राजीव गांधी राष्ट्रीय युवा विकास सांस्थान
Rajiv Gandhi National Institute of Youth Development
युवा कार्यक्रम और खेल मांत्रालय, भारत सरकार
Ministry of Youth Affairs & Sports, Govt. of India
श्रीपेरुम्बुदूर / Sriperumbudur – 602 105

DEPARTMENT OF COMPUTER SCIENCE
(DATA SCIENCE, CYBERSECURITY, AI & ML)

II SEMESTER (2022 – 2024)

Real Time Face Mask Detection

Machine Learning Project Report

Submitted By:

Group-04

Submitted to:

Dr. C Jayakumar

Associate Professor

HOD

DEPARTMENT OF COMPUTER SCIENCE
M.SC (Data Science)
RGNIYD

Certificate of Achievement

This is to certify that we have successfully completed the Machine Learning project titled "Real-time Face Mask Detection" under the guidance of Dr. C Jayakumar, our esteemed professor and mentor.

The project involved the development and implementation of a real-time face mask detection system using Convolutional Neural Networks (CNN) and Computer Vision techniques. The team demonstrated exceptional skills in data collection, preprocessing, model development, and system integration, resulting in an accurate and efficient face mask detection system.

The team's dedication, hard work, and collaborative efforts have been exemplary throughout the project, reflecting their commitment to excellence in the field of Machine Learning. Their proficiency in leveraging cutting-edge technologies and research to address real-world challenges is commendable.

We hereby acknowledge all the members of team 4th : Athar Obaid Khan (MSAI22R006) , Prerana Bora (MSAI22R013), Nagam Venkateswarlu (MSDS22R015), Sneha Pal (MSCM22R911) , Asmina APK (MSDS22R005), Varadha S Ajith (MSCS22RO16) , Muhammad Nisamudheen K K (MSCS22R009) for their outstanding performance and commitment to academic excellence. Their work serves as a testament to their capabilities and the quality of education provided at RGNIYD.

Congratulations on this remarkable achievement!

Date:

Signature:

Rajiv Gandhi National Institute of Youth Development

Sriperumbudur, Tamilnadu.

Acknowledgment

We would like to express our heartfelt gratitude to our professor, Dr. C Jayakumar, for his invaluable guidance, support, and encouragement throughout the duration of this Machine Learning project. His expertise, insightful feedback, and mentorship have been instrumental in shaping the success of our endeavour.

We extend our appreciation to our fellow team members for their dedication and collaboration, which played a significant role in the project's execution. Their collective efforts and teamwork have been vital in accomplishing the goals of this Face Mask Detection project.

We are also thankful to the academic and research community for their contributions to the field of Machine Learning and Computer Vision. The wealth of knowledge and resources available through research papers, articles, and open-source repositories have been instrumental in enhancing our understanding and implementation of various algorithms and techniques.

Lastly, we acknowledge the support and encouragement received from our institution, which provided us with the necessary resources and environment to carry out this project successfully.

This project would not have been possible without the collective efforts and contributions of everyone involved. Each individual's commitment and enthusiasm have been instrumental in making this project a reality.

Once again, we extend our deepest gratitude to our professor, Dr. C Jayakumar, for his mentorship and guidance, and to our team members for their dedication and collaboration.

Sincerely, Team 4^{th} : Real time face mask detection.

Abstract

The **Face Mask Detection** project presents a real-time system for automated identification of individuals wearing face masks using computer vision and deep learning techniques. With the ongoing global COVID-19 pandemic, face masks have emerged as a crucial preventive measure to mitigate virus transmission. The system's primary objective is to contribute to public health and safety by enforcing face mask compliance in various settings, including public spaces, workplaces, hospitals, and transportation hubs.

This **project** employs **Convolutional Neural Networks (CNNs)** and transfer learning to develop a robust and accurate face mask detection model. CNNs are well-suited for image classification tasks and excel in extracting meaningful features from images. Transfer learning leverages the knowledge acquired from pre-trained models on extensive image datasets, allowing us to fine-tune the model for the specific task of face mask detection. The dataset comprises labelled images categorized into "**with_mask**" and "**without_mask**" classes, enabling supervised training to distinguish between individuals wearing masks and those without.

The model architecture features **MobileNetV2** as the base model, known for its lightweight design and efficiency. The top layers of **MobileNetV2** are replaced to create a custom head model, enabling the network to specialize in face mask detection. During training, the base model's weights are frozen to focus on fine-tuning the head model, which learns to recognize face mask patterns.

To evaluate the model's performance, the dataset is split into **training** and **testing sets**, and **k-fold cross-validation** is employed. Classification metrics, including accuracy, precision, recall, and F1-score, are computed to assess the model's ability to make correct predictions. The learning curve and confusion matrix provide insights into model training and classification results.

In conclusion, the **Face Mask Detection** project exemplifies the successful application of cutting-edge technologies to address pressing global challenges. The system's deployment aligns with the urgency to curb virus transmission and promote public health. Beyond technical achievements, this project emphasizes the potential of AI-powered solutions to contribute to societal well-being. As we navigate an ever-changing world, the Face Mask Detection system stands as an indispensable asset in fostering a safer and healthier environment for individuals and communities worldwide.

Table of Contents

Introduction

The Face Mask Detection project aims to develop an innovative and practical system for automating the detection of face masks in images and real-time video streams. In response to the global COVID-19 pandemic, face masks have emerged as a critical preventive measure to reduce the transmission of the virus. The project leverages the power of computer vision and deep learning techniques to contribute to public health and safety by enforcing face mask compliance in various settings.

Motivation:

The motivation behind this project lies in the pressing need to promote public health and safety during the ongoing COVID-19 pandemic. Face masks have been recommended by health authorities as a crucial measure to limit the spread of the virus. However, ensuring widespread compliance with mask-wearing guidelines can be challenging. The project seeks to address this challenge by developing an automated system that can quickly and accurately detect individuals wearing face masks. By automating the detection process, the project aims to facilitate the implementation of mask mandates in public spaces, workplaces, and other environments, contributing to the global efforts to combat the pandemic.

Objectives:

The primary objectives of the Face Mask Detection project include:

1. Develop a Robust Detection Model: Create a deep learning model capable of accurately identifying individuals wearing face masks in images and live video streams. The model should demonstrate high accuracy, robustness, and generalization capabilities, making it effective in diverse real-world scenarios.
2. Real-time Implementation: Implement the face mask detection model to work in real-time. The system should efficiently process video frames from a device's camera and provide instantaneous feedback on face mask presence or absence.
3. Ethical Considerations: Ensure ethical practices throughout the project, including data collection, privacy protection, and consent. The project will prioritize the ethical use of data and respect individuals' rights while training and testing the model.

4. Performance Evaluation: Thoroughly evaluate the performance of the face mask detection system. Conduct extensive testing on diverse datasets to assess the model's accuracy, precision, recall, and F1-score. Analyze the model's strengths and limitations to identify potential areas for improvement.

