**Driver Drowsiness Detection using Convolutional Neural Network**

A **Convolutional Neural Network (CNN)** is a type of Deep Learning neural network architecture commonly used in Computer Vision. Computer vision is a field of Artificial Intelligence that enables a computer to understand and interpret the image or visual data. CNNs are designed to automatically and adaptively learn spatial hierarchies of features from the input images.

The key components of a **CNN** include **convolutional layers**, **pooling layers**, and **fully connected layers**.

* **Convolutional layers** apply convolution operations to the input images, extracting features such as edges, textures, and patterns.
* **Pooling layers** downsample the feature maps obtained from convolutional layers, reducing the spatial dimensions while retaining important information.
* **Fully connected layers**, typically found at the end of the network, perform classification based on the features extracted by earlier layers.



CNNs leverage the concept of **parameter sharing**, meaning that the same set of weights is used for different parts of the input image, enabling the network to learn spatial hierarchies of features efficiently. This property makes CNNs well-suited for tasks such as **object recognition** and **image classification**.

In this approach for drowsiness detection, we have divided the problem into 2 subproblems, i.e.

* Predicting whether person eyes are closed or not
* Predicting whether person is yawning or not

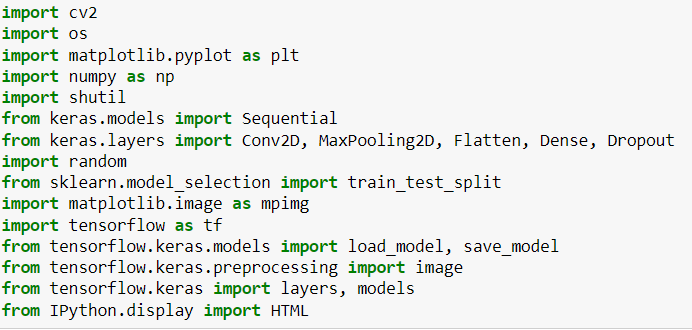
For this, we have trained two separate CNN models, one for the prediction of closed eyes and other for the prediction of yawning.

**Dataset:**

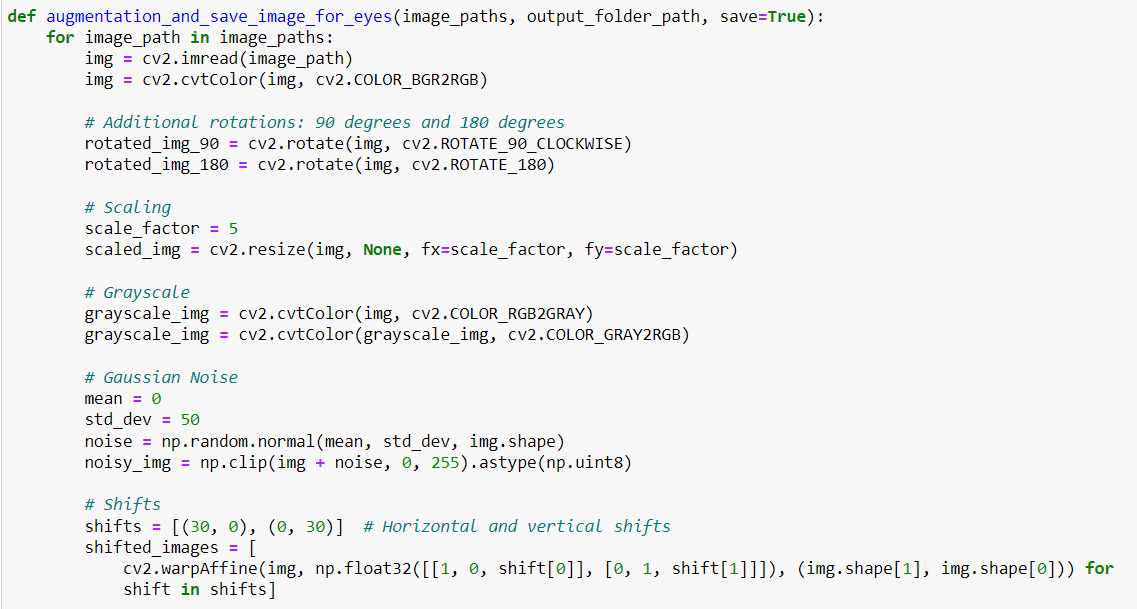
We have used the publicly available dataset that is available on Kaggle. The dataset comprises of 4 different sets of images, i.e. open eyes, closed eyes, yawning and no yawning. Each set comprised of 726 unique images.

