Complex Engineering Problem
Computer Vision (CS-436)

Title: Face Mask Detection

Group Members	**Syeda Maria (CS-21010)** **Aqsa Asif (CS-21039)**
Year	**B.E.**
Batch	**2021**
Course Title & **Course Code**	**Computer Vision** **(CS-436)**
Submitted To	**Ms. Fauzia Yasir**

GRADING RUBRIC

Software Use Rubric				
Skill Sets	Extent of Achievement			
	0	1	2	3
To what extent has the student implemented the solution?	The solution has not been implemented.	The solution has syntactic and logical errors.	The solution has syntactic or logical errors.	The solution is syntactically and logically sound for the stated problem parameters.
How efficient is the proposed model?	The model does not address the problem adequately.	The model exhibits redundancy and partially covers the problem.	The model exhibits redundancy or partially covers the problem.	The model is free of redundancy and covers all aspects of the problem.
How did the student answer questions relevant to the solution?	The student answered none of the questions.	The student answered less than half of the questions.	The student answered more than half but not all of the questions.	The student answered all the questions.
To what extent the student use libraries, tools and download database of images?	The student is unable with the interface.	The student is familiar few library function.	-	The student is proficient with the interface and able to use libraries and download dataset.

Weighted CLO Score	
Remarks	
Instructor's Signature with Date	

Table of Contents

Title: Face Mask Detection **1**

1 Application Background and Introduction **5**

1.1 Background *5*

1.2 Problem Statement *5*

1.3 Project Objective *5*

1.4 Motivation *6*

2 Dataset Overview **6**

2.1 Dataset Used *6*

2.2 Dataset Description *6*

7

2.3 Dataset Distribution *7*

3 Preprocessing for CV Pipeline **7**

3.1 Preprocessing for Mask Detector *7*

3.1.1 Dataset Loading and Labeling 7

3.1.2 Image Resizing and Conversion: 8

3.1.3 Preprocessing for MobileNetV2 8

3.1.4 Label Binarization and One-Hot Encoding: 8

3.1.5 Data Splitting: 9

3.1.6 Data Augmentation: 9

10

3.2 Preprocessing for Mask Video Detector *10*

3.2.1 Frame to Blob (for face detection model) 10

3.2.2 Resize Frame 10

3.2.3 Extract and Prepare Detected Faces (for mask classifier) 11

4 AI/ML Model Architecture **11**

4.1 Base Model: MobileNetV2 .... *11*

4.2 Custom Head Model *12*

4.3 Compilation and Optimization *13*

4.4 Training Strategy *13*

4.4.1 Transfer Learning with Frozen Base 13

4.4.2 Regularization 13

4.4.3 Training Parameters 14

4.4.4 Evaluation During Training 14

5 Experimental Setup 14
5.1 Software and Libraries: 14
5.2 Dataset: 14
5.3 Preprocessing: 14
5.4 Training Configuration: 15
5.5 Evaluation: 15
6 Result 15
6.1 Overall Performance Metrics 15
6.2 ROC Curve 17
6.3 Classification Report 17
6.4 Confusion Matrix 18
7 Post Processing 18
7.1.1 Interpreting the Predictions 18
7.2 Drawing Bounding Boxes and Labels 19
7.3 Displaying the Processed Frame 19
7.4 Handling Exit Conditions 20
8 Screenshot of Video Demo 20

1 Application Background and Introduction

1.1 Background

The COVID-19 pandemic brought unprecedented challenges to global health systems, economies, and everyday life. Among the most effective safety measures recommended by health organizations such as the WHO and CDC was the use of face masks to reduce airborne virus transmission. As a result, wearing masks became mandatory in public spaces worldwide.

However, ensuring compliance with mask mandates—especially in crowded or high-risk environments like airports, hospitals, public transport stations, and educational institutions—proved difficult. Manual monitoring is not only resource-intensive but also inconsistent and prone to human error.

In response, artificial intelligence (AI) and computer vision have provided innovative ways to automate compliance monitoring. Deep learning models, particularly Convolutional Neural Networks (CNNs), have shown exceptional performance in image classification tasks. This project leverages **MobileNetV2**, a lightweight and efficient deep learning model, to build a real-time face mask detection system that can be deployed in real-world scenarios with limited computational resources.

1.2 Problem Statement

Although wearing face masks is a simple and effective method to prevent the spread of contagious viruses, ensuring compliance on a large scale remains a challenge. Manual enforcement is not scalable and may lead to oversight or inconsistent application of safety protocols. There is a pressing need for an automated, real-time system that can detect whether a person is wearing a face mask or not, to help maintain public health standards effectively and efficiently.

1.3 Project Objective

This project aims to design and implement a face mask detection system using deep learning techniques. The key objectives are:

- To classify images of people into two categories: **with mask** and **without mask**.
- To use **transfer learning** with the **MobileNetV2** architecture, pre-trained on ImageNet, for efficient model training.
- To apply **data augmentation** techniques to enhance model generalization and robustness.
- To train, evaluate, and save the model for potential deployment in real-time applications, such as integration into CCTV systems or mobile-based solutions.

1.4 Motivation

The motivation for this project is grounded in the urgent need for scalable safety solutions in response to the COVID-19 pandemic and any similar future outbreaks. An automated face mask detection system can:

- Relieve the burden of manual monitoring and enforcement.
- Enhance public safety in crowded and high-risk areas.
- Be easily deployed in low-resource settings, especially when using lightweight models like MobileNetV2.
- Contribute to the broader adoption of AI in public health infrastructure, improving resilience in the face of future pandemics or public safety crises.

2 Dataset Overview

2.1 Dataset Used

We selected the ***Face Mask Detection Dataset*** on available on **Kaggle**

2.2 Dataset Description

- The dataset contains **3,833 high-quality RGB images** of human faces.

