Real Time Face Mask Detection

Submitted by:
Abhishek Pandey (abhishek230102@gmail.com)
Abu Afza (afza22bd@gmail.com)
Devanshi Singh (devanshisingh6b@gmail.com)
Rupasri Achanta (rupasrikkd147@gmail.com)
Yamparala Venkata Gopi
(yvenkatagopi18bcs@iiitkottayam.ac.in)

Submitted to:
Devanshi Singh
April 03, 2021

Abstract

During pandemic COVID-19, WHO has made wearing masks compulsory to protect against this deadly virus. In this tutorial we will develop a machine learning project – Real-time Face Mask Detector with Python.

We will train the face mask detector model using Keras and OpenCV.

Contents

List of Figures

Chapter 1

Introduction

Face Mask Detector using Convolutional Neural Networks (CNN) Python, Keras, TensorFlow and OpenCV. With further improvements these types of models could be integrated with CCTV or other types cameras to detect and identify people without masks. With the prevailing worldwide situation due to COVID-19 pandemic, these types of systems would be very supportive for many kind of institutions around the world. Please let me know your ideas, suggestions and concerns in the comment section and feel free to improve this project yourself and come up with better results.

1.1 Python

Python is powerful... and fast; plays well with others; runs everywhere; is friendly & easy to learn; is Open. These are some of the reasons people who use Python would rather not use anything else.

- Python can be easy to pick up whether you're a first time programmer or you're experienced with other languages. The following pages are a useful first step to get on your way writing programs with Python!
- The community hosts conferences and meetups, collaborates on code, and much more. Python's documentation will help you along the way, and the mailing lists will keep you in touch.
- The Python Package Index (PyPI) hosts thousands of third-party modules for Python. Both Python's standard library and the community-contributed modules allow for endless possibilities.
- Python is developed under an OSI-approved open source license, making it freely usable and distributable, even for commercial use. Python's license is administered by the Python Software Foundation.

1.2 Convolutional Neural Networks (CNNs)

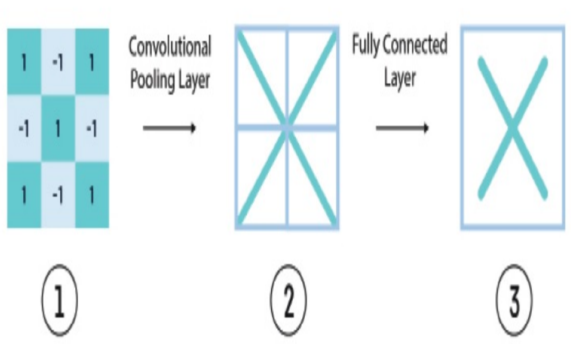
For complicated problems like image identification, it's difficult and time-consuming to try to identify the most important variables before training (feature engineering). This is why deep learning instead applies feature learning, where the machine learns the optimal features and weights on its own.

CNN accepts a fixed-sized vector as an input and produce a fixed-sized vector as an output.

CNN is made of 2 main layers:

- The convolutional and pooling layer(s) extract the optimal features. Each feature is a filter is slide over the target image to break the image into simpler images.
- The fully connected layer identifies the class of the image by comparing it to different images and finding the best match.

Layers in CNN



1. Input Layer
2. Convolution Layer
3. ReLU Layer
4. Pooling Layer
5. Fully connected Layer

CNN Applications are

- Image recognition
- video recognition
- Image analysis
- Image classification
- Media recreation
- Recommendation System
- Natural Language Processing (NLP)

1.3 Keras

Keras is Simple. Flexible. Powerful.

Keras is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library.

Keras gives fundamental reflections and building units for creation and transportation of ML arrangements with high iteration velocity. It takes full advantage of the scalability and cross-platform capabilities of TensorFlow. The core data structures of Keras are layers and models . All the layers used in the CNN model are implemented using Keras. Along with the conversion of the class vector to the binary class matrix in data processing, it helps to compile the overall model.