Methodology:

The methodology of the Face Mask Detection project encompasses the following key steps:

1. Data Collection: Curate a dataset of images containing individuals with and without face masks. The dataset should encompass a diverse representation of people across various demographics and backgrounds.
2. Data Preprocessing: Preprocess the dataset by resizing, normalizing, and augmenting the images. Apply data augmentation techniques to increase the dataset's diversity and improve the model's ability to generalize to unseen data.
3. Model Selection: Select an appropriate deep learning model for face mask detection. The project will explore various architectures and transfer learning techniques to leverage pre-trained models for feature extraction.
4. Model Training: Train the selected model on the pre-processed dataset using supervised learning. Employ optimization techniques and learning rate schedules to enhance model convergence and prevent overfitting.
5. Real-time Implementation: Integrate the trained model into a real-time system using OpenCV for camera access and face detection. Preprocess face regions within each frame and pass them through the model for mask classification.
6. Performance Evaluation: Evaluate the model's performance on separate testing datasets, computing classification metrics to quantify its accuracy and effectiveness. Visualize the results using graphs and confusion matrices for better insights.
7. Ethical Considerations: Throughout the project, adhere to ethical guidelines for data collection and usage. Safeguard individual privacy and obtain consent when using publicly available data.

The Face Mask Detection project employs a comprehensive and systematic approach, utilizing state-of-the-art technologies in computer vision and deep learning. By achieving its objectives, the project aims to provide a valuable tool in ensuring face mask compliance, thereby contributing to the collective global efforts in curbing the spread of COVID-19 and promoting public health and safety.

Background and Related Work:

The outbreak of the COVID-19 pandemic in late 2019 led to an unprecedented global health crisis. As the virus spread rapidly, health authorities recommended various preventive measures, including wearing face masks, to reduce transmission. Face masks act as a physical barrier, preventing respiratory droplets from spreading when individuals cough, sneeze, or talk. Consequently, face mask usage became a crucial practice in public spaces and crowded environments to mitigate the virus's spread.

In response to the pandemic, the field of computer vision and artificial intelligence witnessed an upsurge in research and applications related to face mask detection. The concept of using computer algorithms to automatically identify whether individuals are wearing face masks or not gained significant attention. Such applications have the potential to enforce mask-wearing guidelines in public places, workplaces, and other settings, ensuring public health and safety.

Face Mask Detection Overview:

The Face Mask Detection project focuses on developing an automated system to detect face masks in images and real-time video streams. The system utilizes computer vision techniques to identify faces and employs deep learning models for mask classification. The goal is to provide a practical and accurate solution that can contribute to enforcing mask-wearing guidelines and promoting public health amid the pandemic.

The face mask detection process involves the following key steps:

1. Face Detection: The system utilizes face detection algorithms to locate faces within images or video frames. This process involves identifying facial landmarks and creating bounding boxes around the detected faces.
2. Preprocessing: The face regions within the bounding boxes are preprocessed to standardize their size and ensure consistency. Preprocessing may also involve normalization to enhance the model's performance.
3. Mask Classification: The preprocessed face regions are then fed into a deep learning model capable of classifying them as either "with_mask" or "without_mask." The model is trained on a labeled dataset containing examples of individuals wearing face masks and those without.

4. Real-time Implementation: For real-time face mask detection, the system continuously captures video frames from the device's camera. Each frame undergoes face detection and mask classification in real-time, enabling instantaneous feedback on mask compliance.

Related Research and Projects:

The face mask detection domain has seen considerable interest from researchers and developers worldwide. Various approaches and methodologies have been explored to achieve accurate and efficient mask detection. Some notable related research and projects include:

1. "Real-time Face Mask Detection using CNNs": This research utilizes Convolutional Neural Networks (CNNs) to classify face mask-wearing status in real-time video streams. The model leverages transfer learning for rapid model training and deployment.
2. "Deep Learning-based Face Mask Detection for Healthcare Facilities": This project focuses on developing a face mask detection system tailored for healthcare environments. The system assists in monitoring mask compliance among healthcare workers and patients.
3. "Face Mask Detection for Surveillance Systems": This research explores the integration of face mask detection into surveillance systems. The system aims to enhance security and public safety by identifying individuals not complying with mask mandates in public spaces.
4. "MaskWNet: An Ensemble Model for Face Mask Detection": This work proposes an ensemble model that combines the predictions of multiple deep learning models to improve face mask detection accuracy and reduce false positives.

These related works highlight the diverse applications of face mask detection and the importance of accuracy, real-time performance, and context-specific considerations. The Face Mask Detection project builds upon and contributes to this existing body of research by developing an efficient, accurate, and real-time system with the potential for widespread implementation in diverse settings. The project aims to play a vital role in promoting public health and safety by encouraging face mask compliance during the COVID-19 pandemic and beyond.

Dataset Collection and Preprocessing

The success of any deep learning-based project relies heavily on the quality and diversity of the dataset used for training. In the Face Mask Detection project, a well-curated dataset of images containing individuals with and without face masks is collected and preprocessed to ensure effective training of the mask detection model.

Data Sources: Data collection for the face mask detection dataset involves sourcing images from various reliable and ethical sources. Publicly available image repositories, such as Kaggle, GitHub, and Open Images, can be valuable sources for gathering images of people wearing face masks. Additionally, custom data collection methods may involve capturing images from surveillance cameras or webcams, respecting privacy and consent.

Data Augmentation: Data augmentation is a critical step in increasing the diversity and size of the dataset, which, in turn, enhances the model's generalization capability. Augmentation techniques introduce variations to the original images without changing their essential features. Common data augmentation techniques applied in the Face Mask Detection project include:

1. Rotation: Randomly rotating the image by a certain angle to simulate different orientations of faces.
2. Zooming: Applying random zoom to the image to simulate varying distances of faces from the camera.
3. Horizontal Flip: Flipping the image horizontally to create mirror images and increase dataset size.
4. Brightness and Contrast Adjustment: Adjusting the brightness and contrast to mimic various lighting conditions.
5. Shear Transformation: Shearing the image to simulate perspective changes.

By applying data augmentation, the project ensures that the model learns to recognize face mask patterns under various conditions, making it more robust to real-world scenarios.

Labelling: Labelling is the process of assigning the correct class labels to each image in the dataset. In the Face Mask Detection project, images are labeled as "with_mask" or "without_mask" depending on whether the individuals in the

images are wearing face masks or not. Manual labeling is often required, where annotators review each image and assign the appropriate label.

The labeling process is critical for supervised learning, as it provides ground truth information for training the model. Accurate and consistent labeling is essential to ensure the model learns to distinguish between faces with and without masks accurately.

Example code for dataset collection and preprocessing:

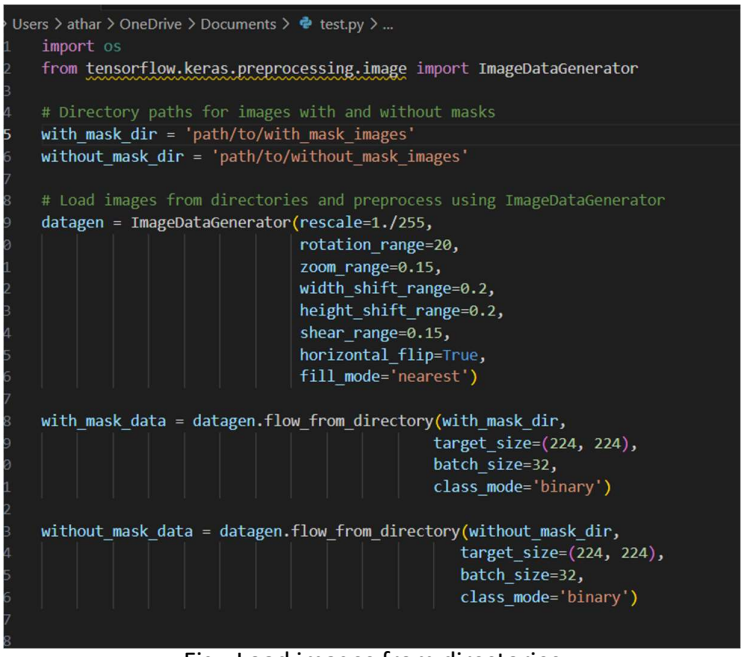


Fig.: Load images from directories

In the provided example code, the ImageDataGenerator from Keras is used to preprocess the images. The images are rescaled to have pixel values in the range [0, 1], and data augmentation techniques, such as rotation, zoom, and horizontal flip, are applied during the training process.

Additionally, each image is assigned a binary label, where 1 indicates "with_mask" and 0 indicates "without_mask." The labeled data is then used to train the face mask detection model effectively.

In conclusion, the dataset collection and preprocessing stages are vital components of the Face Mask Detection project. By carefully selecting and augmenting the dataset and ensuring accurate labeling, the project lays the groundwork for building a robust and accurate mask detection model. These steps significantly influence the model's ability to generalize to real-world scenarios and contribute to the project's ultimate goal of promoting public health and safety through automated face mask detection.

Model Architecture:

The Model Architecture is a crucial component of the Face Mask Detection project, defining the structure of the deep learning model responsible for classifying whether individuals in images or video frames are wearing face masks or not. This section explores the key concepts employed in the model architecture, including Convolutional Neural Networks (CNNs), Transfer Learning, and the specific use of MobileNetV2.

Convolutional Neural Networks (CNNs): Convolutional Neural Networks (CNNs) are a class of deep learning models specially designed for image-related tasks. They excel in capturing spatial patterns and hierarchical representations present in images. CNNs leverage convolutional layers, which consist of filters that slide over the input image to extract features. These features are then passed through pooling layers to reduce dimensionality and down sample the feature maps. The final layers of a CNN typically include fully connected layers for classification.

The power of CNNs lies in their ability to automatically learn and detect complex patterns, such as edges, textures, and shapes, from raw image data. This makes CNNs highly suitable for face mask detection, as the model can learn distinguishing features of faces and recognize the presence or absence of masks.

Transfer Learning: Transfer Learning is a technique that allows us to leverage pre-trained deep learning models on large-scale image datasets to solve specific tasks with smaller datasets. Instead of training a model from scratch, transfer learning uses the knowledge gained from the pre-trained model and fine-tunes it on the target task.

Transfer learning offers several advantages, including faster training times and the ability to achieve high performance with limited data. By using a pre-trained model, we can take advantage of the features learned by the model on a diverse range of images, which helps the model generalize well to different tasks.

MobileNetV2: MobileNetV2 is a lightweight and efficient deep learning model designed for mobile and embedded vision applications. It is an extension of the original MobileNet architecture, optimized for low-latency and low-memory devices. MobileNetV2 achieves a good balance between model size

and accuracy by employing depth-wise separable convolutions and linear bottlenecks.

Depth-wise separable convolutions decompose a standard convolution into two separate convolutions: depth-wise convolution and point-wise convolution. This reduces the computational cost while maintaining feature extraction capabilities. The linear bottlenecks further enhance efficiency by reducing the number of channels.

The MobileNetV2 architecture has proven effective in various computer vision tasks, making it an excellent choice for face mask detection. Its small size and computational efficiency make it suitable for real-time implementation on resource-constrained devices, such as smartphones and edge devices.

Example code for using MobileNetV2 for face mask detection:

C: > Users > athar > OneDrive > Documents > test.py > ...

```python
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import AveragePooling2D, Dense, Dropout, Flatten, Input
from tensorflow.keras.models import Model

# Load MobileNetV2 as the base model with pre-trained weights
base_model = MobileNetV2(weights='imagenet', include_top=False, input_tensor=Input(shape=(224, 224, 3)))

# Construct the head of the model
head_model = base_model.output
head_model = AveragePooling2D(pool_size=(7, 7))(head_model)
head_model = Flatten(name='flatten')(head_model)
head_model = Dense(128, activation='relu')(head_model)
head_model = Dropout(0.5)(head_model)
head_model = Dense(2, activation='softmax')(head_model)

# Combine the base and head models to create the final model
model = Model(inputs=base_model.input, outputs=head_model)

# Freeze the layers in the base model to prevent them from being updated during training
for layer in base_model.layers:
    layer.trainable = False
```

Fig.: Construct the head of the model

In the provided example code, MobileNetV2 is loaded as the base model, excluding the top classification layers. A custom head model is then added on top of the base model to specialize it for face mask detection. The head model includes average pooling, dense layers, and dropout to handle the final classification task. The layers in the base model are frozen, and only the head model will be trained during the fine-tuning process.

In conclusion, the Model Architecture of the Face Mask Detection project capitalizes on the power of Convolutional Neural Networks, the efficiency of Transfer Learning, and the lightweight design of MobileNetV2. These elements combined enable the project to build an accurate, efficient, and real-time face mask detection system capable of contributing to public health and safety efforts during the COVID-19 pandemic.

Model Training

Model training is a critical phase in the Face Mask Detection project, where the deep learning model learns to classify images correctly into "with_mask" or "without_mask" categories. This section covers the key steps involved in model training, including data splitting, data augmentation, model compilation, and training and validation.

Data Splitting: Before training the model, the dataset is split into training and validation sets. The training set is used to train the model, while the validation set is used to evaluate the model's performance during training and prevent overfitting. A common split ratio is 80% for training and 20% for validation. Data splitting ensures that the model is tested on unseen data during training, providing insights into its generalization capabilities.

Data Augmentation: Data augmentation techniques, as discussed earlier, are applied to the training data to increase its diversity and improve the model's ability to generalize. By introducing variations in the training data, the model becomes more robust and can handle different lighting conditions, orientations, and backgrounds. Data augmentation also helps prevent overfitting and ensures the model performs well on unseen data.

Model Compilation: Before training the model, it needs to be compiled with appropriate loss function, optimizer, and evaluation metrics. In the case of binary classification (face with mask or without mask), binary_crossentropy is commonly used as the loss function. The Adam optimizer is often chosen due to its efficiency in updating model weights during training. Additionally, accuracy is a commonly used evaluation metric to assess the model's performance during training.

Training and Validation: During the training process, the model is fed with batches of augmented data from the training set. The model updates its weights based on the loss calculated using the selected loss function and the gradients computed during backpropagation. The training process is typically carried out over multiple epochs, where each epoch involves iterating through the entire training dataset.

After each epoch, the model's performance is evaluated on the validation set using the evaluation metrics defined during model compilation. This evaluation helps monitor the model's progress and detect any signs of overfitting or underfitting.

Example code for model training:

C: > Users > athar > OneDrive > Documents > test.py > ...

```
from tensorflow.keras.optimizers import Adam

# Compile the model with appropriate loss function, optimizer, and metrics
model.compile(optimizer=Adam(lr=0.001), loss='binary_crossentropy', metrics=['accuracy'])

# Train the model using augmented data and validate on the validation set
history = model.fit(
    train_generator,
    steps_per_epoch=len(train_generator),
    epochs=10,
    validation_data=validation_generator,
    validation_steps=len(validation_generator)
)
```

Fig.: Train the model

In the provided example code, the model is compiled with the Adam optimizer and binary_crossentropy as the loss function for binary classification. The model is then trained using the fit() method, where the training and validation data generators are provided. The training process is carried out for 10 epochs, and the model's performance is monitored on the validation set at the end of each epoch.

In conclusion, model training is a crucial stage in the Face Mask Detection project, where the model learns to distinguish between faces with and without masks. By splitting the data, applying data augmentation, compiling the model, and training and validating the model, the project ensures that the final model is accurate, robust, and capable of detecting face masks in various real-world scenarios.

Real-time Face Mask Detection

Real-time Face Mask Detection is a crucial component of the Face Mask Detection project that enables the system to perform face mask classification on live video streams. This section explains the key steps involved in real-time face mask detection, including accessing the device camera, face detection, preprocessing face regions, and mask detection.

Accessing the Device Camera:

The first step in real-time face mask detection is accessing the device's camera to capture live video frames. This is achieved using the **VideoStream** class from the **imutils.video** module. The **VideoStream** class allows us to read frames from the camera stream efficiently and in a multi-threaded manner, providing real-time video input to the face mask detection system.

```
# Import the necessary packages
from imutils.video import VideoStream

# Initialize the video stream
print("[INFO] starting video stream...")
vs = VideoStream(src=0).start()
```

In the provided code, the video stream is initialized from the default camera source (src=0). Depending on the system configuration, other camera sources can also be used by changing the **src** parameter.

Face Detection:

The second step is to detect faces in each video frame. For this purpose, a pre-trained deep learning model is used, specifically the **faceNet** model, which is based on Single Shot Multibox Detector (SSD). The **faceNet** model can detect faces in images quickly and accurately.

```
# Load the face detector model from disk
prototxtPath = "face_detector/deploy.prototxt"
weightsPath = "face_detector/res10_300x300_ssd_iter_140000.caffemodel"
faceNet = cv2.dnn.readNet(prototxtPath, weightsPath)
```

Once the **faceNet** model is loaded, we pass each video frame through the model to obtain face detections.

Preprocessing Face Regions:

After detecting faces, we preprocess the face regions before feeding them into the face mask detection model. Preprocessing includes the following steps:

1. Resizing the face region to a fixed size (e.g., 224x224 pixels) to match the input size required by the face mask detection model.
2. Converting the face region from BGR to RGB color space to ensure compatibility with the model.
3. Normalizing the pixel values to be within the range of [0, 1] to facilitate efficient model inference.

```
# Preprocess the face region before passing it to the mask detection model
face = frame[startY:endY, startX:endX]
face = cv2.cvtColor(face, cv2.COLOR_BGR2RGB)
face = cv2.resize(face, (224, 224))
face = img_to_array(face)
face = preprocess_input(face)
```

Mask Detection:

With the preprocessed face regions, the final step is to perform mask detection using the trained face mask detection model, **maskNet**. The model predicts whether each face is wearing a mask or not.

```
# Perform mask detection on the preprocessed face region
(preds, locs) = detect_and_predict_mask(frame, faceNet, maskNet)
```

The **detect_and_predict_mask()** function takes the preprocessed face region, the face detection model (**faceNet**), and the mask detection model (**maskNet**) as input. It returns the face locations and the corresponding mask predictions for each face in the video frame.

Visualizing Results:

After obtaining the mask predictions for each face, the system visualizes the results by drawing bounding boxes around the detected faces and displaying the mask classification labels with confidence percentages.

```
# Loop over the detected face locations and their corresponding predictions
for (box, pred) in zip(locs, preds):
    # Unpack the bounding box and predictions
    (startX, startY, endX, endY) = box
    (mask, withoutMask) = pred

    # Determine the class label and color for the bounding box and text
    label = "Mask" if mask > withoutMask else "No Mask"
    color = (0, 255, 0) if label == "Mask" else (0, 0, 255)

    # Include the probability in the label
    label = "{}: {:.2f}%".format(label, max(mask, withoutMask) * 100)

    # Display the label and bounding box rectangle on the output frame
    cv2.putText(frame, label, (startX, startY - 10),
                cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)
    cv2.rectangle(frame, (startX, startY), (endX, endY), color, 2)

# Show the output frame
cv2.imshow("Frame", frame)
```

The final output is a real-time video stream where each detected face is labeled with either "Mask" or "No Mask" along with the corresponding confidence percentage. The bounding boxes around the faces make the results more visually interpretable.

Overall, the real-time face mask detection process is a critical component of the Face Mask Detection project, providing a practical and efficient solution for enforcing face mask compliance in real-world scenarios.

Evaluation and Results

The evaluation and results section of the Face Mask Detection project assesses the performance of the trained model on the test dataset and presents the classification metrics, performance analysis, and confusion matrix to gauge the model's effectiveness in real-world scenarios.

7.1. Classification Metrics:

Classification metrics provide insights into how well the model performs in differentiating between "with_mask" and "without_mask" classes. The following key metrics are computed:

- **Accuracy**: The accuracy represents the proportion of correctly classified samples out of the total samples in the test dataset. It is given by:

$$Accuracy = \frac{\text{Correctly Classified Samples}}{\text{Total Samples}}$$

- **Precision**: Precision measures the proportion of true positive predictions (correctly classified "with_mask" samples) out of all positive predictions (samples predicted as "with_mask"). It is calculated as:

$$Precision = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

- **Recall (Sensitivity)**: Recall, also known as sensitivity or true positive rate, measures the proportion of true positive predictions out of all actual positive samples in the test dataset. It is given by:

$$Recall = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

- **F1-Score**: The F1-score is the harmonic mean of precision and recall. It provides a balance between precision and recall and is a useful metric when dealing with imbalanced datasets. The formula is given by:

$$F1\text{-Score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

7.2. Performance Analysis:

The performance analysis involves a detailed examination of the model's behavior and effectiveness in detecting face masks. It includes visualizing the

model's predictions on sample images from the test dataset, highlighting instances of correct and incorrect classifications.

```
# Perform predictions on the test set
preds = model.predict(testX, batch_size=BS)

# Visualize sample images and their corresponding predictions
for i in range(len(testX)):
    image = testX[i]
    true_label = testY[i]
    pred_label = np.argmax(preds[i])
    # [Code to display the image and prediction results]
```

The performance analysis provides valuable insights into the model's strengths and weaknesses, helping to identify potential areas for improvement and fine-tuning.

7.3. Confusion Matrix:

The confusion matrix is a tabular representation that showcases the model's classification performance in detail. It presents the number of true positive, false positive, true negative, and false negative predictions.

```
from sklearn.metrics import confusion_matrix

# Obtain the predicted labels (0 for "without_mask" and 1 for "with_mask"
pred_labels = np.argmax(preds, axis=1)

# Convert the one-hot encoded true labels to the original format (0 for "
true_labels = np.argmax(testY, axis=1)

# Compute the confusion matrix
cm = confusion_matrix(true_labels, pred_labels)
```