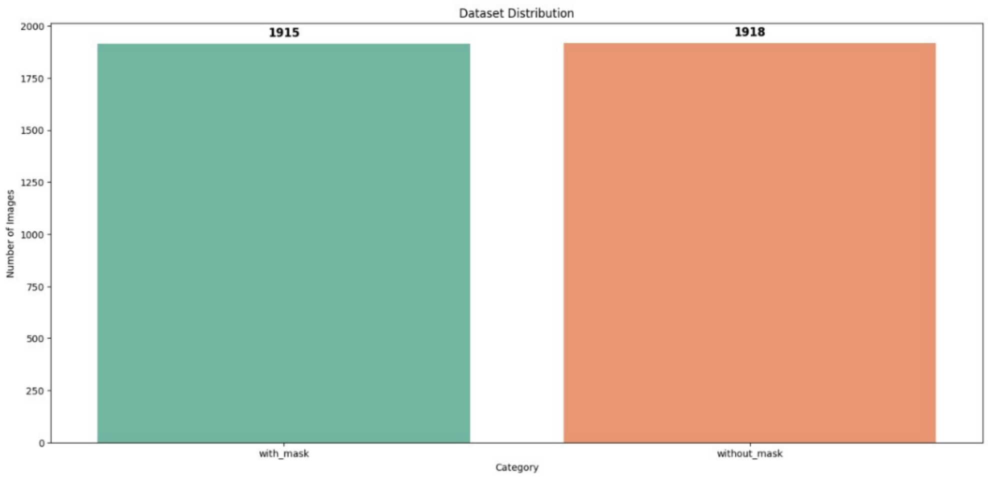
  It is **well-balanced**:

  - **1,915** images with masks
  - **1,918** images without masks
- A balanced dataset helps **reduce bias** and **improves model accuracy**.
- It includes a variety of:
  - **Face angles**
  - **Lighting conditions**
  - **Skin tones**
  - **Backgrounds**
- Previous experiments on this dataset showed up to **96% validation accuracy** using a custom CNN.
- It is ideal for training **lightweight models like MobileNetV2**, which we plan to use in our project.

(a) With Mask

(b) Without Mask

2.3 Dataset Distribution



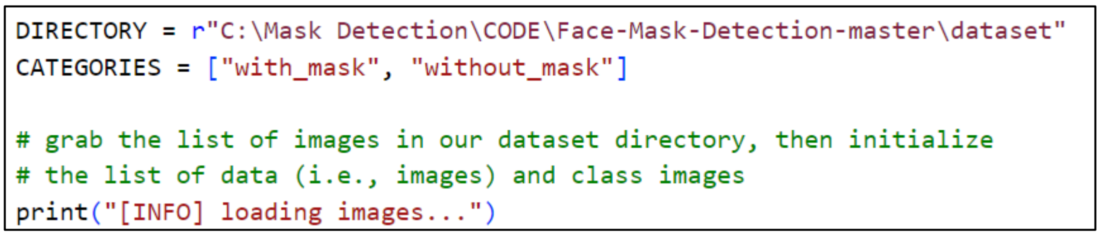
3 Preprocessing for CV Pipeline

The preprocessing phase is a critical component of the computer vision pipeline. In this project, it ensures that the input images are standardized, augmented, and properly labeled to train an accurate and generalizable face mask detection model. The preprocessing steps used are as follows:

3.1 Preprocessing for Mask Detector

3.1.1 Dataset Loading and Labeling

- The dataset consists of two categories: `with_mask` and `without_mask`.
- Images are loaded from these category folders using the `load_img()` function, and their corresponding labels are stored in a list.



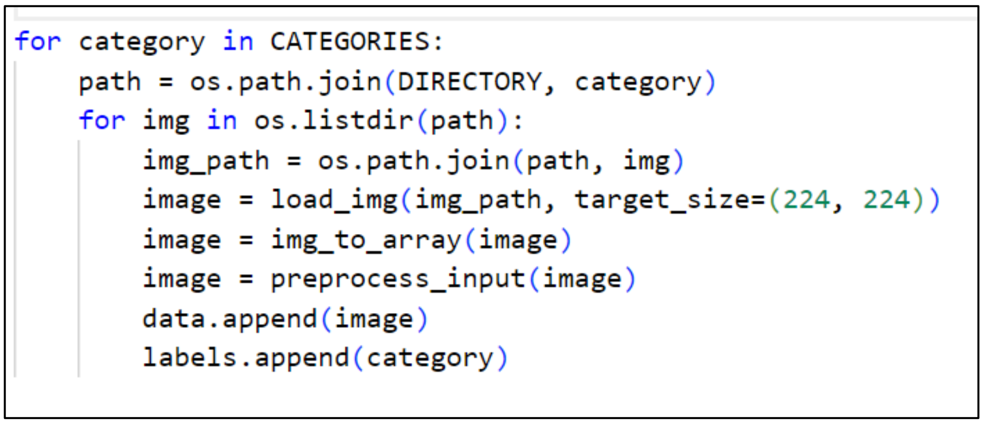
```
DIRECTORY = r"C:\Mask Detection\CODE\Face-Mask-Detection-master\dataset"
CATEGORIES = ["with_mask", "without_mask"]

# grab the list of images in our dataset directory, then initialize
# the list of data (i.e., images) and class images
print("[INFO] loading images...")
```

3.1.2 Image Resizing and Conversion:

- Each image is resized to a fixed shape of **224x224 pixels**, matching the input requirements of the MobileNetV2 architecture.
- Images are converted from PIL format to NumPy arrays using `img_to_array()` for further processing.

