Mid-Term Computer Vision Project

Face Detection and Tracking

Juan Gomez

Charbel Francisco Elias

MoSIG - 2020/2021

juan-daniel.gomez-campo@grenoble-inp.org

charbel-francisco.elias@grenoble-inp.org

I. Neural Networks

A. Multi-Layer Perceptron

We start by training an MLP using the FDDB dataset that includes 5,171 faces. Our best MLP model had the following parameters:

- Window size: 24x24 RGB
- 3 Hidden layers

We found that the model performed better when adding the activation function “swish” from tf.keras. We use the binary cross-entropy loss function and Adam optimizer for the following models. This is summary of the MLP model that is used:

```
Model: "sequential"

Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 1728)              0
dense (Dense)                (None, 864)               1493856
dense_1 (Dense)              (None, 432)               373680
dense_2 (Dense)              (None, 216)               93528
dense_3 (Dense)              (None, 1)                 217
=================================================================
Total params: 1,961,281
Trainable params: 1,961,281
Non-trainable params: 0
```

The Image Data Generator from the keras library is used to augment the dataset by rotating the images in the training set with a range of 20.

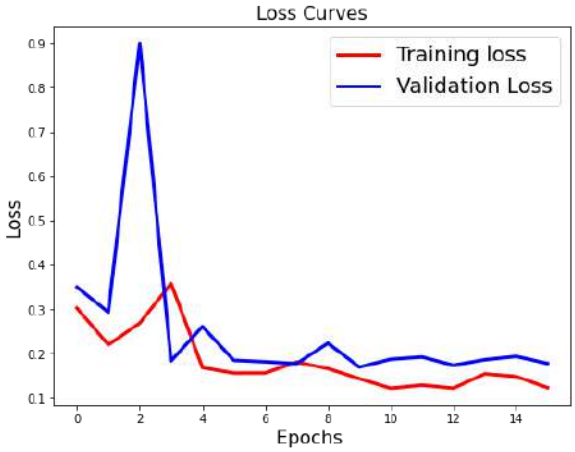
An early stopping callback is added in order to prevent overfitting. It is triggered when the loss did not decrease for 5 epochs.

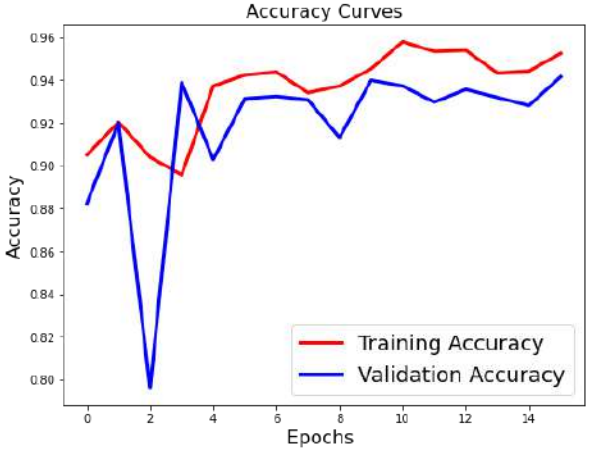
The total training time takes 3m59s on Google Colab on 80% of the dataset.

The following are the results of this MLP:

Loss	Accuracy	Precision	Recall	AUC
0.2005	0.9304	0.9106	0.9546	0.9790

The loss and accuracy curves are as follows:





This is a summary of some of the MLP models that were tested:

	Window Size	Color	Data Augmentation	Trainable Parameters	Training Time	Accuracy	Precision	Recall	AUC	Loss
A	8x8	GRAY	None	3,169	4m35	0.7614	0.7741	0.7382		0.6059
B	8x8	RGB	None	27,937	6m15	0.8754	0.8866	0.8609		0.4046
C	24x24	RGB	None	934,417	5m	0.9111	0.8995	0.9256		0.3101
D	24x24	RGB	Rotation	934,417	5m40s	0.8903	0.8513	0.9459		0.4486
E	16x16	RGB	Rotation	184,897	5m20s	0.8816	0.9440	0.8393		0.4075
F	32x32	RGB	Rotation	1,328,257	6m	0.8725	**0.9671**	0.8132		0.3966
G*	24x24	RGB	Rotation	1,961,281	5m30s	0.9193	0.9133	0.9266	0.9785	**0.1989**
H*+	24x24	RGB	Rotation	1,961,281	**3m59s**	**0.9304**	0.9106	**0.9546**	**0.9790**	0.2005
I*+	24x24	GRAY	Rotation	540,289	6m50s	0.8585	0.8609	0.8551	0.9307	0.4071
J*+	24x24	RGB	Rotation	840,673	5m19s	0.9145	0.9109	0.9188	0.9586	0.3388

* Hidden Layers 1 and 2 have “swish” as an activation function instead of “relu”
\+ Early stopping based on loss is used

We started with a simple 8x8 window size with grayscale data in model A, and as we switched to RGB in model B, we noticed a jump in the results (accuracy, precision, and recall). We can also

visualize the same effect as we switch from RGB color mode in model H to grayscale in model I, as the number of trainable parameters dropped but so did the accuracy, precision, and recall.

When we were training model B, we could see a significant difference between the training accuracy and the validation accuracy, this meant that our model was overfitting; so, we added a layer of Dropout to regularise the model and obtain the model B.

Then we decided to augment the training data in order to better generalize the model, so we added a simple rotation with a range of 20 using ImageDataGenerator from Keras.

As we moved to a larger window size of 24x24 in model C, we noticed an increase in the result metrics.

After having stable results with a window size of 24x24, we tried to decrease the window size to 16x16 in model E and to increase it to 32x32 in model F, but this resulted in a huge drop in the recall, but with a small jump in the precision. So, we decided to continue using the 24x24 window size.

Then, we tried to decrease the number of trainable parameters between models H and J by decreasing the sizes of the hidden layers, but the accuracy and recall dropped and the loss increased.

Now that we had a stable model with decent results, we tried to improve by changing the activation function. We read some articles about the activation function Swish where the highlights were as follows: *"Research by the authors of the papers shows that simply be substituting ReLU units with Swish units improves the best classification accuracy on ImageNet by 0.9% for Mobile NASNet-A and 0.6% for Inception-ResNet-v2." ~ Andre Ye [4].*

So, we decided to put Swish to the test. We used Swish as an activation function for the first and second hidden layers instead of ReLU, and we saw a clear jump in the results. The accuracy in model D (with ReLU) was 0.8903 and it increased to 0.9193 in model G (with Swish). The same goes for the precision that increased from 0.8513 in model D to 0.9133 in model G. However, the recall dropped from 0.9459 in model D to 0.9266 in model G, but the gains in precision and accuracy overweigh the losses in the recall.

We noticed that the difference between the training accuracy and validation accuracy was increasing after a certain number of epochs, so we decided to add an EarlyStopping callback that stops the training if the loss didn't decrease within a specific threshold for 5 epochs. After adding this callback, we were able to get better results.

B. Convolutional Neural Network

After achieving good performance with the previous MLP model, we decided to train a CNN model, which is typically known for better performance in image classification. This CNN model is based on the LeNet CNN model [1].

