# Covid19\_Face\_Mask\_Detection

**Background:**

* As recommended by current CDC guidance, masks slow the spread of COVID-19 because they help keep people who are infected from spreading respiratory droplets to others when they cough, sneeze, or talk.
* More than 35 **states** across the country **have mandated** facial coverings.



**Problem:**

* Wearing a face mask in public helps prevent the spread of COVID-19 — but only if worn properly, covering both your nose and mouth.
* It is expensive and prone to error for human to perform mask-wearing screening in public places such as airport, hospital and office.



**Proposal:**

* Build an AI-based face mask detector. Convolutional Neural Network (CNN) model using TensorFlow with Keras library and OpenCV to detect if you are wearing a face mask.
* To help organizations like Airport, hospital, office, supermarkets improve on the previously implemented safety measures

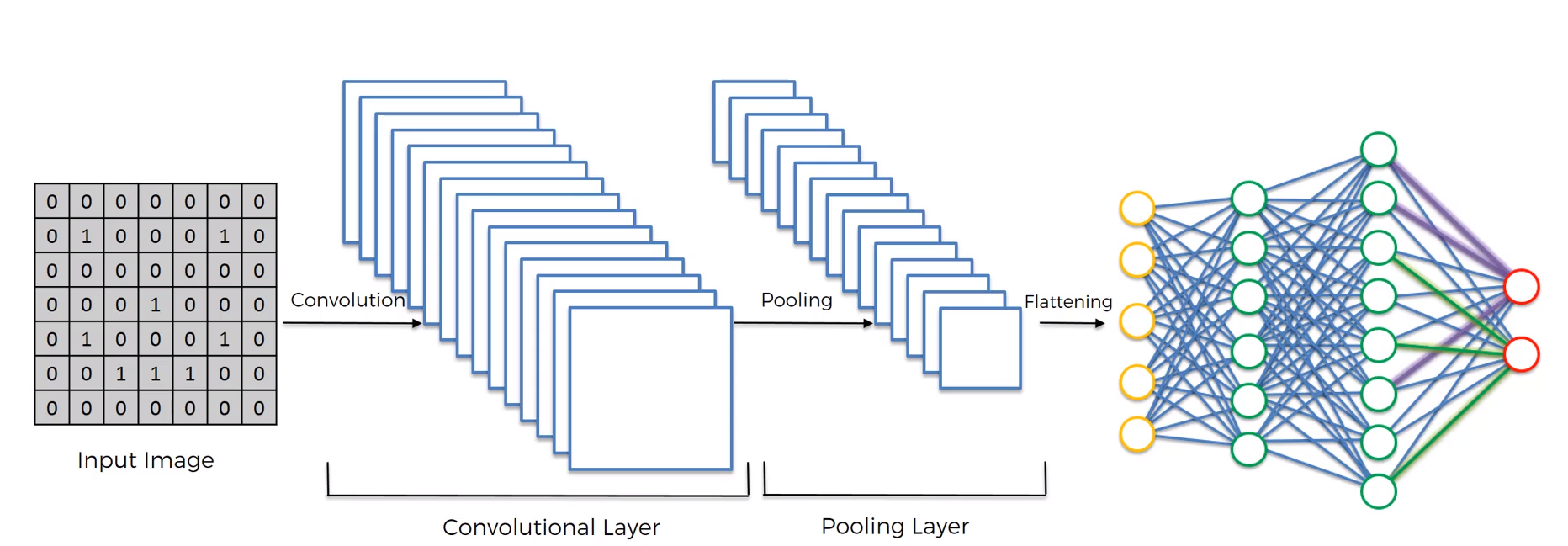
**Data Source:**

1. Flickr-Faces-HQ3 (FFHQ), FFHQ contains 70000 high-quality images of human faces in PNG images at 1024 × 1024 resolution and is free of use. The FFHQ dataset offers a lot of variety in terms of age, ethnicity, viewpoint, lighting, and image background. <https://github.com/NVlabs/ffhq-dataset>.



1. MaskedFace-Net - a dataset of human faces with a correctly or incorrectly worn mask (137,016 images) based on the FFHQ, by Adnane Cabani, Karim Hammoudi, Halim Benhabiles, and Mahmoud Melkemi. <https://arxiv.org/pdf/2008.08016.pdf>

**CNN - Image Analysis**



Given an input image, we apply multiple different feature detectors (also called filters) to create feature maps. And those compose our convolutional layer. On top of this convolutional layer we apply the rectified linear unit to increase non-linearity in our images.

Then we apply a pooling layer to our convolutional layer. So, from every single feature map we created a pooled feature map. The pooling layer has lots of advantages. The main purpose of the pulling layer is to make sure that we have a special invariant in our images. If something is a bit different to the ideal scenario then we can still pick up that feature plus pooling significantly reduces the size of our images. Pooling also helps with avoiding any kind of overfitting of our model to the data because its just simply get rid of lots of data but at the same time preserves the main features we are after.

Then we flatten all the pooled images into one long vector. We input it into an artificial neural network.

The next step is a fully connected artificial neural network where all of these features are processed through a network and then have the final fully connected layer which perform the voting towards the classes that we are after ad then all of this id trained through a forward propagation and back propagation process. Lots of iterations and in the end, we have a very well-defined neural network. Another important thing is not only the weights are trained in artificial neural network part but also the feature detectors are trained and adjusted in that same ingredient decent process and that allows us to come up with the best feature maps.