3.1.3 Preprocessing for MobileNetV2

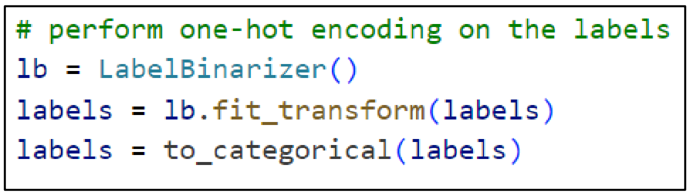
- Images are passed through the `preprocess_input()` function, which:
  - Scales the pixel values to a range suitable for MobileNetV2 (i.e., normalized between -1 and 1).
  - Aligns the data distribution with the statistics used when MobileNetV2 was originally trained on ImageNet.



```
for category in CATEGORIES:
    path = os.path.join(DIRECTORY, category)
    for img in os.listdir(path):
        img_path = os.path.join(path, img)
        image = load_img(img_path, target_size=(224, 224))
        image = img_to_array(image)
        image = preprocess_input(image)
        data.append(image)
        labels.append(category)
```

3.1.4 Label Binarization and One-Hot Encoding:

- Class labels are encoded using `LabelBinarizer()` to transform them into binary format (0 or 1).
- `to_categorical()` is used to convert these binary labels into one-hot encoded vectors, which are suitable for training with the softmax activation function in the final output layer.



```
# perform one-hot encoding on the labels
lb = LabelBinarizer()
labels = lb.fit_transform(labels)
labels = to_categorical(labels)
```

3.1.5 Data Splitting:

- The dataset is split into **training (80%)** and **testing (20%)** sets using `train_test_split()`, ensuring class balance through stratification.

3.1.6 Data Augmentation:

To reduce overfitting and increase the robustness of the model, an `ImageDataGenerator` is used to apply real-time data augmentation to the training images. Augmentations include:

- **`rotation_range=20`**
  - Randomly rotates the image within a range of **±20 degrees**.
  - Helps the model learn to recognize objects even when they are slightly tilted.
- **`zoom_range=0.15`**
  - Randomly zooms the image **in or out by up to 15%**.
  - Helps the model become invariant to the distance between the object and the camera.
- **`width_shift_range=0.2`**
  - Randomly shifts the image horizontally by **up to 20%** of the image width.
  - Simulates slight changes in the horizontal position of the object (e.g., faces not perfectly centered).
- **`height_shift_range=0.2`**
  - Randomly shifts the image vertically by **up to 20%** of the image height.
  - Helps the model tolerate vertical misalignments.
- **`shear_range=0.15`**
  - Applies a **shearing transformation up to 15%**, slanting the image along the horizontal or vertical axis.
  - Introduces affine distortions to help the model generalize better.
- **`horizontal_flip=True`**
  - Randomly flips images horizontally.
  - Useful in face detection tasks since faces can appear in mirrored orientations.
- **`fill_mode='nearest'`**
  - Determines how to fill in new pixels that appear after a transformation (like rotation or shift).

  - `'nearest'` fills these pixels with the value of the **nearest neighboring pixel** to avoid introducing noise.

```python
# construct the training image generator for data augmentation
aug = ImageDataGenerator(
    rotation_range=20,
    zoom_range=0.15,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.15,
    horizontal_flip=True,
    fill_mode="nearest")
```

3.2 Preprocessing for Mask Video Detector

3.2.1 Frame to Blob (for face detection model)

```python
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))
```

Purpose:
Converts the raw video frame into a standardized format (a 4D blob) for the OpenCV DNN face detector.

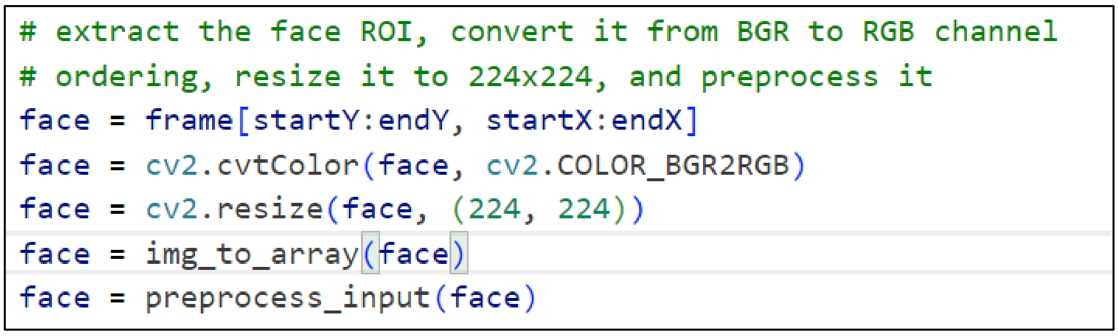
- 1.0: scaling factor (no scaling)
- (224, 224): spatial size for the model input
- (104.0, 177.0, 123.0): mean subtraction values for normalization (used by this particular Caffe model)

3.2.2 Resize Frame

```python
    # grab the frame from the threaded video stream and resize it
    # to have a maximum width of 400 pixels
    frame = vs.read()
    frame = imutils.resize(frame, width=400)
```

Purpose:
Resize the full frame to a fixed width for consistency and possibly better performance/speed.

3.2.3 Extract and Prepare Detected Faces (for mask classifier)



```
# extract the face ROI, convert it from BGR to RGB channel
# ordering, resize it to 224x224, and preprocess it
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)
```

Purpose:
This is preprocessing **each detected face** before passing it to the maskNet (your trained Keras/TensorFlow model).

- **cvtColor**: MobileNetV2 expects RGB format
- **resize:** matches MobileNetV2 input size (224×224)
- **img_to_array:** converts the image to a NumPy array
- **preprocess_input:** scales pixel values to the format expected by MobileNetV2 (values between -1 and 1)

4 AI/ML Model Architecture

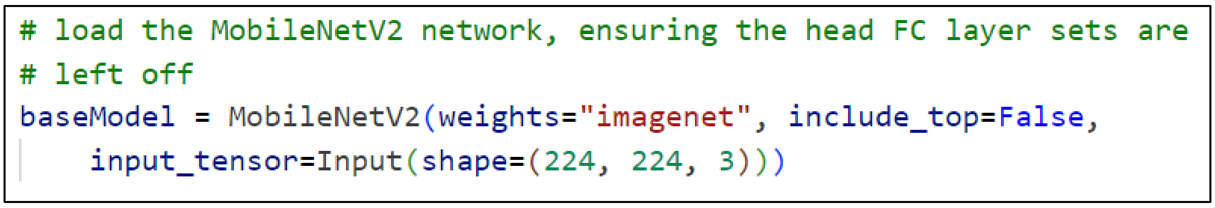
This project utilizes **transfer learning** with the **MobileNetV2** architecture, a lightweight and efficient deep convolutional neural network pre-trained on the ImageNet dataset. The goal is to build a robust and accurate face mask detector capable of distinguishing between masked and unmasked faces.

4.1 Base Model: MobileNetV2

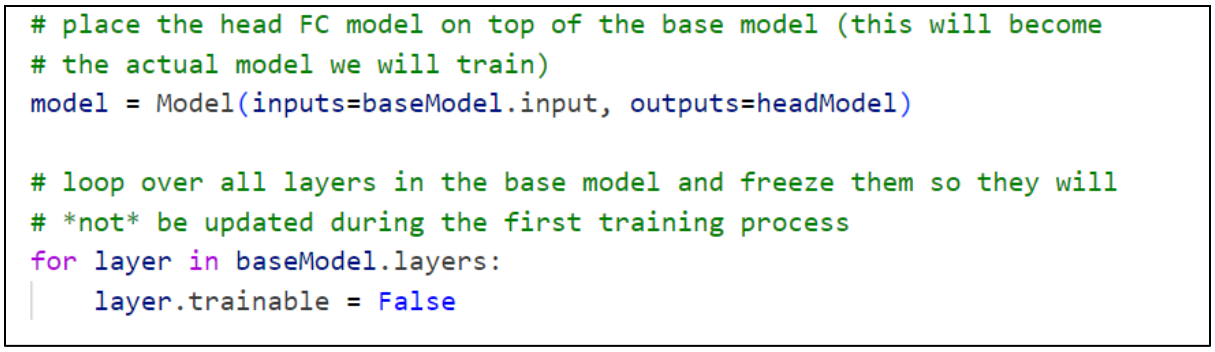
MobileNetV2 was chosen for its speed and efficiency, making it suitable for deployment in real-time and resource-constrained environments. In our architecture:

- We **load MobileNetV2 with pre-trained weights** from ImageNet.
- The `include_top=False` argument ensures that the **final classification layers are removed**, allowing us to attach a new custom head suitable for our binary classification task.
- The input shape is set to **(224, 224, 3)**, which is standard for MobileNetV2.

All layers of the base model are **frozen** (`layer.trainable = False`) during initial training to retain the knowledge learned from ImageNet and prevent overfitting, especially since our dataset is smaller.



```python
# load the MobileNetV2 network, ensuring the head FC layer sets are
# left off
baseModel = MobileNetV2(weights="imagenet", include_top=False,
    input_tensor=Input(shape=(224, 224, 3)))
```



```python
# place the head FC model on top of the base model (this will become
# the actual model we will train)
model = Model(inputs=baseModel.input, outputs=headModel)

# loop over all layers in the base model and freeze them so they will
# *not* be updated during the first training process
for layer in baseModel.layers:
    layer.trainable = False
```

4.2 Custom Head Model

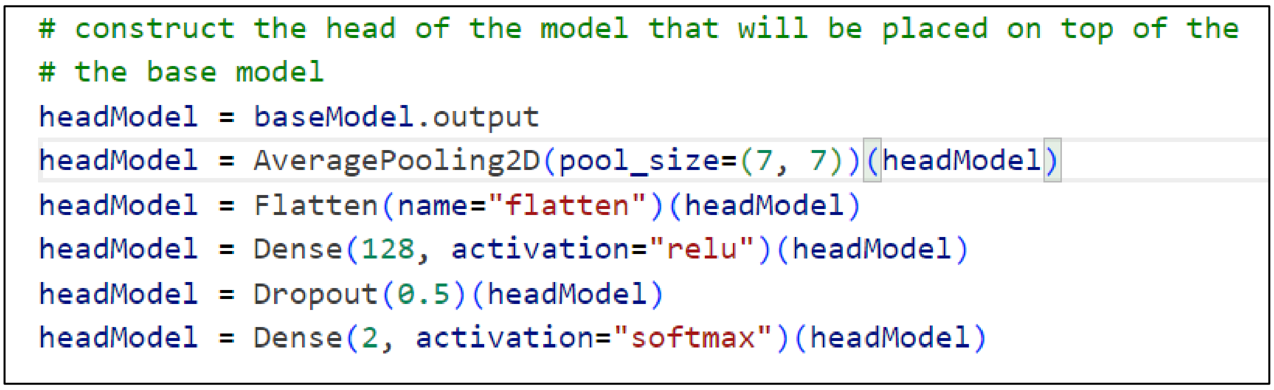
On top of the MobileNetV2 base, a **custom classification head** is added to adapt the model to our specific task:

1. **AveragePooling2D(pool_size=(7, 7))**: Reduces the spatial dimensions (height and width) of the feature maps. A 7x7 pool size reduces computational load and flattens spatial structure.
2. **Flatten()**: Converts the 2D output from the pooling layer into a 1D vector so it can be fed into fully connected layers.
3. **Dense(128, activation="relu")**: A fully connected layer with 128 neurons.

    - activation="relu" introduces non-linearity and helps the model learn complex patterns.

4. **Dropout(0.5)**: Randomly sets 50% of input neurons to zero during training. This prevents overfitting by making the network less reliant on specific neurons.
5. **Dense(2, activation="softmax")**: Final output layer with 2 neurons (one for each class: "with_mask" and "without_mask").
    - activation="softmax" converts the raw output scores into probabilities summing to 1, which is ideal for multi-class (even binary) classification

4.3 Compilation and Optimization



```
# construct the head of the model that will be placed on top of the
# the base model
headModel = baseModel.output
headModel = AveragePooling2D(pool_size=(7, 7))(headModel)
headModel = Flatten(name="flatten")(headModel)
headModel = Dense(128, activation="relu")(headModel)
headModel = Dropout(0.5)(headModel)
headModel = Dense(2, activation="softmax")(headModel)
```

- The model is compiled with the **Adam optimizer** using a learning rate (`INIT_LR`) of 1e-4 and a decay schedule (`INIT_LR / EPOCHS`).
- The **loss function** is `binary_crossentropy`, suitable for binary classification.
- **Accuracy** is used as the evaluation metric.

