Dexter laboratory

- face detection & recognition-

Project Report

Ocnaru Mihai-Octavian

1. Introduction

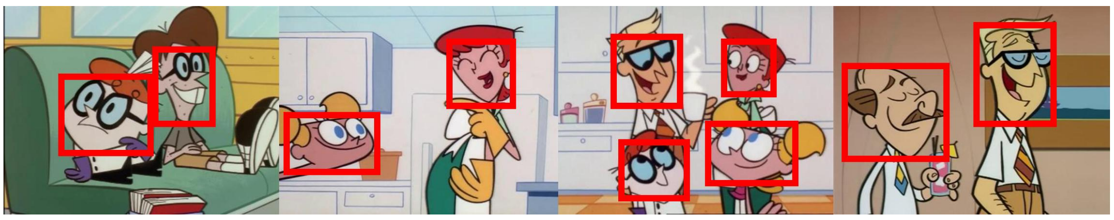
The following project tries to achive a good performance in facial detection and recognition on an images dataset, based on Dexter Laboratory cartoon. More about the cartoon here.

Regarding the implementation, we had to follow certain restrictions on what could be used so that it wouldn't be considered a "tool" for making everything simpler. For example, we couldn't integrate an already trained model or use a pretrained one with transfer learning. Everything we used had to be trained by ourself.

1.1 Objectives

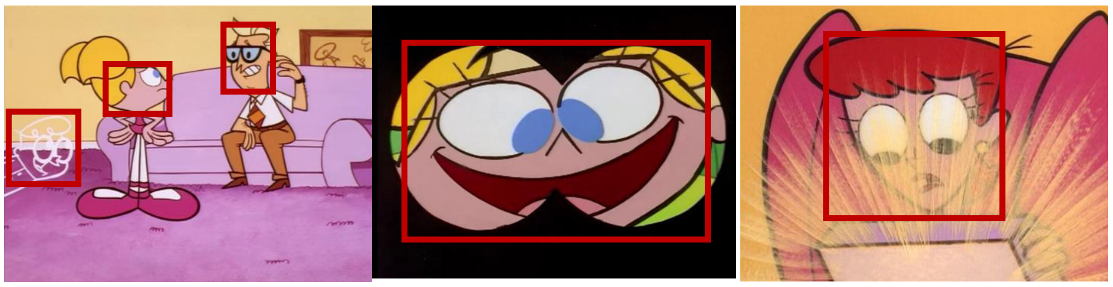
The main two objectives for this project were, as mentioned earlier, the facial dectetion of the characters as long as recognizing them. Those two have also some restrictions:

1.1.1 Facial detection



In the detection process we should aim to find all faces from all images, regarding their class. All the classes from this project were consider the main characters: **mom**, **dad**, **dexter** and **deedee**, but also the others charcters who will fall under the **unknown** class.

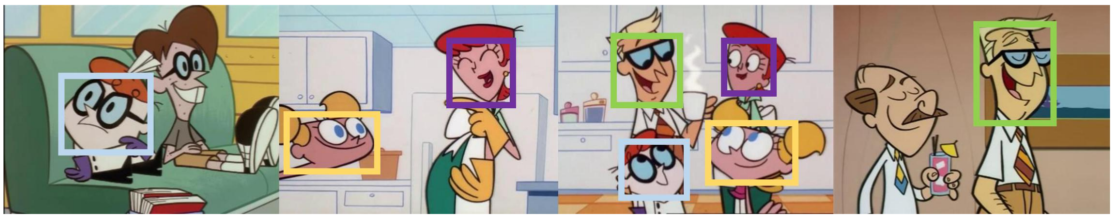
The faces were in a great percentage in a normal looking aspects and positions, but we had to also take into account the outliers from the dataset, for example the following images:



Speaking of detection we will also have to take into account the invariance of the detector for the rotation of faces, different aspect ratios and other environmental changes, like brightness and hue in some pictures.

1.1.2 Recognition phase

In this phase we only had to recognise the main characters. Finding an **unknown** character will affect the precision of the solution. So for above picture those will be the outputs.

