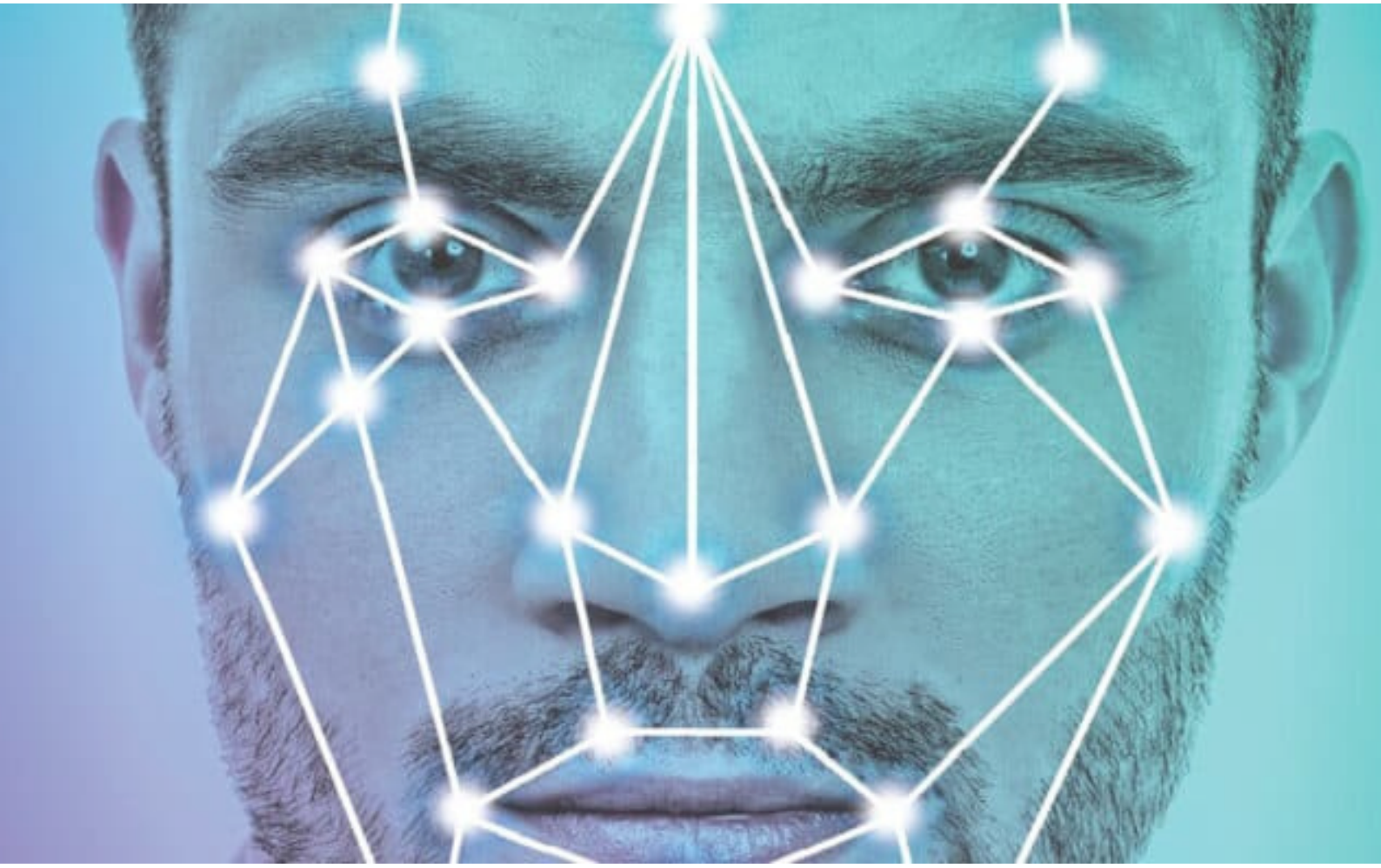
NOVA IMS Information Management School

DROWSINESS DETECTOR

FOR MEDICAL PURPOSES



DATA SCIENCE AND ADVANCED ANALYTICS MASTER

DEEP LEARNING PROJECT

Mariana Camarneiro • r20170744 @novaims.unl.pt
Matilde Pires • r20170783@novaims.unl.pt
Vasco Pestana • r20170803 @novaims.unl.pt