4.4 Training Strategy

To optimize performance and generalization, the following training strategy was employed:

4.4.1 Transfer Learning with Frozen Base

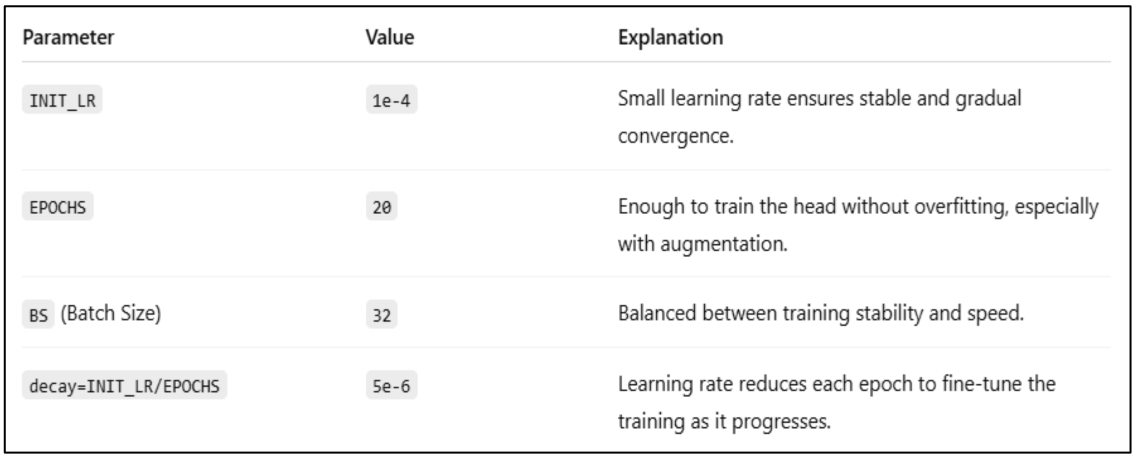
- Only the custom head layers are trainable during initial training.
- This allows the model to quickly adapt to our task without altering the robust features already learned by MobileNetV2.

4.4.2 Regularization

Overfitting is addressed by:

- **Dropout (0.5)** in the classifier head: Randomly disables 50% of neurons during each training batch.
- **Early training of only the head**: Prevents drastic updates to already well-learned features.
- **Data augmentation**: As discussed, it serves as implicit regularization

4.4.3 Training Parameters



Parameter	Value	Explanation
`INIT_LR`	`1e-4`	Small learning rate ensures stable and gradual convergence.
`EPOCHS`	`20`	Enough to train the head without overfitting, especially with augmentation.
`BS` (Batch Size)	`32`	Balanced between training stability and speed.
`decay=INIT_LR/EPOCHS`	`5e-6`	Learning rate reduces each epoch to fine-tune the training as it progresses.

4.4.4 Evaluation During Training

- **Validation set (20%)** is used to monitor the model's performance after each epoch.
- Metrics tracked:
  - **Accuracy** (how many predictions were correct)
  - **Loss** (how confident and correct the predictions were)

5 Experimental Setup

To evaluate the performance of our face mask detection model, we conducted experiments using a structured dataset and a standard training-testing split. The experimental setup is described below:

5.1 Software and Libraries:

- Programming Language: Python 3.x
- Deep Learning Framework: TensorFlow and Keras
- Supporting Libraries: NumPy, Matplotlib, scikit-learn, imutils

5.2 Dataset:

- Total Images: ~3,800
- Classes: with_mask, without_mask
- Image Size: Resized to 224x224 pixels to match MobileNetV2 input shape
- Data Split: 80% for training and 20% for testing, stratified to maintain class balance

5.3 Preprocessing:

- All images were normalized using MobileNetV2's `preprocess_input()` function.

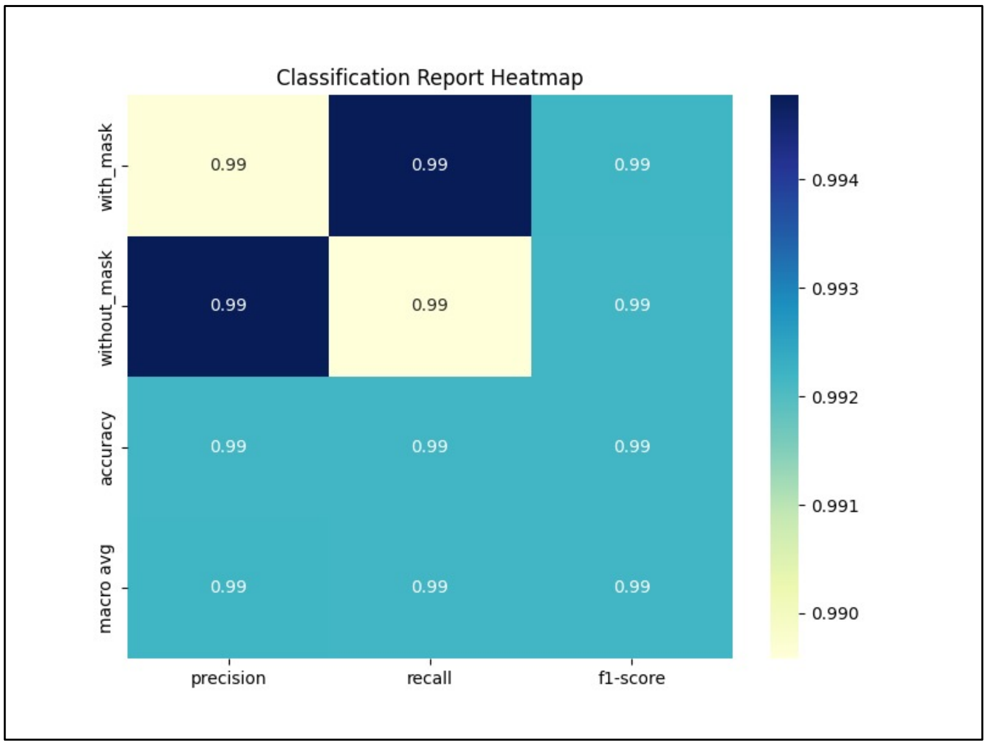
- Labels were binarized and converted to categorical format using `LabelBinarizer`.

