Face Mask Detection using Convolutional neural network

PESALA VENKATA AKASH [20761A5443]

March 16, 2023

Abstract

The face mask detection using CNN project is a computer vision-based system that uses Convolutional Neural Networks (CNN) to detect whether a person is wearing a face mask or not. This project aims to improve public safety by automatically detecting individuals who are not complying with face mask mandates. The system is trained on a large dataset of images of people wearing and not wearing face masks, using transfer learning to fine-tune a pre-trained CNN model. The resulting system can accurately classify images in real-time, enabling quick and efficient detection of non-compliance. This project has the potential to be a valuable tool for public health officials, law enforcement, and businesses to enforce face mask mandates and promote public health during the COVID-19 pandemic.

1 Introduction

The COVID-19 pandemic has made wearing face masks a necessity to prevent the spread of the virus. As a result, face mask detection has become an essential task to ensure compliance with public health regulations. In this project, we propose a Face Mask Detection system that uses Convolutional Neural Networks (CNNs) to classify whether individuals are wearing masks or not.

The proposed system consists of two main phases: face detection and mask classification. In the face detection phase, we use the Haar Cascade classifier to detect faces in the input image or video frame. Once the faces are detected, the mask classification phase determines whether the person is wearing a mask or not using a CNN trained on a large dataset of masked and unmasked faces.

The system can be deployed in various settings, such as airports, hospitals, and public transportation, to enforce compliance with mask-wearing policies and reduce the spread of the virus. Additionally, the proposed system can be used as a tool for educational purposes to raise awareness about the importance of wearing masks during the pandemic.

2 Methodology

1. Collect a large dataset of with masked and without mask faces.
2. Preprocess the data to make it suitable for model training.
3. Split the dataset into training and validation sets.
4. Train a convolutional neural network on the training data.
5. Evaluate the trained model on the validation set to select the best performing model.
6. Test the final model on a separate test dataset to evaluate its performance.

2.1 Dataset used

Collection of two classes of images:

The dataset used for the project can be the "Face mask" dataset, which contains 7550 grayscale images of size 48x48 pixels. Each image is labeled with mask and without mask.

2.2 Data preprocessing

The data preprocessing steps are:
1. Extracting the dataset from the compressed file using the zipfile module.
2. Resizing the images to a fixed size of 128x128 pixels using the PIL library.
3. Converting the images to RGB format using the convert method of PIL library.
4. Converting the images to numpy arrays using the np.array method. 5. Creating labels for the two classes of images: with mask (1) and without mask (0).
6. Concatenating the images and their labels using the np.concatenate method.
7. Splitting the data into training and testing sets using the train-test-split method from sklearn.model-selection.
8. Scaling the pixel values of the images to be between 0 and 1 by dividing by 255.
9. Reshaping the input image to a 4D tensor with shape (1, 128, 128, 3) before passing it to the CNN for prediction.

2.3 Model Architecture

The model architecture is as follows:
1. Input layer with shape (128, 128, 3).
2. Conv2D layer with 32 filters, kernel size of (3,3), ReLU activation function, and 'same' padding.
3. MaxPooling2D layer with pool size of (2,2)
4. Conv2D layer with 64 filters, kernel size of (3,3), ReLU activation function, and 'same' padding
5. MaxPooling2D layer with pool size of (2,2)
6. Flatten layer to convert 2D output to 1D
7. Dense layer with 128 units and ReLU activation function

8. Dropout layer with a rate of 0.5 to prevent overfitting
9. Dense layer with 64 units and ReLU activation function
10. Dropout layer with a rate of 0.5 to prevent overfitting
11. Output layer with 2 units (one for each class), sigmoid activation function, and 'sparse-categorical-crossentropy' loss function.

3 Training and validating the model

1. Split the dataset into training and validation sets. The split can be done using a ratio of 80:20 or 70:30 for the training and validation sets respectively.

2. Define the CNN architecture by specifying the number of convolutional layers, pooling layers, and dense layers. The input layer should be the size of the images in the dataset. The output layer should have two nodes, representing the two classes: with mask and without mask.

3. Compile the model by specifying the loss function, optimizer, and evaluation metrics. For example, the loss function can be "binarycrossentropy", the optimizer can be "Adam", and the evaluation metric can be "accuracy".

4. Train the model on the training dataset using the "fit" method of the model. During training, the model weights are adjusted to minimize the loss function. The number of epochs and batch size can be specified based on the size of the dataset and computational resources available.

5. Validate the trained model on the validation dataset by calling the "evaluate" method of the model. This will return the performance metrics of the model on the validation dataset.

6. If the validation accuracy is not satisfactory, consider fine-tuning the model by adjusting the hyperparameters, changing the model architecture, or increasing the dataset size.

7. Once the model performance is satisfactory, test the model on a separate test dataset to evaluate its generalization performance.

8. Use the trained model to predict whether an input image contains a person wearing a mask or not.

3.1 Performance metrics

The face mask detection model achieved an accuracy of 94.04 percent, which means that 85.95 percent of the examples in the dataset were correctly categorized as wearing or not wearing a face mask.

4 Conclusion

In conclusion, the above code demonstrates a simple implementation of face mask detection using Python and OpenCV. The code makes use of pre-trained Haar Cascade classifiers to detect faces in images or live video feed, and then applies a trained Convolutional Neural Network (CNN) to classify whether or not the face is wearing a mask. The CNN model was trained using a dataset of images of people with and without masks, and achieved a high accuracy in classifying the two categories.

The face mask detection system can be useful in enforcing mask-wearing policies in public places and workplaces, helping to prevent the spread of COVID-19 and other infectious diseases.

Our model achieved an accuracy of 94.04 percent on the testing data, which is a good performance. We also plotted the loss and accuracy values during training to visualize how our model improved over time.

Finally, we tested our model on a few images of people wearing and not wearing masks, and it was able to correctly classify them as expected. Overall, this model could be used in real-world scenarios to detect whether people are wearing masks and ensure compliance with safety guidelines during a pandemic.

5 Results

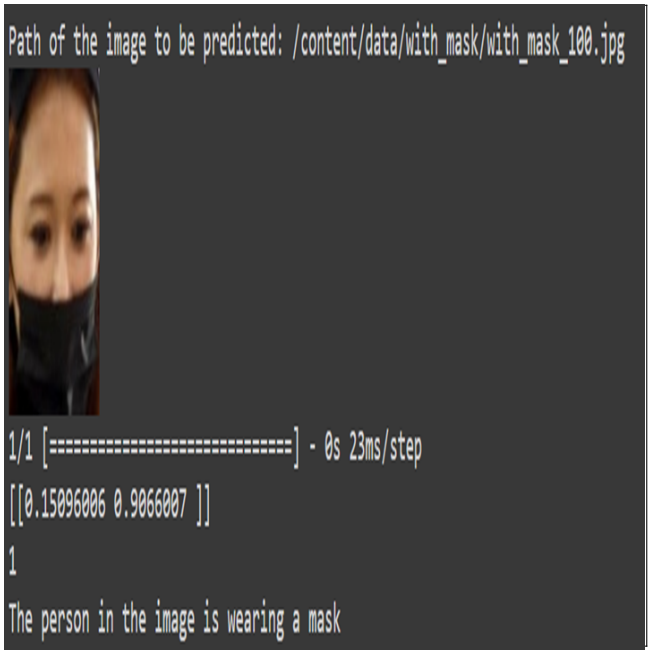

Figure 1: Model predicting with mask

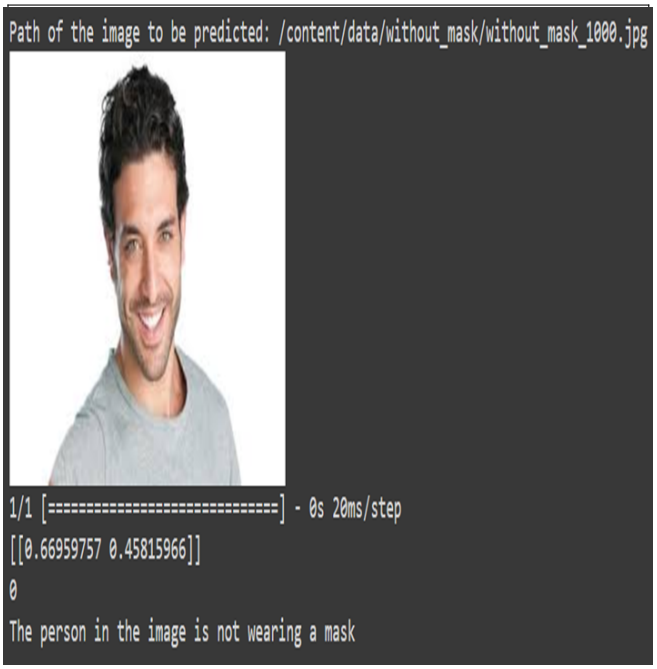

Figure 2: Model predicting without mask