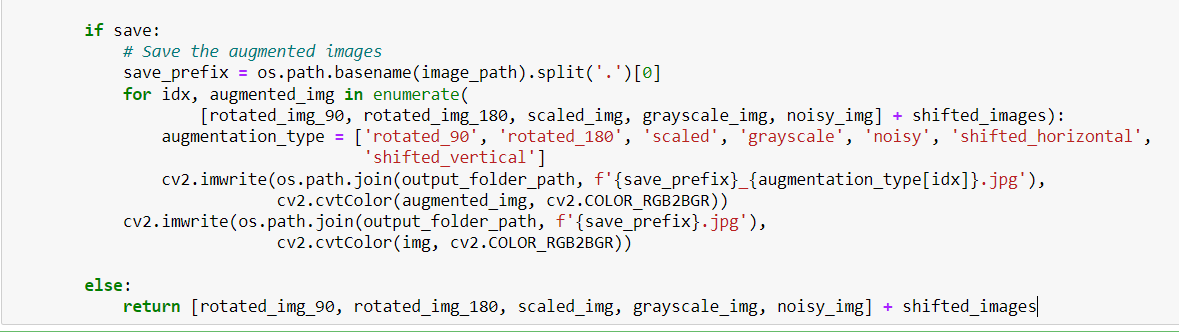


**Code Implementation and Explanation:**

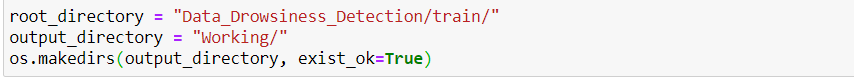
****

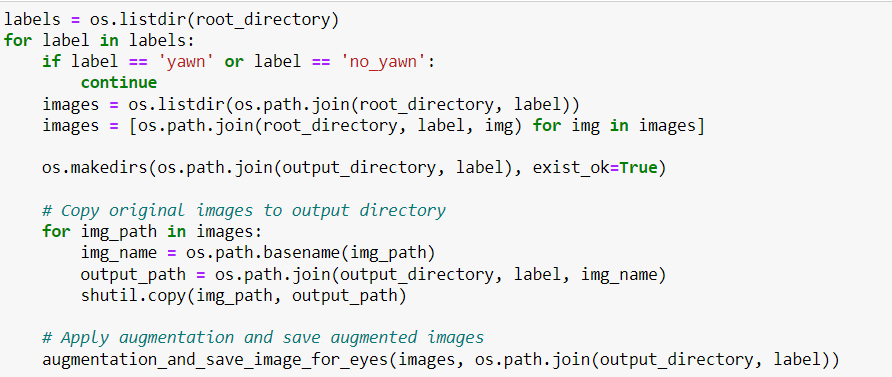
* Various libraries are imported for Code implementation.
* Libraries used: CV2, OS, Matplotlib, NumPy, TensorFlow



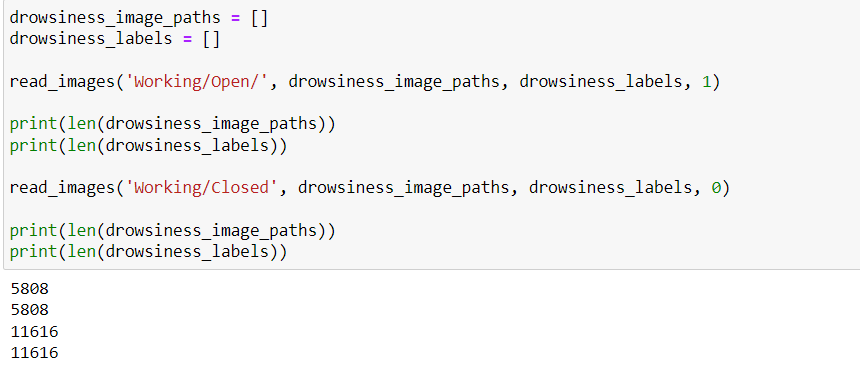
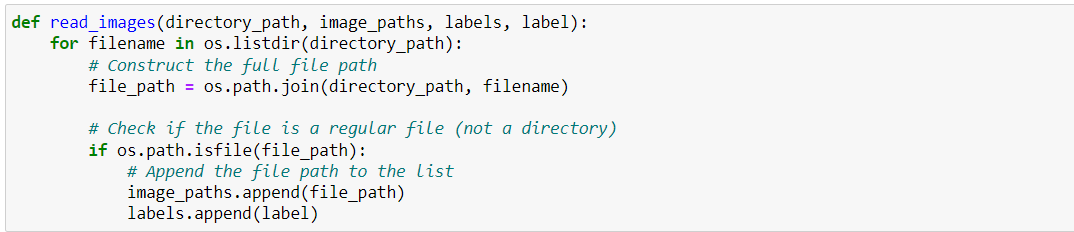


* **Data augmentation** is a technique used in machine learning and deep learning to artificially increase the size of a training dataset by applying various transformations to the existing data.
* For our 1st model for drowsiness detection based on open or closed eyes, we have applied 7 augmentations on each image, i.e. after augmentation our data size increased by 8 times.
* Applied augmentations are:
  + **Image Rotation by 90(degree)**
  + **Image Rotation by 180(degree)**
  + **Scaled image with scaling factor of 5**
  + **Grayscale image**
  + **Image with Gaussian noise**
  + **Horizontal shift by 30 pixels**
  + **Vertical shift by 30 pixels**

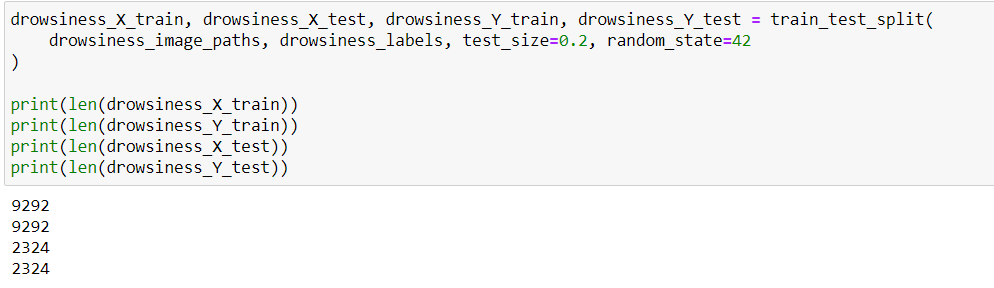




* Here, we ran the above augmentation function on the open and closed eye image dataset and save the corresponding augmented images in a new folder.



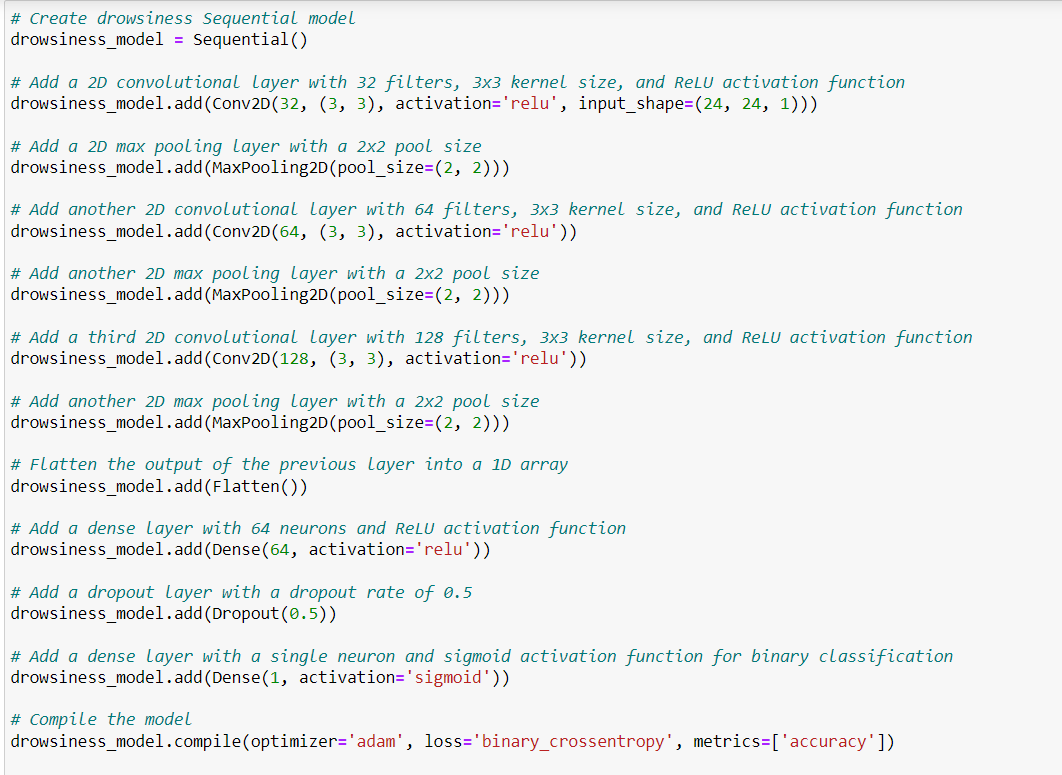
* Here, **read\_images** function is used to define labels for the open and closed eyed. In our case, we have denoted **open eyes** with a **label 1** and **closed eyes** with a **label 0**.
* After that the size of the corresponding folders is verified. And the total number of images came out to be **11616 = 726 \* 8 \* 2**, as it should be.



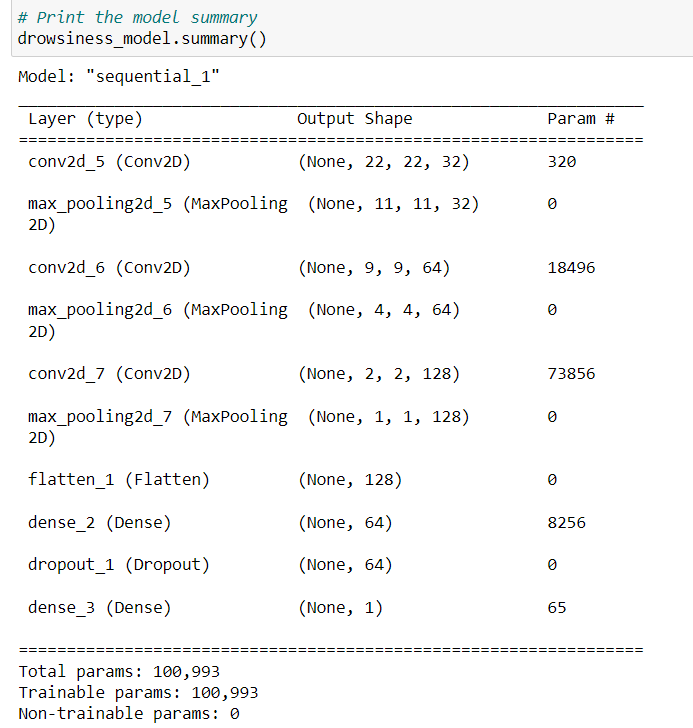
* The dataset is then further divided into train and test components, with the help of **train\_test\_split** function.
* We have used the 80:20 split for training and testing respectively.



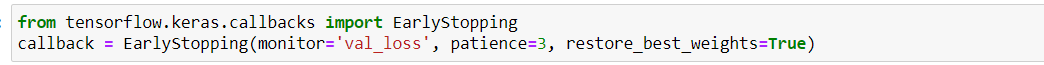
* To train the model, preprocessing of data is necessary. Various preprocessing performed by us are:
  + **Grayscaling :** As the data contains only eye images, it is effective to transform each image into grayscale format, so that our number of parameters get reduced and computation will become fast.
  + **Resizing** each image into (24,24)
  + **Normalizing by dividing each pixel image by 255.**
  + Converting that image array to NumPy array.
* After that the size of respective training and testing dataset is verified.

****

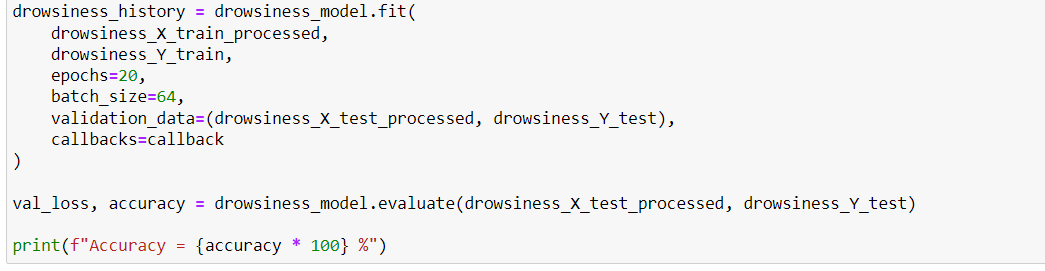
* Here we have defined the overall architecture of our drowsiness model. It is a 10 layered architecture comprising of 3 Convolutional layers, 3 Max Pooling layers, 2 Dense Fully connected layers, 1 Dropout layer and 1 Flattening layer.
* The ‘ReLU’ activation function is used in 3 convolutional layers, while in 1st Dense layer ‘ReLU’ is used and in 2nd  ‘Sigmoid’ activation function is used.
* The Kernels used are of size = (3,3), stride = 1, and number of filters used are 32, 64 and 128 respectively in 3 convolutional layers in order.
* **Dropout layer** randomly selects a subset of neurons to be "dropped out" with a certain probability, typically set between 0.2 and 0.5. When a neuron is dropped out, its output is set to zero, effectively removing it from the network for that iteration. Here, we have used the probability to be dropped out as 0.5.



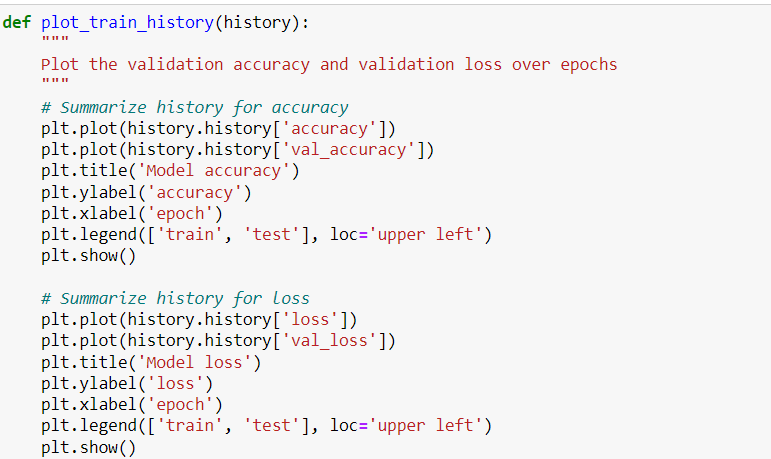
* It is the overall summary of our model and the number of Trainable parameters are ~100,000 as one can see from above summary.

****

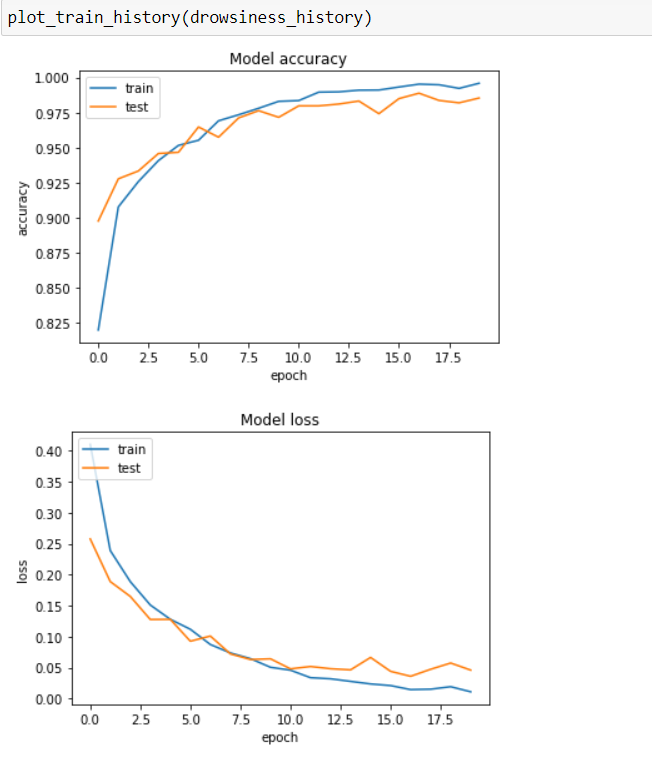
* Early Stopping is used to stop training when a monitored metric has stopped improving.
* Here, we are monitoring the ‘val\_loss’ metric for early stopping.
* **Patience :** Number of epochs with no improvement after which training will be stopped. Defaults to 0. For our case, we have set it to 3.
* **Restore\_best\_weights:** Whether to restore model weights from the epoch with the best value of the monitored quantity. If False, the model weights obtained at the last step of training are used.

****

* The above code implements the training of the drowsiness\_model with number of epochs set to 20 and batch size used is 64.
* Number of training images = 9292, So, number of batches = 9292 / 64 ~ 146. So in each epoch training will be done on 146 batches of data.

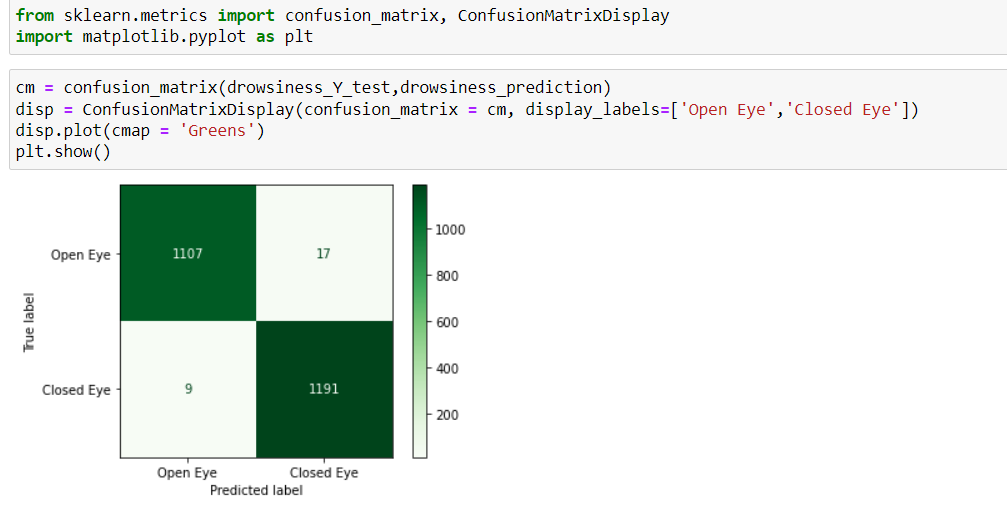
****

* The above code is used to plot the Model loss and Model accuracy with respect to both training and testing data.

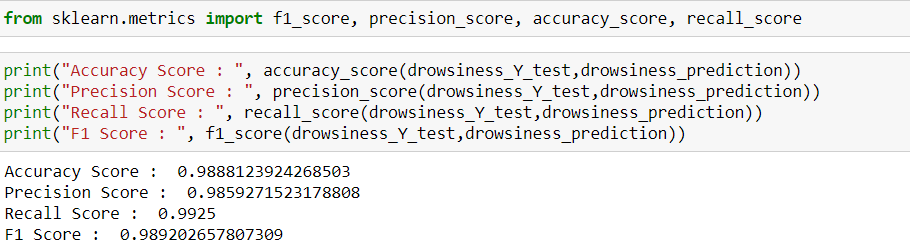
****

* The above plots clearly shows that our model accuracy is constantly improved with the number of epoch while at the same time model loss is constantly reduced with the number of epoch.

**RESULTS:**

****

* Confusion matrix is a table that is often used to evaluate the performance of a classification model.
* Our confusion matrix clearly shows the dominance of our model.



* **Accuracy:** Accuracy measures the proportion of correctly classified instances among all instances. It's calculated as the ratio of the number of correct predictions to the total number of predictions.

****

* **Precision:** Precision measures the proportion of true positive predictions among all positive predictions made by the model.

****

* **Recall:** Recall measures the proportion of true positive predictions among all actual positive instances in the dataset.

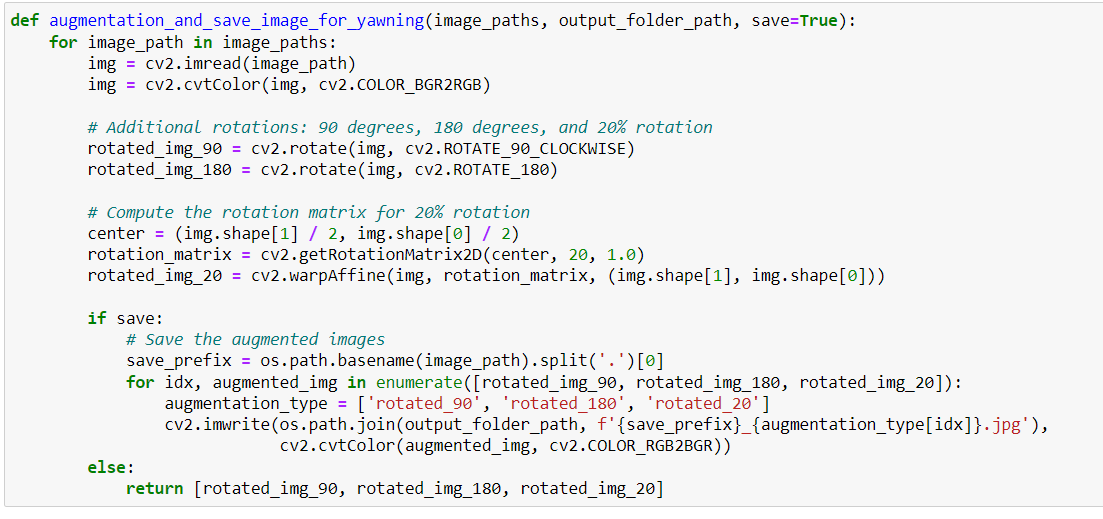
****

* **F1 Score:** The F1 score is the harmonic mean of precision and recall. It provides a single metric that balances both precision and recall**.**

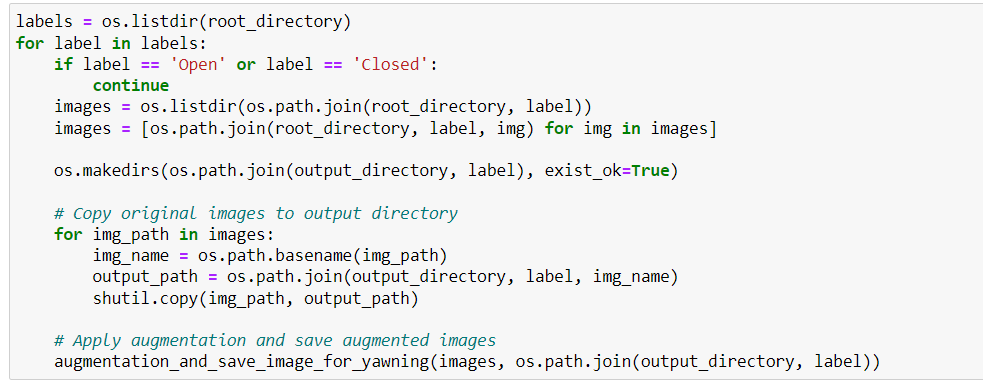
****

* For our drowsiness model all four metrics are above 98.5%.

**CNN Model for Yawning Detection:**

****

* As before, first we are doing augmentation of the data to increase our training data.
* Here, we have done 3 augmentations on each single image as:
  + Image rotation by 90(degree)
  + Image rotation by 180(degree)
  + Image rotation by 20(degree)

****

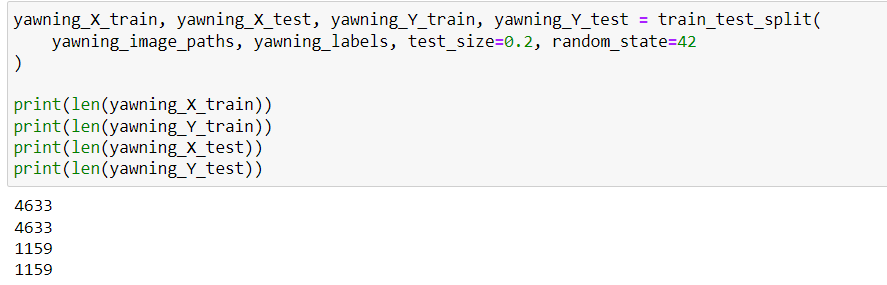
* The above code performs the augmentation and store the result in a new folder.

****

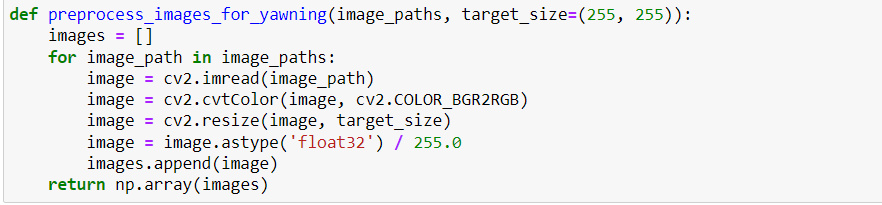
* The 3 types of augmentations that we have performed can be visualised with the help of above code snippet.

****

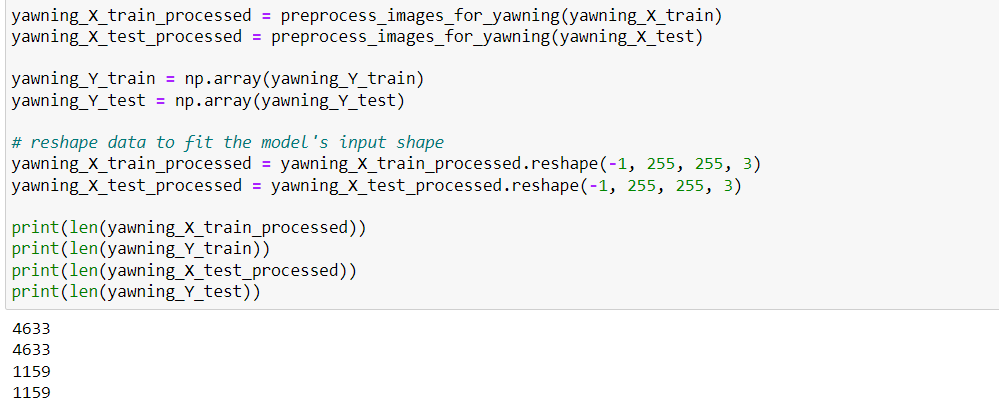
* The number of images after augmentation can be verified with above code snippet.

****

* The data is split into training and testing component in the ratio of 80:20 as before.

****

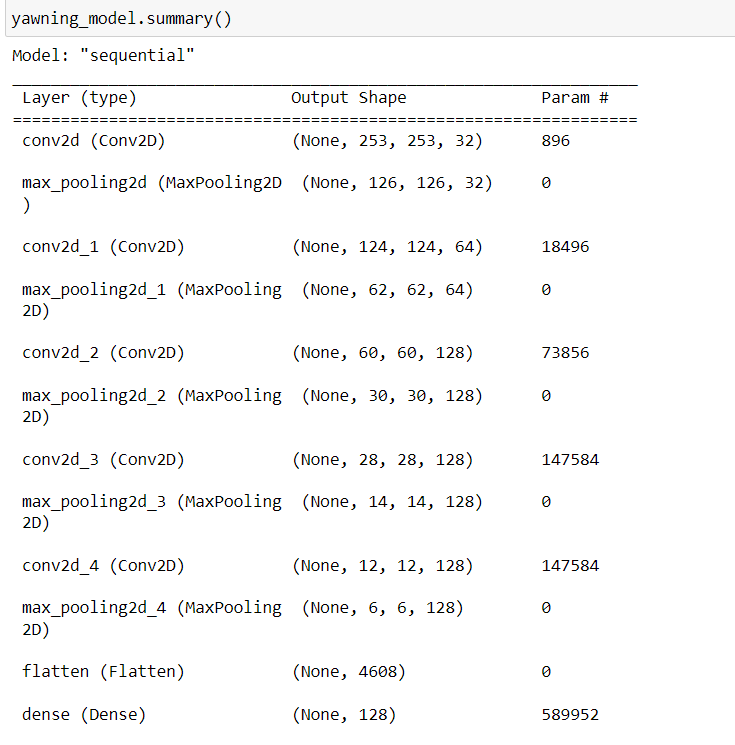
* Preprocessing of images is done to train the model as:
  + Resizing all images to (255,255)
  + Then normalizing each pixel value by dividing it by 255.
  + **Important**: Here we not grayscaling the image as here we have complete face image as our data and the corresponding colour of the pixels do matter.

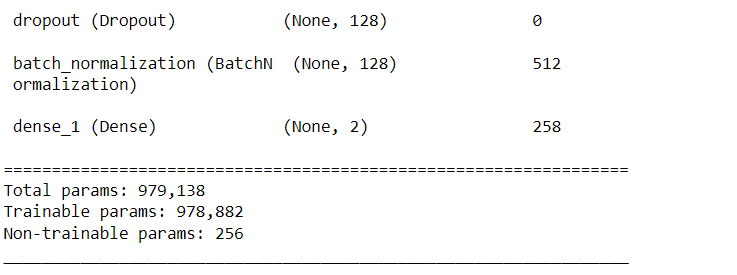
****

* Here the respective preprocessing is performed on the training and testing dataset and is converted into the desired array for training.

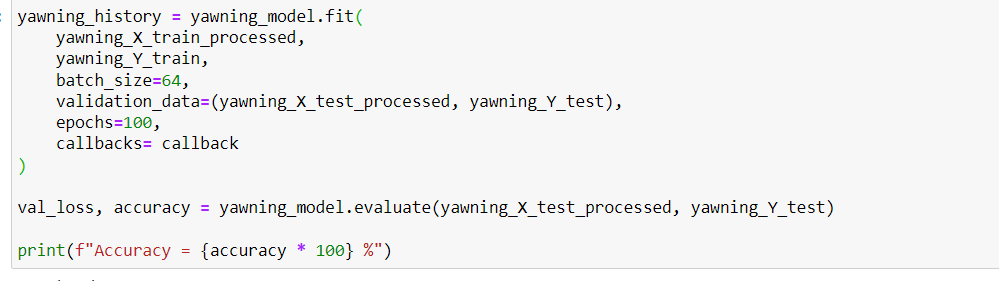
****

* Here we have defined the model framework for our yawning model. It comprises of 15 layers in total with 5 convol utional layers, 5 max pooling layers, 2 dense fully connected layers, 1 Dropout layer, 1 Flattening layer and 1 Batch Normalization layer.
* Kernel Size = (3,3), Stride = 1, Activation Function = ‘Relu’ for all the 5 convolutional layers. The number of filters used are 32, 64, 128, 128, 128 in order for the 5 convolutional layers.
* For Pooling layer the kernel size = (2,2)
* Dropout Probability = 0.3
* First dense layer has 128 outputs with ‘ReLU’ activation.
* Second dense layer has 2 outputs (i.e. Yawn, No Yawn) with ‘softmax’ activation.

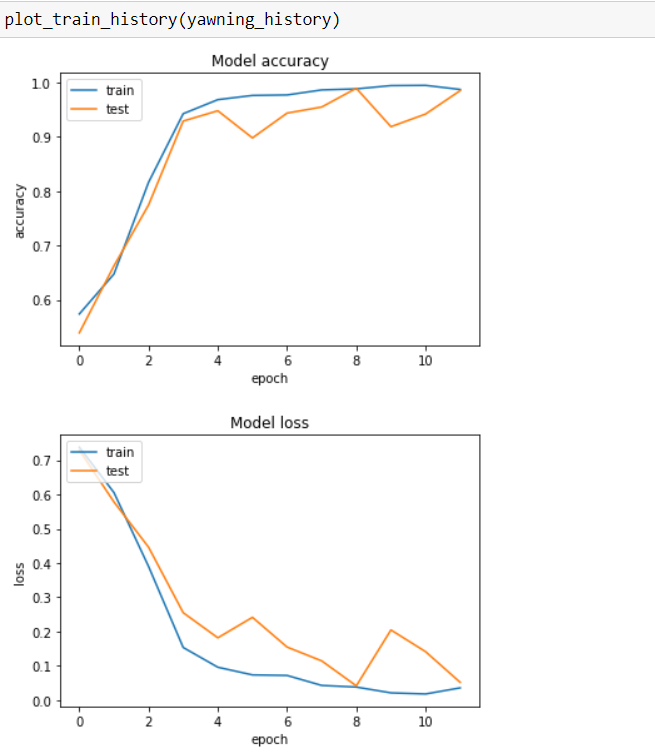
****

****

* The overall model summary can be seen as above with total trainable parameters ~980,000 while 256 non-trainable parameters.

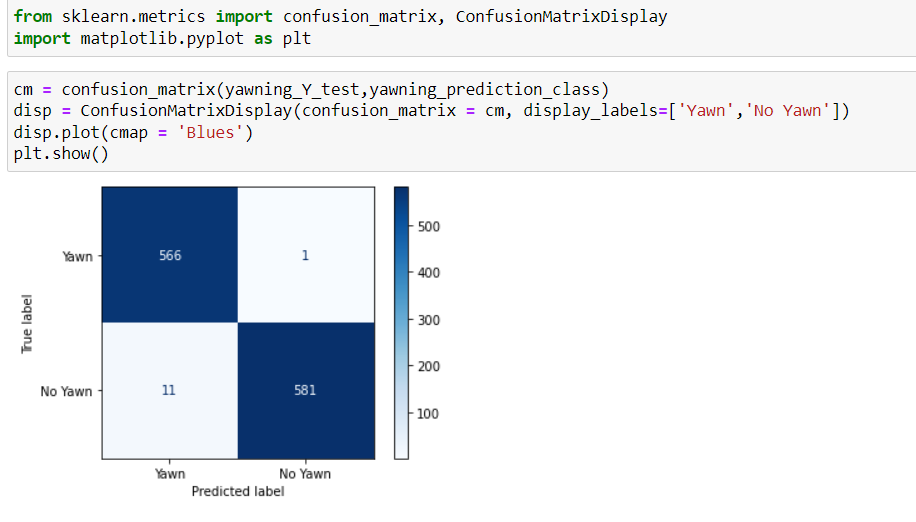
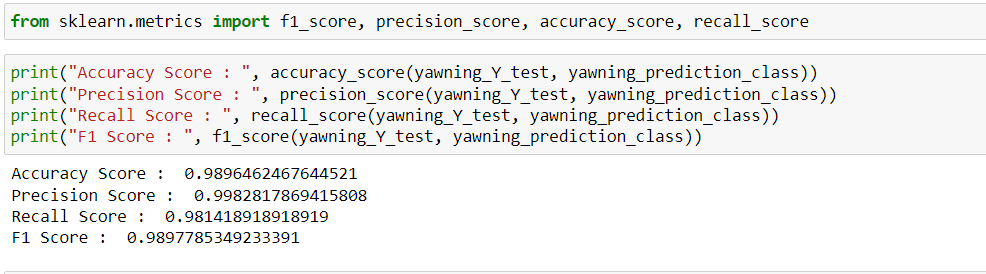
****

* Training the yawning model with batch size set to 64.
* Initially number of epochs set to 100, but after 13 epochs we have achieved our maximum accuracy. So with early stopping we stopped training our model and restored the best weights.

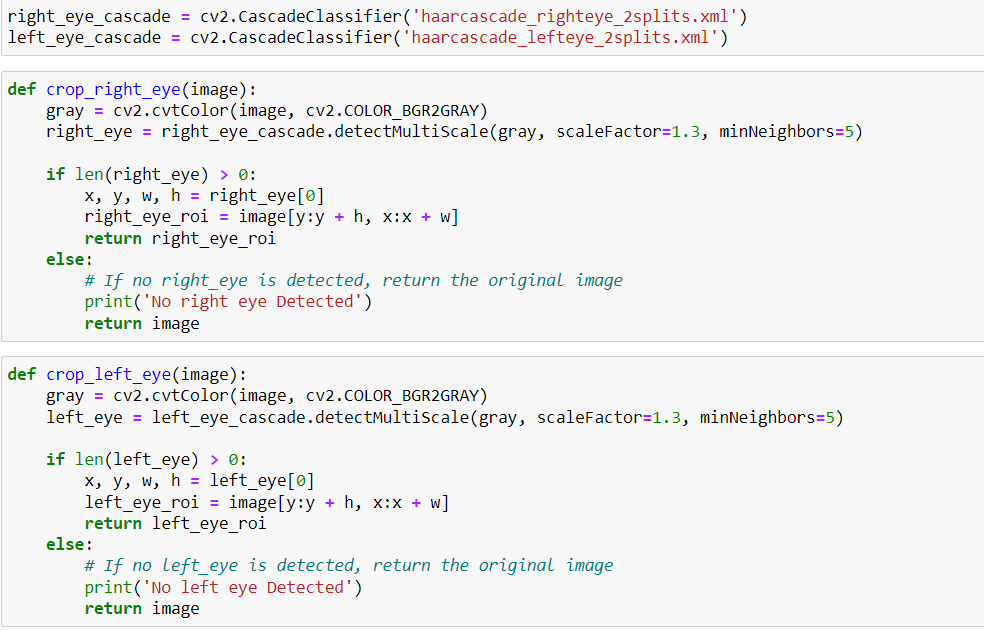
****

* The above graphs represents the Model accuracy and Model loss with number of epochs for both training and testing data.

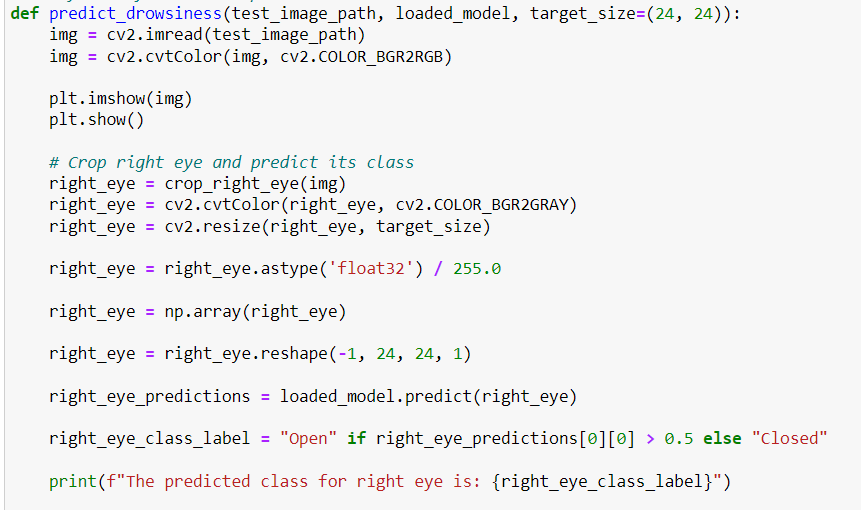
**Metrics**

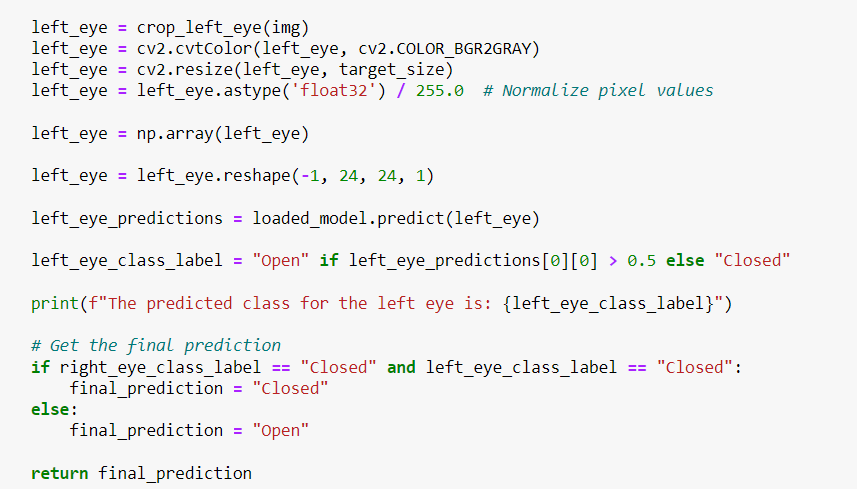
** **

**Testing on our custom images**



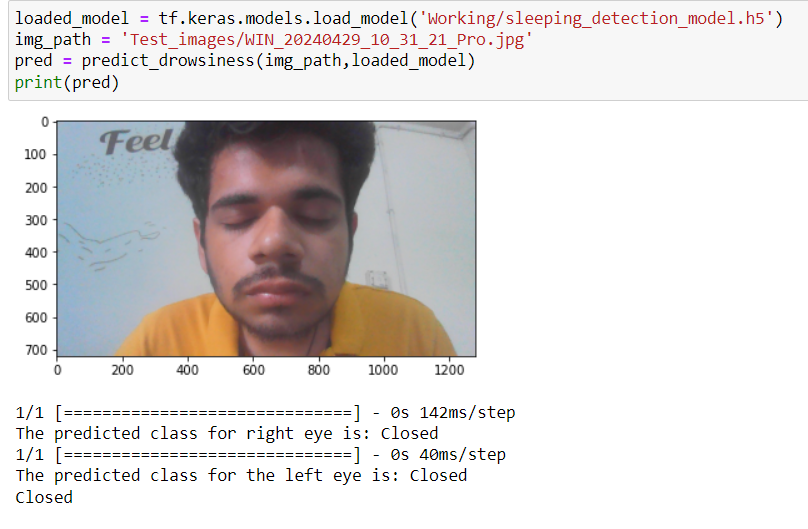
* To detect whether our eyes are closed or open, firstly we have to extract the eye images from the overall face image. We have to necessarily perform this step as we have trained our drowsiness model on eye images only.
* So for eye extraction, we have used the publicly available HaarCascade extensions for left and right eye extractions.
* Then, we defined the crop\_right\_eye and crop\_left\_eye functions to get the coordinates of the desired extracted images.





* Here, the overall logic for closed or open eye prediction is implemented.
* We have used the threshold of 0.5 for classifying into respective classes.
* If prediction for both eyes is “closed” then only the final output of “closed” is given otherwise if either of one is “open” the final output is “open”.





* The above outputs represent the testing of drowsiness model on custom images.



* The above code is for prediction of the yawning as per yawning model.