Abstract

This project aims to develop a system that distinguishes closed from open eyes, to further be combined with a real-time calculation of the level of drowsiness the person being analysed through a camera presents, for medical purposes. Being the main goal of this deep learning project to reach the highest possible accuracy on the predictions of a test dataset, several parameters were tested to obtain the best possible model. To accomplish that, a simple model was defined as a baseline, where a different number of layers were tested, together with different activation functions. We obtained unsatisfactory results on our baseline model, presenting a significant amount of overfitting. This overfit was controlled by performing data augmentation along with drop-out and, once again, several parameters were tested to obtain higher performance. Later, callbacks were used to ensure a good trade-off between the number of epochs and performance, leading to the best result obtained with no evidence of overfitting, and little underfitting, and a f1-score of 0.9501 on the training dataset, 0.9750 on the validation dataset and 0.97 on the test dataset, over 47 epochs. Lastly, the model chosen was combined with a real-time connection to the computer's camera to calculate a dynamic drowsiness score. The final product was extremely satisfactory and possible to be tested by any individual with access to the code.

Keywords: Deep Learning, Convolutional Neural Networks, Binary Classification, Supervised Learning, Predictive Modelling, Image Recognition.

Task definition

This project's primary goal is to develop a system able to assess surgeons' level of drowsiness in order to deem them fit or unfit to perform a surgery. As it is known, these professionals often take long shifts and perform several crucial surgeries in a row, neglecting their rest in between and it is imperative to have some level of confidence a priori to whether they are pushing the limits of safety. In fact, this problem is a matter of safety, not only to patients undergoing surgical procedures, but also to doctors because it is another valid tool that can be used as proof of capacity for eventual situations of negligence or malpractice lawsuits.

For this problem our proposed solution is simple enough: build and train an image-classification model that identifies if a person has their eyes opened or closed. With this model created, the idea was to implement a system that records the surgeon before the operation for a defined period of seconds to assess if they are sleepy/drowsy, according to how many times they close their eyes and how much time they remain closed. Therefore, the deep learning model is applied to the captures and registers, for each, if the surgeon's eyes were opened or closed, so a "drowsiness score" can be computed.

For this system to account for variability in blinking behaviour of different people, a first recording should be done under appropriate conditions where the surgeon is not sleepy, so that it can be used as a baseline for comparison. This way, each person has their own standard reference score that is used to truly understand if a score obtained later is unusual for each individual. For demonstration purposes, the system was implemented to emit a sound when the score reaches the defined value of 15.

Approach

After defining the problem described above, we proceeded to collect a dataset with images of people both with their eyes open and closed. This dataset [1] contains 1232 images with people with opened eyes and 1192 with closed eyes, which, although consists of a balanced dataset (similar number for both classes), can be considered initially relatively small numbers. The images were retrieved in their original format, with no pre-processing done.

Once the data was collected and downloaded, and the folders where to store the images were created, we then proceeded to partition the data to divide the existing dataset into three parts:

- Train - where we use 50% of images from the original dataset to train the model to recognize whether an individual has their eyes open or closed.
- Validation - where we use 25% of the images from the dataset that will serve to assess the performance level of our model and control overfitting.
- Test - here we use the remaining 25% of the images and verify that our model has the desired performance level by testing it with images that the model has never seen before.

For this computer vision problem, we will use Convolutional Neural Networks, using Keras, more specifically, we decided to use the LeNet architecture. Due to the characteristics of our input and output data, making it a binary classification problem, a few parameters were defined from the beginning: the loss function used is the binary cross entropy and the activation function for the output layer is the Sigmoid.

Our following step was to decide how to build the network and how to pre-process the data. For that, we decided to use an approach of testing with increasing complexity, starting with the simplest model possible and sequentially applying more layers and other elements.

An important alteration that was implemented was using data augmentation while preparing the data. This is because, as stated previously, the original dataset had a limited number of images and after partitioning the data it became a fact that the model would need more images to train the network successfully. Therefore, this step allowed to introduce a greater variety of data to the training phase, which could be very beneficial to reduce overfitting and improve the model's generalization ability.

To clearly assess improvement in each step of this process, we mainly focused on the validation accuracy, while simultaneously checking for cases of over or underfitting. In addition, we also considered running time, so as to make sure the model was still efficient and did not run for too long unnecessarily. Moreover, for the best model, we also decided to analyse other metrics, namely the precision, recall, f1-score and support, in order to better understand how well the model is performing.

As for the system that computes the drowsiness score, we took a similar system [2] applied to drivers as a basis and adjusted it to our model and desired requirements.

Results

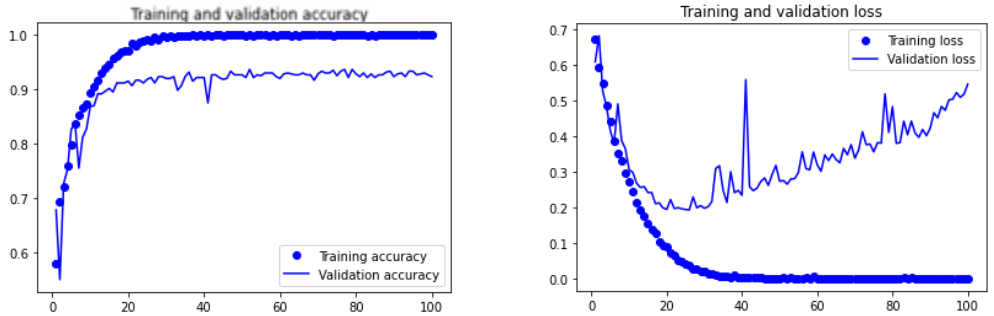
The first trial was a simple model, with neither dropout nor data augmentation. This model was used as a baseline, where several changes were incrementally implemented in order to achieve the best possible performance.

In the several attempts made over this model, the train set presented an extremely high accuracy, but there was a significant amount of overfitting.

The first result obtained was using only one layer, the default batch size of 32 and without specifying the activation function, resulting in a performance of 0.9487 in training and 0.8395 in validation. To improve the results and to lower the running time the batch size was decreased to 20, and a performance of 0.9858 was obtained in training and of 0.855 in validation. This exact model but with ReLU as activation function led to a slightly better performance – 0.9864 in train and 0.8633 in validation - and for this reason the following tests were made using ReLU, which adds non-linearity to the images. The next step was to increase the number of layers to try to achieve a higher performance. Using two layers instead of one increased both the train and the validation accuracy and when using three the difference was even greater, and thus this number of layers was chosen to pursue the tests. The final score of this baseline was 1 for the training set and 0.9183 for validation, a considerably good performance but with a significant amount of overfitting.

BASELINE						
# Epochs	**Batch size**	**# Layers**	**Activation Func.**	**Optimizer**	**Train**	**Validation**
100	32	1	Default	RMSprop	0.9487	0.8395
	20		ReLU		0.9858	0.855
					0.9864	0.8633
				Adam	1	0.8900
		2		RMSprop	0.9959	0.8900
		3			1	0.9233

Table 1: Train and Validation accuracy of Baseline



Graph 1 - Training and validation accuracy and loss curves (best model of this series of attempts)

Having this final version of the baseline, the next phase was to reduce the overfitting. Baring this in mind, data augmentation was performed, and a dropout layer was added. The initial dropout rate was set to 0.5 and afterwards tuned down until the performance was maximized.

With a dropout rate of 0.5, the train accuracy scored 0.9136 and the validation 0.9332, being overfitting no longer a problem. At this point, the batch size parameter was increased to 32 in order to observe the impact. As expected, this bigger batch size led to worse results and presented a slower learning, and thus the analysis was followed using the previously stablished batch size of 20. Using a dropout rate of 0.3 the validation accuracy was slightly better – 0.9417 - and the same was verified with a dropout rate of 0.2 – 0.9533. As the results were similar, but a higher underfitting was noticed with the dropout rate at 0.2, we decided to move forward focusing slightly more on the more parameter value that registered a greater balance between the train and validation performance, which is 0.3.