1.4 TensorFlow

TensorFlow, an interface for expressing machine learning algorithms, is utilized for implementing ML systems into fabrication over a bunch of areas of computer science, including sentiment analysis, voice recognition, geographic information extraction, computer vision, text summarization, information retrieval, computational drug discovery and flaw detection to pursue research. In the proposed model, the whole Sequential CNN architecture (consists of several layers) uses TensorFlow at backend. It is also used to reshape the data (image) in the data processing.

1.5 OpenCV

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc. OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 18 million. The library is used extensively in companies, research groups and by governmental bodies.

Along with well-established companies like Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota that employ the library, there are many startups such as Applied Minds, VideoSurf, and Zeitera, that make extensive use of OpenCV. OpenCV's deployed uses span the range from stitching streetview images together, detecting intrusions in surveillance video in Israel, monitoring mine equipment in China, helping robots navigate and pick up objects at Willow Garage, detection of swimming pool drowning accidents in Europe, running interactive art in Spain and New York, checking runways for debris in Turkey, inspecting labels on products in factories around the world on to rapid face detection in Japan.

It has C++, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS. OpenCV leans mostly towards real-time vision applications and takes advantage of MMX and SSE instructions when available. A full-featured CUDAand OpenCL interfaces are being actively developed right now. There are over 500 algorithms and about 10 times as many functions that compose or support those algorithms. OpenCV is written natively in C++ and has a templated interface that works seamlessly with STL containers.

Chapter 2

Details of the Work

We will discus the whole project using following way.

2.1 Project Overview

We are planing to do we want to get the image we apply the casecade classifier and casecade classifier due us the return of intern face give us x, y with parameter width (W) and height (H). Then we get this ROY the region of interest then it will be get 100/100 image then we will be pass it into convolutional neural network (CNN). Then the probability will be shows the output with Mask or without Mask. Figure 2.1 shows project overview.

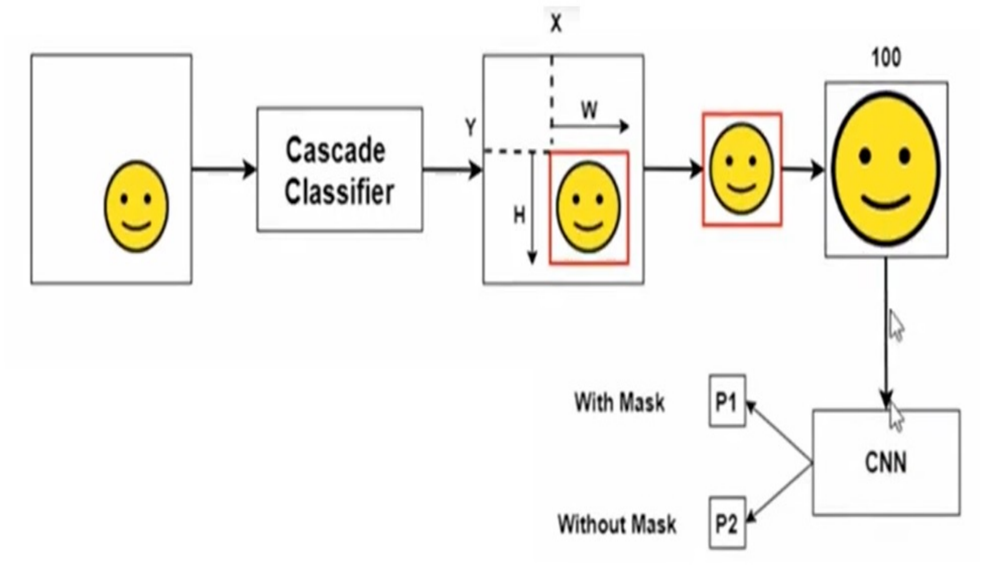


Figure 2.1: Project Overview

2.2 Experimental Setup

The experiment is performed on Google Colaboratory 1 under python 3 using TensorFlow and Graphics Processing Unit (GPU). An efficient and user-friendly python library also known as Keras, Tensorflow and OpenCV is also utilized to design and implement our Real Time Face Mask Detection.