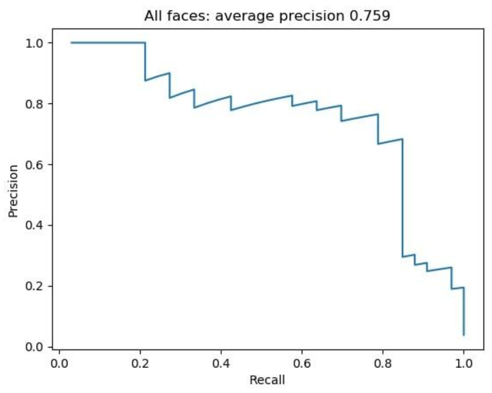


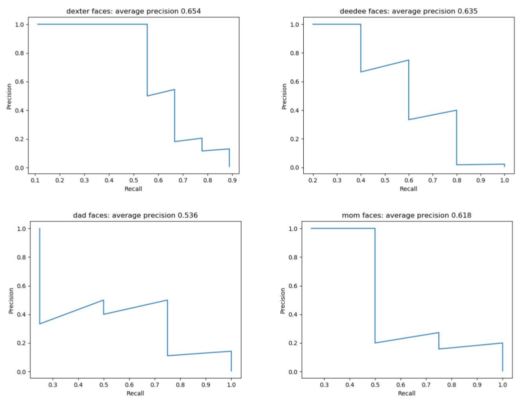
1.2 Evaluation metrics

For the detection phase we will plot the precision and recall and compute the area under the curve. The predictions are sorting in descending order based on the confidence score. This score reflects the confidence that in the predicted bounding box there is a face, where 1.0 will suggest strongly the presence of a face and 0 that there is no evidence of any face at all.

To be considered in the plot, all predictions will have to pass a certain intersection over union(IOU) threshold with the ground truth bounding box. For this project the IOU threshold is set to **0.3**.

Regarding the recognition phase, we will plot the precision/recall curve for every class, and compute the mean average precision(mAP).



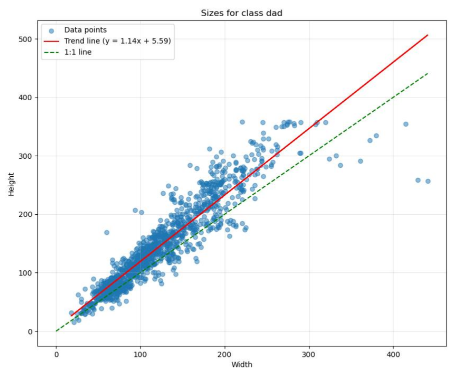


2. Implementation

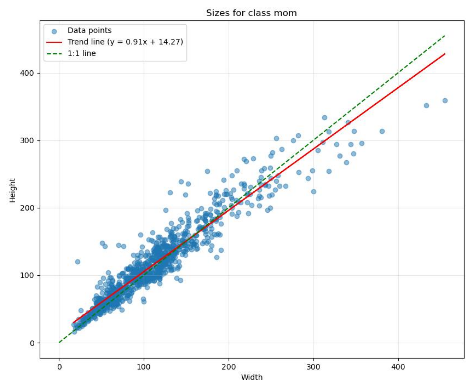
2.1 Finding the windows sizes

The requested implementation for solution was to be based on a sliding window techinique to which we should add a binary classifier for face/non-face detection.

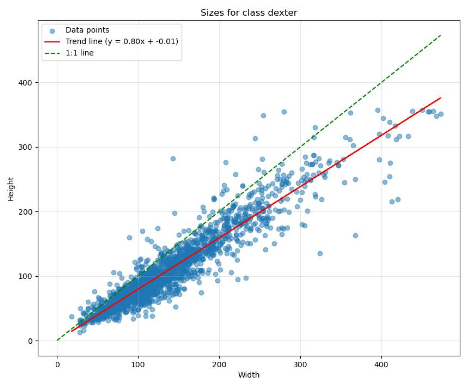
For this part I have analyzed all the classes and plotted relevant statistics to ensure a good decision deciding the correct sizes. Let's look at each class width and height plot and try to understand:



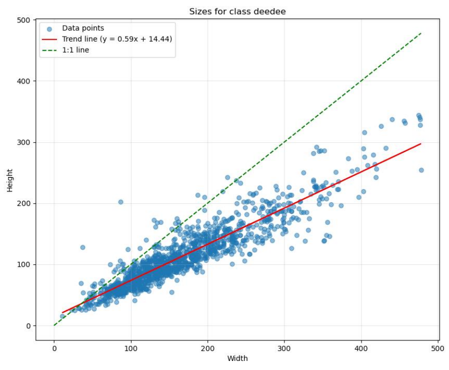
Dad plot



Mom plot

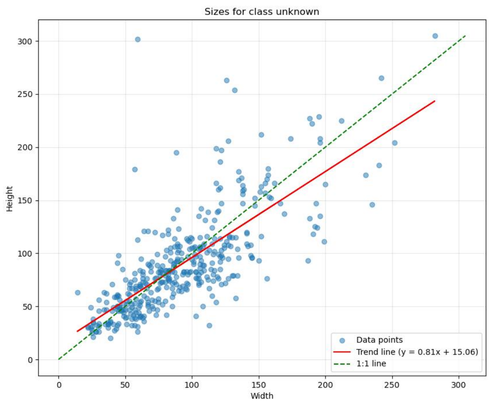


Dexter plot



Deedee plot

Unknown plot

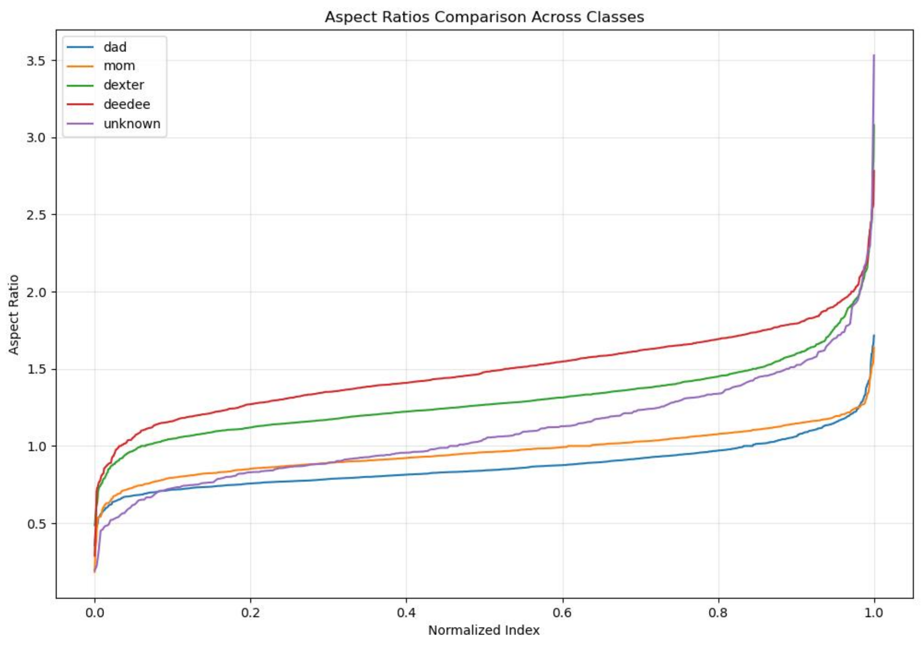


We can clearly observe some intersting patterns in the aspect ratios of the classes. First of all, we can see that each main characters's classes has a little deviation from the trend line and a more norrow shape, compared to the unknown class which has a spread plot and a more distributed aspect. This is a consequence of the variation in the unknown class, caused by so many different characters.

Second, the plots have a **1:1** line there for the refferences, helping us understand the different shapes of each character. For example, we can observe that mom tend to have a more square version of her face, rather than a rectangular one, compared to deedee.

The position of the trend line related to the green line suggest the position of the rectangular shape. For instance, Deedee's face is a rectangle with a higher width than height, opposed to dad's face configuration, which is a 90º "vertical" rectangle, with the height bigger than the width.

For a more easier visualisation let's look into the plot with all the aspect ratios:



Here we can understand that we are dealing with some concentration in some areas. Looking at mom's line we can see that it tends to stay from **0.8** to **1.2**, but the dense region here is around the 1 index mark.

Seeing those region of interests I have thought about using a clustering method for finding the representative region in an optimal way. The problem with this technique arrise in the way we implement it.

Suppose we choose to have **k** windows in our project. Let's ignore the fact that we have to carrefully choose this hyperparameter which will affect the training time as well as the complexity of eliminating duplicates detection, by increasing the work needed to compute the IOU threshold for the aggregator.

We have two options now:

1. We compute the **k** windows based on the all data from all classes
2. We compute only **a fraction of k** windows for each class

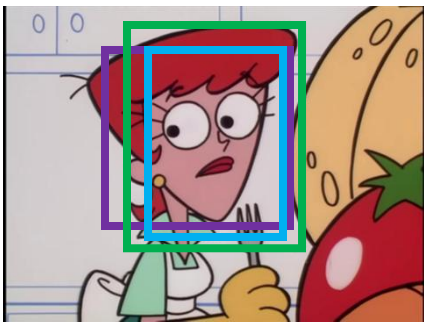
Let's start analyzing the second approach. We can assume that the class distribution is uniform, meaning that each class has **n** different dimensions. This will mean that we have to find **k/5** windows for each class that will represent the dimensions and aspect ratio of that class. Even tho, this is a false affirmation (see the plot below), there is another huge problem with this approach.