5.4 Training Configuration:

- Base Model: MobileNetV2 (pre-trained on ImageNet, `include_top=False`)
- Batch Size: 32
- Epochs: 20
- Optimizer: Adam with an initial learning rate of 1e-4 and decay over epochs
- Loss Function: Binary Crossentropy
- Evaluation Metric: Accuracy
- Data Augmentation: Rotation (20°), Zoom (15%), Shifts (20%), Shear (15%), Horizontal Flip

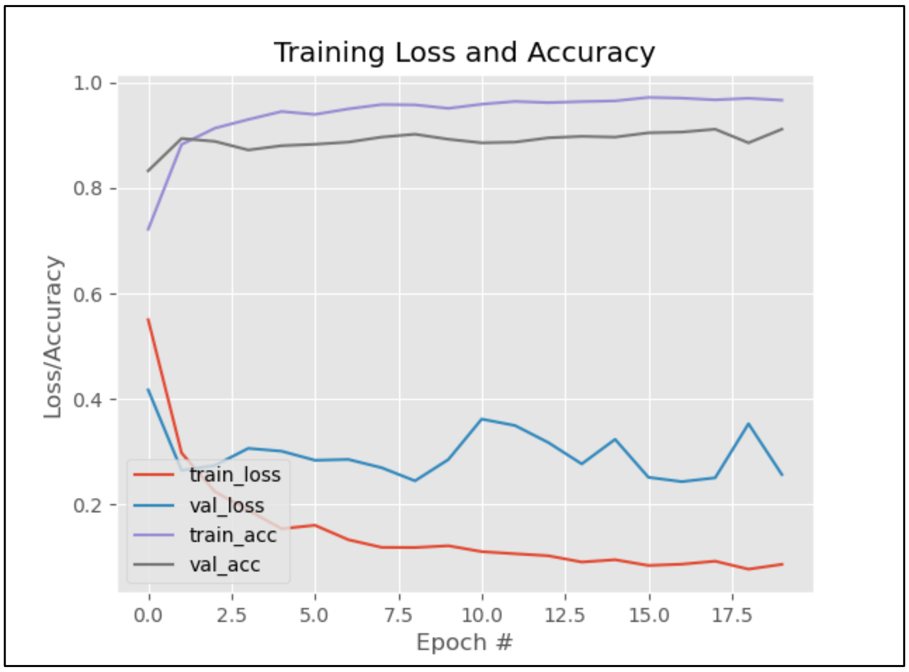
5.5 Evaluation:

- The model was evaluated using a classification report that includes precision, recall, F1-score, and accuracy.
- A training curve showing loss and accuracy over epochs was plotted and saved for analysis.



6 Result

6.1 Overall Performance Metrics



Lines Explained

🔴 **Red Line (train_loss)**: The loss on the training data.

🔵 **Blue Line (val_loss)**: The loss on the validation (test) data.

🟣 **Purple Line (train_acc)**: Accuracy on the training data.

⚫ **Black Line (val_acc)**: Accuracy on the validation (test) data.

What the Graph Tells You

1. High Accuracy Achieved

- **Training Accuracy** rises quickly and stays above **95%**.
- **Validation Accuracy** also rises and stabilizes around **90–95%**.

This means the model is **learning well** and **generalizing reasonably**.

2. Loss Behavior

- **Training Loss** (red) decreases steadily — good sign, as the model is minimizing error.
- **Validation Loss** (blue) decreases initially but **fluctuates slightly** after epoch 5.

This up-down pattern in validation loss is **normal**, especially with small datasets or aggressive augmentations. Your model is still learning, but it's likely hitting small plateaus or reacting to noise in the validation data.

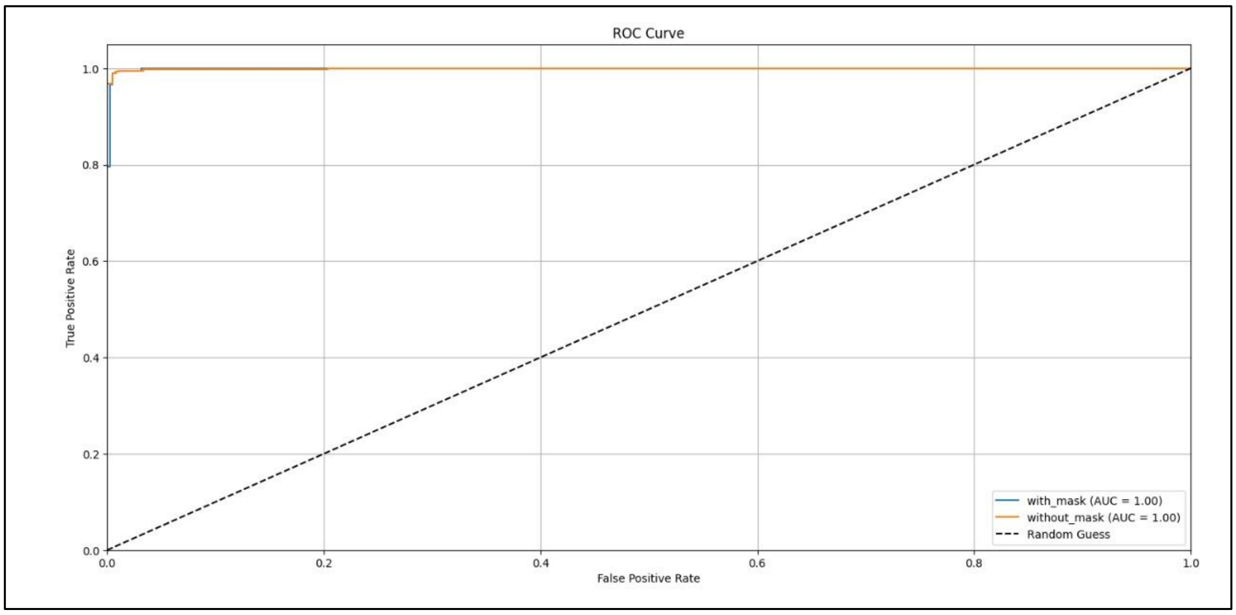
3. Is There Overfitting?

Not significantly. Here's why:

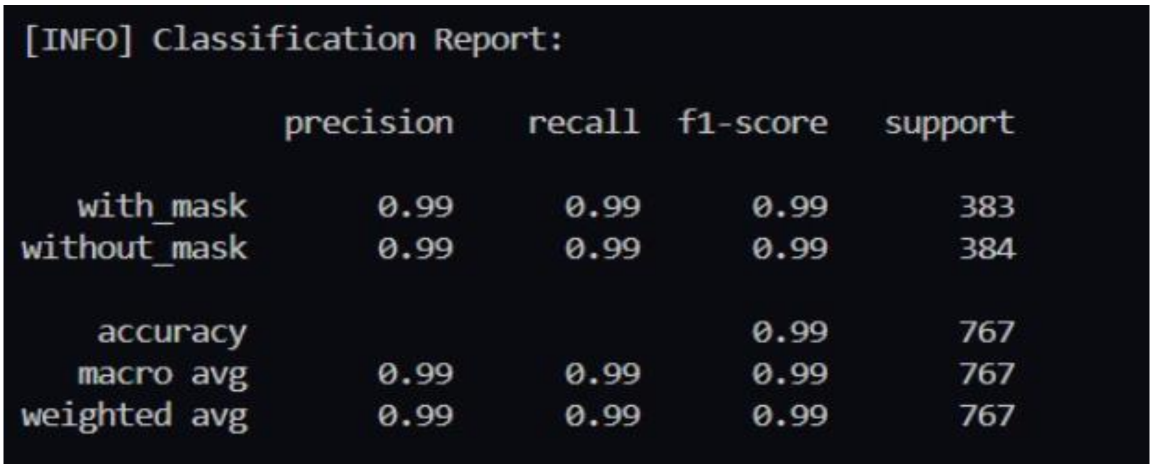
- Validation accuracy **remains close to** training accuracy.
- Validation loss doesn't rise drastically.

This means the model is **not memorizing** the training data — it is still learning useful general features.

6.2 ROC Curve

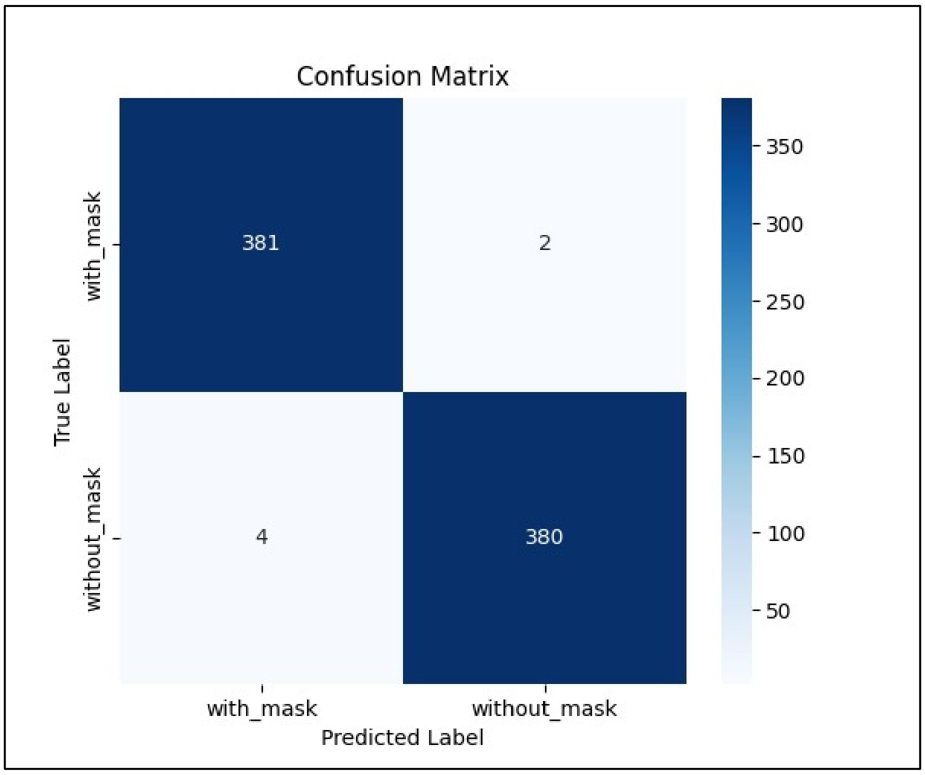


6.3 Classification Report



```
[INFO] Classification Report:

              precision    recall  f1-score   support

   with_mask       0.99      0.99      0.99       383
without_mask       0.99      0.99      0.99       384

    accuracy                           0.99       767
   macro avg       0.99      0.99      0.99       767
weighted avg       0.99      0.99      0.99       767
```

