DJ SYNAPSE TASK 2 – SYNOPSIS/COMPUTER VISION

Name: Devang Shah
Branch: FE Computer engineering
SAP ID/Class: 60004200158/J2

Problem Statement: Given a dataset of human face images taken from a dashboard of a car, detect if the driver is drowsy or not. Note that a driver can be classified as drowsy if their eyes are closed more often than open.

- Describe the preprocessing steps required on the image (if any).
- Describe techniques you would use to detect the eyes in the entire image.
- Describe techniques you would use to detect drowsiness in the eyes.
- End the synopsis with a conclusion which should contain information about why you selected a particular algorithm over other existing solutions.

Note: All modelling techniques must be described in detail. Mathematical explanations would fetch you brownie points! (Code for the same is optional). Consider this as a supervised learning problem where you have been given images of open and closed eyes for training.

1.) Pre-processing steps required on the image:

(a.) Image resizing - Neural networks need all input images to have the same shape and size, because processor applies the same instruction and operations to a batch of images at the same time in order to be super-fast. We can use transforms.compose(pytorch) or cv2.resize function.

(b.) Normalization - We can use the transforms.normalize() or cv2.normalize() function to apply visual normalization in order to fix very dark/light/contrast/saturated pictures because the model needs similar types of images to be fed.

(c.) Gray scaling (not mandatory) - In order to avoid distractions in facial image classification it could be a good idea to use black and white pictures. But this may make eye detection a bit difficult because eye is also a block spot surrounded by white. You may use PIL attribute image.convert('L') or cv2.cvtColor(image, cv2.COLOR_BGR2GRAY).

Additional image processing steps after detecting face and eyes in the image may include cropping of portion of our interest to minimize noise, face-straightening to clean up the image, albumentations, etc.

2.) Techniques you would use to detect the eyes in the entire image:

(Pls check my task 1. ipynb file. I have used eye detection code there)

I would be using the Haar Feature-based Cascade Classifier for Face and eye detection. It is included in open cv library itself. Will just need to fetch the .xml files beforehand. The Haar feature continuously traverses pixel by pixel from the top left of the image to the bottom right to search for the particular feature. Haar Cascade will detect only open eyes and draw a bounding box around it. So, if there are no detections or boxes found in the image using the Haar

Cascade Algo then it is obviously a closed eye. The model will learn(train) itself on these images of closed or open eyes.

3.) Techniques you would use to detect drowsiness in the eyes:

Now the model has learnt from the training dataset, we can use it on dynamic data also. We will need a running video of a car driver here. We then split the video into various frames using OpenCV. We now consider a single frame → detect faces in those individual frames → If there are faces, then we will detect eyes (then, we may crop and remove the other portions not useful to us) → then we determine whether the eyes are closed or open (as done in step 2) → If the eyes are detected to be open, then no issues. But if eyes are detected to be closed continuously for more time (that is specific no. of frames based on frame rate), then it means the driver is drowsy. However, if the closed state of eyes is not continuous, then it can be a blink.

4.) Conclusion:

Haar Cascade Detection is one of the oldest yet powerful feature detection algorithms invented. The models are stored on GitHub, and we can access them with OpenCV methods easily with just 5 lines of code you can actually detect features. The algorithm is highly accurate and works well with most of the images. It is pre-trained model, so the training-time is saved as compared to other CNNs (calculating loss, setting optimizer, fine tune hyper-parameters, etc). These Haar Cascades algo. are very fast. Computationally cheap and works well even when resources are constrained.

END