|  |  |
| --- | --- |
| A practical solution to real-world face mask detection  Face Mask detection | ASHISH ALICHEN  2K19CSUN01127  B.TECH CSE 5A |

**Abstract**

The present scenario of COVID-19 demands an efficient face mask detection application. The project’s main goal is to implement this system at entrances of colleges, airports, hospitals, and offices where chances of spread of COVID-19 through contagion are relatively higher. Reports indicate that wearing face masks while at work reduces the risk of transmission. It is object detection and classification problem with two different classes (Mask and Without Mask). A hybrid model using deep and classical machine learning for detecting face masks will be presented. A dataset is used to build this face mask detector using Python, OpenCV, and TensorFlow and Keras.

**Introduction**

In 2020, the largest pandemic in recent history spread through the world: COVID-19. As of May 1st, 2021, there have already been 152 million cases and 3 million deaths around the world. In many regions, those numbers are considerably under-counted. Beyond that, many parts of the world have slowed or stopped due to the human, economic, and social impacts of distancing and protection measures. For the ongoing pandemic and predictions for future pandemics, this project seeks to create a mask detection system that can recognize whether people in surveillance-type video streams are correctly wearing their masks.

**Methodology**

* Proposed workflow

build a very simple and basic Convolutional Neural Network (CNN) model using TensorFlow with Keras library and OpenCV to detect if you are wearing a face mask to protect yourself. All the aspects of our work are described below.

* Deep learning architecture

The deep learning architecture learns various important nonlinear features from the given samples. Then, this learned architecture is used to predict previously unseen samples.

* Image processing

Haar Cascade Classifier will detect the input from the video cam. The images captured by the system's webcam required pre-processing before going to the next step. In the pre-processing step, the image is transformed into a grayscale image because the RGB colour image contains so much redundant information that is not necessary for face mask detection. Then, we resized the images into (224x224) size to maintain the uniformity of the input images to the architecture. Then, the images are normalized and after normalization, the value of a pixel resides in the range from 0 to 1. Normalization helped the learning algorithm to learn faster and captured necessary features from the images.

* Dataset

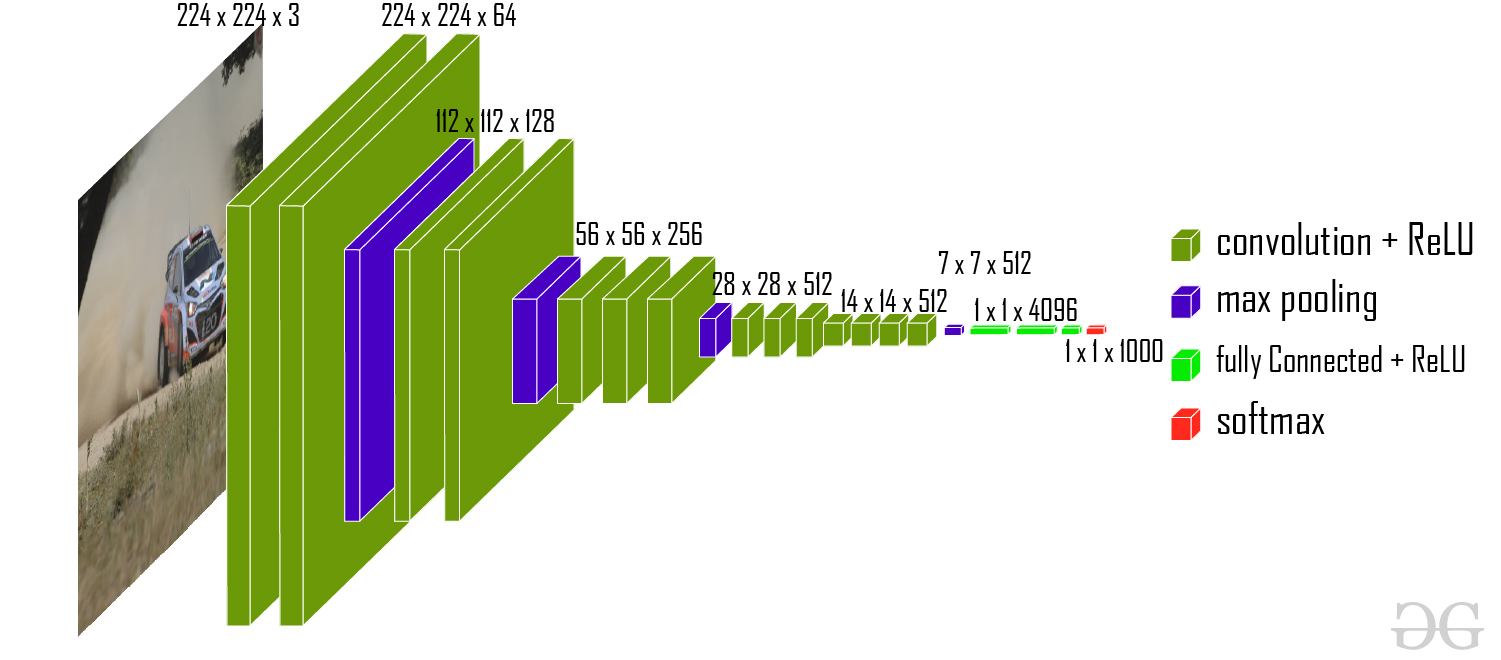
To train our deep learning architecture, we collected images. The architecture of the learning technique highly depends on CNN. Data from an online source is collected for training and testing the model. The dataset contains images of faces only. It consists of about 1,376 images of which 690 images containing people with face masks and 686 images containing people without face masks. For training purposes, 80% of images of each class are used and the rest of the images are utilized for testing purposes.