2.3 Implementation

In this section, we will learn the basic steps to implement a face mask detection system. For implementation, we subdivide our task into a set of three sub tasks as follow.
Task 1: Data collection and preprocessing.
Task 2: Building and training the learning model.
Task 3: Deploying the learning model with OpenCV.

2.3.1 Task 1: Data collection and preprocessing

The system uses a large volume of data set consisting of face images that have labeled and used for the training of our model. We have used the dataset, which comprises images from github repo face mask detection data set and some additional images (clicked by us). Our dataset contains more than 1376 images, 690 face images with masks and 686 without masks.

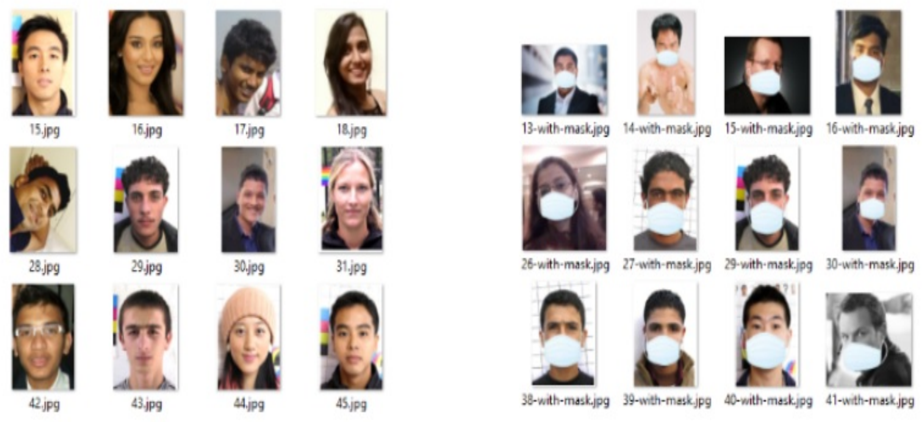


Figure 2.2: Sample Datasets

- **Data Visualization:**

  Data visualization is the process of transforming abstract data to meaningful representations using knowledge communication and insight discovery through encodings. It is helpful to study a particular pattern in the dataset.

  The total number of images in the dataset is visualized in both categories – with mask and without mask.

  The statement $categories = os.listdir(data\_path)$ categorizes the list of directories in the specified data path. The variable categories now looks like: [with mask, without mask]

  Then to find the number of labels, we need to distinguish those categories using $labels = [iforiinrange(len(categories))]$. It sets the labels as: [0, 1]

  Now, each category is mapped to its respective label using $label\_dict = dict(zip(categories, labels))$ which at first returns an iterator of tuples in the form of zip object where the items in each passed iterator is paired together consequently. The mapped variable label_dict looks like: [with mask: 0, without mask: 1]

- **Conversion of RGB image to Gray image:**

  Modern descriptor-based image recognition systems regularly work on grayscale images, without elaborating the method used to convert from color-to-grayscale. This is because the color-to-grayscale method is of little consequence when using robust descriptors. Introducing nonessential information could increase the size of training data required to achieve good performance. As grayscale rationalizes the algorithm and diminishes the computational requisites, it is utilized for extracting descriptors instead of working on color images instantaneously.

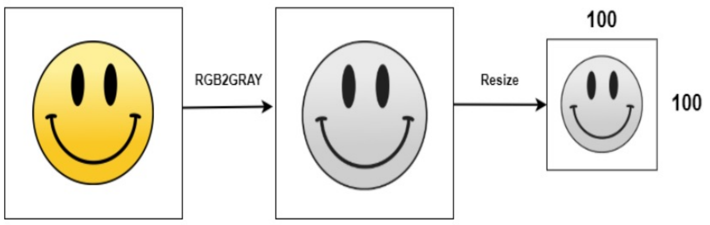


Figure 2.3: Conversion of a RGB image to a Gray Scale image of 100x100 size

We use the function cv2.cvtColor(input_image, flag) for changing the color space. Here flag determines the type of conversion. In this case, the flag cv2.COLOR_BGR2GRAY is used for gray conversion.

Deep CNNs require a fixed-size input image. Therefore we need a fixed common size for all the images in the dataset. Using cv2.resize() the gray scale image is resized into 100 x 100.