If we look closely at the 5 plots of the dimensions, we can observe a high density in the **[50, 200] x [50, 150]** zone. This is normal as our image size is only **480 x 360**. So this mean that even if we will find a correct rectangle for, let's say class Deedee, that rectangle could also work for Dexter, according to the similarities between the two plots.

Considering also the fact that for a valid facial detection the IOU treshold with the ground truth should be only above 0.3, we can denote that a valid Deedee's rectangle can be use for a mom or dad bounding box.

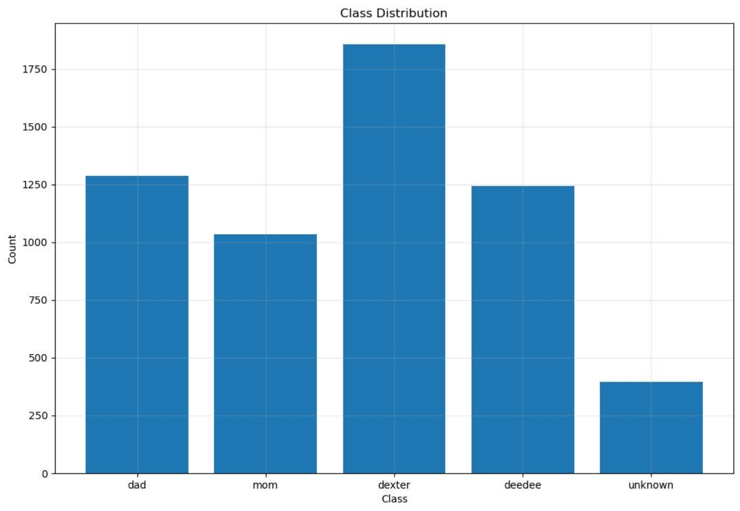
As an example suppose the following picture of mom. Her native aspect ratio is somewhere around **1.0**. This means that a square face should be just fine, how to find a good width for it will be discussed later. If we try to detect her face with a dad or Dexter face, we won't have any problem as it will have a higher IOU score than our threshold.

The purple square would be her real face, the green and blue one will be for a dad's detection and Dexter's one, respectively.



So the second approach will fail because of the use of redundant heights and widths which will cause duplicates in the clusterization process, but also won't cover the dataset as well as we wished.

Let's observe than, the distribution of data along all classes:



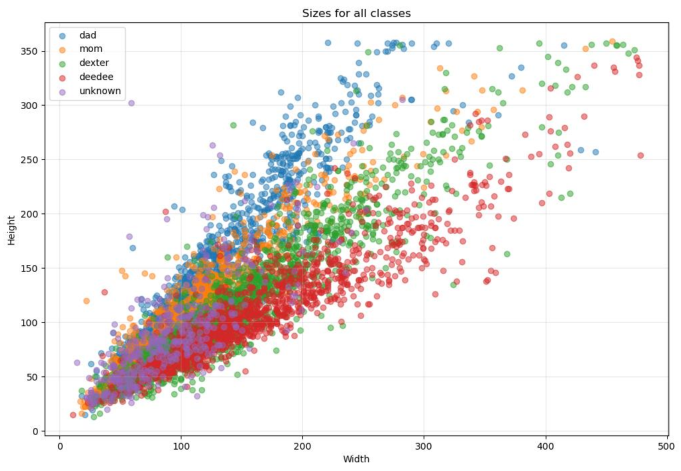
Analyzing the data we can cleary see that the Dexter have far more examples that the next represented class, dad, by at least **38%**. We will continue than with the first approach in which we will consider all the points in the dataset and apply the clusterization method on them.

We can try various method for clusterization, such as:
- K Means
- Gausian Mixture model
- HDBScan
- DBScan.

From experimentation and searching that best value for **k** would be 30, assuring a faster processing and a easy to find IOU threshold for the aggregator.