* CNN model architecture

We’ve used the VGG-16 model. The input to the network is an image of dimensions (224, 224, 3). The first two layers have 64 channels of 3\*3 filter size and the same padding. Then after a max pool layer of stride (2, 2), two layers have convolution layers of 256 filter size and filter size (3, 3). This is followed by a max-pooling layer of stride (2, 2) which is the same as the previous layer. Then there are 2 convolution layers of filter size (3, 3) and 256 filters. After that, there are 2 sets of 3 convolution layers and a max pool layer. Each has 512 filters of (3, 3) size with the same padding. This image is then passed to the stack of two convolution layers. In these convolution and max-pooling stack layers, the filters we use are of the size 3\*3 instead of 11\*11 in AlexNet and 7\*7 in ZF-Net. In some of the layers, it also uses 1\*1 pixel which is used to manipulate the number of input channels. There is a padding of 1-pixel (same padding) done after each convolution layer to prevent the spatial feature of the image.



After the convolution and max-pooling layer, we got a (7, 7, 512) feature map. We flatten this output to make it a (1, 25088) feature vector. After this there are 3 fully connected layers, the first layer takes input from the last feature vector and outputs a (1, 4096) vector, the second layer also outputs a vector of size (1, 4096) but the third layer output 1000 channels for 1000 classes of ILSVRC challenge, then after the output of 3rd fully connected layer is passed to softmax layer to normalize the classification vector. After the output of classification vector top-5 categories for evaluation. All the hidden layers use ReLU as its activation function. ReLU is more computationally efficient because it results in faster learning and it also decreases the likelihood of vanishing gradient problems.



**Working**

* Required libraries

import os

import cv2

import random

import numpy as np

from keras import Sequential

from keras.layers import Dense

from keras.preprocessing import image

from keras.applications.vgg16 import VGG16

from sklearn.model\_selection import train\_test\_split

* Functions

Following code resize the image of datasets to 224x224 and append the image with their labels.

categories = ["with mask", "without mask"]

data = []

for category in categories:

path = os.path.join('train',category)

label = categories.index(category)

for file in os.listdir(path):

img\_path = os.path.join(path,file)

img = cv2.imread(img\_path)

img = cv2.resize(img,(224,224))

data.append([img,label])

Following function returns ‘0’ for with mask and ‘1’ for with mask

def detect\_face\_mask(img):

y\_pred = model.predict\_classes(img.reshape(1,224,224,3))

return y\_pred[0][0]

