

Task 2- Face Mask Detector

Introduction to our solution and other possible solutions to the problem

We were assigned a task to build a face mask detector.

We decided to create a classification machine learning algorithm for this purpose. We used the technique of support vector machine (SVM for short) in this case. Other machine learning models like logistic regression or decision trees can also be used for the purpose. Another approach for the problem could be to train a convolutional neural network which is a deep learning algorithm. Since deep learning models are best suited for unstructured data types, hence going with a deep learning algorithm would certainly provide the best results. But for simplicity we have used a SVM model which was trained and tested using datasets taken from a website called Kaggle. We can also use our own created dataset by using an OpenCV code that could read our webcam video feed and capture hundreds of frames of our faces, once with a mask on, and once with a mask off. But that approach is highly time consuming and webcam video qualities are low on laptops, so we chose to take already created datasets from the internet.

Then we used scikit learn, numpy and opencv to load the datasets, check their elements for presence of haar features and filtered the dataset to be used, created labels, and then finally train and test our model as we will discuss in the explanation of the code. With this done, we will have created a mask detector program. To test this ourselves we wrote an OpenCV code that helps us to read out webcam feed and check how good the model works.

Note that the accuracy of your model depends on a lot of factors like the learning algorithm used, the size of dataset, the nature of train-test split, etc.

The best performance will be achieved by deep learning algorithms and use of large datasets with around a 75-25 train-test split.

As for our ML model, an SVM can have its accuracy increased if we increase the number of data points it is being trained on, or if we augment the datasets to reuse the same datasets after few modifications to train the model, or if we pre-process the data in a better manner.

Hence even if one uses the same code as ours, all these factors will determine the accuracy of the model on your machines. We have achieved an accuracy of 73% in our machine, which can be further improved by more and more optimisations.

The code has been done in a Jupyter notebook present in the same github repository as this file.

The 2 main portions of the code in the Jupyter notebook are explained below:

PART 1: LOADING DATASETS, TRAINING AND TESTING THE SVM MODEL:

```
import cv2
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
import os
```

The above lines import the required libraries for the face mask detection system.

```
# Load the face detection classifier
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
```

This line loads the Haar cascade classifier for face detection. It is used to detect faces in the images and video frames. In the provided code, a Haar cascade classifier is used for face detection. Haar cascade classifiers use simple rectangular features called Haar features to detect objects or patterns. These features capture variations in pixel intensities, such as edges and corners. By combining multiple Haar features and utilizing a machine learning algorithm, the classifier determines whether a region of an image contains the object of interest. In this case, the Haar cascade classifier (face_cascade) is used to detect faces in the grayscale frames. The detected faces are then used for subsequent tasks like extracting face regions and conducting face mask detection.

```
# Load the dataset
mask_dir = r'C:\Users\divsh\Downloads\archive (2)\images\masks'
no_mask_dir = r'C:\Users\divsh\Downloads\archive (2)\images\without_masks'
```

These lines specify the paths to the directories containing the mask and no-mask images. Please make sure to update these paths with the appropriate directory locations on your system.

```
mask_images = [os.path.join(mask_dir, f) for f in os.listdir(mask_dir)]
no_mask_images = [os.path.join(no_mask_dir, f) for f in os.listdir(no_mask_dir)]
```

These lines create lists of file paths for the mask and no-mask images by combining the directory paths with the file names in the directories.

```
# Check that both mask and no-mask images are present in the dataset
if len(mask_images) == 0 or len(no_mask_images) == 0:
    raise ValueError("Both mask and no-mask images must be present in the dataset")
```

This code block checks if there are at least one mask image and one no-mask image present in the dataset. If either of the lists is empty, it raises a `ValueError`.

```
# Create labels for the dataset
mask_labels = np.ones(len(mask_images))
no_mask_labels = np.zeros(len(no_mask_images))
```

These lines create labels for the mask and no-mask images. The mask labels are set to 1 and the no-mask labels are set to 0.

```
# Combine the datasets and labels
images = mask_images + no_mask_images
labels = np.concatenate([mask_labels, no_mask_labels])
```

These lines combine the lists of mask and no-mask images into a single list called `images`. Similarly, the mask and no-mask labels are concatenated into a single array called `labels`.

```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(images, labels, test_size=0.25, random_state=42)
```

This line splits the dataset into training and testing sets using the `train_test_split` function from `sklearn.model_selection`. It takes the `images` and `labels` arrays, and splits them into training and testing sets with a test size of 0.25 (25% of the data). The `random_state` parameter ensures reproducibility of the same train-test split.

```
# Define a function to extract features from the images
def extract_features(image_path):
    image = cv2.imread(image_path)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))
    if len(faces) == 0:
        return None
    (x, y, w, h) = faces[0]
    face_roi = gray[y:y+h, x:x+w]
    face_roi = cv2.resize(face_roi, (100, 100))
    return face_roi.flatten()
```

This function, `extract_features`, takes an image path as input and returns the flattened grayscale face region after detecting and cropping the face. It uses the loaded face cascade classifier to detect faces in the image, extracts the region of interest (ROI) corresponding to the first detected face, resizes it to 100x100 pixels and returns the flattened array.

```
# Extract features from the training set
X_train_features = []
y_train_subset = []
for i, image_path in enumerate(X_train):
    features = extract_features(image_path)
    if features is not None:
        X_train_features.append(features)
        y_train_subset.append(y_train[i])

X_train_features = np.array(X_train_features)
y_train_subset = np.array(y_train_subset)
```

These lines iterate over the images in the training set and extract features using the `extract_features` function. The features are then added to the `X_train_features` list, and the corresponding labels are added to the `y_train_subset` list.

```
# Check the number of unique classes in y_train
classes = np.unique(y_train_subset)
if len(classes) < 2:
    raise ValueError("At least two classes are required in the training data")
```

This code block checks if there are at least two unique classes in the training data. If not, it raises a `ValueError`. The minimum requirement is two classes (mask and no-mask) for the face mask detection task.

```
# Select a random subset of features that corresponds to the indices in y_train_subset
idx = np.random.choice(len(X_train_features), size=len(y_train_subset), replace=False)
X_train_subset = X_train_features[idx]
```

This line randomly selects a subset of features from the training data to balance the classes. It selects indices using `np.random.choice` to match the length of `y_train_subset` and ensures that the selection is done without replacement (`replace=False`). The selected indices are then used to extract the corresponding features from `X_train_features`, resulting in a balanced subset called `X_train_subset`.

```
# Train the SVM model
svm = SVC(kernel='linear', C=1, random_state=42)
svm.fit(X_train_subset, y_train_subset)
```

These lines create an SVM (Support Vector Machine) classifier object with a linear kernel and regularization parameter `C=1`. The `random_state` parameter is set for reproducibility. The classifier is then trained on the balanced subset of training features (`X_train_subset`) and labels (`y_train_subset`) using the `fit` method.

```
# Extract features from the testing set
X_test_features = []
y_test_subset = []
for i, image_path in enumerate(X_test):
    features = extract_features(image_path)
    if features is not None:
        X_test_features.append(features)
        y_test_subset.append(y_test[i])

X_test_features = np.array(X_test_features)
y_test_subset = np.array(y_test_subset)
```

These lines extract features from the testing set images in a similar manner to the training set. The extracted features are added to the `X_test_features` list, and the corresponding labels are added to the `y_test_subset` list.

```
# Test the SVM model
accuracy = svm.score(X_test_features, y_test_subset)
print('Accuracy:', accuracy)
```

This line computes the accuracy of the SVM model on the testing set by using the `score` method of the SVM classifier with the testing features (`X_test_features`) and labels (`y_test_subset`). The accuracy is then printed to the console.

In our model's case, accuracy is about 73%.

.....

2ND PART: READING WEBCAM VIDEO FEED AND CLASSIFYING:

```
# Open the webcam
cap = cv2.VideoCapture(0)

while True:
    # Capture frame-by-frame
    ret, frame = cap.read()

    # Convert the frame to grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Detect faces in the frame
    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))

    for (x, y, w, h) in faces:
        face_roi = gray[y:y+h, x:x+w]
        face_roi = cv2.resize(face_roi, (100, 100))
        features = face_roi.flatten().reshape(1, -1)

        # Predict the mask status using the SVM model
        mask_status = svm.predict(features)

        # Draw a rectangle around the face
        color = (0, 255, 0) if mask_status == 1 else (0, 0, 255)
        cv2.rectangle(frame, (x, y), (x+w, y+h), color, 2)

        # Add a label to the rectangle
        label = "Wearing mask" if mask_status == 1 else "Not wearing mask"
        cv2.putText(frame, label, (x, y-10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)

    # Display the resulting frame
    cv2.imshow('Face Mask Detection', frame)

    # Press 'q' to exit the loop
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release the webcam and close all windows
cap.release()
cv2.destroyAllWindows()
```

This part of the code is responsible for opening the webcam, continuously capturing frames, detecting faces in each frame, predicting the mask status using the SVM model, drawing rectangles and labels around the detected faces based on the predicted mask status, and displaying the resulting frames with the annotations. Here's a summarized documentation for this code block:

The code starts by opening the webcam using `cv2.VideoCapture(0)`. It enters a while loop that runs indefinitely until the user presses the 'q' key. Within the loop, it captures each frame from the webcam using `cap.read()`, converts the frame to grayscale using `cv2.cvtColor`, and detects faces in the grayscale frame using the Haar cascade classifier (`face_cascade`). For each detected face, it extracts the face region, resizes it to 100x100 pixels, and flattens the image to a 1D array of features. The SVM model (`svm`) predicts the mask status based on the extracted features. The code then draws a rectangle around each face on the original frame, with a green color for a predicted mask status of 1 (wearing mask) and red color for a predicted mask status of 0 (not wearing mask). A label indicating the mask status is added above each rectangle. Finally, the annotated frame is displayed using `cv2.imshow`. If the user presses the 'q' key, the loop breaks, and the webcam is released using `cap.release()` and all windows are closed using `cv2.destroyAllWindows()`.

The following 2 images show how a webcam reads the image and classifies it on my computer.

