Pyck project- Face Mask Detector

By - Sudarshan Gupta

Introduction

Problem Definition-

This project uses a Deep Neural Network, more specifically a Convolutional Neural Network, to differentiate between images of people with and without masks. The CNN manages to get an accuracy of 99.12% on the training set and 95.88% on the test set. Then the stored weights of this CNN are used to classify as mask or no mask, in real time, using OpenCV. With the webcam capturing the video, the frames are preprocessed and fed to the model to accomplish this task. The model works efficiently with no apparent lag time between wearing/removing mask and display of prediction.The model is capable of predicting multiple faces with or without masks at the same time.

Importance-

In today's scenario, the world is going through this covid-19 pandemic and one of the safety measures (precaution) is to wear the mask. In public places, to ensure that people are wearing masks or not, it becomes really difficult to keep an eye on the persons who are not wearing masks. To get rid of this problem, this project helps us to monitor all those wearing masks or not.

Motivation-

We want to explore the computer vision domain and to implement it in the real time domain, we decided to combine it with a deep learning

technique to finally detect whether a person is wearing a mask or not. Having completed a course on Machine learning last semester, we got a basic idea of deep learning techniques and had the confidence to complete a project on it.

Implementation

1. **What is CNN?**
    - Computer vision is evolving rapidly day-by-day. One of the reasons is deep learning. When we talk about computer vision, a term convolutional neural network(abbreviated as CNN) comes in our mind because CNN is heavily used here. Examples of CNN in computer vision are face recognition, image classification etc. It is similar to the basic neural network. CNN also has learnable parameters like neural network i.e, weights, biases etc.
2. **Layers in CNN-**

    There are five type of layers in CNN-

    - Input layer
    - Convo layer (Convo + ReLU)
    - Pooling layer
    - Fully connected(FC) layer
    - Softmax/logistic layer

- Output layer

Implementation of CNN layers in Code

- Input layer in CNN should contain image data. Image data is represented by three dimensional matrix
- Convo layer is sometimes called feature extractor layer because features of the image are extracted within this layer.
- Pooling layer is used to reduce the spatial volume of the input image after convolution. It is used between two convolution layers. If we apply FC after the Convo layer without applying pooling or max pooling, then it will be computationally expensive and we don't want it. So, max pooling is the only way to reduce the spatial volume of the input image.
- Fully connected layer involve weights, biases, and neurons. It connects neurons in one layer to neurons in another layer. It is used to classify images between different categories by training.
- Output layer contains the label which is in the form of one-hot encoded.

3. Keras Implementation

The data used can be downloaded through this https://data-flair.training/blogs/download-face-mask-data/ . There are 1314 training images and 194 test images divided into two categories, with and without mask.

1. Importing all necessary modules and libraries (numpy, keras, opencv)

```
import numpy as np
import keras
from keras.layers import Conv2D,MaxPooling2D,SpatialDropout2D,Flatten,Dropout,Dense
from keras.models import Sequential,load_model
from tensorflow.keras.optimizers import Adam
from keras.preprocessing import image
from keras.preprocessing.image import ImageDataGenerator
import cv2
```

2. Load the dataset and normalize them by dividing with 255 (train data and test data)

```
train_dataset = ImageDataGenerator(rescale=1./255)
test_dataset = ImageDataGenerator(rescale=1./255)

training_set = train_dataset.flow_from_directory(
        'train',
        target_size=(150,150),
        batch_size=16 ,
        class_mode='binary')

test_set = test_dataset.flow_from_directory(
        'test',
        target_size=(150,150),
        batch_size=16,
        class_mode='binary')
```

3. Setting up the CNN model (adding different layers)

```
model=Sequential()
model.add(Conv2D(32,(3,3),activation='relu',input_shape=(150,150,3)))
model.add(MaxPooling2D() )
model.add(Conv2D(64,(3,3),activation='relu'))
model.add(MaxPooling2D() )
model.add(Conv2D(128,(3,3),activation='relu'))
model.add(MaxPooling2D() )
model.add(Conv2D(256,(3,3),activation='relu'))
model.add(MaxPooling2D() )
model.add(Flatten())
model.add(Dense(100,activation='relu'))
model.add(Dense(1,activation='sigmoid'))

model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
```

(Convolutional layers, MaxPooling layers, Flatten layer, Dense(fully connected layer))
Compiled the model using adam optimizer. There are many different other optimizers, we can also try with them.

4. Parameters and shapes in each layers in our model

We can get this information using the model.summary() command in cmd. Output of this command for our model is-

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 148, 148, 32)      896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 72, 72, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 36, 36, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 34, 34, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 17, 17, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 15, 15, 256)       295168
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 7, 7, 256)         0
_________________________________________________________________
flatten (Flatten)            (None, 12544)             0
_________________________________________________________________
dense (Dense)                (None, 100)               1254500
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 101
=================================================================
Total params: 1,643,017
Trainable params: 1,643,017
Non-trainable params: 0
```

We can observe that the total parameters are- 1643017 and total trainable parameters are 1643017 and Non-trainable parameters are 0.

5. After compiling our model, we trained our model by fit() method

```
model_saved=model.fit_generator(
        training_set,
        epochs=8,
        validation_data=test_set,)
model.save('CNN_model.h5',model_saved)
```

We saved our model in a file named CNN_model.h5, so that we don't need to train our model everytime we use our program. We can

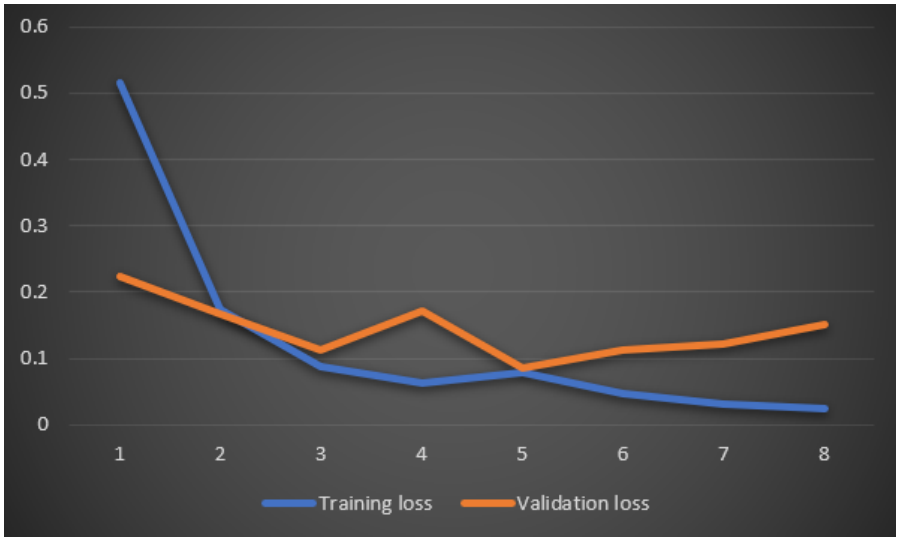
comment out the training code after compiling it one time, and load the saved model next time directly.

6. Evaluating the model after training-

```
Epoch 1/8
83/83 [==============================] - 36s 308ms/step - loss: 0.5168 - accuracy: 0.7064 - val_loss: 0.2238 - val_accuracy: 0.9175
Epoch 2/8
83/83 [==============================] - 26s 308ms/step - loss: 0.1745 - accuracy: 0.9430 - val_loss: 0.1672 - val_accuracy: 0.9433
Epoch 3/8
83/83 [==============================] - 27s 324ms/step - loss: 0.0883 - accuracy: 0.9744 - val_loss: 0.1118 - val_accuracy: 0.9588
Epoch 4/8
83/83 [==============================] - 27s 325ms/step - loss: 0.0636 - accuracy: 0.9757 - val_loss: 0.1710 - val_accuracy: 0.9433
Epoch 5/8
83/83 [==============================] - 26s 315ms/step - loss: 0.0786 - accuracy: 0.9670 - val_loss: 0.0865 - val_accuracy: 0.9742
Epoch 6/8
83/83 [==============================] - 26s 309ms/step - loss: 0.0480 - accuracy: 0.9860 - val_loss: 0.1117 - val_accuracy: 0.9742
Epoch 7/8
83/83 [==============================] - 25s 307ms/step - loss: 0.0307 - accuracy: 0.9892 - val_loss: 0.1217 - val_accuracy: 0.9742
Epoch 8/8
83/83 [==============================] - 26s 313ms/step - loss: 0.0235 - accuracy: 0.9912 - val_loss: 0.1519 - val_accuracy: 0.9588
Model: "sequential"
```

After training, we got an accuracy of 99.12% on the training set and 95.88% on the test set.

7. Graph for training and validation loss and accuracy



Training and validation loss

Training and validation accuracy

4. Live Detection Result

Without Mask

With Mask

Problems Faced-

1. While dealing with the layers, I need to find out about the different arguments and the output shape of each layer to predict which layer to use next.
2. Use of each layer and the role of dimensions in layers input and output.
3. Handling train and test data. It took much time to get done with the CNN model overall.
4. Face Detection using Open CV. Didn't know initially about the haarcascade frontalface default.

Conclusion-

The goal of this project was to build a CNN model which can distinguish between two labels. We implemented this idea to detect whether a person is wearing a mask or not. We applied different

convolutional layers and pooling layers, and successfully built the CNN model. We also implemented the live detection of a person's face using the OpenCV library of python and linked it with CNN model to output the result of wearing a mask or not. We got successful in building the model and testing it with an accuracy nearly 95%.