Face Mask Detection using Convolution Neural Network

Name:- P Ashok Reg. No:- 11710358 Roll No:- RKM042A18

Abstract:-

Face Detection has evolved as a very popular problem in Image processing and Computer Vision. Many new algorithms are being devised using convolutional architectures to make the algorithm as accurate as possible. In the present scenario due to Covid-19, there are no efficient face mask detection applications which are now in high demand for transportation means, densely populated areas, residential districts, large-scale manufacturers and other enterprises to ensure safety.

This system can therefore be used in real-time applications which require face-mask detection for safety purposes due to the outbreak of Covid-19.

Methodology:-

Here, we will be using a face mask dataset made by Prajna Bandhary. It consists of 1316 images having 2 classes of faces i.e., with mask and without mask.

The main aim is to identify and differentiate people with masks and without masks.

In brief, we get the image of a face and run it through a cascade classifier. The cascader will focus on the region of interest and give us details like height and width (a bounding box of the face). Then we resize the height and width to 100x100 and pass it on to the trained CNN model to get the probabilities as an output.

```
[ ] import cv2,os
    from google.colab.patches import cv2_imshow


    data_path='/content/drive/My Drive/dataset'
    categories=os.listdir(data_path)
    labels=[i for i in range(len(categories))]

    label_dict=dict(zip(categories,labels))

    print(label_dict)
    print(categories)
    print(labels)
```

```
{'without mask': 0, 'with mask': 1}
['without mask', 'with mask']
[0, 1]
```

Step 1:- Data Preprocessing

This dataset consists of images having various colours, sizes and orientations. Hence, we convert the images into grayscale to avoid the colour being a critical point for detecting masks. Later, we also check and make sure that all the images are of the size 100x100, before giving it to the neural network.

```
img_size=100
data=[]
target=[]
for category in categories:
    folder_path=os.path.join(data_path,category)
    img_names=os.listdir(folder_path)

    for img_name in img_names:
        img_path=os.path.join(folder_path,img_name)
        img=cv2.imread(img_path)

        try:
            gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
            #Coverting the image into gray scale
            resized=cv2.resize(gray,(img_size,img_size))
            #resizing the gray scale into 100x100, since we need a fixed common size for all the images in the dataset
            data.append(resized)
            target.append(label_dict[category])
            #appending the image and the label(categorized) into the list (dataset)

        except Exception as e:
            print('Exception:',e)
            #if any exception rasied, the exception will be printed here. And pass to the next image
```

```
[ ] import numpy as np
    from keras.utils import np_utils

    data=np.array(data)/255.0
    data=np.reshape(data,(data.shape[0],img_size,img_size,1))
    target=np.array(target)
    new_target=np_utils.to_categorical(target)
```

```
[ ] np.save('data',data)
    np.save('target',new_target)
```

Step 2:- Training the CNN

Our model contains 2 convolutional layers(two conv2D 100@3x3). In the first layer we need to load the dataset after the data preprocessing step. Later we have to arrange / build our convolutional architecture. I've also added a model.add(Dropout(0.5)) to avoid overfitting. As we have only 2 categories, we can use binary_crossentropy as our loss function. We start training for 20 epochs with a model checkpointing for every epoch.

```
[ ]  data=np.load('data.npy')
     target=np.load('target.npy')
     #loading the save numpy arrays in the previous code
```

```
[ ]  from keras.models import Sequential
     from keras.layers import Dense,Activation,Flatten,Dropout
     from keras.layers import Conv2D,MaxPooling2D
     from keras.callbacks import ModelCheckpoint

     model=Sequential()
     model.add(Conv2D(64,(3,3),input_shape=data.shape[1:],activation='relu'))
     model.add(MaxPooling2D(pool_size=(2,2)))
     #The first CNN layer followed by Relu and MaxPooling layers
     model.add(Conv2D(128,(3,3),activation='relu'))
     model.add(MaxPooling2D(pool_size=(2,2)))
     #The second convolution layer followed by Relu and MaxPooling layers
     model.add(Flatten())
     model.add(Dropout(0.5))
     #Flatten layer to stack the output convolutions from second convolution layer
     model.add(Dense(50,activation='relu'))
     #Dense layer of 64 neurons
     model.add(Dense(2,activation='softmax'))
     #The Final layer with two outputs for two categories
     model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
```

```
[ ]  from sklearn.model_selection import train_test_split
     train_data,test_data,train_target,test_target=train_test_split(data,target,test_size=0.1)
```

```
[ ]  checkpoint = ModelCheckpoint('model-{epoch:03d}.model',monitor='val_loss',save_best_only=True,mode='auto')
     history=model.fit(train_data,train_target,epochs=20,batch_size=10,callbacks=[checkpoint],validation_split=0.2)
```

Step 3:- Mask Detection

Now, we have to load the model that we created. Then, we give the path of the image which we want to predict as input to our model.

Next, we have to label the two classes (**0** for **without mask** and **1** for **with mask**). We also set the rectangle bounding box color using **RGB** values. I've used RED and GREEN as the two colors.

```
[ ]  model = load_model('model-017.model')

     face_clsfr=cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')

     labels_dict={0:'without_mask',1:'with_mask'}
     color_dict={0:(0,0,255),1:(0,255,0)}
```

```
[ ]  img=cv2.imread('/content/drive/My Drive/02.jpg')
     gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

     faces = face_clsfr.detectMultiScale(gray,1.3,5)
     if len(faces)>0:
       [x,y,w,h]=faces[0]
       face_img=gray[y:y+w,x:x+w]
       resized=cv2.resize(face_img,(100,100))
       normalized=resized/255.0
       reshaped=np.reshape(normalized,(1,100,100,1))
       result=model.predict(reshaped)

       label=np.argmax(result,axis=1)[0]

       cv2.rectangle(img,(x,y),(x+w,y+h),color_dict[label],2)
       cv2.rectangle(img,(x,y-40),(x+w,y),color_dict[label],-1)
       cv2.putText(img, labels_dict[label], (x, y-10),cv2.FONT_HERSHEY_SIMPLEX,0.8,(255,255,255),2)


       cv2_imshow(img)
     else:
       print("Face not found")
```

OUTPUT:-

Conclusion:-

The project takes images as input and converts them to grayscale images of size 100x100 (our input size). These images are then given as input to the Convolutional Neural Network model and trained. The trained models are saved after every epoch and the model having the best performance metric is chosen as the best model. The model with highest accuracy, i.e, the best model is used for predicting the user uploaded images. After predicting the image, we display whether the face is covered with a mask or not using OpenCV.

The limitation of this project is that, after observing keenly it is found that the model could detect surgical masks compared to other types of mask with higher accuracy.

This project with slight modifications can be integrated with embedded systems for application in airports, railway stations, offices, schools, and public places to ensure that public safety guidelines are followed.