The first picture is where the camera detect a face without mask on my phone screen that I've held in my hand:

Jupyter Task2-MASK-DETECTOR Last Checkpoint: 5 minutes ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

```

In [*]: # Open the webcam
cap = cv2.VideoCapture(0)

while True:
    # Capture frame-by-frame
    ret, frame = cap.read()

    # Convert the frame to grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Detect faces in the frame
    faces = face_cascade.detectMultiScale(gray, 1.1, 4)

    for (x, y, w, h) in faces:
        face_roi = gray[y:y+h, x:x+w]
        face_roi = cv2.resize(face_roi, (100, 100))
        features = face_roi.flatten().tolist()

        # Predict the mask status using SVM
        mask_status = svm.predict(features)

        # Draw a rectangle around the face
        color = (0, 255, 0) if mask_status == 0 else (0, 0, 255)
        cv2.rectangle(frame, (x, y), (x+w, y+h), color, 2)

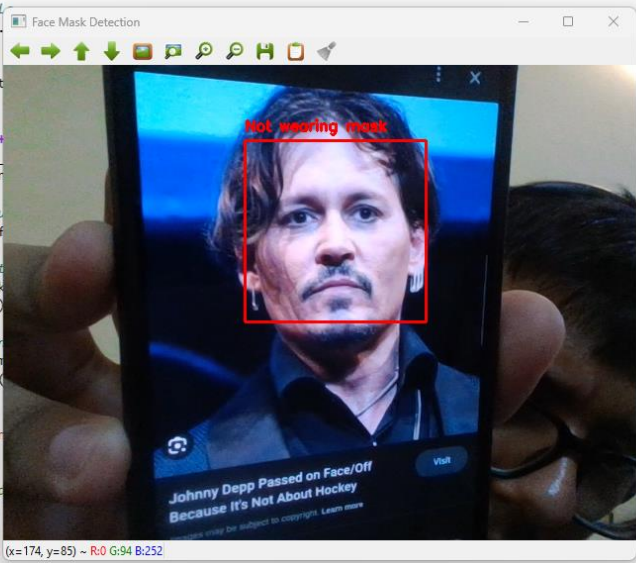
        # Add a label to the rectangle
        label = "Wearing mask" if mask_status == 0 else "Not wearing mask"
        cv2.putText(frame, label, (x+w, y+h), cv2.FONT_HERSHEY_SIMPLEX, 0.8, color, 2)

    # Display the resulting frame
    cv2.imshow('Face Mask Detection', frame)

    # Press 'q' to exit the loop
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release the webcam and close all windows
cap.release()
cv2.destroyAllWindows()

```



In []:

In the picture below, it detects a face with a mask on it through the webcam:

Jupyter Task2-MASK-DETECTOR Last Checkpoint: 14 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

```

In [*]: # Open the webcam
cap = cv2.VideoCapture(0)

while True:
    # Capture frame-by-frame
    ret, frame = cap.read()

    # Convert the frame to grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Detect faces in the frame
    faces = face_cascade.detectMultiScale(gray, 1.1, 4)

    for (x, y, w, h) in faces:
        face_roi = gray[y:y+h, x:x+w]
        face_roi = cv2.resize(face_roi, (100, 100))
        features = face_roi.flatten().tolist()

        # Predict the mask status using SVM
        mask_status = svm.predict(features)

        # Draw a rectangle around the face
        color = (0, 255, 0) if mask_status == 0 else (0, 0, 255)
        cv2.rectangle(frame, (x, y), (x+w, y+h), color, 2)

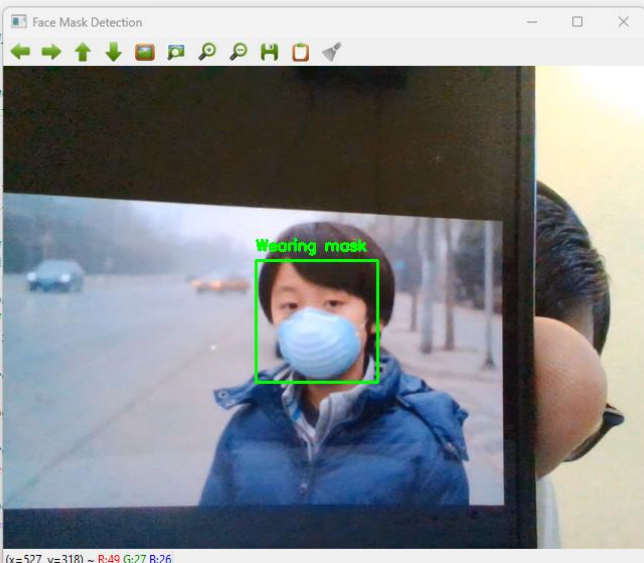
        # Add a label to the rectangle
        label = "Wearing mask" if mask_status == 0 else "Not wearing mask"
        cv2.putText(frame, label, (x+w, y+h), cv2.FONT_HERSHEY_SIMPLEX, 0.8, color, 2)

    # Display the resulting frame
    cv2.imshow('Face Mask Detection', frame)

    # Press 'q' to exit the loop
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release the webcam and close all windows
cap.release()
cv2.destroyAllWindows()

```



In []: