# Driver Fatigue Detection Based on MTCNN and EM-CNN

(Computer Vision)

There are many techniques of detecting drowsiness or fatigue in a driver. There are many algorithms which uses steer-wheel angle, psychological indicators, eye-detection, full face detection, respiratory sensors, etc. In this algorithm, the multitask cascaded convolutional network (MTCNN) architecture is employed in face detection and feature point location (eyes, nose and mouth), and the region of interest (ROI) is extracted using feature points. A convolutional neural network, named EM-CNN, is proposed to detect the states of the eyes and mouth from the ROI images. The percentage of eyelid closure over the pupil over time (PERCLOS) and mouth opening degree (POM) are two parameters which are used for fatigue detection.

EM-CNN is based on a state recognition network which is used to classify eye and mouth states (i.e., open or closed). In machine vision-based fatigue driving detection, blinking frequency of the eye and yawning are important indicators for judging driver fatigue. EM-CNN can reduce the influence of factors such as changes in lighting, sitting, and occlusion of glasses and some other complex environmental situations. This method combines multiple levels of features by cascading two unique CNN structures. Face detection and feature point location are performed based on MTCNN, and the state of eyes and mouth is determined by EM-CNN. Binocular images (image consisting of both eyes), rather than monocular images) are detected to obtain abundant eye features.

## MTCNN:

MTCNN or Multi-Task Cascaded Convolutional Neural Networks is a neural network which detects faces and facial landmarks on images. It was published in 2016 by Zhang et al.

Face detection is challenging in real-world scenarios due to changes in driver posture and unconstrained environmental factors, such as illumination and occlusion. By using the depth-cascading multitasking

MTCNN framework, face detection and alignment can be completed simultaneously, the internal relationship between the two is exploited to improve performance, and the global face features are extracted; thus, the positions of the face, left and right eyes, nose, and the left and right corners of the mouth can be obtained. MTCNN comprises of three cascaded subnetworks, i.e., P-Net (Proposal Network), R-Net (Refined network), and O-Net (Output network).

*P-Net:* After passing the images in the program, an image pyramid is created, in order to detect faces of all different sizes. In other words, different copies of the same image in different sizes to search for different sized faces within the image. A 12 x 12 stage 1 kernel that will go through every part of the image, scanning for faces. This portion of the image is passed to P-Net, which returns the coordinates of a bounding box if it notices a face. Then, it would repeat that process by shifting the 12 x 12 kernel 2 pixels right or down at a time. The shift of 2 pixels is known as the stride, or how many pixels the kernel moves by every time. Having a stride of 2 helps reduce computation complexity without significantly sacrificing accuracy.

The weights and biases of P-Net have been trained so that it outputs a relatively accurate bounding box for every 12 x 12 kernel. However, the network is more confident about some boxes compared to others. Thus, there is a need to parse the P-Net output to get a list of confidence levels for each bounding box, and delete the boxes with lower confidence (i.e., the boxes that the network isn't quite sure contains a face). Even after picking the boxes with higher confidence, still a lot of bounding boxes get left out, and a lot of them overlap. Non-Maximum Suppression, or NMS, is a method that reduces the number of bounding boxes.

The area of each of the kernels are calculated, as well as the overlapping area between each kernel and the kernel with the highest score. The kernels that overlap a lot with the high-scoring kernel get deleted. Finally, NMS returns a list of the "surviving" bounding boxes. Afterward, the bounding box coordinates are converted to coordinates of the actual image. The coordinates of each bounding box are a value between 0 and 1, with (0,0) as the top left corner of the 12 x 12 kernel and (1,1) as the bottom right corner. By multiplying the coordinates with the actual image width and height, the bounding box coordinates are converted to standard, real-sized image coordinates.

Since the bounding boxes may not be square, the bounding boxes are reshaped to a square by elongating the shorter sides (if the width is smaller than the height, it is expanded sideways; if the height is smaller than the width, it is expanded vertically).

*R-Net:* Sometimes, an image may contain only a part of a face peeking in from the side of the frame. In that case for every bounding box, an array of the same size is created, and the pixel values (the image in the bounding box) are copied to the new array. If the bounding box is out of bounds, only the portion of the image in the bounding box is copied to the new array and everything else is filled with a 0. This process of filling arrays with 0s is called padding. After padding the bounding box arrays, they are resized to 24 x 24 pixels, and normalized to values between -1 and 1.

Now, they are fed to R-Net and their output is gathered. R-Net's output is similar to that of P-Net: It includes the coordinates of the new, more accurate bounding boxes, as well as the confidence level of each of these bounding boxes. Once again, the boxes with lower confidence are not discarded, and NMS is performed on every box to further eliminate redundant boxes. Since the coordinates of these new bounding boxes are based on the P-Net bounding boxes, they need to be converted to standard coordinates.

After standardizing the coordinates, the bounding boxes are reshaped to a square to be passed on to O-Net.

*O-Net:* Before the bounding boxes are passed from R-Net, first padding is done to any boxes that are out-of-bounds. Then, after the boxes are resized to 48 x 48 pixels, the bounding boxes can be passed into O-Net. The outputs of O-Net are slightly different from that of P-Net and R-Net. O-Net provides 3 outputs: the coordinates of the bounding box (out [0]), the coordinates of the 5 facial landmarks (out [1]), and the confidence level of each box (out [2]).

Once again, the boxes with lower confidence levels are discarded, and both the bounding box coordinates and the facial landmark coordinates are standardized. Finally, they are run through the last NMS. At this point, there should only be one bounding box for every face in the image.

## Region Of Interest (ROI) Extraction:

Generally, most eye detection methods only extract one eye to identify a fatigue state. However, when the driver's head shifts, using information from only a single eye can easily cause misjudgement. Therefore, to obtain more eye information and to accurately recognize eye state, a two-eye image is extracted to determine whether the eyes are open or closed. The position of the driver's left and right eyes is obtained by using the MTCNN network. Here, the position of the left eye and right eye are considered as $a_1(x_1, y_1)$ and $a_2(x_2, y_2)$ respectively, the distance between the left and right eye is $d_1$, and the width of the eye image is $w_1$. The height is $h_1$, according to the proportion of the face "three courts and five eyes," the binocular images are intercepted, and the correspondence between width and height is expressed as follows:

$$d_1 = \sqrt{(x_1 - x_2)^2 - (y_1 - y_2)^2}$$

$$w_1 = 1.5d_1$$

$$h_1 = d_1$$

A driver's mouth region changes significantly when talking and yawning. Here, the positions of the left and right corners of the mouth are obtained using the MTCNN network. The position of the left corner and the right corner of the mouth are considered as $b_1(x_3, y_3)$ and $b_2(x_4, y_4)$ respectively. The distance between the left and right corners is $d_2$, the width of the mouth image is $w_2$, and the height is $h_2$, similar to the eye region extraction. The correspondence between width and height is expressed as follows:

$$d_2 = \sqrt{(x_3 - x_4)^2 - (y_3 - y_4)^2}$$

$$w_2 = d_2$$

$$h_2 = d_2$$

## EM-CNN:

When the driver enters the fatigue state, there is usually a series of physiological reactions, such as yawning and closing the eyes. According to the EM-CNN, multiple states of the eyes and mouth are acquired, and the fatigue state of the driver is evaluated by calculating the eye closure degree PERCLOS and mouth opening degree POM.

*PERCLOS:* PERCLOS parameter indicates the percentage of eye closure time per unit time.

$$PERCLOS = \frac{\sum_i^N f_i}{N} \times 100\%$$

Here, $f_i$ represents the closed frame of the eye, $\sum_i^N f_i$ represents the number of closed-eye frames per unit time, and $N$ is the total number of frames per unit time. To determine the fatigue threshold, many frame sequences or image datasets have to be collected to test and calculate their PERCLOS value. The PERCLOS value will vary from data set to data set.

*POM:* Similar to PERCLOS, POM represents the percentage of mouth closure time per unit time:

$$POM = \frac{\sum_i^N f_i}{N} \times 100\%$$

Here, $f_i$ indicates the frame with the mouth open, $\sum_i^N f_i$ indicates the number of open mouth frames per unit time, and $N$ is the total number of frames per unit time.

## Fatigue State Recognition:

After the neural network pretraining is completed, the fatigue state is identified based on the fatigue threshold of PERCLOS and POM. First, the face and feature point positions of the driver frame image are obtained by the MTCNN, and the ROI area of the eyes and mouth is extracted. Then, the state of the eyes and mouth are evaluated by the EM-CNN. Here, the eye closure degree and mouth opening degree of the continuous frame image are

calculated, and the driver is determined to be in a fatigue state when the threshold is reached. The threshold values will be based on the model used and the training data.

Combined with the temporal correlation of eye state changes, judging by the blink frequency, fatigue driving is to mark the state of the eyes in the video frame sequence. If the detected eye state is open, it is marked as “1”; otherwise, it is marked as “0”, and the blink process in the video frame sequence can be expressed as a sequence of “0” and “1”. The frequency of yawning is used to judge the fatigue driving and is to mark the mouth state in the video frame sequence. If the detected mouth state is open, it is marked as “1”; otherwise, it is marked as “0”, then the beat in the videoframe sequence, the yawning process can be expressed as a sequence of “0” and “1.” Thresholds are related to different methods used in different studies. Thresholds for PERCLOS and POM should be obtained from experiments.

According to the research article on “*Driver Fatigue Detection Based on Convolutional Neural Networks Using EM-CNN*”, this proposed algorithm EM-CNN outperforms other CNN-based methods, i.e., AlexNet, VGG-16, GoogLeNet, and ResNet50, showing accuracy and sensitivity rates of 93.623% and 93.643% respectively.

**Pre-processing Steps:** The images recorded from the video frames needed to be converted to RGB images using the OpenCV library function cv2.cvtColor(“Image”, cv2.COLOR_BGR2RGB), so that the eyes, nose and corner points of the mouth can be detected clearly.

**Reason for selecting this technique:** A model based on an algorithm or algorithms should just not only focus high rate of success, but it should also be able to execute the given operation in less amount of time. MTCNN isn’t just a plain old neural network: it utilizes some interesting techniques to achieve high accuracy with less run-time.

Low computation complexity results in a fast run-time. To achieve real-time performance, it uses a stride of 2, reducing the number of operations to a quarter of the original. It also doesn’t start finding facial landmarks until the

last network (O-Net), after the bounding box coordinates have been calibrated, which narrows down the coordinates of facial features even before it begins its calculations. This makes the program a lot faster as it only has to find facial landmarks in the few boxes that pass-through O-Net.

High accuracy is achieved with a deep neural network. Having 3 networks — each with multiple layers — allows for higher precision, as each network can fine-tune the results of the previous one. In addition, this model employs an image pyramid to find faces both large and small. Even though this may produce an overwhelming amount of data, NMS, as well as R-Net and O-Net, all help reject a large number of false bounding boxes.

Also, I do not believe that fatigue or drowsiness detection can only be detected by the fact that eyes are opened or close. There's no use telling the driver that he/she is drowsy after they have closed their eyes for some time, in which the accident may already occur, so it is beneficial to predict it beforehand. It has to be done by the entire posture of the person, but since the driver is sitting, we will only be able to detect its face from far, but then also there are many signs of fatigue that can be detected by only seeing the face, like fast blinking, angle between the head and shoulder, yawning, droopy shoulders, etc.

The driver should also feel comfortable while driving, otherwise there might be other consequences which may happen due to the physiological sensors or respiratory function detectors, because they have to attached to the drivers (or human skin).

## Code for MTCNN:

```
import mtcnn
# print version
print(mtcnn.__version__)

import matplotlib.pyplot as plt
# load image from file
filename = "glediston-bastos-ZtmmR9D_2tA-unsplash.webp"
pixels = plt.imread(filename)
print("Shape of image/array:",pixels.shape)
imgplot = plt.imshow(pixels)

# draw an image with detected objects
def draw_facebox(filename, result_list):
# load the image
data = plt.imread(filename)
```

```
# plot the image
plt.imshow(data)
# get the context for drawing boxes
ax = plt.gca()
# plot each box
for result in result_list:
# get coordinates
x, y, width, height = result['box']
# create the shape
rect = plt.Rectangle((x, y), width, height, fill=False, color='green')
# draw the box
ax.add_patch(rect)
# show the plot
plt.show()

# filename = 'test1.webp' # filename is defined above, otherwise uncomment
# load image from file
# pixels = plt.imread(filename) # defined above, otherwise uncomment
# detector is defined above, otherwise uncomment
#detector = mtcnn.MTCNN()
# detect faces in the image
faces = detector.detect_faces(pixels)
# display faces on the original image
draw_facebox(filename, faces)

# draw the dots
for key, value in result['keypoints'].items():
# create and draw dot
dot = plt.Circle(value, radius=20, color='orange')
ax.add_patch(dot)
```