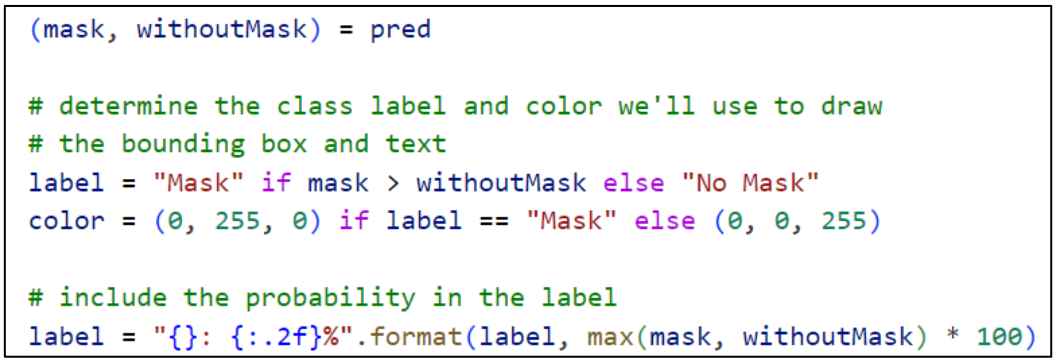
6.4 Confusion Matrix



7 Post Processing

7.1.1 Interpreting the Predictions

Post-processing refers to the operations performed **after the model has made its predictions**, in order to **interpret**, **format**, and **present** the results to the user in a meaningful and usable way. In this face mask detection system, post-processing ensures that the output from the neural networks is not just raw numerical data, but rather **visually informative and actionable** feedback on the video stream.



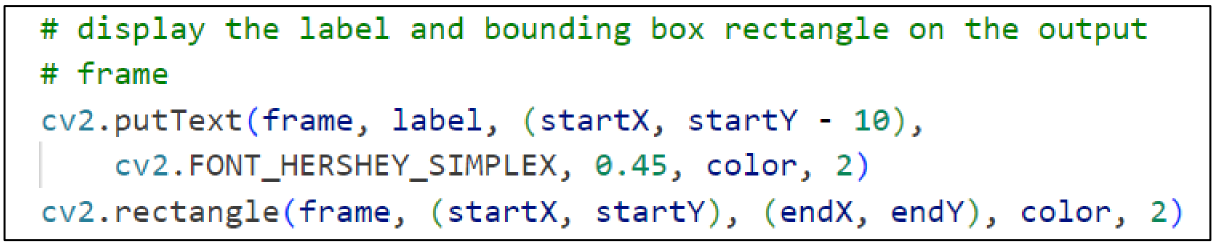
```
(mask, withoutMask) = pred

# 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)
```

Explanation:
The mask classification model outputs two probabilities: one for "Mask" and one for "No

Mask". This step compares the two and selects the higher one to determine the final label for each detected face.

Why it's done:
Machine learning models typically return numerical probabilities, which need to be **converted into human-readable decisions**. The use of a clear label ("Mask"/"No Mask") along with a confidence score helps convey both the classification and its certainty, which is critical for user trust and usability.

7.2 Drawing Bounding Boxes and Labels

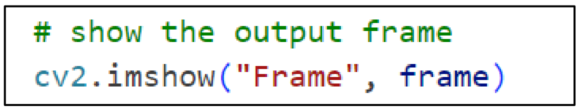


```python
# display the label and bounding box rectangle on the output
# frame
cv2.putText(frame, label, (startX, startY - 10),
    cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)
cv2.rectangle(frame, (startX, startY), (endX, endY), color, 2)
```

Explanation:
A rectangle (bounding box) is drawn around each detected face, and the classification label along with its confidence percentage is placed just above it.

Why it's done:
This visual feedback allows users or operators to **quickly identify which individuals are not wearing masks**, and potentially take real-time action. The colored boxes (green for mask, red for no mask) are especially helpful in distinguishing between cases at a glance, even in a crowded frame.

7.3 Displaying the Processed Frame



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

Explanation:
The processed video frame is displayed in a window using OpenCV.

Why it's done:
Real-time visualization is critical in applications such as surveillance or access control. Without displaying the output frame, the results of detection and classification would remain hidden to the end user, defeating the purpose of real-time monitoring.

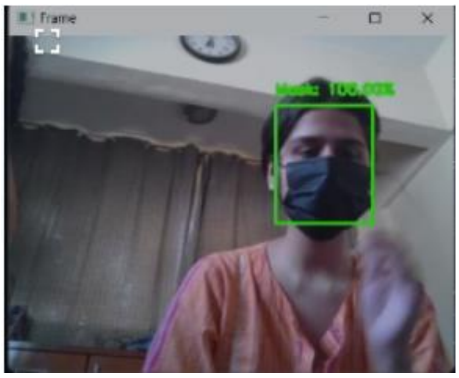
7.4 Handling Exit Conditions

```
key = cv2.waitKey(1) & 0xFF

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

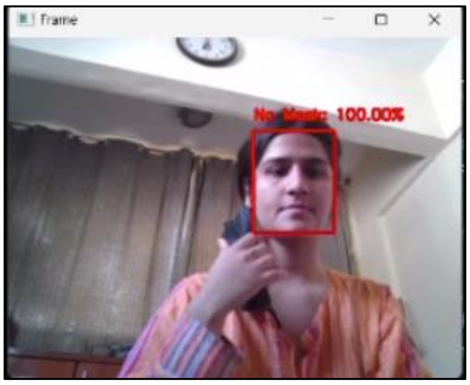
Explanation:
This code checks if the user has pressed the 'q' key and exits the video loop if so.

Why it's done:
It provides a clean and user-controlled way to **terminate the system**, which is important in practical deployments for control and safety.

8 Screenshot of Video Demo



(a) Mask Detected



(b) No Mask Detected

As seen in the ***figure (a)***, the system draws a green rectangle around the detected face and labels it as "Mask" with a confidence score (e.g., 100%). This result demonstrates that: • The face was accurately located in the frame. • The preprocessed face image was correctly classified by the model. • Post-processing logic successfully annotated the frame for visual feedback.	In the ***figure (b),*** the system displays a red bounding box around the detected face and labels it as "No Mask" with a corresponding confidence score (e.g., 100%). This verifies: • The model can **differentiate facial features** that correspond to an unmasked face. • The preprocessing and classification pipeline works consistently for both classes. • Visual post-processing (labeling and bounding box coloring) helps quickly identify violations in public spaces.