This is the model summary:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 16, 16, 20)        1520
activation (Activation)      (None, 16, 16, 20)        0
max_pooling2d (MaxPooling2D) (None, 8, 8, 20)          0
conv2d_1 (Conv2D)            (None, 8, 8, 50)          25050
activation_1 (Activation)    (None, 8, 8, 50)          0
max_pooling2d_1 (MaxPooling2 (None, 4, 4, 50)          0
flatten (Flatten)            (None, 800)               0
dense (Dense)                (None, 500)               400500
activation_2 (Activation)    (None, 500)               0
dense_1 (Dense)              (None, 2)                 1002
activation_3 (Activation)    (None, 2)                 0
=================================================================
Total params: 428,072
Trainable params: 428,072
Non-trainable params: 0
_________________________________________________________________
```

We trained the model with the same dataset (FDDB) as our MLP, but we wanted to go ahead and train it on the WIDER dataset as well. As it is a much larger dataset with more variety that can be useful for our final application of face detection.

Model	Dataset	Window Size	Color	Data Augmentation	Trainable Parameters	Training Time	Accuracy	Precision	Recall	AUC	Loss
MLP	FDDB	24x24	RGB	Rotation	1,961,281	3m59s	0.9304	0.9106	0.9546	0.9790	0.2005
CNN	FDDB	16x16	RGB	Rotation, Zoom, Flip	**428,072**	**2m5s**	**0.9565**	**0.9478**	**0.9651**	**0.9867**	**0.0434**
CNN	WIDER	16x16	RGB	Rotation,Zoom, Flip	**428,072**		**0.934**	**0.9436**	**0.9501**	**0.9854**	**0.066**

We can clearly notice the boost in performance in all the computed metrics between the previous MLP model and the CNN models.

This is the expected behavior as for the majority of computer vision problems, CNNs tend to outperform MLPs because it accounts for local connectivity as each filter is panned around the entire image according to a specific size and stride, thus allowing the filter to find and match patterns no matter where the pattern is located in a given image. Another advantage for the CNN over the MLP is that it is less wasteful because the weights are smaller and shared and the layers are sparsely connected partially connected in CNN and not fully connected like the MLP which might be inefficient because of the redundancy when it comes to higher dimensions.

Between the CNNs trained on different datasets, the results are very similar. Surprisingly, the one trained on FDDB has slightly better metrics, even though it is a much more limited dataset.

Still, we think the one trained with WIDER may perform better with a completely different dataset for testing, as it has more variety. For our final application we will be using only the CNN models for the reasons explained above.

II. Face Detection and Tracking

The goal of this part of the project is to use computer visión and machine learning techniques to detect and track faces in videos. The results are evaluated with the AVDIAR data set of videos.
In order to accomplish the task we implemented two main methods for object tracking and detection well known in computer vision applications: Adaptive Background Subtraction and Skin Color Detector. In fact, we used these methods to track the people and extract a region of interest (ROI) out of the frame. And then we used multiple DNN models (either made by us or from OpenCV library) combined with a Sliding Window technique for detection to detect the faces. Lastly, we evaluated the results of each of the combinations and we tabulate them for further explanation.
Before we do that, it is of the most important for us to explain the choice of metrics for the evaluating of the models.

- Precision

$$P = \frac{TP}{TP + FP}$$

The precision metric measures the model's confidence in classifying an example as positive. So if we take this measure alone it may not give us the whole picture on the performance of the model. Because the result may be biased sometimes, as we are going to see in the results.

- Recall

$$R = \frac{TP}{TP + FN}$$

On the other hand, the recall metric measures how many of the positive examples were actually classified as such, in other words it does not only take into account if the positive classification is correct, but it takes into account how many of the positive examples were missed. Again, this alone may not give a proper indication of the performance. We need to find a way to combine the two.

- F1 Score

$$F1 = 2 * \frac{(P * R)}{(P + R)}$$

The F1 Score combines both the precision and the recall of the model, it measures the balance between the two. The lower the F1 score, the more imbalance the model is regarding precision and recall.

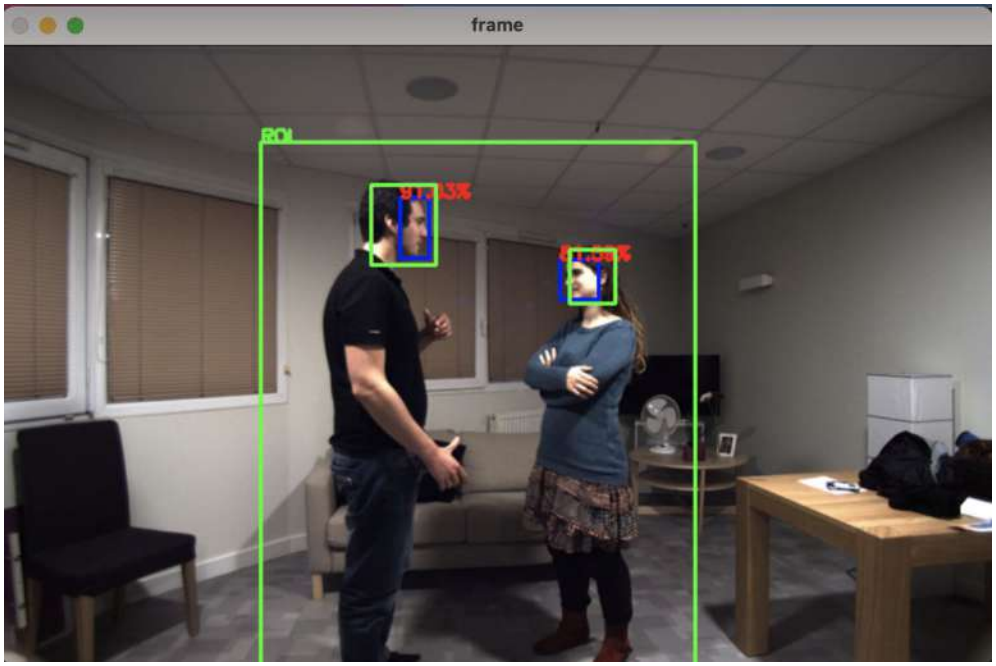
- Computation time (FPS)

Not everything is about metrics performance, it is also important to see the impact on computation for each of the methods used. Is it worth it to use a certain method over the other even though the performance is much worse?

- Important remarks:

Given the imbalance nature of our dataset, we decided to not use the Accuracy Metric, as it takes into account the True Negatives. This is problematic because running a sliding window through a ROI, the network will mostly predict negatives as there is a lot of background and just 1-3 faces at a time.

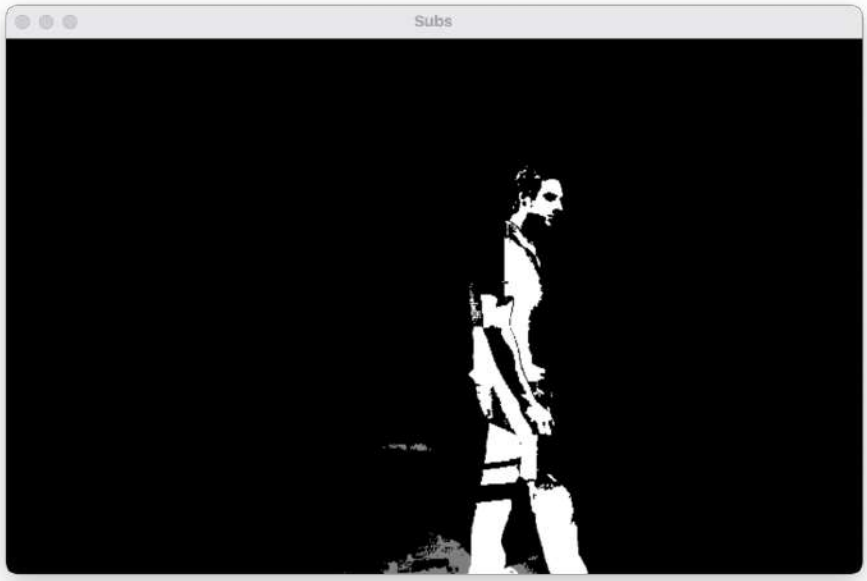
All of the metrics were calculated with an specific IOU (Intersection over union) threshold, normally a value such as 0.5 is picked. But we consider that a detection is correct if the IOU with the ground truth is larger than 0.3. This threshold is honestly hand picked, as we considered that 0.5 is an unforgiving threshold for our models. A demonstration of this can be seen in the following picture:



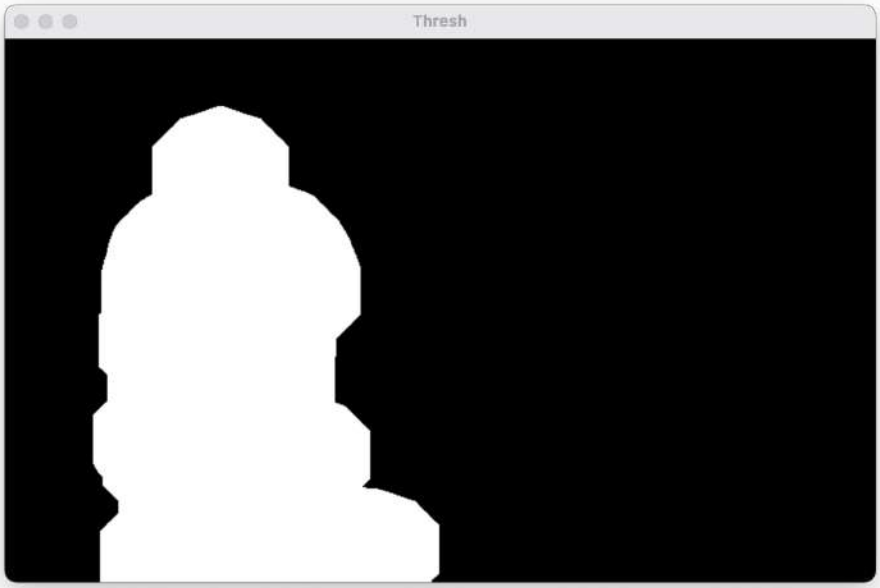
The prediction is in blue and the ground truth is in green. Here we can clearly see that the example is correctly classified, even though it leaves out the ears and the back part of the head. The IOUs respectively are 0,31 and 0,36 for this example. If we were to pick 0.5 as our threshold these predictions would not be accepted. This is the case for most of the predictions, as the bounding boxes for the ground truth include the back of the head when they are sideways talking to each other (very common in multiple-people videos) and sometimes de prediction includes only the face. So we computed the metrics with a IOU threshold of 0.3 for the rest of the project.

1. Adaptive Background Subtraction: Body Tracker

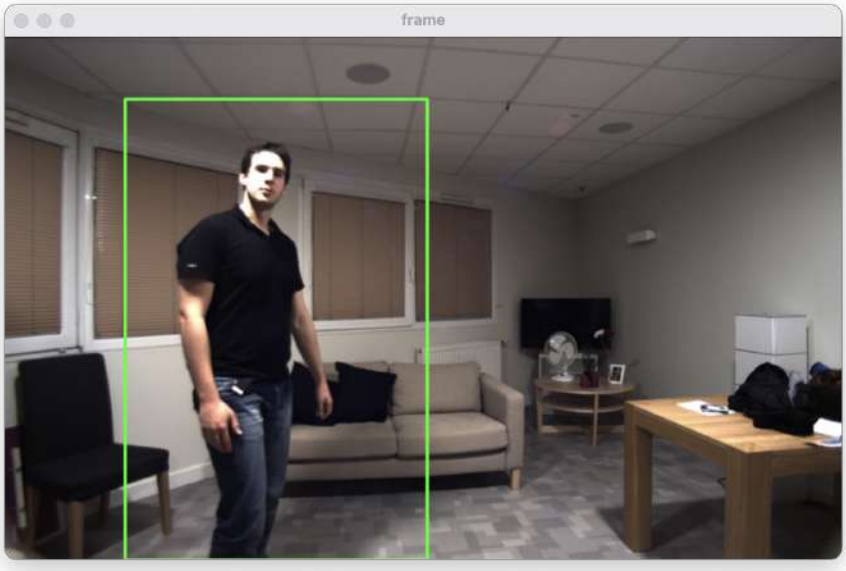
Adaptive Background Subtraction is a common method used in object tracking and is really simple to implement in OpenCV. The method ‘createBackgroundSubtractorMOG2’ from the OpenCv library is used to create the subtraction with respect to the first frame of the image which should be an empty image of the background. There lies our first limitation, this method only works with the videos that start with no people on them, meaning the first frame (frame of reference for the subtraction) is empty. An example of the result of the subtraction is as follows:



As we can see it seems to be working fine, we could go ahead and extract the contours from that image but our goal is to get only one big contour enclosing the whole body. For that, we need to do some further processing. So we adjust the brightness and threshold the image. Then, we apply a series of dilations to get a closing effect on the contours. The result is as follows:

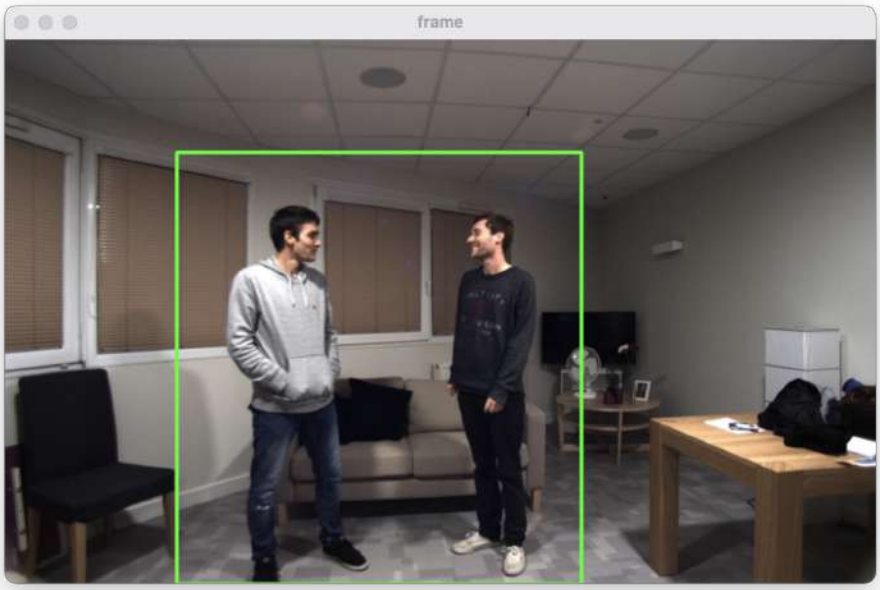


With this, we manage to get a single contour in this mask, so we can draw a bounding box around it to obtain our final result, a bounding box for the body:



We find that most of the time it works correctly (correct being the enclosure of the whole body and getting only one bounding box per person). Although, it is sometimes possible to get more than one bounding box per person or even leaving the face completely out because the processing is not enough sometimes. This is off course a source of error and a cause of increasing computing time. It is important to say that in the case of videos with two people, if they are close together we then get a combined bounding box, which we can easily detect by its

particular larger width (which in theory should be around two times the average width of a single person's bounding box). We account for all this in our code. An example of a combined bounding box:



As stated before, this method only works with a frame of reference (first frame empty), this is why we can only use it with the first seven videos of the AVDIAR dataset. For the rest of them we implement the Skin Detector Method.

At first we wanted to evaluate the results directly with the bounding box of the whole body but the ground truth files for the bodies are missing in the dataset. So we decided to take the bounding boxes for the body and use them as ROIs in order to find a face inside them with a DNN.

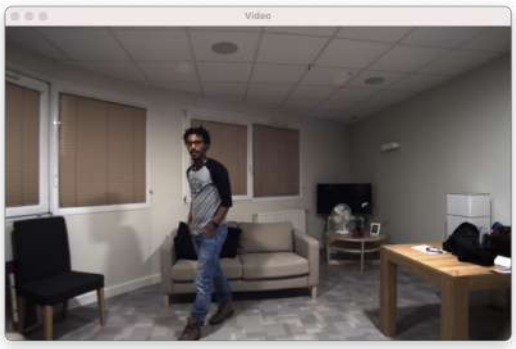
Now that we have isolated a ROI per person, we can feed it to a DNN to get the detection of the face in each frame. We will explain the principle of each combination and then show the results and compare them.

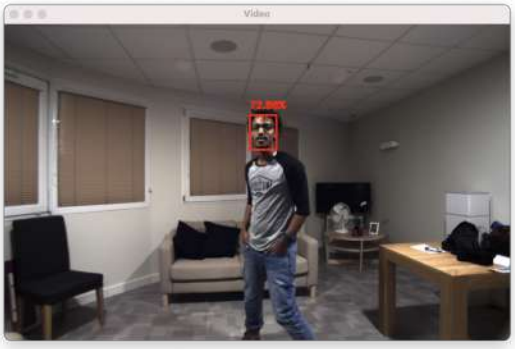
2. OpenCV pre-trained network

Firstly, we used a convolutional neural network that is included in the OpenCV library and is trained specifically for face detection. Somebody with more resources and experience already did it and made it possible for people like us to use it and compare the results with our own.

The network comes from the deep neural network module of OpenCV and was trained using the framework Caffe. This model works very simply, just input an image into it and it will try to detect any faces on that image by outputting their predicted bounding boxes. No sliding window technique from our part is necessary. This Caffe-based model can be found in [2]. For the

purpose of demonstration, we first use the model without cropping the frame by the ROI, feeding it the whole frame for it to find the faces on it. Unsurprisingly, the model by itself struggles a lot as we can see in the result charts as it is difficult to find a little face in a whole image, especially with the different lightning conditions presented in the video. But this can not be seen if we only take a look at the Precision Chart. And there is a simple reason for this, precision only measures if a positive prediction is in fact positive. With this model, it can predict a face in only a few frames of the video, but everytime it predicts one it is correct. The difficulty relies on all of the positive examples it misses (false negatives), as we can see in the Recall Chart. The recall metric shows that there are a lot of examples that are not classified as positive when they should be, and it is even more apparent in the videos 5,6 and 7, which correspond to the ones with two people on them. This can be explained as most of the time they are facing each other (sideways from the point of view of the camera), so it is really hard for the model to detect the faces as they are far away and sideways. On the other hand, the computation time for this tracker is the best out of any of the others as we can see in the FPS Chart, as it does not need to do much pre-processing. We can see examples of the output on the model in the images below:



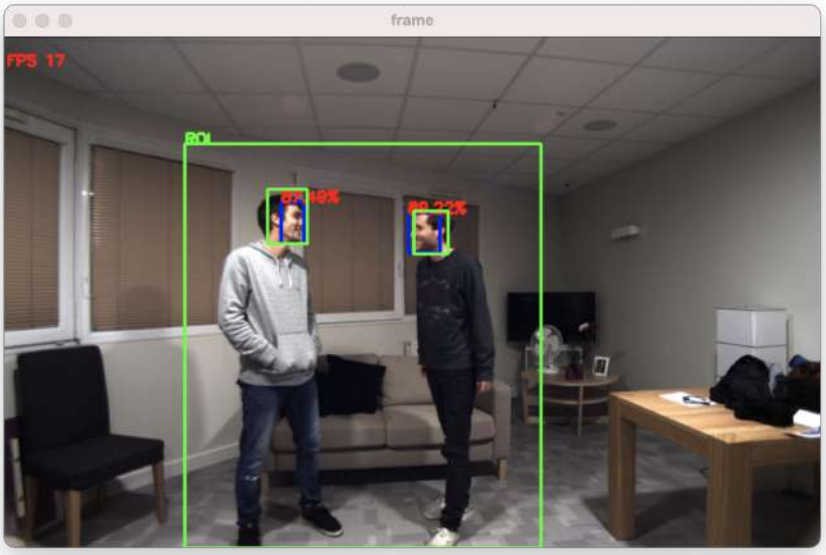


As we can see, the model struggles when the person in the video is far from the camera as the proportion face to background is much smaller than when they are close.

3. ABS + OpenCV pre-trained network

So let's help the network, it is clearly difficult for it to search in the whole frame for each of the faces and it is missing most of them. If we can find a way to increase the face to background proportion it may give better results. So here is where Adaptive Background Subtraction comes into play, each of the ROIs represents one person on the frame (hopefully) so it should make the prediction in that smaller area easier. The only difference with model 2 is that instead of feeding the whole frame to the network we feed it each of the ROIs, on top of that we also cropped the ROI even more as a pre-processing step. As we know that we are looking for faces, we should only look in the top part of the ROI so the model can detect even faster, so we only take the first third of the ROI's height. The results are much better as we can see in the results charts. Regarding the precision, we can see it is almost on par with the model 2**.** The improvement is

more obvious in the Recall Chart, as we can see it increases by a wide margin. It is clear that it continues to struggle in the videos with two people, but still the improvements are considerable. This can be seen as well in the F1 Score Chart, as it gives the highest percentage of all of the models on average. Regarding the computation time, this takes a considerable decrease, as we can see in the FPS Chart, it goes from a somewhat smooth 21 FPS to a 16 FPS. But taking into account the increase in performance it is worth it. Even though it is still far from perfect, it is the tracker to beat so far. An example of the visual results is shown in the following image:



4. ABS + CNN

After the success of the last tracker, we now want to see how our own CNN model compares to the OpenCV one. We tested with three variations of the CNN model, one trained with only FDDB dataset, another with the much larger WIDER dataset, and in the end with one trained with both the WIDER dataset and some of the videos from AVDIAR dataset (not the first 7 of course). It is important to say that we can not just feed a whole frame to the model (or even a complete ROI) and expect a detection, as our models only detect if the input image is a face or not. This is why in this case we must use a Sliding Window technique, so we can actually get decent results.

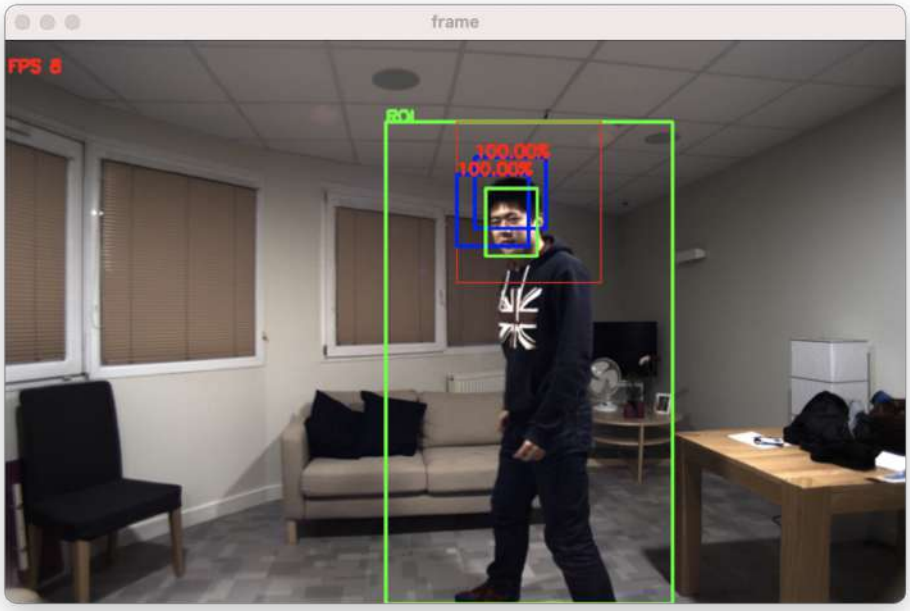
4.1 Sliding Window Technique

The Sliding Window technique is necessary for our model to make a prediction, so at first we tried a simple way to implement it. We took the ROIs given by the ABS method and went through them with different window sizes. At this stage we encountered many problems with this method. In fact the program relied on our CNN models, even though we trained our models with multiple datasets, we found that most of the time it was just detecting skin. This could be

explained by the complexity of the model or the limited data for training. Even though we tried to get more databases it proved to be very hard, as we ended up with only FDDB, WIDER and AVDIAR. So the results of this first try were not good, as most of the time the models would predict as a face any part of the frame that had skin color. It is important to say that the amount of apparent miss-classifications decreased according to the training database of the models. Still, it was overall doing the same, detecting skin. This is why we did not compute the performance of the Sliding Window detector by itself, as it was mostly false positives. Furthermore, in the first seven videos (our test videos) the background had a similar color to the skin so we were also getting a lot of predictions in the background. This can be further demonstrated in section 5.

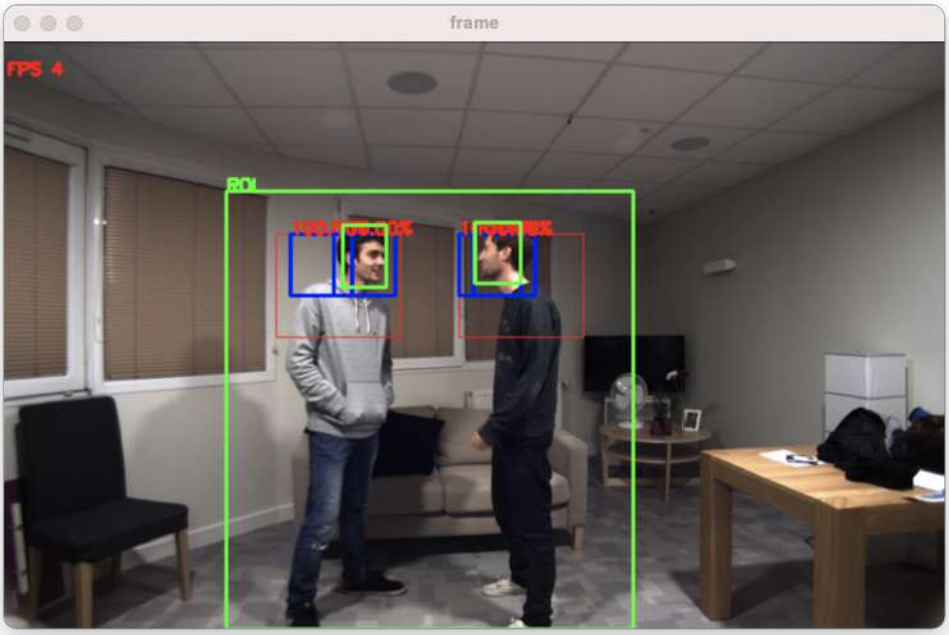
All things considered we needed to find a way to make it work, so despite our limitations we further decreased the ROI to focus more on the possible position of the face. Most of the time (if not always) the face is placed in the top part of the ROI extracted by the ABS method. So we made sure to only take into account the first third of the ROI, which should include the face every time (if the face is indeed in the ROI).

Having done this the results were better, but it could still be improved. We decreased the size of the ROI once more considering the possible position of the face. Instead of taking the whole width, the face would probably be around the center of the ROI's width. This way we could cut out some of the space at the sides of the face. The final results can be observed in the following picture:



As we can see the model now seems to be more precise. The green area marked as ROI is the bounding box generated by the body tracker, the red box is the actual ROI we feed to the sliding window detector according to the processing we described earlier. The blue boxes represent the

prediction of the model along with the corresponding confidence. Finally, the green box represents the ground truth for the sake of visual representation.
Similarly, when there is more than one person present in the video and they are close enough to each other they create a big ROI easily determined by its width size, which is almost double (or triple, etc) the one of a single person. We applied similar techniques for when this happens by subdividing the big ROI into multiple parts, the results are as follows:

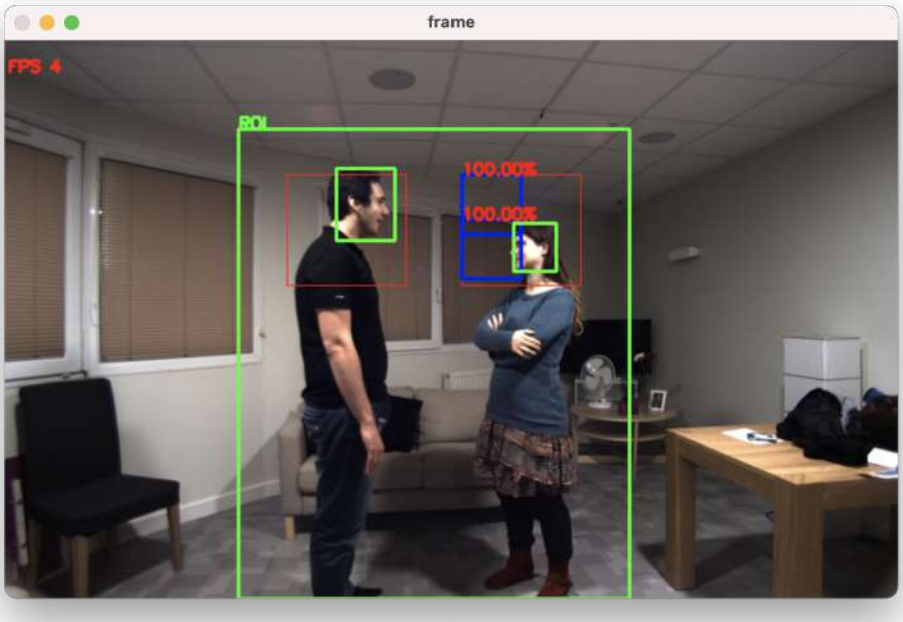


The only thing that remains to be answered is the window size for the sliding window, we chose to pick the sizes dynamically instead of fixed. The windows sizes depend on the size of the red ROI, being half and one fourth of the size of this region. This makes sense with our model and we do get decent results. Even though, it is important to mention that this could be a source of error, as a lot of this processing is determined by us and not by the capacity of our model.
The trackers in 4.2, 4.3 and 4.4 all use the sliding window technique. For now on and until 4.4, the only thing that changes between trackers is the database in which the DNN models have been trained with, so we will analyze the effects of each database in the final performance.

4.2 ABS + CNN_FDDB

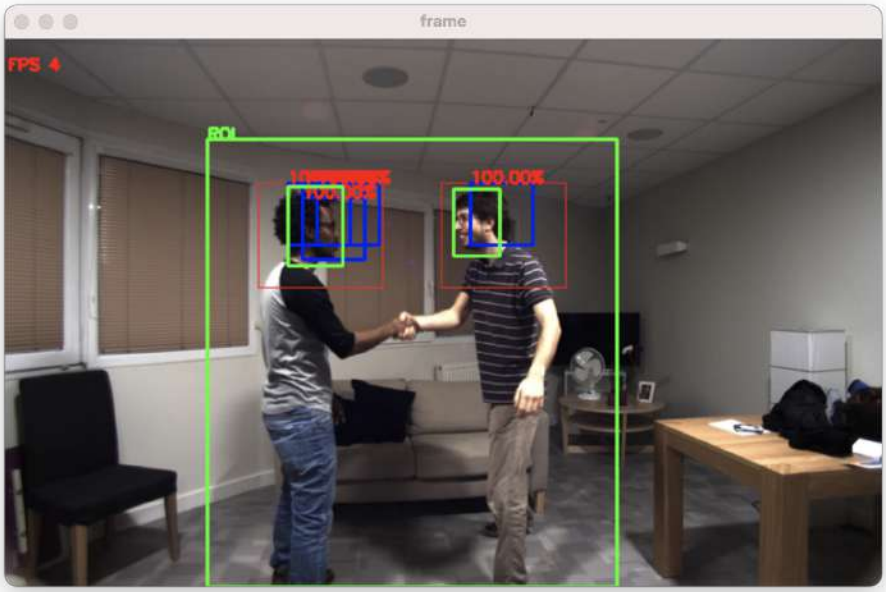
First off we implement the tracker with adaptive background subtraction and our model I.B, as we can see in the results charts, it has the lowest values for the metrics in all of the videos. Precision on average is 40,88%, Recall 43,58% and F1 Score 41,75%. The model is clearly not doing a great job, as all of the metrics are below 50%, meaning that for more than half the time it makes some kind of mistake, either in false positives or false negatives. For what is worth, it is a

balanced result as both Precision and Recall are close in values, as we can see by the F1 Score. This poor performance can be explained by the limitation of the FDDB dataset. It only has 7500 images for training and validation, and most of the faces in the dataset are the front part of the faces, so it struggles a lot when there are faces facing sideways, which is common in the AVDIAR dataset. And example of the classification of this tracker can be seen in the following image:



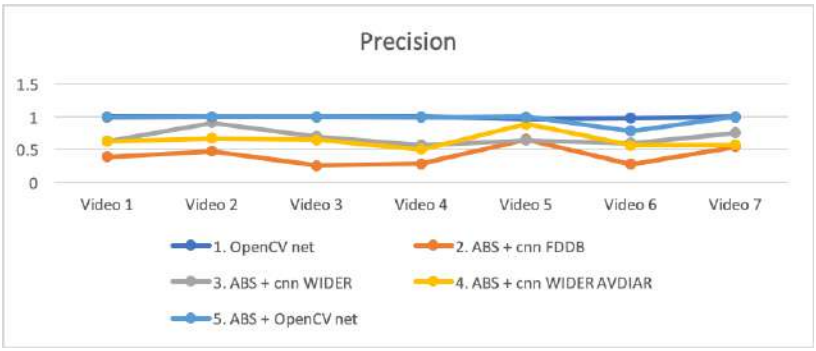
4.2 ABS + CNN_WIDER

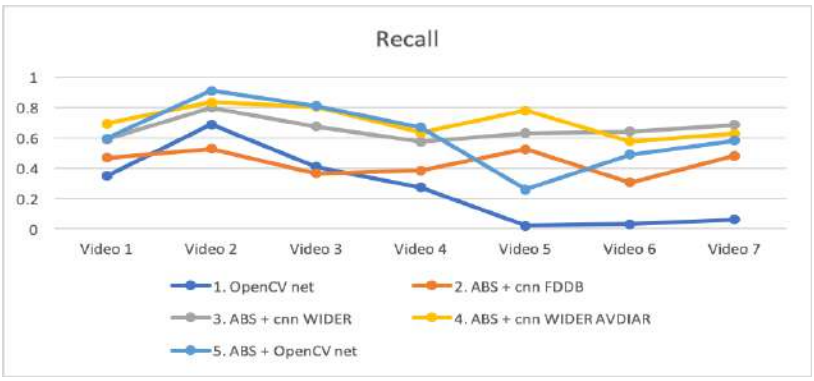
Now we evaluate our tracker with the CNN model trained on the WIDER dataset, we should get better results than with 4.1 as WIDER is a much more diverse dataset. Also, it has up to 32000 images for training and validating. So it covers a lot more ground. As we can see in the results charts, it is fact superior in every metric in comparison with tracker 4.1. Surprisingly, the recall values for the seven videos are more constant than the ones from our best tracker yet (3). And on average the Recall value is 65,51% compared to the 61,53% of tracker 3. This means it is actually giving better results in Recall than any other tracker yet. Despite this, the F1 Scores are consistently lower than tracker 3 because its precision is always larger. These results can be explained by the presence of the Sliding Window technique and a more comprehensive dataset than FDDB. The Sliding Window technique predicts more positives per frame, which makes it more likely to have less false negatives (therefore increasing recall), while also increasing the chance of getting more false positives (therefore decreasing precision). All things considered the best tracker still is ABS + OpenCV net. An example result can be seen in the following image:

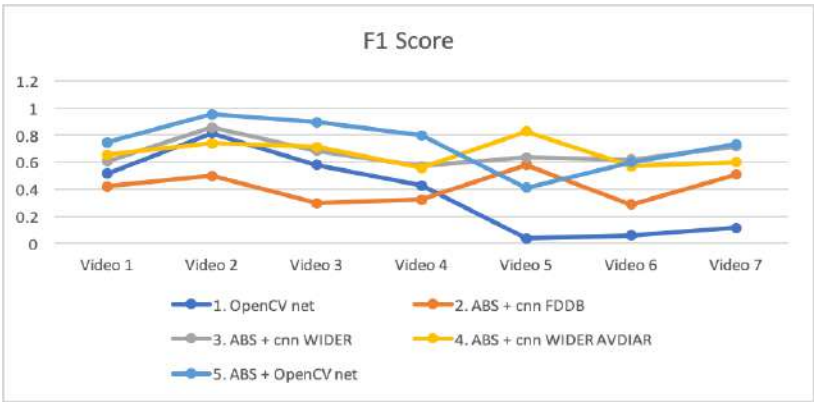


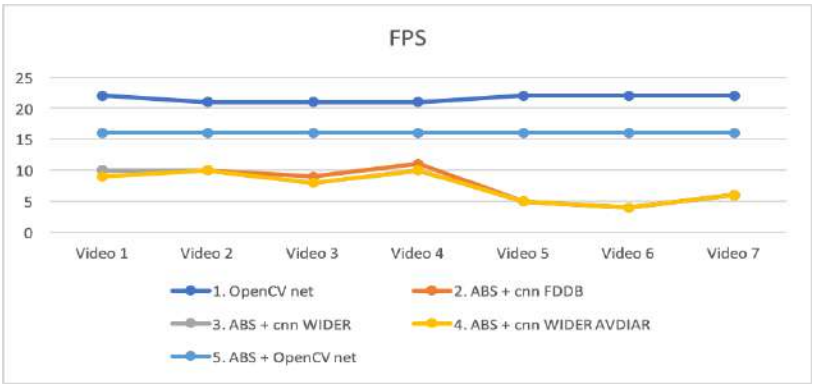
4.3 ABS+ CNN_WIDER_AVDIAR