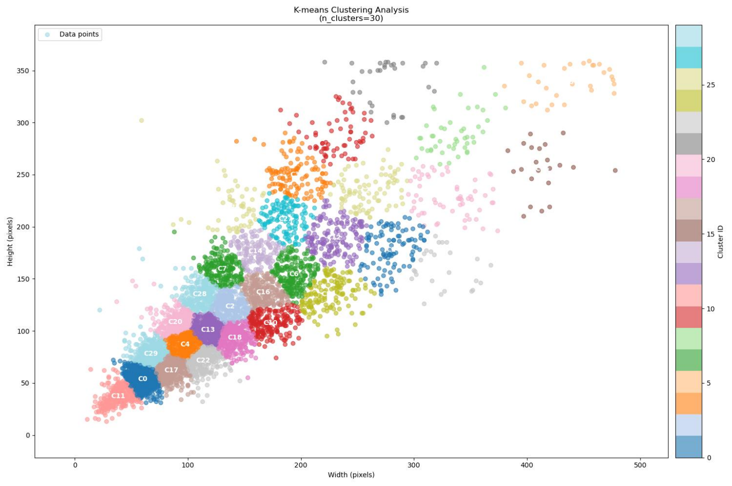
The hard problem was finding the best clustering method, for my problem. Plotting the data points we can clearly see that we have some high density region in the left-bottom corer and a high spread zone in the top right corner.

So choosing a cluster algorithm should be about finding the optimal partioning of the points, without segmenting the high concentration zones.

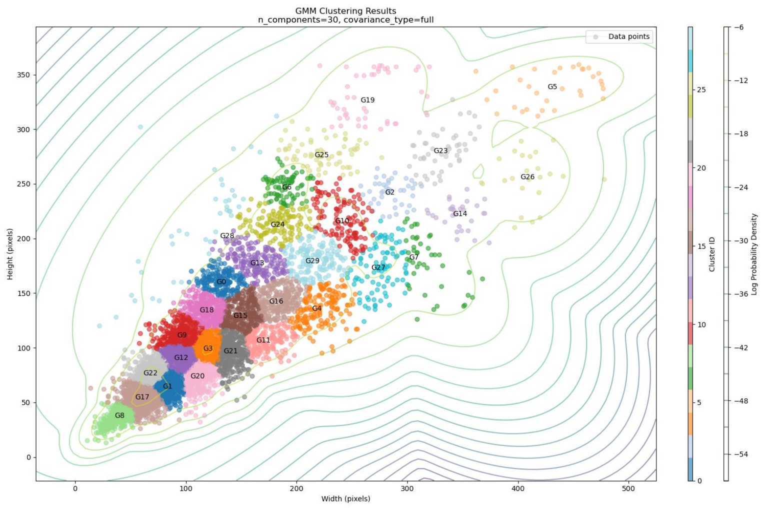


All classes

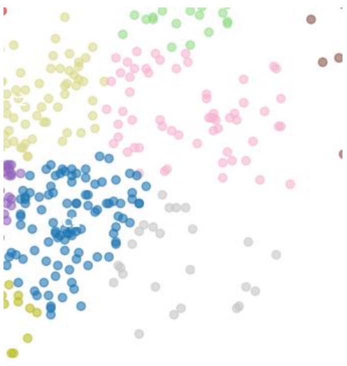
After some tests I have come down to two options: Kmeans and GMM. The below plots illustrated both clusters.

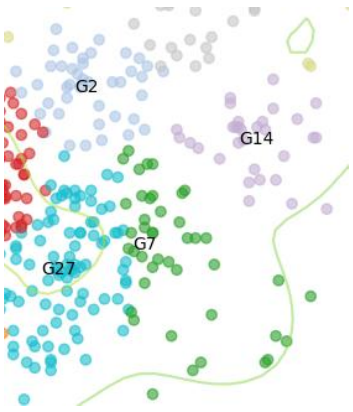


Kmeans



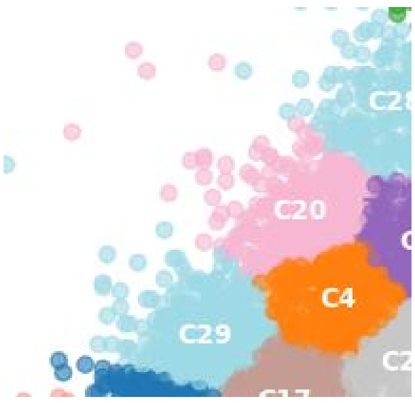
GMM


Kmeans

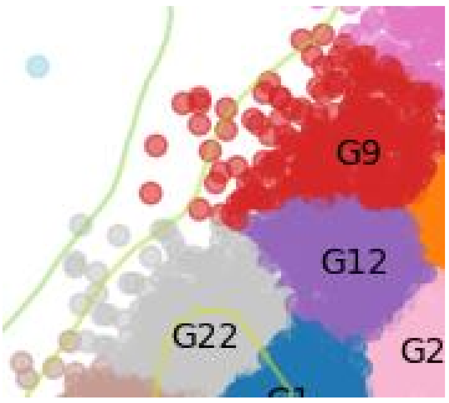


GMM

Choosing between those two comes down to observing the differences at handling the outliers in the spreaded zone. The GMM tends to incorporates the spreaded points in a closer graph, while the Kmeans creates a new cluster, this way ensuring a better representation of the dataset.



Kmeans



GMM

Looking closer at the dense zone, at the beging of the plot we can also observe that the region betweens two clusters are better incorporate under the Kmeans strategy for our use case.

2.2 Sliding window

Now that we have found the desired window sizes we can slide over an image. For this part I have choose the stride to be **0.3** of the window size. This will give the bellow table regarding the founded windows at each step:

Cluster Size	**Y Iterations**	**X Iterations**
335 x 289	7	1
275 x 339	5	4
330 x 228	12	1
231 x 287	7	6
258 x 234	11	5
325 x 160	20	2
279 x 178	18	3
194 x 249	10	9
229 x 189	17	6
186 x 207	14	10
139 x 218	13	17
224 x 135	27	7
193 x 154	22	9
159 x 176	18	14
166 x 137	27	13
130 x 159	22	18
172 x 107	38	12
137 x 123	30	18
110 x 135	27	23
141 x 93	43	16
117 x 101	38	23
88 x 108	38	34
97 x 87	50	30
113 x 71	59	23
85 x 61	70	35
67 x 77	58	49
60 x 53	86	50
38 x 37	148	108

Total window iterations: 35,073.

2.3 Facial detector

With all the computed windows all we have to do is to build a face detector that should output a probability of finding a face in the current window.

For this part we first build a non-faces dataset which consists of random patches from all images provided with the following rules:

- the patches can overlap one-another no more than **40%**
- the patches can overlap a face no more than **45%**
- we try to extract **60** patches from each image

The resulting dataset consists of **190.000** images.

After that we have trained a convolutional neural network, for about 200 epochs, using a **0.2** split factor for my test/validation dataset. As for the optimizer we used **Adam**, with a decaying learning rate and **binary crossentropy** as the loss function.

The training process was also optimized to use batches for data loader, to ensure that all images will fit into RAM. The resulted model had a 95% accuracy of the provided data.

With the founded window computed in the previous step we create a batch and use the model to predict. The use of a batch reduced the time prediciton by **500%**, ensuring a faster compution time.

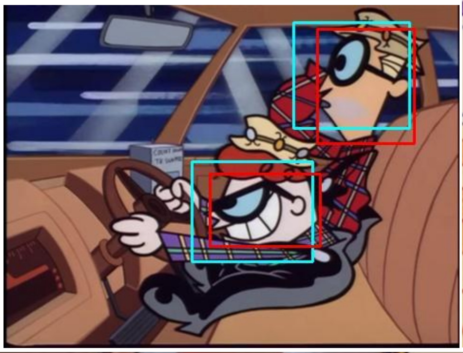
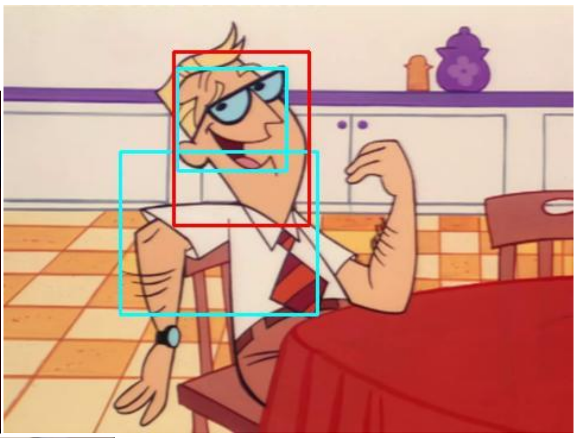
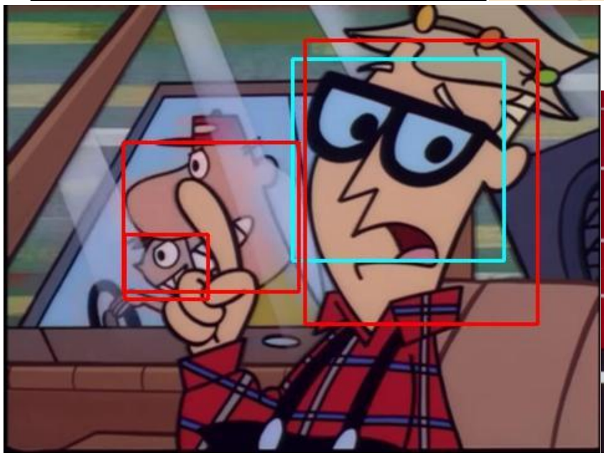
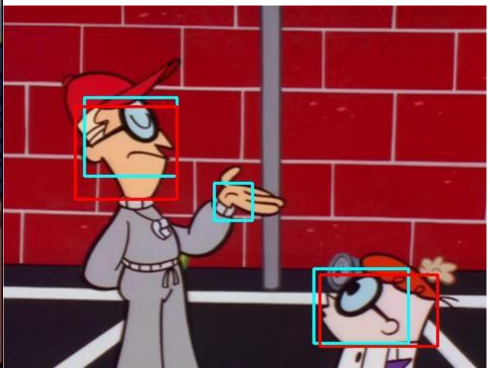
2.4 Aggregating detections duplicates

Having all the bounding boxes predicted from the model as well as each one having its confidence score we will have to remove the duplicates from the detections. The first attempt was to use the non maximum suppression and try to find the best IOU threshold for it. After some tries I decided to switch to another mechanism, as it gived poor performance, although the good predictions found.

In order to aggregate those we first cluster the windows based on a given IOU threshold. After some try-error approach the best threshold found was **0.18**. This is so low due to the presence of numerous images with faces being close one to another. We will also consider the prediction that are above a certain threshold, in our case it will be set to **0.8.**

After clusterization, the clusters are each one processed resulting a final aggregated window and score. The window is obtained by a ponderate average of all coordinates of the windows made based on the score. This way a high confident prediction will impose more than a smaller one. The final score is calculated as an average of all scores.

Following the last step, the founded predictions are written to their coresponding folders. Here's are some prediction, the red rectangle is for the ground truth and the cyan one is the predicted.

We can see that it can track and find faces with a good accuracy, but il will also produce mistakes, mainly because of the IOU threshold for the aggretaor.

There is a trade-off that has to be made, either we lower the threshold ensuring a better split between two close faces, loosing the accuracy of isolating the faces from false positives (like in the last image, where the hand is taken into account because of the lack of aggregation) or we use a higher treshold accepting the fact that our process won't be able to split between closer faces (see the third images, with the father, son and the dad), but with a better overall accuracy for detection.

2.5 Face classification

For making the distinction between founded faces, we will use another CNN trained with the provided faces. This time it will use **categorical crossentropy**, the same optimizer, the same strategy for learning rate, but it will be trained on a lower number of epochs.

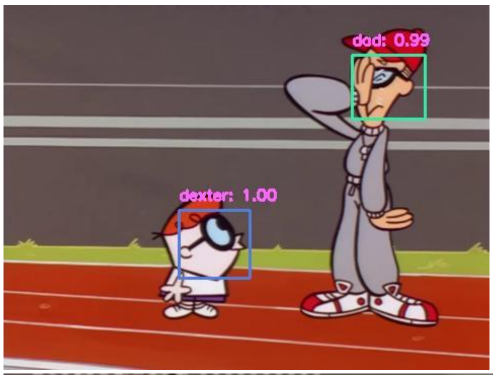
The input size will be the same as for the detection model **130 x 116**, this number representing the median values for all widths and heights.

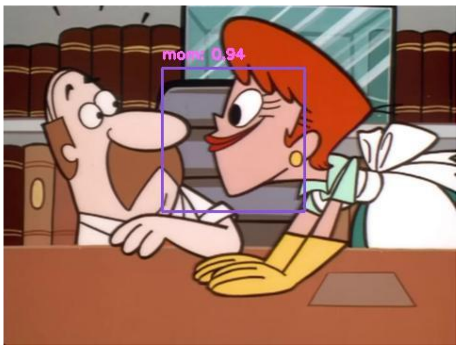
The dataset used here are simply the provided images, with a split factor of **0.2**. The output of the model will be a probability across 5 nodes, each one representing the percentage of belonging to a class, including the **unknown class.**

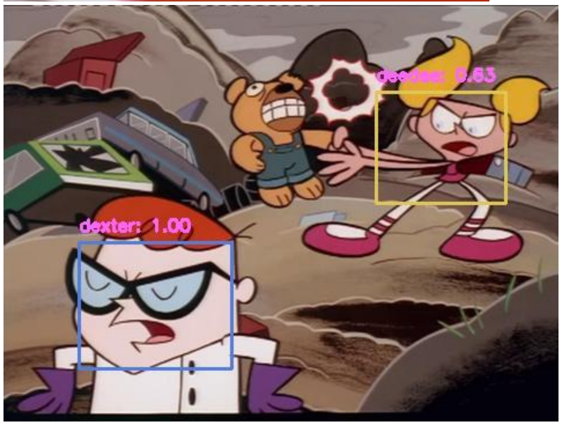
The process will be the same as previous, using batches technique, for decreasing computation time, each founded prediction from previous step will be passed to the model and it will try to guess its correct class.