- **Image Reshaping:**

  The input during relegation of an image is a three-dimensional tensor, where each channel has a prominent unique pixel. All the images must have identically tantamount size corresponding to 3D feature tensor. However, neither images are customarily coextensive nor their corresponding feature tensors. Most CNNs can only accept fine-tuned images. This engenders several problems throughout data collection and implementation of model. However, reconfiguring the input images before augmenting them into the network can help to surmount this constraint.

  The images are normalized to converge the pixel range between 0 and 1. Then they are converted to 4 dimensional arrays using data=np.reshape(data,(data.shape[0], img_size,img_size,1)) where 1 indicates the Grayscale image. As, the final layer of the neural network has 2 outputs – with mask and without mask i.e. it has categorical representation, the data is converted to categorical labels.

2.3.2 Task 2: Building and training the learning model

- **Building the model using CNN architecture:**

  CNN has become ascendant in miscellaneous computer vision tasks. The current method makes use of Sequential CNN.

  The First Convolution layer is followed by Rectified Linear Unit (ReLU) and MaxPooling layers. The Convolution layer learns from 200 filters. Kernel size is set to 3 x 3 which specifies the height and width of the 2D convolution window. As the model should be aware of the shape of the input expected, the first layer in the model needs to be provided with information about input shape. Following layers can perform instinctive shape reckoning. In this case, input_shape is specified as data.shape[1:] which returns the dimensions of the data array from index 1. Default padding is "valid" where the spatial dimensions are sanctioned to truncate and the input volume is non-zero padded. The activation parameter to the Conv2D class is set as "relu". It represents an approximately linear function that possesses all the assets of linear models that can easily be optimized with gradient-descent methods. Considering the performance and generalization in deep learning, it is better compared to other activation functions. Max Pooling is used to reduce the spatial dimensions of the output volume. Pool_size is set to 3 x 3 and the resulting output has a shape (number of rows or columns) of: shape_of_output = (input_shape − pool_size + 1) / strides), where strides has default value (1,1).

  As shown in figure 2.4, the second Convolution layer has 100 filters and Kernel size is set to 3 x 3. It is followed by ReLu and MaxPooling layers. To insert the data into CNN, the long vector of input is passed through a Flatten layer which transforms matrix of features into a vector that can be fed into a fully connected neural network classifier. To reduce overfitting a Dropout layer with a 50% chance of setting inputs to zero is added to the model. Then a Dense layer of 64 neurons with a ReLu activation function is added. The final layer (Dense) with two outputs for two categories uses the Softmax activation function.

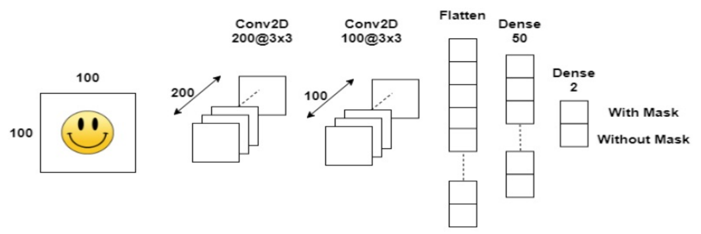


Figure 2.4: Convolutional Neural Network architecture

The learning process needs to be configured first with the compile method. Here "adam" optimizer is used. categorical_crossentropy which is also known as multiclass log loss is used as a loss function (the objective that the model tries to minimize). As the problem is a classification problem, metrics is set to "accuracy".

2.4 Task 3: Deploying the learning model with OpenCV

- **Splitting the data and training the CNN model:**

  After setting the blueprint to analyze the data, the model needs to be trained using a specific dataset and then to be tested against a different dataset. A proper model and optimized train_test_split help to produce accurate results while making a prediction. The test_size is set to 0.1 i.e. 90% data of the dataset undergoes training and the rest 10% goes for testing purposes. The validation loss is monitored using ModelCheckpoint. Next, the images in the training set and the test set are fitted to the Sequential model. Here, 20% of the training data is used as validation data. The model is trained for 20 epochs (iterations) which maintains a trade-off between accuracy and chances of overfitting. Figure 2.1 depicts visual representation of the proposed model.

2.5 Result And Analysis

The model is trained, validated and tested upon two datasets. Corresponding to dataset 1, the method attains accuracy up to 95.77% (shown in figure 2.6). Figure 2.5 depicts how this optimized accuracy mitigates the cost of error. Dataset 2 is more versatile than dataset 1 as it has multiple faces in the frame and different types of masks having different colors as well. Therefore, the model attains an accuracy of 94.58% on dataset 2 depicts the contrast between training and validation loss corresponding to dataset 2. One of the main reasons behind achieving this accuracy lies in MaxPooling. It provides rudimentary translation invariance to the internal representation along with the reduction in the number of parameters the model has to learn. This sample-based discretization process down-samples the input representation consisting of image, by reducing its dimensionality. Number of neurons has the optimized value of 64 which is not too high. A much higher number of neurons and filters can lead to worse performance. The optimized filter values and pool_size help to filter out the main portion (face) of the image to detect the existence of mask correctly without causing over-fitting.

The system can efficiently detect partially occluded faces either with a mask or hair or hand. It considers the occlusion degree of four regions – nose, mouth, chin and eye to differentiate between annotated mask or face covered by hand. Therefore, a mask covering the face fully including nose and chin will only be treated as "with mask" by the model.

The main challenges faced by the method mainly comprise of varying angles and lack of clarity. Indistinct moving faces in the video stream make it more difficult. However, following the trajectories of several frames of the video helps to create a better decision – "with mask" or "without mask".

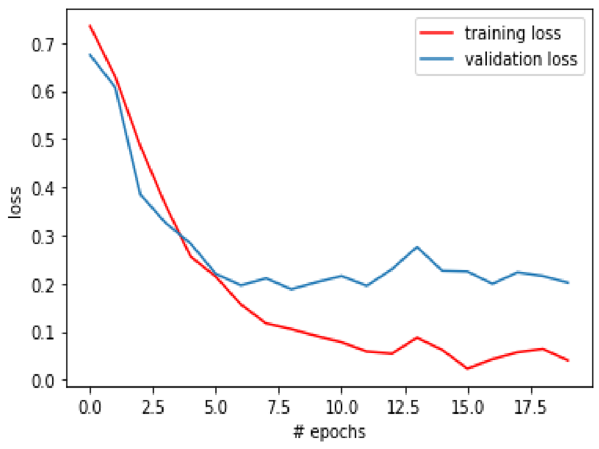


Figure 2.5: epochs vs loss corresponding to dataset 1

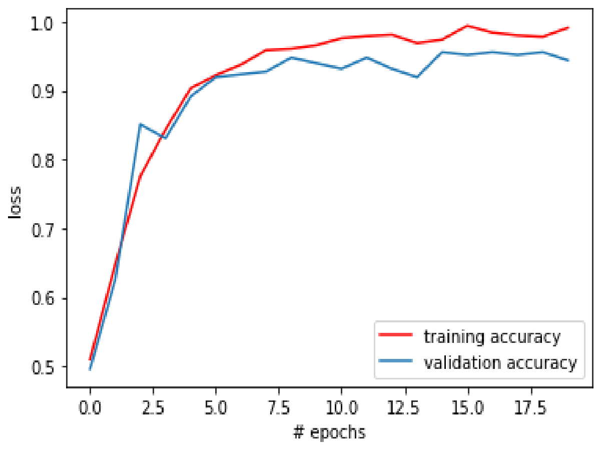


Figure 2.6: epochs vs accuracy corresponding to dataset 1

Chapter 3

Conclusion

Her we briefly explained the motivation of the work at first. Then, we illustrated the learning and performance task of the model. Using basic ML tools and simplified techniques the method has achieved reasonably high accuracy. It can be used for a variety of applications. Wearing a mask may be obligatory in the near future, considering the Covid-19 crisis. Many public service providers will ask the customers to wear masks correctly to avail of their services. The deployed model will contribute immensely to the public health care system.