Now we wanted to try and re-train the last CNN model with the AVDIAR dataset, for this faces and backgrounds from videos different from the ones we were testing with. Better results than in 4.2 are expected as the faces in AVDIAR repeat themselves in other videos, and also there would be a lot more faces sideways in the training dataset. In spite of this, the results were not as good as we expected. The average values for precision, recall and f1 score were 63.67%, 70.59% and 66.67% respectively. This means that compared with the tracker 4.2 is a little less precise. We did notice that it gave more positives results, which resulted in more false positives (resulting in less precision). But the average recall is higher by almost 5%. It can be explained as well by the larger number of overall positive detections. It is important to say that when evaluating the model by itself we found that it was overfitting. This can explain the behavior of the results. Even though the results were not as high as expected, we believe that if we were to re-train a model with the AVDIAR dataset and prevent it from overfitting the results would be superior. But unfortunately, we did not have enough time to re-train the model as it normally takes more than half a day to do it.


Precision
1.5
1
0.5
0
Video 1
Video 2
Video 3
Video 4
Video 5
Video 6
Video 7
1. OpenCV net
2. ABS + cnn FDDB
3. ABS + cnn WIDER
4. ABS + cnn WIDER AVDIAR
5. ABS + OpenCV net


Recall
1
0.8
0.6
0.4
0.2
0
Video 1
Video 2
Video 3
Video 4
Video 5
Video 6
Video 7
1. OpenCV net
2. ABS + cnn FDDB
3. ABS + cnn WIDER
4. ABS + cnn WIDER AVDIAR
5. ABS + OpenCV net


F1 Score
1.2
1
0.8
0.6
0.4
0.2
0
Video 1
Video 2
Video 3
Video 4
Video 5
Video 6
Video 7
1. OpenCV net
2. ABS + cnn FDDB
3. ABS + cnn WIDER
4. ABS + cnn WIDER AVDIAR
5. ABS + OpenCV net


FPS
25
20
15
10
5
0
Video 1
Video 2
Video 3
Video 4
Video 5
Video 6
Video 7
1. OpenCV net
2. ABS + cnn FDDB
3. ABS + cnn WIDER
4. ABS + cnn WIDER AVDIAR
5. ABS + OpenCV net

5. Face Detector using Skin Color

Similar to the Adaptive Background Subtractor, our goal is to get the region of interest based on the skin color. Then we input this region through different face detectors, and we will be able to detect the appropriate faces. Using this method, we would be able to increase the FPS since we wouldn't be running the face detector on the full image, just the regions of interest.
In order to find the proper region of interest, several types of skin detectors could be used.
One approach would be to implement a simple classifier where we input a single pixel color, and detect if it's a skin color or not. Another approach would be to implement a semantic segmentation of the skin.
However, we decided to follow a much simpler approach, one that is implemented by Adrian Rosebrook of pyimagesearch [3]. This method relies on a range of possible color pixels that mostly represent the skin color. After experimenting with RGB, HSV, and YCrCb color spaces, the YCrCb gave the best results.

5.1. Skin Detection Algorithm

The first step in the algorithm we implemented is converting the color space to YCrCr, then creating a skin mask based on the range of possible skin colors taken from [3], then we apply erosion and dilation, then Gaussian blur, to reduce the noise. After that, we find the contours of the resulting previous mask. Then, we get the bounding boxes of the contours that will be our regions of interest.
Moreover, we manually tune the valid skin color ranges based using trial and error.
To further improve the results, we add some constraints on the bounding boxes that will have a higher probability of being a face.
For the bounding box to be valid, it has to pass these constraints:

- Area greater than a minimum area value
- Height:Width ratio in the range [0.9, 1.9]
- Centroid is in the top 65% of the frame

In the figure below, we can visualize the skin mask that is computed based on our algorithm and all the detected bounding boxes that are classified into ROI (in yellow) and not ROI (in red) based on the constraints above. In this example, we dilate the skin mask further in order to get the region around the face/skin and not just the skin. We can also notice the false skin detections when the color of the shirt or chair resembles the skin color, but they did not pass the constraints on the shape and size of the bounding box.

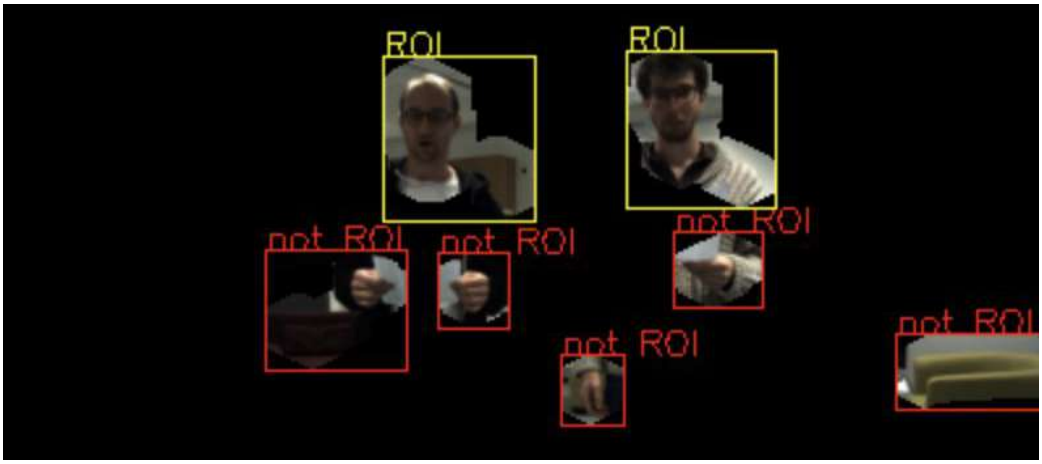


Figure 5.a

One major problem that we came across when testing the skin detector on the AVDIAR dataset is that in videos of sequences 1 to 21, the lighting in the room is sometimes too bright and the color of the ceiling, walls, and furniture is very close to the color of the skin thus rendering the skin detector useless. When testing the skin detector on this subset of video (1 to 21), it was detecting around 60% of the frame as a possible region of interest.

Figure 5.b

When testing the same detector on videos (22 to 44), it was able to properly detect the skin with different skin tones, as seen in the figure below.

Figure 5.c

In order to solve the problem of different illumination in different scenarios, we tried using a method presented in this paper [5]:*"The proposed method tries to minimize both false positives and false negatives, taking into account the illumination conditions of the image and it results to be computationally efficient for real-time applications."* This method relies on computing the global luminance of the image in the YCrCb color space, and based on this, the range of possible skin pixels would vary. However, we were not able to complete this implementation because of programming constraints.