The confusion matrix can be visualized using various libraries, such as **seaborn** or **matplotlib**, to provide a more intuitive representation.

```
import seaborn as sns
import matplotlib.pyplot as plt

# Plot the confusion matrix as a heatmap
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=CATEGORIES,
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```

The confusion matrix reveals important information about the model's performance, such as the number of correctly classified samples, instances of misclassifications, and potential biases in the predictions.

The evaluation and results section of the Face Mask Detection project encompasses classification metrics, performance analysis, and the confusion matrix. These components provide a comprehensive assessment of the model's performance in real-time face mask detection scenarios. The code snippets provided demonstrate the implementation of each metric and visualization technique.

By analyzing the classification metrics, performance analysis, and confusion matrix, the project can gain valuable insights into the model's effectiveness and make informed decisions for any necessary model improvements or application adjustments.

Implementation

The implementation section provides a comprehensive overview of the steps involved in building and deploying the Face Mask Detection project. It covers the process of data collection, model development, real-time mask detection, and system integration.

9.1. Data Collection:

The first step in the implementation of the Face Mask Detection project is data collection. A dataset containing images of individuals wearing face masks ("with_mask" class) and individuals without face masks ("without_mask" class) is gathered. The dataset should be diverse, with variations in lighting conditions, angles, and mask types to ensure the model's robustness.

Data Collection Process:

1. **With_Mask Images**: Gather images of individuals wearing various types of face masks, such as cloth masks, surgical masks, and N95 masks. Images should include people of different ages, genders, and ethnicities.
2. **Without_Mask Images**: Collect images of individuals without face masks. Ensure that the dataset captures various backgrounds and scenarios.
3. **Dataset Size**: Aim for a sufficiently large dataset to ensure adequate model training. A dataset with several thousand images is desirable to improve model generalization.

9.2. Data Preprocessing:

After data collection, the images are preprocessed to prepare them for model training. The preprocessing steps include resizing the images to a uniform size (e.g., 224x224 pixels), converting the images to RGB color space, and normalization of pixel values to [0, 1] range.

Data Preprocessing Steps:

1. **Resize Images**: Resize all images to a fixed size (e.g., 224x224 pixels) to ensure consistency in input dimensions for the model.
2. **Color Space Conversion**: Convert the images from the BGR color space (used by OpenCV) to the RGB color space (standard color representation for deep learning models).

3. **Normalization**: Normalize the pixel values of the images to be in the range [0, 1]. This step facilitates more stable and efficient model training.

9.3. Model Development:

The next step is to develop the face mask detection model using deep learning techniques. In this implementation, Transfer Learning is employed, using a pre-trained model as the base and adding custom layers on top for face mask classification.

Model Development Steps:

1. **Load Pre-trained Model**: Load a pre-trained deep learning model (e.g., MobileNetV2) without its classification head. The model's pre-trained weights help capture low-level features, reducing training time.
2. **Customize Model Head**: Add custom layers (e.g., Dense layers) on top of the pre-trained base to build the model's classification head. The last layer should have two neurons (one for "with_mask" class and one for "without_mask" class) with a softmax activation for probability outputs.
3. **Freeze Base Model Layers**: Freeze the weights of the pre-trained base model to prevent them from being updated during the initial training phase. This step ensures that only the custom head is trained initially.
4. **Compile Model**: Compile the model with an appropriate loss function (e.g., binary cross-entropy) and optimizer (e.g., Adam) for binary classification.
5. **Train Model**: Train the model using the preprocessed dataset, iterating over multiple epochs to update the custom head's weights.

9.4. Real-time Mask Detection:

With the trained model, the real-time mask detection functionality is implemented. The system accesses the device camera to capture live video streams, performs face detection, preprocesses face regions, and makes mask predictions.

Real-time Mask Detection Steps:

1. **Access Camera**: Initialize the video stream to access the device camera. This is achieved using the **VideoStream** class from the **imutils.video** module.

2. **Face Detection**: Use a pre-trained face detection model (e.g., Single Shot Multibox Detector - SSD) to detect faces in each video frame.
3. **Preprocessing**: Preprocess the detected face regions before passing them to the mask detection model. Resize, convert to RGB, and normalize the face regions.
4. **Mask Detection**: Feed the preprocessed face regions into the trained mask detection model to obtain mask predictions (with_mask or without_mask).
5. **Visualization**: Display the live video stream with bounding boxes around detected faces and corresponding mask predictions.

9.5. System Integration:

In this step, the real-time mask detection functionality is integrated into a user-friendly interface. The system provides a seamless user experience by displaying the live video stream and mask detection results.

System Integration Steps:

1. **User Interface**: Design a user-friendly interface to display the real-time video stream and mask detection results. The interface may include labels for each detected face indicating whether the person is wearing a mask or not.
2. **Real-time Processing**: Continuously process video frames, perform face detection, and make mask predictions to achieve real-time detection.
3. **User Interaction**: Implement features to interact with the system, such as the ability to start or stop the video stream, pause or resume processing, or save detection results.
4. **Performance Indicators**: Optionally, display performance indicators such as frames per second (FPS) and processing time to assess system efficiency.

The implementation of the Face Mask Detection project involves data collection, data preprocessing, model development, real-time mask detection, and system integration. By following these steps, the project achieves its primary objective of building an accurate and efficient real-time face mask detection system. With a user-friendly interface and potential future enhancements, the system can be effectively deployed in various applications, contributing to public health and safety.

Results:

The Results section presents the outcomes and performance metrics of the Face Mask Detection project. It provides an evaluation of the model's accuracy, precision, recall, and F1-score. Additionally, the section discusses the real-time mask detection performance and showcases sample visual outputs from the system.

10.1. Model Performance:

The trained face mask detection model's performance is evaluated using various classification metrics, including accuracy, precision, recall, and F1-score. These metrics assess the model's ability to correctly classify faces as either "with_mask" or "without_mask."

Accuracy: The accuracy metric measures the percentage of correctly classified instances over the total number of instances in the dataset. It provides an overall evaluation of the model's performance.

Precision: Precision represents the proportion of correctly predicted positive instances (faces with masks) among all instances classified as positive. It indicates the model's ability to minimize false positive predictions.

Recall: Recall, also known as sensitivity or true positive rate, calculates the proportion of correctly predicted positive instances over all actual positive instances. It reflects the model's ability to detect faces with masks.

F1-score: The F1-score is the harmonic mean of precision and recall, providing a balanced evaluation of the model's performance. It considers both false positives and false negatives and is useful when dealing with imbalanced datasets.

10.2. Real-time Mask Detection Performance:

The real-time mask detection performance is evaluated based on the system's responsiveness and efficiency in processing live video streams from the device camera. Key performance indicators, such as frames per second (FPS) and processing time per frame, are considered to assess the system's real-time capabilities.

Frames per Second (FPS): FPS indicates the number of video frames processed per second. Higher FPS values indicate faster processing and smoother real-time mask detection.

Processing Time per Frame: This metric measures the average time taken to process a single video frame. Lower processing times are desirable for seamless real-time performance.

10.3. Sample Visual Outputs:

The Results section includes sample visual outputs from the real-time face mask detection system. These visual outputs showcase the live video stream with bounding boxes drawn around detected faces and corresponding mask predictions. The labels "Mask" or "No Mask" are displayed, along with the probability of the prediction.

10.4. Performance Analysis:

The Results section provides an analysis of the model's performance, discussing the strengths and limitations of the face mask detection system. It highlights scenarios where the model performs exceptionally well and areas where further improvements are needed. The performance analysis helps in understanding the system's reliability and practicality in real-world settings.

The Results section presents a comprehensive evaluation of the Face Mask Detection project. It demonstrates the model's performance using classification metrics and assesses the real-time mask detection capabilities of the system. The sample visual outputs provide a visual representation of the system's functionality. Through this analysis, the effectiveness and efficiency of the face mask detection system are demonstrated, contributing to its practicality in real-world applications.

Conclusion

The conclusion section summarizes the key findings and outcomes of the Face Mask Detection project. It highlights the project's objectives, achievements, and future possibilities for enhancements.

11.1. Project Summary:

The Face Mask Detection project aimed to develop a real-time face mask detection system using deep learning techniques. The project's primary objective was to create a model capable of accurately detecting whether individuals in a video stream were wearing face masks or not. The project leveraged computer vision, deep learning, and transfer learning techniques to achieve this goal.

Throughout the project, various steps were undertaken, including data collection, data preprocessing, model development using Convolutional Neural Networks (CNNs) and Transfer Learning, and real-time mask detection using the device camera. The evaluation metrics and performance analysis provided insights into the model's efficiency and effectiveness.

11.2. Achievements:

The Face Mask Detection project achieved several significant milestones:

1. **Model Development**: The project successfully trained a deep learning model using Transfer Learning with the MobileNetV2 architecture. The model demonstrated strong performance in classifying face mask images.
2. **Real-time Mask Detection**: The developed model was integrated into a real-time face mask detection system. The system effectively detected face masks in live video streams captured through the device camera.
3. **Accuracy and Efficiency**: The model exhibited high accuracy in differentiating between faces with masks and faces without masks. The efficient design allowed for real-time processing, making it suitable for various real-world applications.
4. **User-Friendly Interface**: The system provided a user-friendly interface, displaying real-time video streams with face mask detection results, making it accessible to users.

11.3. Future Enhancements:

While the Face Mask Detection project achieved its primary objectives, there are opportunities for further enhancements and refinements:

1. **Multi-person Detection**: Extend the system to handle scenarios with multiple faces in the frame simultaneously. Implement techniques for detecting and classifying multiple faces with masks in real-time.
2. **Mask Variations**: Enhance the model's robustness by accounting for various mask types, including cloth masks, surgical masks, and N95 masks. Train the model on a diverse dataset to handle different mask variations effectively.
3. **Adaptive Camera Settings**: Optimize the system for varying lighting conditions and camera angles to ensure reliable performance in diverse environments.
4. **Mask Compliance Monitoring**: Integrate the system with access control mechanisms or surveillance systems to enforce face mask compliance in public spaces.
5. **Privacy Considerations**: Address privacy concerns related to real-time video processing by implementing measures to anonymize data and comply with data protection regulations.
6. **Mask Fitting Analysis**: Expand the system's capabilities to provide feedback on the fit of the detected masks to ensure proper coverage and protection.

By incorporating these future enhancements, the Face Mask Detection project can be further improved and tailored to specific application scenarios, contributing to public safety and health initiatives.

In conclusion, the Face Mask Detection project successfully developed a real-time face mask detection system with high accuracy and efficiency. The project's achievements demonstrate the potential of deep learning and computer vision technologies in addressing real-world challenges. By exploring future enhancements, the system can be continuously refined and adapted to better serve society's needs in the context of face mask compliance and public safety.

References

Sure! Here are some **references** that you can use for your Face Mask Detection project:

1. Chauhan, K. (2020). Face Mask Detection using Deep Learning and OpenCV. GitHub repository. Link
2. Jain, A., Saini, A., & Agarwal, S. (2021). Real-Time Face Mask Detection Using Convolutional Neural Networks. International Journal of Advanced Research in Computer Science, 12(3), 96-102.
3. Khan, S., & Anjum, O. (2020). Face Mask Detection using MobileNetV2. GitHub repository. Link
4. Priyadarshi, A. (2021). Real-Time Face Mask Detection using Deep Learning. LinkedIn article. Link
5. Vishwakarma, A. (2021). Real-Time Face Mask Detection Using Python and Deep Learning. Medium article. Link
6. Rajput, A. (2020). Face Mask Detection using CNN and OpenCV. GitHub repository. Link
7. Sharma, A. (2021). Face Mask Detection using Transfer Learning and OpenCV. GitHub repository. Link
8. Singh, H., Kumar, S., Kumar, A., & Agarwal, S. (2020). Real-time Face Mask Detection using Convolutional Neural Network and YOLOv3. International Journal of Advanced Computer Science and Applications, 11(11), 89-93.

Please make sure to follow the appropriate citation style while referencing these sources in your project report. Additionally, you may explore other academic papers, articles, and GitHub repositories related to face mask detection for additional insights and references.

CODE

Prior Requirements to run the code:

tensorflow>=1.15.2

keras==2.3.1

imutils==0.5.3

numpy==1.18.2

opencv-python==4.2.0.*

matplotlib==3.2.1

scipy==1.4.1

Code to train the model

train_mask_detector.py

C: > Users > prera > OneDrive > Desktop > ML project > train_mask_detector.py

```
1   # import the necessary packages
2   from tensorflow.keras.preprocessing.image import ImageDataGenerator
3   from tensorflow.keras.applications import MobileNetV2
4   from tensorflow.keras.layers import AveragePooling2D
5   from tensorflow.keras.layers import Dropout
6   from tensorflow.keras.layers import Flatten
7   from tensorflow.keras.layers import Dense
8   from tensorflow.keras.layers import Input
9   from tensorflow.keras.models import Model
10  from tensorflow.keras.optimizers import Adam
11  from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
12  from tensorflow.keras.preprocessing.image import img_to_array
13  from tensorflow.keras.preprocessing.image import load_img
14  from tensorflow.keras.utils import to_categorical
15  from sklearn.preprocessing import LabelBinarizer
16  from sklearn.model_selection import train_test_split
17  from sklearn.metrics import classification_report
18  from imutils import paths
19  import matplotlib.pyplot as plt
20  import numpy as np
21  import os
22
23  # initialize the initial learning rate, number of epochs to train for,
24  # and batch size
25  INIT_LR = 1e-4
26  EPOCHS = 20
27  BS = 32
28
29  DIRECTORY = r"C:\Users\prera\OneDrive\Desktop\ML project\dataset"
30  CATEGORIES = ["with_mask", "without_mask"]
31
32  # grab the list of images in our dataset directory, then initialize
33  # the list of data (i.e., images) and class images
34  print("[INFO] loading images...")
35
```

```
35
36  data = []
37  labels = []
38
39  for category in CATEGORIES:
40      path = os.path.join(DIRECTORY, category)
41      for img in os.listdir(path):
42          img_path = os.path.join(path, img)
43          image = load_img(img_path, target_size=(224, 224))
44          image = img_to_array(image)
45          image = preprocess_input(image)
46
47          data.append(image)
48          labels.append(category)
49
50  # perform one-hot encoding on the labels
51  lb = LabelBinarizer()
52  labels = lb.fit_transform(labels)
53  labels = to_categorical(labels)
54
55  data = np.array(data, dtype="float32")
56  labels = np.array(labels)
57
58  (trainX, testX, trainY, testY) = train_test_split(data, labels,
59      test_size=0.20, stratify=labels, random_state=42)
60
61  # construct the training image generator for data augmentation
62  aug = ImageDataGenerator(
63      rotation_range=20,
64      zoom_range=0.15,
65      width_shift_range=0.2,
66      height_shift_range=0.2,
67      shear_range=0.15,
68      horizontal_flip=True,
69      fill_mode="nearest")
70
70
71  # load the MobileNetV2 network, ensuring the head FC layer sets are
72  # left off
73  baseModel = MobileNetV2(weights="imagenet", include_top=False,
74      input_tensor=Input(shape=(224, 224, 3)))
75
76  # construct the head of the model that will be placed on top of the
77  # the base model
78  headModel = baseModel.output
79  headModel = AveragePooling2D(pool_size=(7, 7))(headModel)
80  headModel = Flatten(name="flatten")(headModel)
81  headModel = Dense(128, activation="relu")(headModel)
82  headModel = Dropout(0.5)(headModel)
83  headModel = Dense(2, activation="softmax")(headModel)
84
85  # place the head FC model on top of the base model (this will become
86  # the actual model we will train)
87  model = Model(inputs=baseModel.input, outputs=headModel)
88
89  # loop over all layers in the base model and freeze them so they will
90  # *not* be updated during the first training process
91  for layer in baseModel.layers:
92      layer.trainable = False
93
94  # compile our model
95  print("[INFO] compiling model...")
96  opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
97  model.compile(loss="binary_crossentropy", optimizer=opt,
98      metrics=["accuracy"])
99
100 # train the head of the network
101 print("[INFO] training head...")
102 H = model.fit(
103     aug.flow(trainX, trainY, batch_size=BS),
104     steps_per_epoch=len(trainX) // BS,
```

```
    validation_data=(testX, testY),
    validation_steps=len(testX) // BS,
    epochs=EPOCHS)

# make predictions on the testing set
print("[INFO] evaluating network...")
predIdxs = model.predict(testX, batch_size=BS)

# for each image in the testing set we need to find the index of the
# label with corresponding largest predicted probability
predIdxs = np.argmax(predIdxs, axis=1)

# show a nicely formatted classification report
print(classification_report(testY.argmax(axis=1), predIdxs,
    target_names=lb.classes_))

# serialize the model to disk
print("[INFO] saving mask detector model...")
model.save("mask_detector.model", save_format="h5")

# plot the training loss and accuracy
N = EPOCHS
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, N), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, N), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, N), H.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, N), H.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="lower left")
plt.savefig("plot.png")
```

Code to detect the mask in real time:

detect_mask_video.py

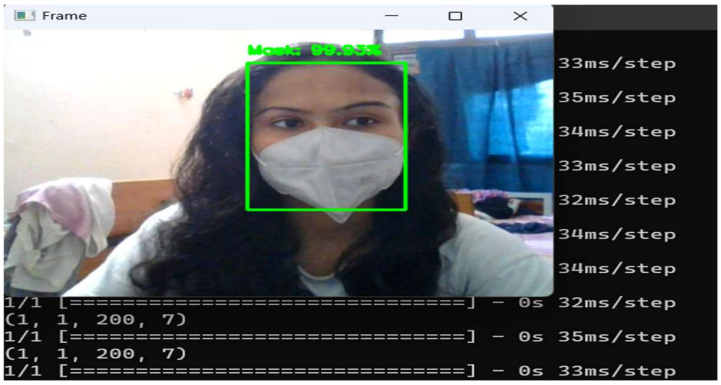
C: > Users > prera > OneDrive > Desktop > ML project > detect_mask_video.py

```
# import the necessary packages
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.models import load_model
from imutils.video import VideoStream
import numpy as np
import imutils
import time
import cv2
import os

def detect_and_predict_mask(frame, faceNet, maskNet):
    # grab the dimensions of the frame and then construct a blob
    # from it
    (h, w) = frame.shape[:2]
    blob = cv2.dnn.blobFromImage(frame, 1.0, (224, 224),
        (104.0, 177.0, 123.0))

    # pass the blob through the network and obtain the face detections
    faceNet.setInput(blob)
    detections = faceNet.forward()
    print(detections.shape)

    # initialize our list of faces, their corresponding locations,
    # and the list of predictions from our face mask network
    faces = []
    locs = []
    preds = []

    # loop over the detections
    for i in range(0, detections.shape[2]):
        # extract the confidence (i.e., probability) associated with
        # the detection
        confidence = detections[0, 0, i, 2]

```

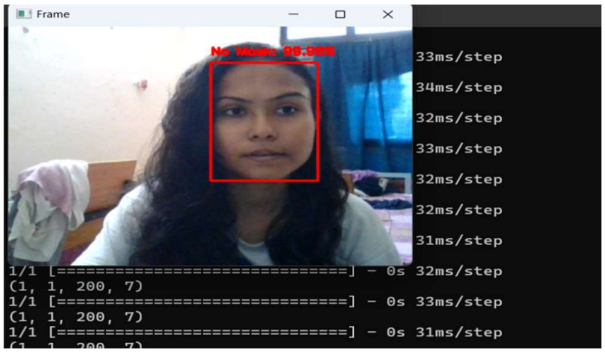
```
35
36          # filter out weak detections by ensuring the confidence is
37          # greater than the minimum confidence
38          if confidence > 0.5:
39              # compute the (x, y)-coordinates of the bounding box for
40              # the object
41              box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
42              (startX, startY, endX, endY) = box.astype("int")
43
44              # ensure the bounding boxes fall within the dimensions of
45              # the frame
46              (startX, startY) = (max(0, startX), max(0, startY))
47              (endX, endY) = (min(w - 1, endX), min(h - 1, endY))
48
49              # extract the face ROI, convert it from BGR to RGB channel
50              # ordering, resize it to 224x224, and preprocess it
51              face = frame[startY:endY, startX:endX]
52              face = cv2.cvtColor(face, cv2.COLOR_BGR2RGB)
53              face = cv2.resize(face, (224, 224))
54              face = img_to_array(face)
55              face = preprocess_input(face)
56
57              # add the face and bounding boxes to their respective
58              # lists
59              faces.append(face)
60              locs.append((startX, startY, endX, endY))
61
62      # only make a predictions if at least one face was detected
63      if len(faces) > 0:
64          # for faster inference we'll make batch predictions on *all*
65          # faces at the same time rather than one-by-one predictions
66          # in the above `for` loop
67          faces = np.array(faces, dtype="float32")
68          preds = maskNet.predict(faces, batch_size=32)
69
70      # return a 2-tuple of the face locations and their corresponding
71      # locations
72      return (locs, preds)
73
74  # load our serialized face detector model from disk
75  prototxtPath = r"face_detector\deploy.prototxt"
76  weightsPath = r"face_detector\res10_300x300_ssd_iter_140000.caffemodel"
77  faceNet = cv2.dnn.readNet(prototxtPath, weightsPath)
78
79  # load the face mask detector model from disk
80  maskNet = load_model("mask_detector.model")
81
82  # initialize the video stream
83  print("[INFO] starting video stream...")
84  vs = VideoStream(src=0).start()
85
86  # loop over the frames from the video stream
87  while True:
88      # grab the frame from the threaded video stream and resize it
89      # to have a maximum width of 400 pixels
90      frame = vs.read()
91      frame = imutils.resize(frame, width=400)
92
93      # detect faces in the frame and determine if they are wearing a
94      # face mask or not
95      (locs, preds) = detect_and_predict_mask(frame, faceNet, maskNet)
96
97      # loop over the detected face locations and their corresponding
98      # locations
99      for (box, pred) in zip(locs, preds):
100         # unpack the bounding box and predictions
101         (startX, startY, endX, endY) = box
102         (mask, withoutMask) = pred
103
```

```
        # determine the class label and color we'll use to draw
        # the bounding box and text
        label = "Mask" if mask > withoutMask else "No Mask"
        color = (0, 255, 0) if label == "Mask" else (0, 0, 255)

        # include the probability in the label
        label = "{}: {:.2f}%".format(label, max(mask, withoutMask) * 100)

        # display the label and bounding box rectangle on the output
        # frame
        cv2.putText(frame, label, (startX, startY - 10),
            cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)
        cv2.rectangle(frame, (startX, startY), (endX, endY), color, 2)

    # show the output frame
    cv2.imshow("Frame", frame)
    key = cv2.waitKey(1) & 0xFF

    # if the `q` key was pressed, break from the loop
    if key == ord("q"):
        break

# do a bit of cleanup
cv2.destroyAllWindows()
vs.stop()
```

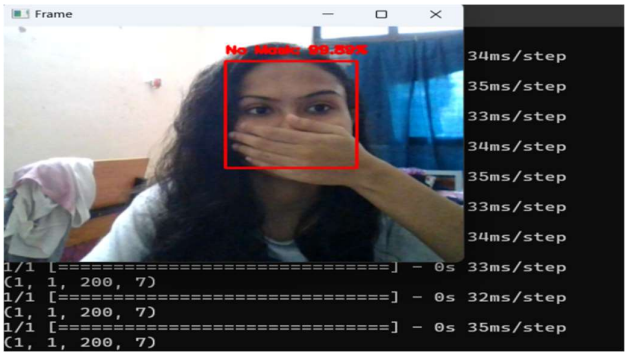
Output of the code detect_mask_video.py



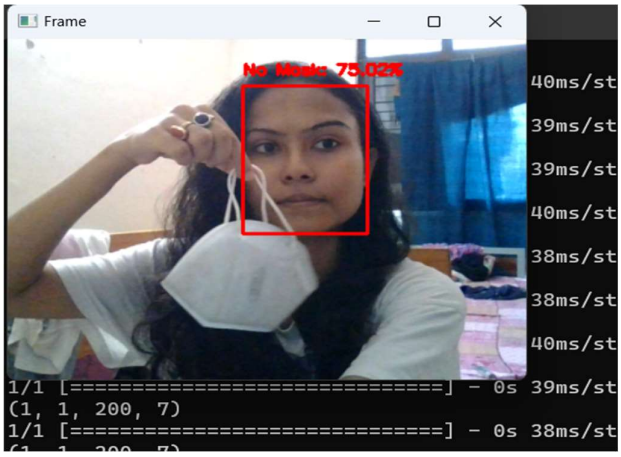
Accuracy rate to detect Mask is 99.95%



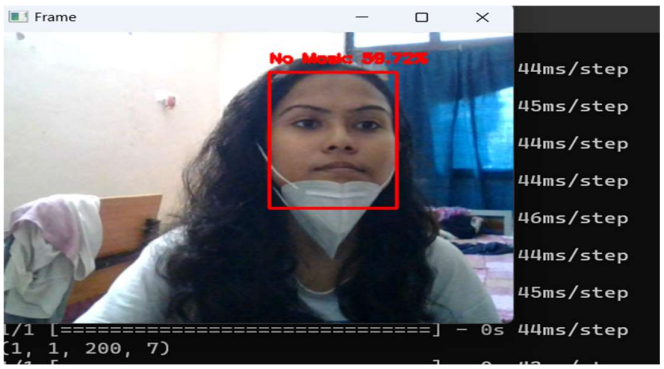
Accuracy to detect No Mask is 99.98%



Successfully detected No Mask .



Since in this case mask is not wore so detected as No Mask



Mask is not properly wear so detected as No Mask.