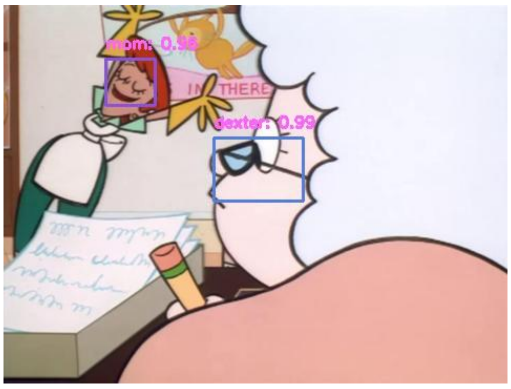
Once all of this will be completed, the result is then written to files, in a special format.

Below is a sample of how good the classifier is doing its task:



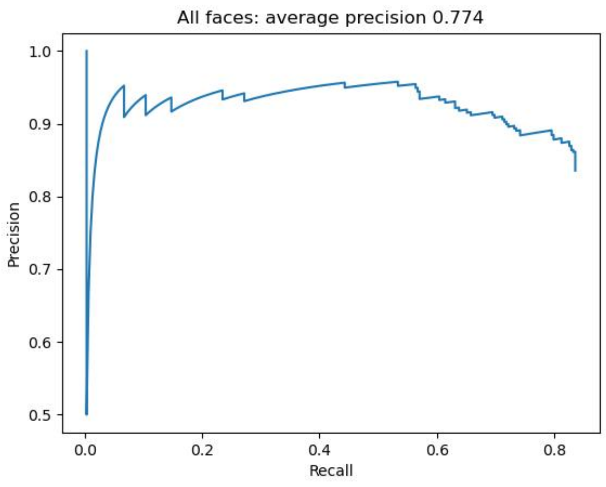


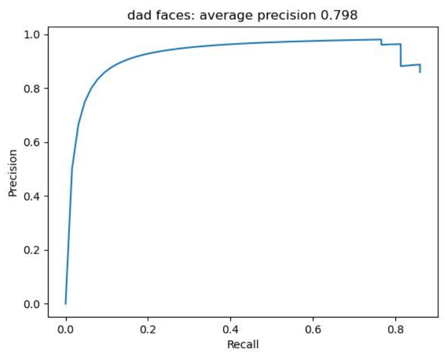


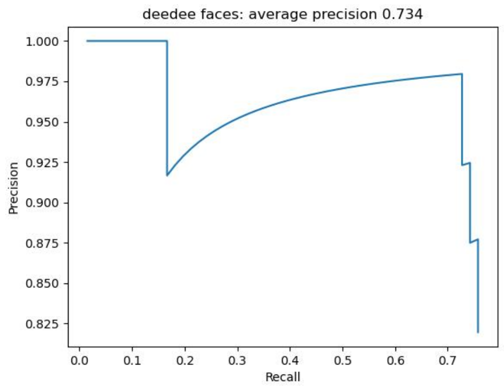


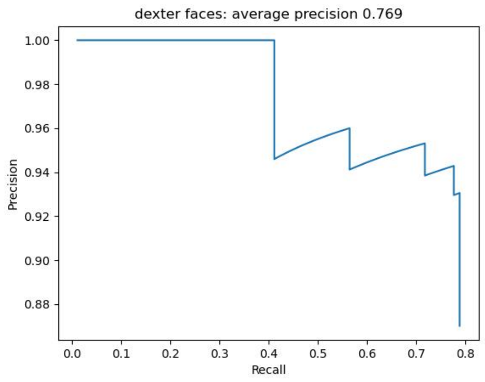
3. Results

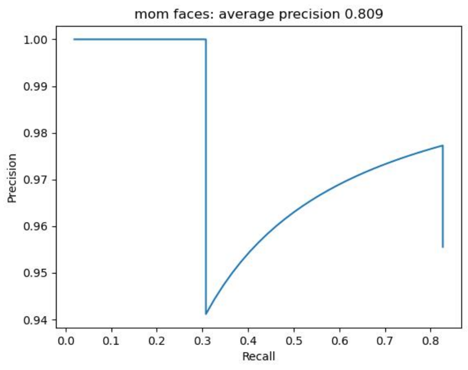
We were provided a special **200** images dataset as validation test, and below are the results obtained using the above method:











For the detection part this approach grants **0.774** score as well as a **0.777** mAP score for classification.