After getting the final regions of interest in the frame, we input them in several models to evaluate and compare their performance and results based on the same metrics used in Adaptive Background Subtraction: precision, recall, F1 score, and FPS.

5.2. Evaluation

When it comes to the evaluation, we will use the skin detector to detect faces in the AVDIAR dataset. Therefore, we need to tighten the constraints mentioned in the previous sections by minimizing the range of the valid ratios of the faces, and adjusting the minimum area of a bounding box. We also have to manually tune the number of iterations for the erosion and dilation steps with a specific kernel size in order to get the best results. All of this tuning was done by trial and error.

After these adjustments, we are able to somehow differentiate between the bounding box of a face and that of a hand/arm or other objects as shown in figure 5.d. For example, the arm is not an ROI because the ratio of the height:width. However, in some cases, it may mistake a random bounding box with similar characteristics to that of a face as a face; such as a hand or an object as shown in figure 5.e.

Another minor problem is that in some cases where the skin of the neck is included the bounding fox of the skin of the face thus increasing the ratio between the height and the width therefore the detector will disregard this as not a face as shown in figure 5.f.
An interesting successful failure is that the skin detector also detected blond hair which has similar color to the skin of the person, thus being able to detect the back of the head of that person. And since the ground truth includes faces even if only the back of the head is visible, this becomes a successful detection.

Figure 5.d

Figure 5.e

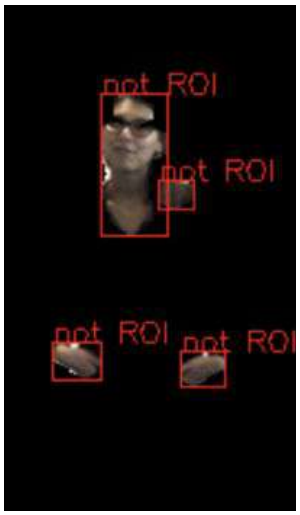


Figure 5.f

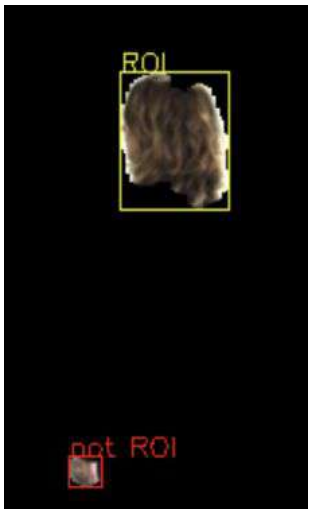


Figure 5.g

5.2.1. Using skin detector

The metrics of the evaluation of the face detector using skin color are surprising as the precision have an average precision of 80% varying from 33% in Video 22 where the combination of the skin of the neck, chest and the face are combined thus not satisfying the set constraints, and

reaching a maximum precision of 98.2% in videos 28, 29 and 40. It performs worse when it comes to the recall metric, as it averages at 43% with a minimum of 32% and a maximum of 48%. This means that when it detects a face, the model is confident that this is a face thus having a low number of false positives ; however, it can produce a relatively larger number of false negatives. This gap between the precision and the recall is visualized in the F1 score that averages at 55.4% which is pretty good compared to the models that we will later test.
The best feature of this detector is that it performs in real-time since it is a relatively simple algorithm that can detect faces in a single frame at around 0.04 ms per frame (without displaying the frame).

5.2.2. Using skin detector and CNN

After having promising results with the face detector using skin color only, we will add our own CNN model.
The algorithm is similar to that used in the Adaptive Background Subtraction. First, we run the skin detector on the frame with slightly more loosened constraints in order to achieve a larger set of possible faces. Then we input the images in the bounding boxes through the sliding window technique which uses the CNN model to detect if there is a face in the bounding box.
We will use the same CNN model we previously trained on several datasets: FDDB, WIDER, WIDER+AVDIAR.

When comparing the models trained on FDDB and WIDER, we can clearly say that in general model 8 trained on WIDER performed better, in all of the evaluation metrics, than model 7 trained on FDDB.
The average precision of model 8 is 62% with a low standard deviation. However, the average precision of model 7 is much lower at 36%. It is the same for the recall where model 8 averages at 66% compared to model 7 that averages at 55%. Both models 7 and 8 perform the same when comparing the frames per second which has an average of 10 fps.

After achieving good results with the WIDER dataset, we will see how it would perform if we extracted the faces from a different subset of AVDIAR videos and re-train the model. The results that we get are quite surprising as we notice that it performs worse than model 8 (only WIDER). This could be due to the fact that the model is detecting more false positives thus decreasing the precision. This may happen because of the existence of a lot of faces in the AVDIAR dataset where the head is not visible or not completely visible, such as the back of the head or an extreme pose away from the camera. This would affect the model as it would learn to detect different features than that of the faces which can increase the false positives.

5.2.3. Using skin detector and OpenCV pre-trained model

As discussed earlier, we will also use an OpenCV pre-trained CNN model for face detection in order to compare our models with the state of the art. When it comes to evaluating the pre-trained OpenCV model alone and then with the additional skin detector, it gets really interesting.

