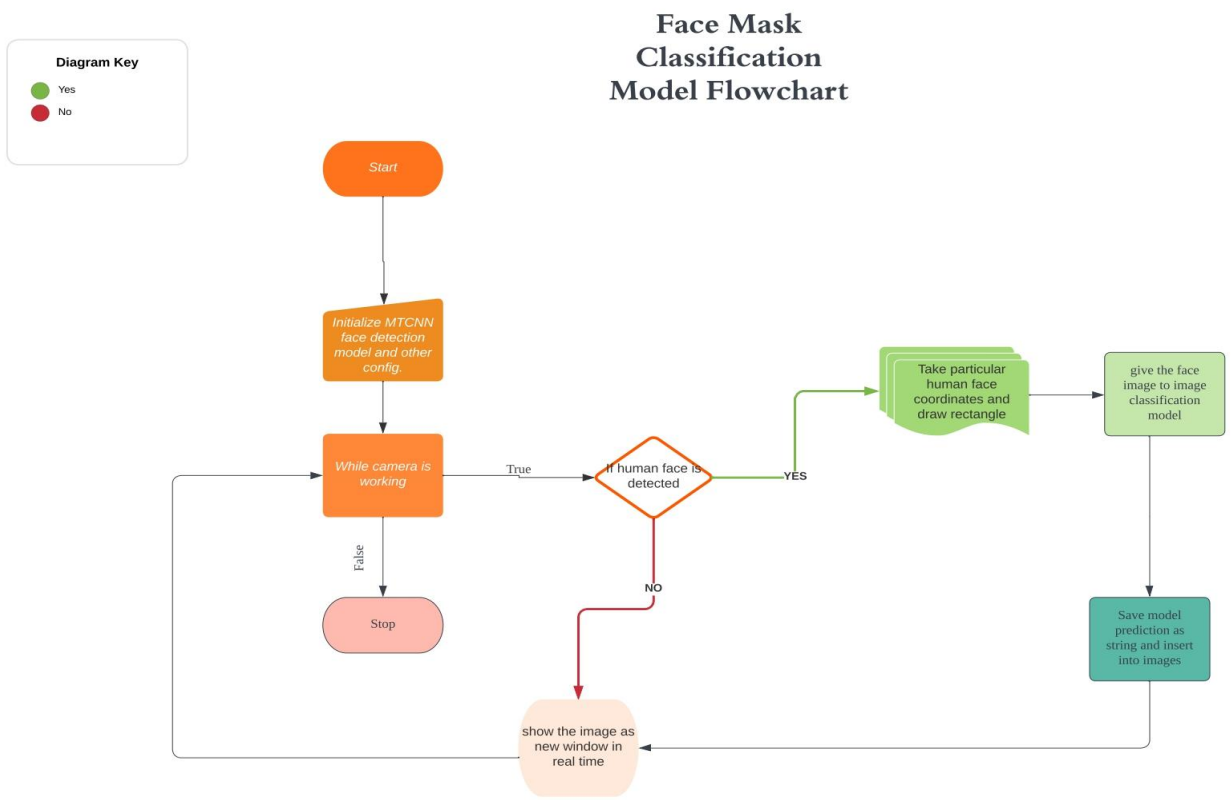


We have saved the trained model to be used in the second part of our code. This allows us to easily integrate it into different applications, such as surveillance systems or entrance checkpoints, or our smart device's camera where It is capable of effectively differentiating individuals based on whether they are wearing face masks or not.

Real-Time Webcam Access and Face Mask Detection: A Comprehensive Implementation

During the real-time face mask detection phase, the OpenCV library is utilised to capture video frames from a webcam. Each frame is processed using the MTCNN model for face detection, which returns bounding boxes indicating the location of each detected face. From the bounding boxes, the face region is extracted and converted to a PIL Image format. The preprocessed face image is then passed through the trained face mask detection model for classification. The model predicts whether the person in the image is wearing a mask or not, and the result is displayed on the screen by drawing a rectangle around the face and adding a label indicating mask or no-mask. This process is repeated for each frame, enabling real-time face mask detection. Here is the Face

Mask Classification Model Flowchart:



Also, the following code allows for the utilisation of a web camera to detect whether individuals are wearing facemasks or not.

```
from PIL import Image, ImageDraw
import numpy as np
import cv2
from mtcnn import MTCNN
import torch
import torch.nn as nn
from torchvision import transforms, models

# Loading ready model
state_dict=torch.load('Final_FaceMask_detection_model.pt',
map_location=torch.device('cpu'))
class FaceMaskModel(nn.Module):
    def __init__(self):
```

```
        super(FaceMaskModel, self).__init__()
        self.model = models.resnet50(pretrained=True)
        num_features = self.model.fc.in_features
        self.model.fc = nn.Linear(num_features, 2)

    def forward(self, x):
        return self.model(x)


# Face mask detection model
model = FaceMaskModel()
model.load_state_dict(state_dict)
model.eval()

# Transformation for input image
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
        transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224,
0.225))
])

# labels (with mask and without mask)
labels = ['Without Mask', 'With Mask']
detector = MTCNN()

# Initialise the video capture
cap = cv2.VideoCapture(0)
if not cap.isOpened():
    print("Failed to open the camera")
    exit()

# Loop over frames from the video stream
while True:
    ret, frame = cap.read()
    if not ret:
        print("Failed to capture frame")
        break
    rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
```

```
    # face detection using MTCNN
    boxes = detector.detect_faces(rgb_frame)

    # detected faces
    for box in boxes:
        x, y, width, height = box['box']

        #face region from the frame
        face = frame[y:y+height, x:x+width]


        #  face to PIL Image format
     pil_face=Image.fromarray(cv2.cvtColor(face, cv2.COLOR_BGR2RGB))


        transformed_face = transform(pil_face)

        # a batch dimension to the face image
        transformed_face = transformed_face.unsqueeze(0)


        with torch.no_grad():
            outputs = model(transformed_face)
            _, predicted = torch.max(outputs, 1)



        label = labels[predicted.item()]

        # bounding box and label
       cv2.rectangle(frame, (x, y), (x+width, y+height), (0, 255, 0),
2)
       cv2.putText(frame, label, (x, y-10), cv2.FONT_HERSHEY_SIMPLEX,
0.9, (0, 255, 0), 2)

    cv2.imshow('Face Mask Detection', frame)

    if cv2.waitKey(1) & 0xFF == ord('q'):
```

```
        break

cap.release()
cv2.destroyAllWindows()
```

After executing the provided code, our model successfully started working and gained real-time access to the webcam feed. The model actively detects faces in the video stream and classifies them based on the presence or absence of a face mask.

The output image shows a side-by-side comparison of two images. On the left side, there is a picture of a person wearing a mask, while on the right side, there is a picture of the same person without a mask. The images are displayed with a border around them to enhance visibility.

The output image shows the excellent performance of our custom face mask detection model. Our model is distinguished by its ability to accurately distinguish masked and unmasked individuals.

Using advanced algorithms, our system precisely defines and highlights each face with incredible precision using bounding boxes. These bounding boxes provide a clear visual representation of the detected faces, enabling fast and accurate estimation.

The **"mask"** or **"unmasked"** labels offer a brief and informative representation of the mask status for each individual. This simplified presentation facilitates easy interpretation of the results.

Overall, this figure demonstrates the excellent performance of our self-developed face mask detection model. Its reliable performance and accurate classification provide

valuable insights for monitoring and implementing face mask adherence in a variety of settings. By employing computer vision techniques and deep learning models, we have developed a robust face mask detection system capable of real-time monitoring.

Conclusion

The proposed real-time face mask detection system offers a solution to the challenges of monitoring and enforcing mask-wearing guidelines. By leveraging PyTorch, OpenCV, and the MTCNN model, the system achieves accurate face detection and mask classification in real-time. This technology contributes to public safety by automating monitoring processes and minimising conflicts arising from non-compliance. Additionally, the system has applications beyond the current pandemic, including healthcare, industrial settings, and security. Further research is needed to address existing challenges and optimise the balance between accuracy and efficiency in real-time face mask detection systems.

In conclusion, the development of a real-time face mask detection system using PyTorch, OpenCV, and the MTCNN model demonstrates the potential of computer vision and deep learning techniques in ensuring mask compliance and enhancing public safety. The proposed methodology combines the strengths of MTCNN for accurate face detection, a pre-trained ResNet model for mask classification, and OpenCV for real-time video processing. By addressing the challenges of real-time performance, generalisation, and accuracy, this system provides an effective tool for enforcing mask-wearing guidelines in various high-density settings. Through continuous research and improvement, real-time face mask detection systems can play a crucial role in mitigating the spread of infectious diseases and promoting public health and safety.

Resources:

To learn more about our research and access the resources used, we invite you to visit the following:

- GitHub repository: Our GitHub repository contains the code and resources related to our project. You can find the implementation details and code for training the models and accessing the web camera in real-time. Visit the repository at GitHub Repository.

- **YouTube Channel:** We have created a YouTube channel where you can find comprehensive video tutorials explaining the implementation details of our research. These tutorials cover topics such as training the models and demonstrating real-time web camera access. Check out our channel at YouTube Video Tutorial.

- **Dataset:** If you are interested in accessing the dataset used in our research, it is available for download. You can find more information about the dataset, including its size, format, and any preprocessing steps applied, on our GitHub repository. Visit the repository and download the dataset at Dataset.

We hope these resources will provide valuable insights and help you understand the implementation aspects of our research project. Should you have any questions or need further assistance, please don't hesitate to reach out to us. Email: valievkoyiljon112@gmail.com .