The following function prompts the whether the face has mask or not

def draw\_label(img,text,pos,bg\_color):

text\_size = cv2.getTextSize(text,cv2.FONT\_HERSHEY\_COMPLEX,1,cv2.FILLED)

end\_x = pos[0] + text\_size[0][0] + 2

end\_y = pos[1] + text\_size[0][1] - 2

cv2.rectangle(img,pos,(end\_x,end\_y),bg\_color,cv2.FILLED)

cv2.putText(img,text,pos,cv2.FONT\_HERSHEY\_SIMPLEX,1,(0,0,0),1,cv2.LINE\_AA)

The following function detects the face. Detecting face is not necessary, however, I did detect face to improve the efficiency because it gives only face as input, without any background. Usually during the live detection background is also captured along with the face, to avoid the background I performed face detection with the help of a cascade classifier.

haar = cv2.CascadeClassifier('haarcascade\_frontalface\_default.xml')

def detect\_face(img):

cods = haar.detectMultiScale(img)

return cods

The following code does the training part. I used 10 epochs which 97% of accuracy

model.compile(optimizer = 'Adam', loss = 'binary\_crossentropy', metrics = ['accuracy'])

model.fit(x\_train,y\_train,epochs = 10,validation\_data = (x\_test,y\_test))

**Code**

import os

import cv2

import random

import numpy as np

from keras import Sequential

from keras.layers import Dense

from keras.preprocessing import image

from keras.applications.vgg16 import VGG16

from sklearn.model\_selection import train\_test\_split

categories = ["with mask", "without mask"]

data = []

for category in categories:

path = os.path.join('train',category)

label = categories.index(category)

for file in os.listdir(path):

img\_path = os.path.join(path,file)

img = cv2.imread(img\_path)

img = cv2.resize(img,(224,224))

data.append([img,label])

random.shuffle(data)

x = []

y = []

for features, label in data:

x.append(features)

y.append(label)

x = np.array(x)

y = np.array(y)

x = x/255

x\_train,x\_test,y\_train,y\_test = train\_test\_split(x,y,test\_size=0.2)

vgg = VGG16()

model = Sequential()

for layer in vgg.layers[:-1]:

model.add(layer)

for layer in model.layers:

layer.trainable = False

model.add(Dense(1,activation='sigmoid'))

model.compile(optimizer = 'Adam', loss = 'binary\_crossentropy',metrics =['accuracy'])

model.fit(x\_train,y\_train,epochs = 10,validation\_data=(x\_test,y\_test))

def detect\_face\_mask(img):

y\_pred = model.predict\_classes(img.reshape(1,224,224,3))

return y\_pred[0][0]

def draw\_label(img,text,pos,bg\_color):

text\_size = cv2.getTextSize(text,cv2.FONT\_HERSHEY\_COMPLEX,1,cv2.FILLED)

end\_x = pos[0] + text\_size[0][0] + 2

end\_y = pos[1] + text\_size[0][1] - 2

cv2.rectangle(img,pos,(end\_x,end\_y),bg\_color,cv2.FILLED)

cv2.putText(img,text,pos,cv2.FONT\_HERSHEY\_SIMPLEX,1,(0,0,0),1,cv2.LINE\_AA)

cap = cv2.VideoCapture(0)

haar = cv2.CascadeClassifier('haarcascade\_frontalface\_default.xml')

def detect\_face(img):

cods = haar.detectMultiScale(img)

return cods

while True:

ret, frame = cap.read()

img = cv2.resize(frame,(224,224))

y\_pred = detect\_face\_mask(img)

cods = detect\_face(cv2.cvtColor(frame,cv2.COLOR\_BGR2GRAY))

for x,y,w,h in cods:

cv2.rectangle(frame,(x,y),(x+w,y+h),(255,0,0),3)

if y\_pred == 0:

draw\_label(frame,'Mask',(30,30),(0,255,0))

else:

draw\_label(frame,'No Mask',(30,30),(0,0,255))

cv2.imshow("window",frame)

if cv2.waitKey(1) & 0xFF == 27:

break

cv2.destroyAllWindows()

**Output**

|  |  |
| --- | --- |
| **A screenshot of a computer  Description automatically generated** | A screenshot of a computer  Description automatically generated with medium confidence |