We notice that the precision of model 6 is at 100% but with a very low and unstable recall that varies from 1% to 66%. We notice that the F1 score is the lowest between the other techniques used. This is the same behavior that is discussed in section II.2.

This is why we apply the skin detector before inputting the image into the model, thus we would have much smaller inputs (only ROIs).

After applying this method in model 10, we notice almost the same precision as model 6 at an average of 97% with a low standard deviation. What is interesting is that we see a huge jump in the recall going from an average of 22% with a huge instability in model 6 to an average of 57% with a very low standard deviation in model 10, which is the second highest recall between all the methods tested. This is due to the fact that the skin detector is generating bounding boxes with a high percentage of being a face, thus massively reducing the region where the model should detect faces thus producing less false negatives. This also improves massively the F1 score which has an average of 70.75% which is also the highest between the different methods.

We also notice a faster detection from an average of 18 fps in model 6 to 22 fps in model 10.

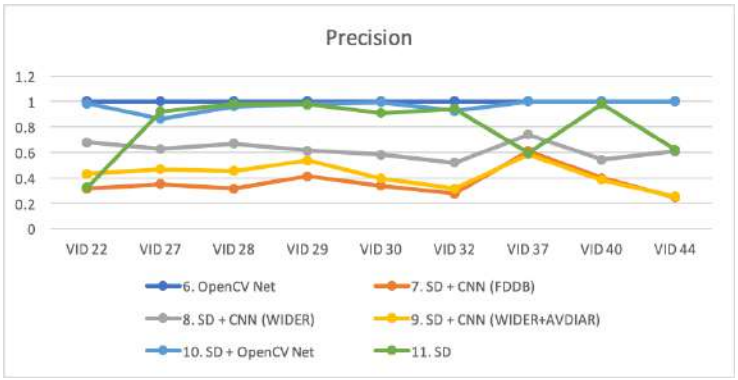
We can easily say that after introducing the skin detection method to the pre-trained OpenCV model, we improve the performance in general.

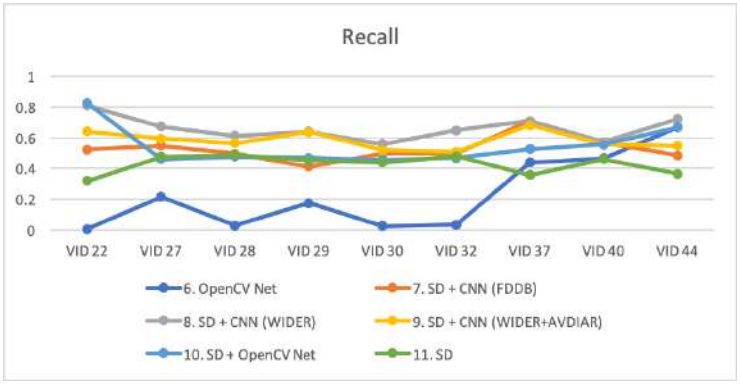
5.2.4. General Comparison Skin Detector Tracker

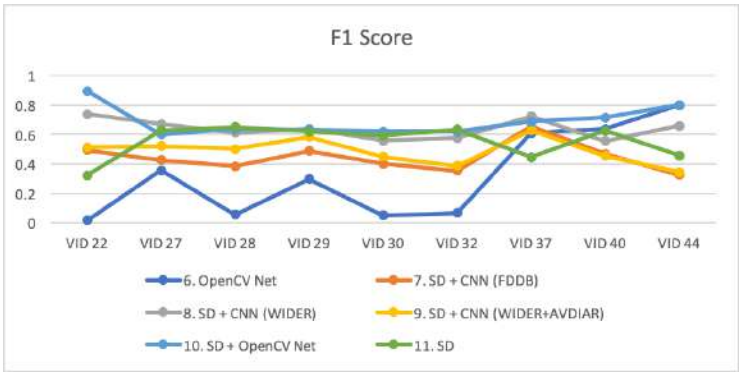
As an overall comparison between the methods used with the skin detector, we can notice that model 10 performs best, with the second highest precision, the second highest recall, highest F1 score and second fastest algorithm.

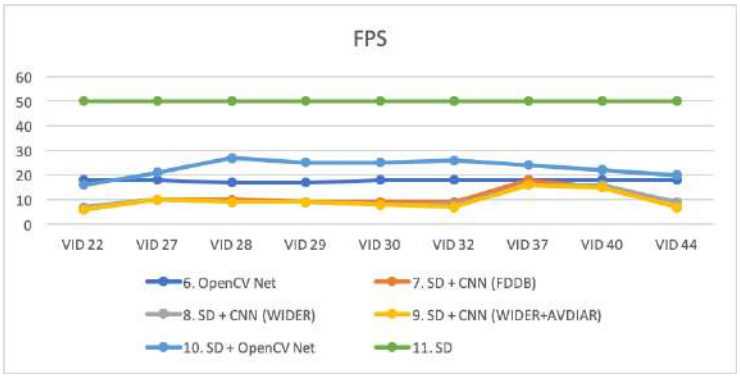
We can see in the recall graphs, almost all models tend to perform better in videos 37, 40, and 44. This is due to the fact that these are the videos of 2 persons standing facing the camera and almost not moving; therefore, this increases the possibility of false negatives, thus increasing the recall and therefore the F1 score.

We can also see the gap between the fps of the skin detector alone and the rest of the models, as it is expected.


Precision
1.2
1
0.8
0.6
0.4
0.2
0
VID 22
VID 27
VID 28
VID 29
VID 30
VID 32
VID 37
VID 40
VID 44
6. OpenCV Net
7. SD + CNN (FDDB)
8. SD + CNN (WIDER)
9. SD + CNN (WIDER+AVDIAR)
10. SD + OpenCV Net
11. SD


Recall
1
0.8
0.6
0.4
0.2
0
VID 22
VID 27
VID 28
VID 29
VID 30
VID 32
VID 37
VID 40
VID 44
6. OpenCV Net
7. SD + CNN (FDDB)
8. SD + CNN (WIDER)
9. SD + CNN (WIDER+AVDIAR)
10. SD + OpenCV Net
11. SD

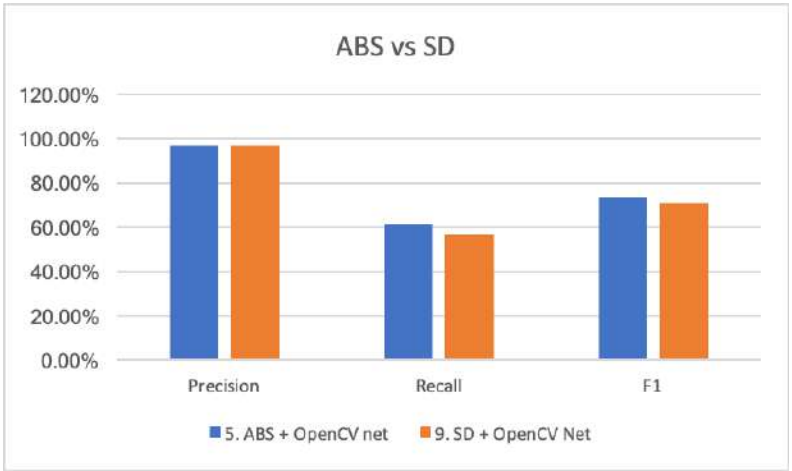

F1 Score
1
0.8
0.6
0.4
0.2
0
VID 22
VID 27
VID 28
VID 29
VID 30
VID 32
VID 37
VID 40
VID 44
6. OpenCV Net
7. SD + CNN (FDDB)
8. SD + CNN (WIDER)
9. SD + CNN (WIDER+AVDIAR)
10. SD + OpenCV Net
11. SD


FPS
60
50
40
30
20
10
0
VID 22
VID 27
VID 28
VID 29
VID 30
VID 32
VID 37
VID 40
VID 44
6. OpenCV Net
7. SD + CNN (FDDB)
8. SD + CNN (WIDER)
9. SD + CNN (WIDER+AVDIAR)
10. SD + OpenCV Net
11. SD

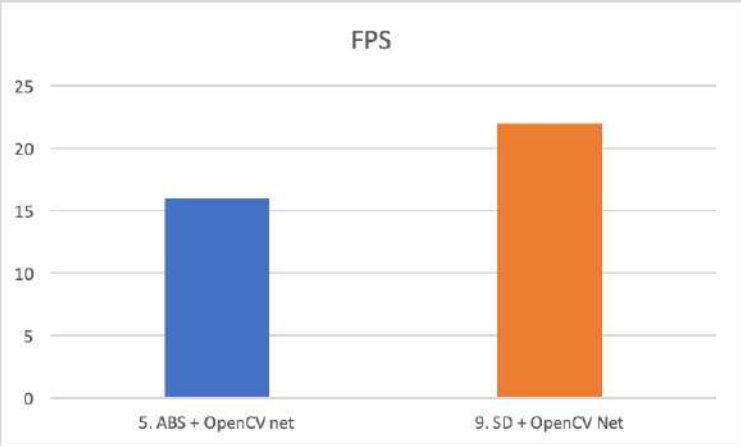
6. Overall comparison

This chart presents the results of all the trackers that we did, these values correspond to the average of the metrics for each video. As we can see easily, the best ones are the ones that combine either ABS or Skin Detector with the OpenCV Caffe-based network.

Taking a closer look at the top two methods to see which one performs better:



Here we can see that the answer is not that obvious, at least not at first sight. The average precision values are almost the same, and the recall of the Skin Detector one is just under the one from ABS. Same thing happens with the F1 Score as it is proportional to the two. So it may seem like they are almost the same, but we have to find a tie breaker. So we take a closer look to the last metric: computation time.



Now it is easier to choose, there is a distinct improvement in computation performance when using the Skin Detector one. This is due to the simple fact that the region of interest generated by the skin detector is significantly smaller than that generated by the ABS. This results not only in a smoother viewing experience (therefore more discernible results) but also in a faster computation time that may be very important when testing on a large batch of videos.

References:

[1] LeCun et Al. 1998. http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf
[2] Caffe-based model in OpenCV:
https://github.com/opencv/opencv/tree/master/samples/dnn/face_detector
[3] Skin detection algorithm:
Adrian Rosebrock. 2014.
https://www.pyimagesearch.com/2014/08/18/skin-detection-step-step-example-using-python-opencv/
[4] Swish: Booting ReLU from the Activation Function Throne
https://towardsdatascience.com/swish-booting-relu-from-the-activation-function-throne-78f87e5ab6eb
[5] N. Brancati, G. De Pietro, M. Frucci, L. Gallo.
Dynamic clustering for skin detection in YCbCr colour space
https://elib.bsu.by/bitstream/123456789/158528/1/Brancati_Pietro_Frucci_Gallo.pdf