BASELINE WITH DROPOUT AND DATA AUGMENTATION I						
# Epochs	**Batch size**	**# Layers**	**Activation Func.**	**Dropout Rate**	**Train**	**Validation**
100	20	3	ReLU	0.5	0.9136	0.9332
	32				0.8871	0.9142
	20			0.2	0.9100	0.9533
				0.3	0.9114	0.9417

Table 2: Train and Validation accuracy of Baseline with dropout and data augmentation

However, mild underfitting could be encountered, and the performance had room for improvement. To boost the performance and try to decrease the underfitting, we added a fourth layer, resulting in a train performance of 0.9542 and a validation of 0.9733. This was the best result so far, with not only a high score in both training and validation, but also with little underfitting. To better examine the trade-off between time to train the model and performance, the number of epochs was decreased by half. The accuracy in both the train and validation set reduced significantly – 0.9036 in train and 0.9467 in validation, but it was still noticeable that it would be possible to reach a good performance with less epochs.

BASELINE WITH DROPOUT AND DATA AUGMENTATION II						
# Epochs	**Batch size**	**# Layers**	**Activation Func.**	**Dropout Rate**	**Train**	**Validation**
100	20	4	ReLU	0.3	0.9542	0.9733
50					0.9036	0.9467

Table 3: Train and Validation accuracy of Baseline with dropout, data augmentation and 4 layers

To reduce the running time, the subsequent stage was to perform callbacks in order to stop the training when the validation accuracy stopped improving substantially. Consider the batch size of 20 for all attempts. We used the Early Stopping callback applied to monitor the maximization of the validation accuracy, with several different patience values.

CALLBACKS							
Patience	**# Layers**	**Activation Func.**	**Optimizer**	**Dropout Rate**	**Train**	**Validation**	**# Epochs**
5	4	ReLU	RMSprop	0.3	0.7983	0.8167	30
10					0.9446	0.9750	83
7					0.9113	0.9467	54
8					0.9224	0.9617	76
7				0.2	0.9283	0.9500	58
			Adam		0.9301	0.9750	47

Table 4: Train and Validation accuracy of models using callbacks

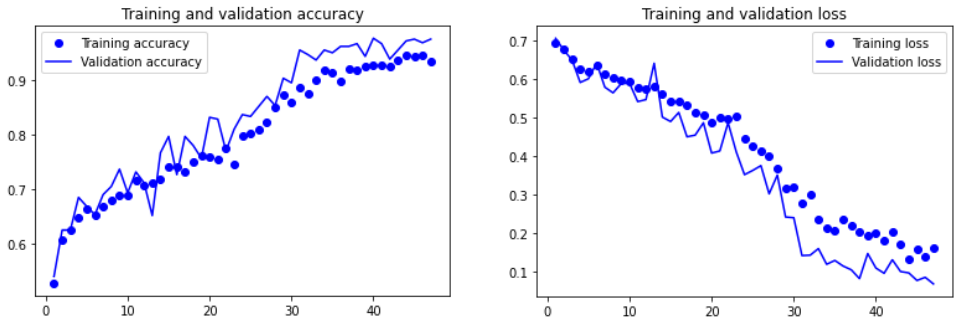
From this set of attempts, we first focused on deciding the patience value to use in the callbacks and it was clear that the smaller the patience value, the less epochs the model would be trained for and the smaller the accuracy obtained (both in the training and validation set). From the experiments made we decided that the increase in performance from using patience 8, instead of 7, did not compensate for the increased running time, since it took more than 20 epochs longer, to reach a similar accuracy score.

Therefore, we moved on with that value to the next tries and decided to decrease the dropout rate to 0.2, to understand if there would be an improvement and, as it can be observed, there was. Due to this, we decided to try again using the Adam optimizer instead of the RMSprop (which in the beginning proved to yield good results as well).

This experiment resulted on the best model in terms of the balance of the trade-off between accuracy and efficiency, since it was able to achieve a score of 0.975 on the validation set, with only 47 epochs of training. For this reason, we considered this to be the best model from the whole process and tested it on the test set created in the beginning, where it scored an accuracy of 97%. Aside from this metric, we also evaluated the model based on a few other metrics from Scikit-learn, to deepen the understanding of its performance.

	precision	recall	f1-score	support
close	0.96	0.97	0.97	297
open	0.97	0.96	0.97	307
accuracy			0.97	604
macro avg	0.97	0.97	0.97	604
weighted avg	0.97	0.97	0.97	604

Figure 1: Classification Report for the best model (based on test set)



Graph 2: Training and validation accuracy and loss curves of the best model

From the figure above it is clear that the model performed rather well on unseen data, for both classes. Focusing on the precision, it measures the percentage of eyes detected as open that were in fact open, so, this is a very important metric for this problem, and it is crucial that it scored high. Since having their eyes closed is the basis for assessing the drowsiness in the system, a model that misclassifies eyes open, when in reality they were closed is dangerous, since it could result in classifying a surgeon as not-drowsy and fit to perform a surgery when that was not the case.

As a final attempt to decrease the slight underfit registered, weight regularization (L2) was implemented on the exact basis of the best model. Nevertheless, although it was able to reach a score of 0.9501 on the training set and of 0.9767, thus reducing the underfit as intended, it did so by training for 66 epochs (nearly 20 more than our best model). Considering the running time of each epoch in each model, this increase represents around double the running time of the whole process. Taking these factors into consideration, it was our decision that this result was less satisfactory than the one obtained previously.

Finally, with the model chosen, it was left to combine the model with a system that would receive a real-time image and compute the drowsiness score. Therefore, we adapted the code to make a real-time connection to the computer's camera, and through the live image being received, calculate a dynamic score. Thus, as soon as the user runs the code, the camera will automatically open and an iterative score with a value of 0 appears on the screen, as well as information on whether the eyes are open or closed at that moment. The score then increases or decreases depending on the eye movement of the person using the system. If it reaches the value of 15, the system will emit a warning sound.

To clearly understand if this system was working properly, we tested it ourselves, by filming our face, closing our eyes and we confirmed that it is functioning according to our expectations (a video can be found in attachment). This test can also be made by any other person, as long as the code is run appropriately.

It is important to note that no time has been set to start and stop receiving the image and calculate the score so as not to condition it. However, the idea would be to make the system start when defined - for example by pressing a key - and stop at the end of the defined time.

Conclusion

For this project we developed an end-to-end deep learning project, more specifically a computer vision one, with the goal of detecting if someone's eyes are open or closed, so as to further detect signs of drowsiness. We consider this project to present extremely satisfactory results given the time and objectives defined. The final accuracy was considerably high - 97% on the test set -, and the system perfectly fulfils its purpose. Still, we believe that it presents a little room for improvement and modifications to be developed in a subsequent phase. To realistically apply this project to the real world, there are limitations to overcome and details that require a deeper analysis.

Firstly, the main idea of this project is to detect drowsiness, to further relate that with the doctors' ability to perform a risky medical operation. However, it would not be possible to train a model with drowsiness as a binary label, since there is no dataset with that information, and it is not feasible to predict if whether a person is sleepy. Given this, the focus was to train the model to distinguish closed eyes from open eyes and test it considering that.

Secondly, the dataset used does not contain many images, leading to the need of using data augmentation. Despite the high accuracy already obtained, it would have been preferable to have a bigger dataset, since we believe an even higher accuracy might have been obtained using a greater number of images as a basis.

In addition, it was challenging to decide a perfect trade-off between the number of epochs to use and the performance obtained, being this the reason behind the use of callbacks.

Moreover, due to time constraints, it was not possible to further develop this model. However, to make the most of this system, it would be advisable to include other relevant external variables such as the number of hours of sleep and the number of surgeries performed, both within a defined time frame. Furthermore, it would be interesting that the image classification model could take into account the mouth, for example, and not only the eyes, as it could be another factor relevant to detect drowsiness.

References

[1] http://parnec.nuaa.edu.cn/_upload/tpl/02/db/731/template731/pages/xtan/ClosedEyeDatabases.html

[2] https://data-flair.training/blogs/python-project-driver-drowsiness-detection-system/

https://keras.io/ja/

https://keras.io/api/preprocessing/image/

https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/Iterator

https://www.gitmemory.com/issue/tensorflow/tensorflow/32764/548759291

https://www.tensorflow.org/api_docs/python/tf/keras/callbacks