Automatic face mask detection

Obviously in the current climate, we must wear face masks. We thought this was an apt problem to attempt to solve and involved many different tasks, such as working with opencv, keras and flask. https://www.pyimagesearch.com/2020/05/04/covid-19-face-mask-detector-with-opencv-keras-tensorflow-and-deep-learning/. This tutorial served as inspiration for this project.

Face detection

Using Opencv, a computer vision library usable in C++ and python, read in a video stream of the camera from your pc to get a video stream. Now we want to isolate the face of the person in the video stream. A video stream is just a continuous array of images, so each frame is isolated and then treated as an image. A very simple fast method is known as the Haar cascade classifier which is what is used here for speed purposes. It is less accurate at detecting faces than the Opencv DNN Caffe based face detector. Both work in the same way and identify n faces in an image and then returns n bounding box co-ordinates for each of n faces in the image. So now we have identified the faces/faces in the video stream.

Face mask model creation

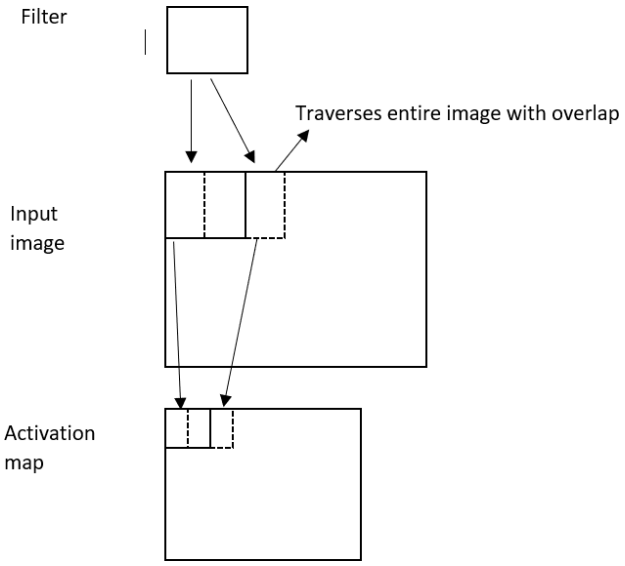
Now we must create the model which can identify if a face possesses a mask or not. Simply, we need a dataset to train a model to do this. One can obtain a massive dataset of faces online and then obtain an image of a mask and supersede a mask over faces in half of these images which creates a balanced dataset. However, a premade dataset exists here, https://github.com/prajnasb/observations/tree/master/experiements/data. With around 700 images each per class, all are colour images with the majority being white and light blue face masks.

The type of model we create is a CNN, a convolutional neural network.

CNN's are a category of Neural networks that have proven very effective in image recognition and classification tasks. A CNN generally consists of four components:

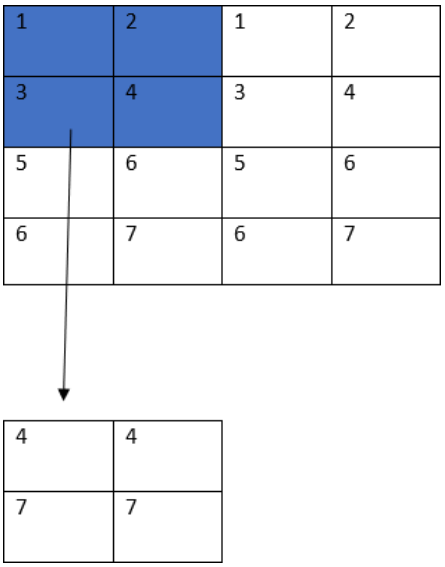
1. Convolutional Layer
2. Activation Function
3. Pooling Layer
4. Fully connected Layer

1. A Convolutional layer is used to extract features from the input image. Essentially a kernel or filter of mxn pixels in size hovers over each mxn section of the input image and performs point wise multiplication and then adds these values which results in a single pixel value. The traversing of this kernel across a whole filter produces something known as an activation map which will be smaller than the original image. The values which are present in this filter/kernel is what is learnt through the training process.



2. Rectified Linear Unit (RELU) is a very commonly used activation *function*. RELU adds non-linearity to the activation map. Features within an image such as edges are naturally non- linear as there is an abrupt difference between these features and the surrounding objects and have no smooth linear relationship between them. Using RELU aids the model in extracting these non-linear features. RELU has also been shown to train networks faster than other activation functions.

3. A pooling layer is used to down sample the activation map. A window is chosen in which to traverse the activation map. Max pooling (choosing the max values from the window) is commonly used in image recognition tasks due the fact it is good at highlighting the most prevalent features in the activation map. Average pooling will smooth the image so the most prevalent features may not be retained so max pooling was chosen. The pooling layer helps to down sample the activation map and reduce computation time and memory usage of the CNN.



4. The output from Convolutional and Pooling Layers are representative of high-level features. Using a fully connected layer is a way to combine all these high-level features in multiple combinations to classify input images to different classes. The high-level features could be used themselves to classify but combing these features together can result in better predictions. The output of a fully connected layer is the final prediction of the input image. A common function to output a prediction per class for the image is to use the Softmax function. Softmax will output a prediction probability per class which sum to 1.

We can produce a model using this data ourselves which would not be hard but using a pretrained model which is designed to detect similar features to our model would be better as it has been trained on millions of images. A trained network called mobileVnet2 is lightweight and can be run on a device with limited computational capability like a smartphone CPU.

Building on top of a model like we have done is known as transfer learning. We essentially download the fully trained mobilevnet2 model and retain the entire model except for its final fully connected layer which links its learned features to its n number of classes which it was trained to classify. We then add our own fully connected layer which allows MobileVnet2 to link its learned features to the 2 classes in our dataset, face mask or not face mask.

Once this model is trained, you now have a trained model to identify whether someone is wearing a face mask or not. To test this model, we must input images of the exact same size that mobilevnet2 has been trained on which is the size of 224x224. These images also must be colour images as mobilevnet2 only works on colour images.

So now both parts need to be joined, your face detection segment returns bounding boxes with faces inside it, the trick now is to isolate this bounding box and resize to 224x224 and this is what you will use to test the model with.

This will work well if you sit with nothing in the background as the haar cascade face detection model is quite inaccurate, and if you wear a white or light blue mask, otherwise it may not work as you expect. This is the limit of this project as with time constraints, more was not doable.

Further work

Firstly, one can separate the model prediction step into an independent service which can be setup in flask in python where it is a service open on a port on your localhost. Then you can hit this service by sending the video stream of your pc camera using a http get request to receive a probability of a face with a face mask being present in the video stream. This is quite basic as a prediction is made per stream. We are not sure of how to make this more efficient as something like Kafka is not optimized for video stream data.

Once this software as a service architecture is setup, you can place the service section of the code inside a docker container and test locally. The goal of this is to be able to then deploy this docker container on AWS ECS. This will work exactly like using a local flask service, only the address of the http request you send will now be a live web address and not an address on your local pc network.