In the end, we get a fully trained convolutional neural network which can recognize images and classify them.

**Implantation:**

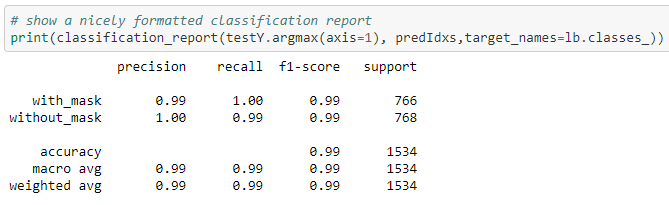
**Step 1: Training Mask Detector**

1. Import libraries (**install tensorflow on 64-bit Python**)
2. Data preprocessing
   * Download with mask and without mask images from data source.
   * Transfer multiple dataset formats (jpg, jepg, png) into jpg in batch.
   * Identify invalid data and remove them.
   * Normalize data size. From 1024 by 1024 into 224 by 224.
3. Transfer image into numpy array
4. image = load\_img(img\_path, target\_size = (224, 224))
5. image = img\_to\_array(image)
6. image = preprocess\_input(image)
7. Building and Training the model

tensorflow.keras.applications.MobileNetV2

MobileNetV2 is a convolutional neural network architecture that seeks to perform well on mobile devices. It is based on an inverted residual structure where the residual connections are between the bottleneck layers.

1. Model accuracy:



1. Serialize the model to disk

model.save("mask\_detector.model", save\_format="h5")

#### **Step 2: Realtime Mask Detection**

1. Import libraries
2. Define detect\_and\_predict\_mask funtion, it requires three parameters:
   * Image frame captured from video
   * Face detection model
   * Mask detection model
3. Initialize the video stream
4. Run the mask detection model

**Demos: (Screenshots from real-time detection demo):**

1. With mask

|  |  |
| --- | --- |
| A picture containing indoor, shelf, clothing, person  Description automatically generated | A picture containing indoor, shelf, person, clothing  Description automatically generated |
| With sunglasses | A person in a green hat  Description automatically generated |
| A person standing in front of a book shelf  Description automatically generated | A picture containing person, indoor, shelf, book  Description automatically generated |
| A picture containing indoor, clothing, person, shelf  Description automatically generated | A picture containing person, indoor, clothing, shelf  Description automatically generated |

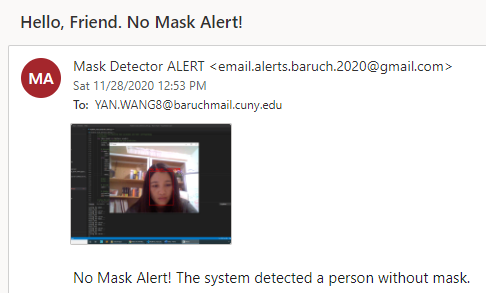
1. Without mask

|  |  |
| --- | --- |
| A person standing in front of a book shelf  Description automatically generated | A person standing in front of a book shelf  Description automatically generated |
| A picture containing indoor, shelf, person, clothing  Description automatically generated | A picture containing indoor, shelf, person, clothing  Description automatically generated |

**Step 3: Addition step:**

Send real-time email alert once without mask people detected.

The security team receive an email with this person’s screenshot.



**Challenges:**

Dataset related:

1. Transfer multiple dataset formats (jpg, jepg, png) into jpg in batch.
2. Normalize data size. From 1024\*1024 into 224\*224.
3. Identify invalid data and remove them.

Performance related:

1. Build and train the model: *tensorflow.keras.applications.MobileNetV2*

MobileNetV2 is a convolutional neural network architecture that seeks to perform well on mobile devices. It is based on an inverted residual structure where the residual connections are between the bottleneck layers. <https://www.tensorflow.org/api_docs/python/tf/keras/applications/MobileNetV2>

1. Learn how to reuse the CNN model without retraining them (In computer science, the process of saving an object to be re-used later is called serialization, while loading a saved object is called deserialization.) <https://towardsdatascience.com/how-to-reuse-your-python-models-without-retraining-them-39cd685659a5>

**Helpful References:**

1. [Deep Learning Tutorials](https://www.udemy.com/course/machinelearning/learn/lecture/6761138#overview)
2. [OpenCV Python Tutorial For Beginners](https://www.youtube.com/watch?v=eX7wXfNLFDw&list=PLS1QulWo1RIa7D1O6skqDQ-JZ1GGHKK-K&index=18)
3. [Deep learning: How OpenCV’s blobFromImage works](https://www.pyimagesearch.com/2017/11/06/deep-learning-opencvs-blobfromimage-works/)
4. [COVID-19: Face Mask Detection using TensorFlow and OpenCV](https://towardsdatascience.com/covid-19-face-mask-detection-using-tensorflow-and-opencv-702dd833515b)
5. [Face Mask Detection using Python, Keras, OpenCV and MobileNet | Detect masks real-time video streams](https://www.youtube.com/watch?v=Ax6P93r32KU&t=918s)
6. Simple Introduction to Convolutional Neural Networks https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac