Eye Control using Deep Learning

Feras Zaher Alnaem
TH Köln
Feras.Zaher_alnaem@smail.th-koeln.de

Bassem Boutros
TH Köln
Bassem.Boutros@smail.th-koeln.de

Abstract:

In this paper we proposed an implementation of a hybrid model of convolutional neural network (CNN) and recurrent neural network (RNN) in order to detect eye motions that can be used later for controlling devices. We validated our work by creating a real-time vision system which accomplishes the task of face detection, eyes detection and eye motions classification, to distinguish between 4 different motions. The four motions are No Eye blinking, Both Eye Blinking, Left Eye Blinking and Right Eye blinking.

In the scope of our project we created our own dataset which was prepared using our own code to capture pictures frames from webcam, detect the face and Eye using HAAR cascade, processing those raw images, resize them and then append all frames together to form a video that captures the eye motion. Each sample of the dataset should contain two videos one for the left Eye and the other for the right eye.

We Followed by creating the model, in which we tried and tested different new algorithms till we achieved good training results in short time and with high accuracy. That was accomplished after implementing the same concept of Attention layer in video frames, where different frames of the videos were given different weights to ensure that less relevant frames which doesn’t contain any motion are ignored. All that together ended with accuracy reaching up to 96%.

Our captured data was around 30 samples only per motion which means a total of around 120 samples for the four different motions, that reached after augmentation to around 600 sample. Where augmentation is done by creating our own customized code to rotate each frame of the video by different predefined angel and then append the whole frames back to have new videos. We might need to take more samples for future improvement in detection where we can say that 100 sample per motion would be more than enough to classify different eye motions on different eyes.

Finally, we created our own code to run live detection, by capturing frames and detect faces on these frames and once face is detected it detects the eyes and do live classification based on the stored trained model. That can be customized in a later phase to perform different actions according to the detected motion.

1.Introduction:

Eye tracking technologies play an important role in many fields, and it will control the future of mobile devices and applications. However, changes in face pose, illumination, variation, eye occlusions, etc. can make it difficult to detect eyes well from facial images. Our project proposes a model for eye detection in order to trigger actions on computer. This project could help people in their daily activities by using the computer while they are just laying down. It could be helpful for people with congenital disorders (the absence of arms and legs). Also, could be used in various professions as Doctors in operations rooms to send commands to computers. Our model is trained to detect eye blinks but can be trained and used for other eye motions or any other kind of motions. It can distinguish between different motions. (No blinks, both eyes blinks, left eye blink and right eye blink). That is done through the following steps: see Figure 1

- Capture videos using webcam.
- Detecting faces followed by Eye detection in each frame.
- Process images and extract features using neural networks.
- Keep a running history of last few frame's extracted features.
- Predict the current eye movement based on history.

2. Related work:

The idea of combining CNN's and RNN's was already implemented in previous works but with different model structure. Our concern was to create different model with higher accuracy, that was done by implementing different new algorithms as adding a new layer between CNN and RNN, which we call it, the attention layer. By using this idea, we make it easier for the RNN to predict the motion through giving different weights to frames, where frames that contains the motions were given high weights and other frames contains less weights. That improved the accuracy of our model where it reached around 96% using only a small amount of data set. Where we recorded only 30 samples per motion.

3. Data:

To build our dataset we wrote our own code that record videos using computers webcam. This has the advantage of using the same source for both training the model and making the predictions.

As our proposal to detect the motion of

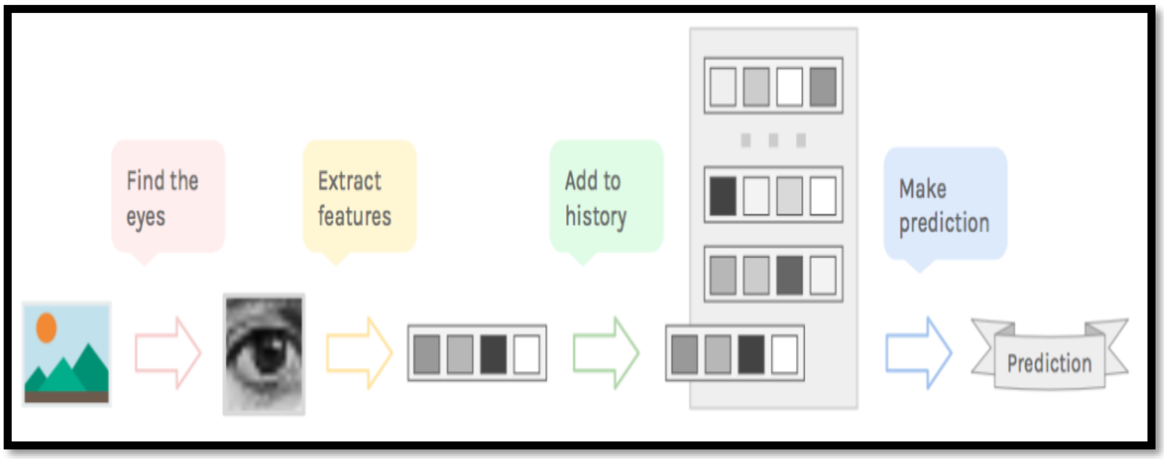


Figure 1. Project architecture

eyes, we captured frames of our eyes. Each video should contain 20 captured frames then we computed the differences between each two consecutive frames to represent the motion. The difference between two frames is a kind of normalization to the data.

The process of preparing the dataset is simply consist of the following steps:

- Capturing frames or pictures.
- Face detection using HAAR Cascade in each frame.
- After successful face detection, HAAR Cascade is used again to detect eyes, a frame is accepted if both eyes are detected in same frame else the whole frame will be ignored.
- Resizing of frames to ensure that all frame sizes are equals.
- Repeat the above steps till 20 accepted frames are captured.
- Compute the different between two consecutive frames as a kind of normalization to data.
- Append the new frames together to have a video.
- Finally, data augmentation.

3.1 Recording:

In the recording phase as mentioned above we used HAAR cascade for both face detection then eye detection. We used the HAAR tool since its easy and fast. A frame is accepted if and only if both eyes are detected in same frame else whole frame is ignored.

In our scenario we recorded 30 samples for each of the motions listed below which means a total of around 120 sample.

- N: No blinks.
- B: Both eyes blinks.
- R: Right eye blink.
- L: Left eye blink.

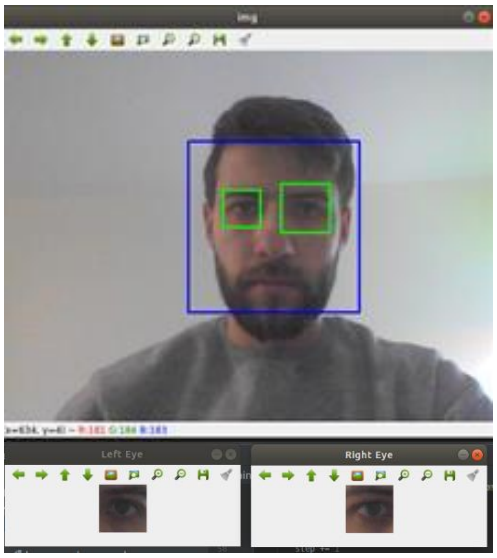


Figure 2.Detection using HAAR cascade

3.2. Preprocess the data:

After successful detection of both eyes, we have to process the captured frames to use them in the dataset. We do that by first resizing all frames to a fixed size (24*24 pixels). As in case a moving head back and forth infront of the webcam while recording the motion will result in different eye sizes in the captured frames. So, we have to do resizing to ensure that all the video frames of the eyes have same dimensions.

That is followed by converting the frames to grayscale images to reduce the data size. Finally comes an important step for dataset preparation which is computing the difference between two frames, that step is kind of normalization to the pixel value, Now we have to append the frames together to get a valid video or sample. During the simulation we noticed sometimes there is shuffle of the eyes (left eye stored in the right eye file and vice versa.), so we have to make sure before saving the two pictures that there is no shuffle. That was done by checking the y-axis position of both eyes on the main face frame and re-shuffle the two frames if shuffle of eyes is detected.

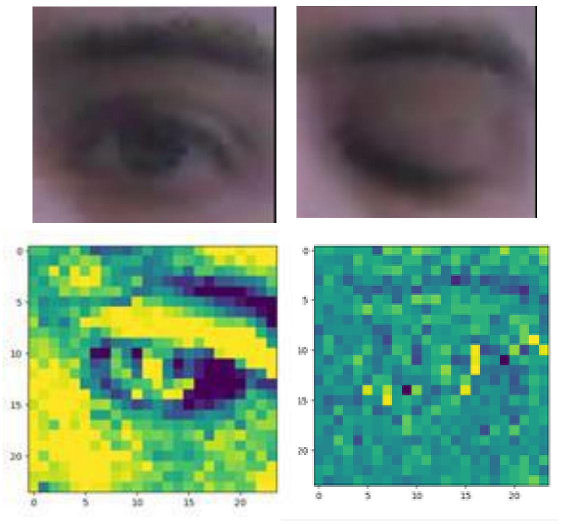


Figure 3. Difference Between Frames

Above steps should be repeated till a successful number of accepted frames is captured (means both eyes fulfill the requirements), in our case we set it to 20 frames. That was achieved using counter that counts the number of accepted frames and stop recording after the count limit is reached

3.3. Data augmentation:

As the total number of samples is only 120 samples which is not enough to train our model we have to use a data augmentation technique in order to artificially expand the size of our data set by creating modified versions of images in the data set. In this way we can improve the performance and ability of our fit model to generalize what it has learned to new images. In our project we used our own code to augment the videos, where we obtained the frames from the videos, then do rotation for each frame of the video and finally append back the new rotated frames. We applied different frames rotation in different angels, since we need to preserve the position of eyes in frames we rotated the videos by small angels. In our case we chose -5,-3,3,5. The rotation will likely rotate pixels out of the image frame and leave areas of the frame with no pixel data that must be filled in from the nearest neighbor. In this way we increased our data set to 600 samples.

Our final dataset will be in the following shape (N, W, H, C, L)

- N: Number of frames per sample - In our case 20.
- W: Width of the image frames. In our case 24.
- H: Height of the image frames. In our case 24
- C: Number of channels in image. Since we are using gray scale, so it is only 1 channel.
- L: Label or motion classification.

The model input is 2 sequence of frames (one for each eye), each eye has 20 frames and the size of each frame is 24* 24 and we use gray scale so for each eye we have a dataset shape of (20*24*24*1).

4.The model:

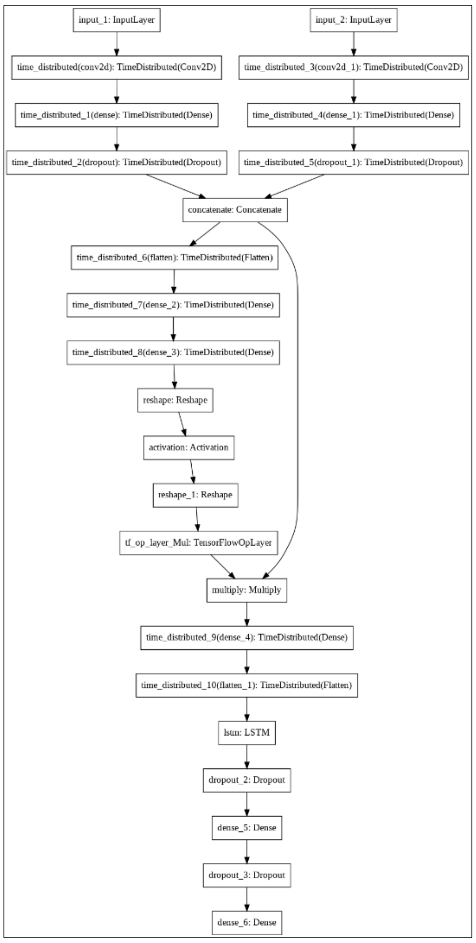

input_1: InputLayer
input_2: InputLayer
time_distributed(conv2d): TimeDistributed(Conv2D)
time_distributed_3(conv2d_1): TimeDistributed(Conv2D)
time_distributed_1(dense): TimeDistributed(Dense)
time_distributed_4(dense_1): TimeDistributed(Dense)
time_distributed_2(dropout): TimeDistributed(Dropout)
time_distributed_5(dropout_1): TimeDistributed(Dropout)
concatenate: Concatenate
time_distributed_6(flatten): TimeDistributed(Flatten)
time_distributed_7(dense_2): TimeDistributed(Dense)
time_distributed_8(dense_3): TimeDistributed(Dense)
reshape: Reshape
activation: Activation
reshape_1: Reshape
tf_op_layer_Mul: TensorFlowOpLayer
multiply: Multiply
time_distributed_9(dense_4): TimeDistributed(Dense)
time_distributed_10(flatten_1): TimeDistributed(Flatten)
lstm: LSTM
dropout_2: Dropout
dense_5: Dense
dropout_3: Dropout
dense_6: Dense

Figure 4. The model

We can divide our model into three layers. The first layer is the CNN layer, Followed by the Attention layer and finally the RNN layer or the LSTM layer. Figure 4 shows the complete model design.

4.1. The CNN Layer:

The main aim of that layer is to extract visual features from the images of both eyes at each time step, combine these features to predict the motion executed with the eyes.

Commonly convolutional layers are used to extract the visual features, as they are good at processing images to squeeze out visual features in order to guess the object of that images.

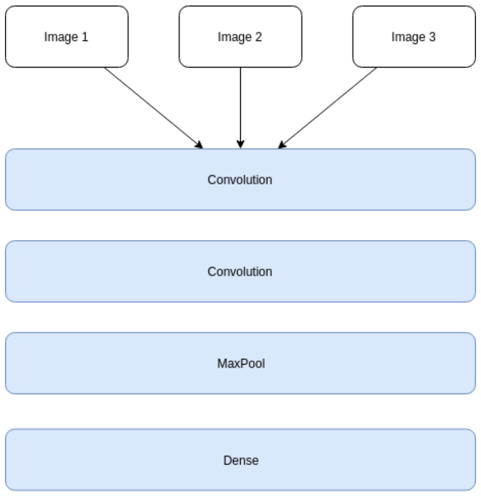


Figure 5. Common CNN without time distribution.

However, in our case we need to present several images that are chronologically ordered to detect movements (eye movements), where we have to extract the visual features in each frame in each time step. That can not be achieved using the commonly used convolutional layers

For this purpose, we used the time distributed layers, which allows us to apply convolutional layers along the time dimension as seen in Figure 6.

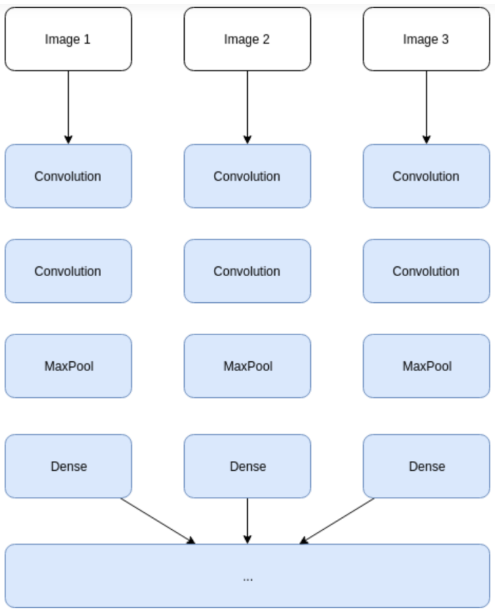


Figure 6. Time Distributed Convolutional Layers.

Now we can apply time distributed convolutional layers to each frame of the 20 video frames. Applying the same for each eye independently as we needed to extract the feature of each eye separately, and then merge the features obtained per frame for both eyes together through another time distributed concatenated layer.
The output of the time distributed concatenated layer will be 20 frames each frame contains the features of both eyes together in that single frame.

The resulting convolutional neural network (CNN) will learn to extract relevant knowledge from pairs of eyes on each frame.

Figure 7 shows the detailed model diagram for the CNN layer of our model and using the time distributed CNN

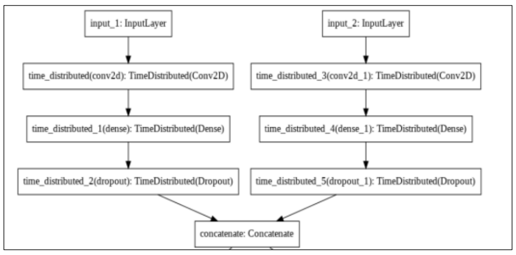

input_1: InputLayer
input_2: InputLayer
time_distributed(conv2d): TimeDistributed(Conv2D)
time_distributed_3(conv2d_1): TimeDistributed(Conv2D)
time_distributed_1(dense): TimeDistributed(Dense)
time_distributed_4(dense_1): TimeDistributed(Dense)
time_distributed_2(dropout): TimeDistributed(Dropout)
time_distributed_5(dropout_1): TimeDistributed(Dropout)
concatenate: Concatenate

Figure 7. Time Distributed layer in our model

We used the tanh activation function inside the time distribution layers, since as mentioned earlier we computed the difference of consecutive frames which could result in pixels in negative values which needs to be preserved.
The range of the tanh function output is from -1 to 1, while it is for the sigmoid function from 0 to 1 as it's shown in Figure 8. The advantage is that the negative inputs will be mapped strongly negative and the zero inputs will be mapped near zero in the tanh graph.
The tanh activation function is:

$$tanh(x) = 2\cdot\sigma(2x)-1$$

Where $\sigma(x)$, the sigmoid function, which is defined as:

$$\sigma(x) = \frac{e^x}{e^x+1}$$

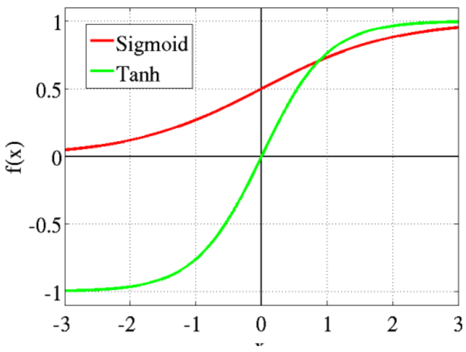
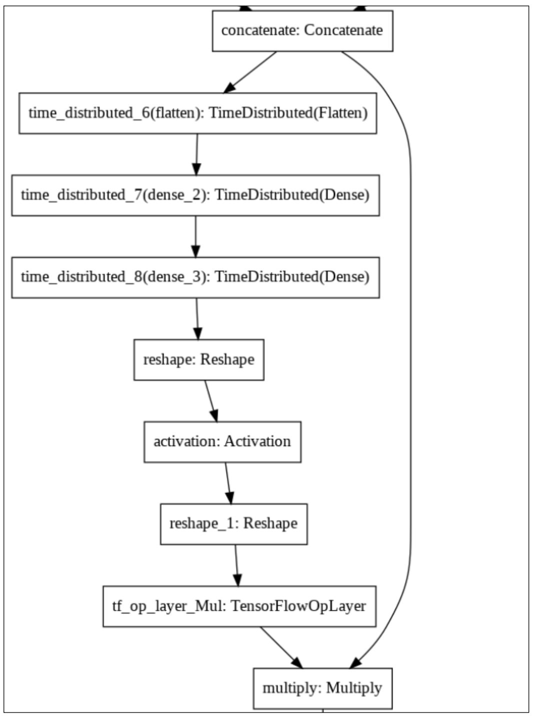


Figure 8. Sigmoid and Tanh Function

4.2. The Attention Model:

Since the motion could be in any of the frame sequence (20 frames) so we need to give different weights to different frames where frames that include the motion should have high weights and other frames should have less weights or ignored. Here we used the idea of attention model but with some modification.
As seen from Figure 9, a flatten layer is added to get all the feature of each frame. In this way we have an output of features of 20 frames (each frame has its own features). After that we apply a dense layer, its input is the features of each frame and it will output one weight for each one of them.


concatenate: Concatenate
time_distributed_6(flatten): TimeDistributed(Flatten)
time_distributed_7(dense_2): TimeDistributed(Dense)
time_distributed_8(dense_3): TimeDistributed(Dense)
reshape: Reshape
activation: Activation
reshape_1: Reshape
tf_op_layer_Mul: TensorFlowOpLayer
multiply: Multiply
Figure 9. Attention layer

The output of last step is of shape (20,1), which if feed into SoftMax will give a vector of all ones which is not relevant. To fix that we must reshape the output to have a shape of (1,20) instead of (20,1). So, we introduced the Reshape layer to achieve that task. Now we can apply the SoftMax whose mission is giving weights to each frame. Where after training it should be able to give high weights to frames with higher accuracy and low weights to other frames. After that we reshape the output again back to its old shape (20,1).
The next step is to multiply those weights with the original frames. We did that by creating a new matrix of ones with the same size of the original frames (24,24,256) where 24 is the frame height and width and 256 is the frame depth after applying the convolutional layer in the previous step. And multiplying this matrix with the output of the SoftMax. Now we

have 20 frames each frame has same shape or size of the original frames, but all values of each of the 20 frame is the same. Now we can scaler multiply this matrix with the original frames. In this way we have given weights to different frames as same as the concept of attention layer.

4.3. The LSTM layer:

The final step in our model is to check the eye movements to detect which motion does it correspond to (N: no blinks, B: both eyes blinks, R: right eye blink or L: left eye blink) so we need to detect the relation between the different 20 frames and LSTM (Long Short Term Memory) is a very sufficient neural computation for that. LSTM is a Recurrent Neural Network (RNN) architecture, which can get several chronological inputs to find what is very useful to predict. The LSTM updates its state using both the extracted features at the current time step and its own previous state.

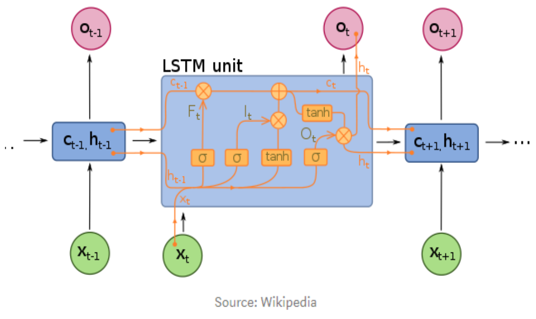


Figure 10. LSTM Unit

There are commonly 2 possibilities:

- make convolution or other neural computation before LSTM.
- make the same kind of work after LSTM.

As we need to check an object in motion, so we need to search the object before detecting the movement. So, here we need to make convolutions before LSTM and then we need to use LSTM layers to treat the data in time range.

Finally, when we have processed the whole sequence of images, the state of the LSTM is then fed to a *SoftMax classifier* to predict the probability of each motion.

Figure 11 shows the LSTM layer which is the third and last layer of our model.

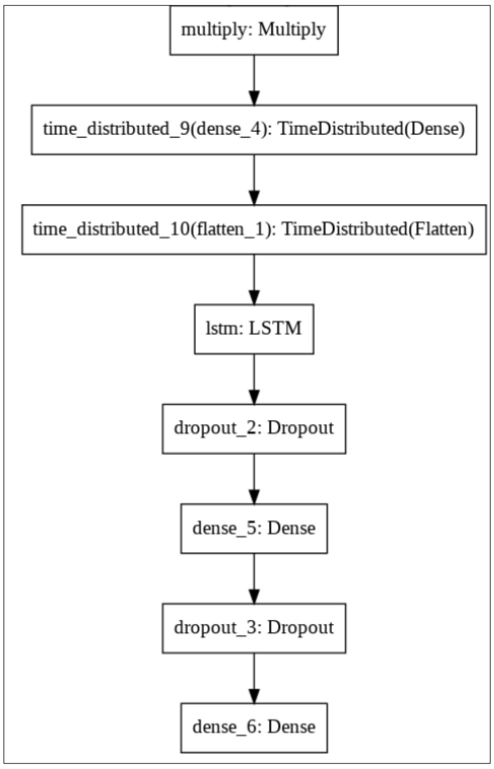

multiply: Multiply
time_distributed_9(dense_4): TimeDistributed(Dense)
time_distributed_10(flatten_1): TimeDistributed(Flatten)
lstm: LSTM
dropout_2: Dropout
dense_5: Dense
dropout_3: Dropout
dense_6: Dense

Figure 11. The LSTM Layer

5.Results :

Our trained model reached 96% of accuracy on the validation set after 7 Epochs of training as seen in Figure 12. This is quite good, considering the fact that our data set was pretty small.

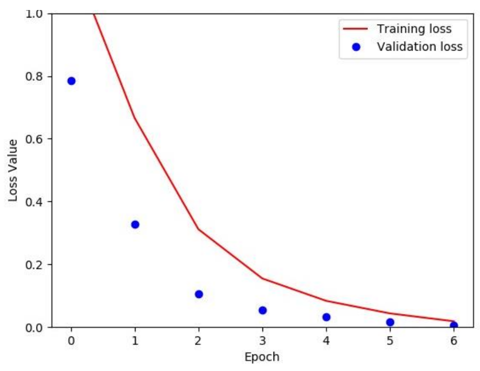


Figure 12. Training and validation loss

Training the model with same data and same number of epochs but not using the attention layer will result in higher loss values as seen below where loss is almost 0.55 compared to around 0.01 with attention layer

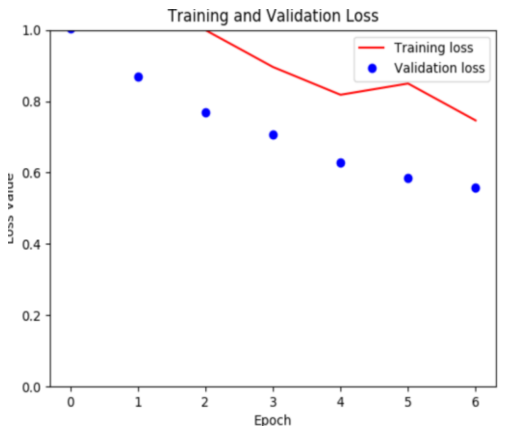


Figure 13.Validation loss without attention Layer

Looking more into how the attention layer we can see that it gives high weights to same frames and less weights to other frames as seen in below figure which shows the weight per each frame of the 20 videos frames, where the sum of the weights should be equal to one.

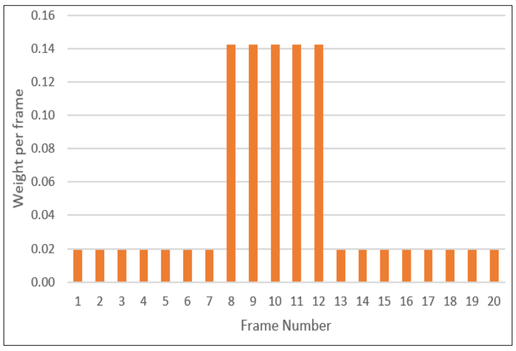
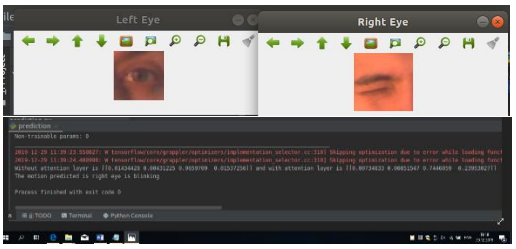


Figure 14. Frame weights

Finally, we write another code than can do live prediction through capturing live frames and to detect the eyes in each frame. The capturing will continue running until capturing 20 accepted frames based on the criteria explained earlier. Once those 20 frames are captured the recording will stop and predict the motion using the trained saved model. Further actions can be implemented based on the predicted motion. Below is an example of our real time predicting.



```
Non-trainable params: 0

2019-12-29 11:39:23.550827: W tensorflow/core/grappler/optimizers/implementation_selector.cc:310] Skipping optimization due to error while loading funct
2019-12-29 11:39:24.400908: W tensorflow/core/grappler/optimizers/implementation_selector.cc:310] Skipping optimization due to error while loading funct
Without attention layer is [[0.01434428 0.00431225 0.9659709  0.0153725 ]] and with attention layer is [[0.00734033 0.00051547 0.7446059  0.2295302 ]]
The motion predicted is right eye is blinking

Process finished with exit code 0
```

Figure 15. Real Time prediction

6. Conclusions:

We have proposed and tested a general design for creating a real-time vision system. We have seen how we use the HAAR cascade to detect the eyes from recorded frames and how image difference could help in motion-related projects. We have also seen how to artificially augment the size of our dataset and to make it suitable for our model. In addition, we have seen the useful combination of CNN'S and RNN's and why to use the time distributed CNN's. We also tried different activation function and noticed the need of using tanh function to preserve the negative values. Finally, we have introduced our attention model and seen its contribution for making the training more effective and easier.

Our model could be improved for further implementations like trigger commands on the computer or any other systems. We can even train this model on other data sets to predict other motions.

7. References

1. Julien Despois, Jan 19.2017, https://medium.com/hackernoon/talk-to-you-computer-with-you-eyes-and-deep-learning-a-i-odyssey-part-2-7d3405ab8be1
2. Patrice Ferlet, Jul 23.2015, https://medium.com/smileinnovation/how-to-work-with-time-

distributed-data-in-a-neural-network-b8b39aa4ce00

3. Hinkelmann, Knut. "Neural Networks, p. 7". University of Applied Sciences Northwestern Switzerland.
4. Sutskever, L.; Vinyals, O.; Le, Q. "Sequence to Sequence Learning with Neural Networks". Electronic Proceedings of Neural Information Processing Systems Conference