Documentation - Face Detection and Recognition

Ciuperceanu Vlad-Mihai

1 Introduction

This project aims to detect and recognize the faces of characters from Dexter's Laboratory in the provided images. These tasks guide the two main objectives, while the bonus task allows the use of a state-of-the-art detector to solve these problems. For the first two tasks, we will use classifiers trained on HOG descriptors, following the sliding window paradigm, while for the bonus task, we will use the YOLO model to achieve near-perfect results.

2 Precalculations and Observations

Before deciding on the parameters for our solutions, we can first analyze the images in the training set to gather more information about them.

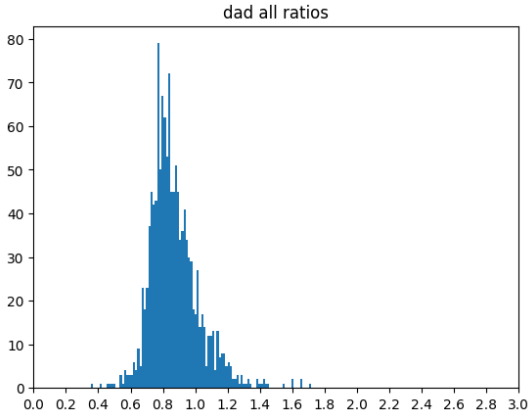
Clearly, we observe that the faces vary significantly in both size and shape (some characters have faces that can be easily framed with a square, others with a longer rectangle, and others with a taller rectangle).

By extracting the annotations for each face, we notice that their dimensions are quite large, with both dimensions being around 150 pixels. Of course, they can also go as low as 50-60 pixels or as high as 200 pixels. Thus, we obtain an upper limit for the window size, although we can, of course, resize the images in certain situations.

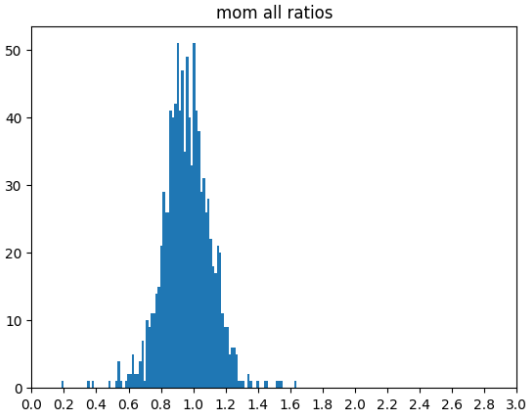
Since a multi-scale approach is needed, where we resize the image, we realize that it is easier to work with this aspect and that the choice of size is not so strict, as it can work with multiple values with relatively large differences, by calibrating the parameters in the multi-scale approach.

However, during resizing, the aspect ratio is preserved, so we must pay attention to the ratio of the face dimensions. These ratios will translate into the windows (as mentioned above, the faces have different shapes, so it would be helpful to use multiple windows).

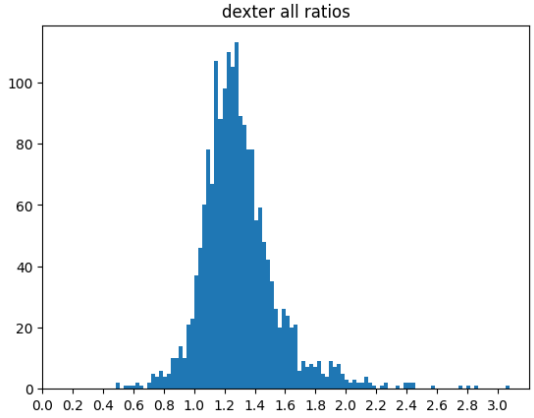
By analyzing the data obtained from the annotations and extracting the coordinates of the faces, we can calculate the aspect ratio for each face. For better visualization, we created a histogram for each of the four main characters and the unknown character category to see the distribution of ratios within the identified faces. We obtained:



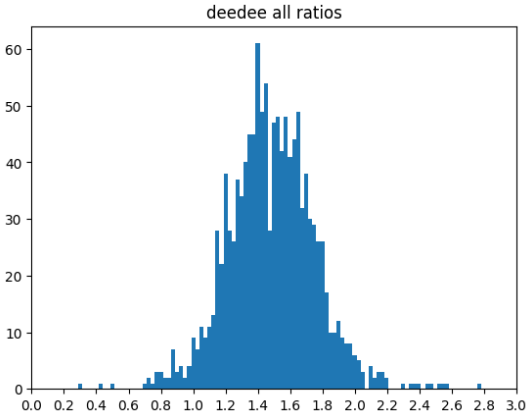
(a) Dad



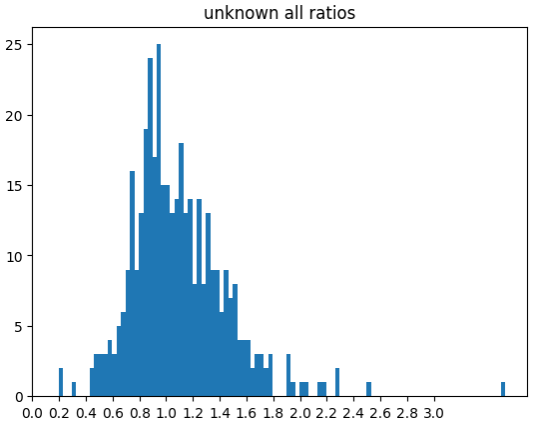
(b) Mom



(c) Dexter



(d) DeeDee



(e) Unknown

Figure 1: Histograms of the aspect ratios of the characters

By analyzing the obtained histograms, we observe that the most condensed area for each character is in the intervals:

- $[0.75 - 0.85]$ for Dad
- $[0.9 - 1.1]$ for Mom
- $[1.1 - 1.3]$ for Dexter
- $[1.4 - 1.6]$ for DeeDee
- $[0.9 - 1.1]$ for Unknown

Thus, observing the variety of the data, we decided to use 4 windows for 4 different aspect ratios: 0.75, 1.0, 1.25, 1.5. The numbers were chosen to represent the condensed areas in each histogram, focusing on the four main characters of interest, as they appear most frequently. At the same time, we aimed for a balanced distance between them, trying to capture both square-framed faces and rectangular faces (both taller and wider).

3 General Approach

Given the variation in the characters' faces, as mentioned above, we will use 4 types of windows to cover the faces of the four main characters, as well as the unknown ones, which can be detected by at least one of the windows (for task 1).

As a general approach to the two tasks, the steps we follow are:
- obtaining positive and negative HOG descriptors
- training the models based on the obtained descriptors
- predicting detections and classifications for the found faces

3.1 Number of Positive and Negative Examples

The number of positive examples depends on the number of faces in the training folder. We chose to flip them left-right to double the number of positive examples, reaching over 11,000 such examples. For the number of negative examples, we chose a value of 48,000, because training models with the RBF kernel is more time-consuming, and a large number of negative examples (and the training dataset in general) significantly increased the training and prediction time.

Additionally, the discrepancy between the number of positive and negative examples should not be neglected, as it could lead to poor learning and a lack of generalization capacity. That is why we tried to find values that would help the models in this process.

3.2 Window Sizes

We will keep the ratios mentioned above, so we need to choose only one size, with the second being calculated based on it.

We chose to start with a size of 90 pixels (the dim_window parameter), with $dim\_hog\_cell = 18$, so that for rectangular windows, the smaller side would be of this size, maintaining the desired ratios (e.g., the window with an aspect ratio of 1.5 will have dimensions of 90 and 135). The starting size for the window was chosen through multiple trials, with numbers such as 36, 48, 50, 56, 60, 75, 100, trying to find windows that would be suitable for the average face size in the images. Of course, for smaller faces, a greater reduction in image size was needed, starting from the initial image, while for larger windows, an initial enlargement was needed, followed by gradual reductions with smaller steps. All these parameter adjustments aimed to detect both smaller and larger images.

For dim_hog_cell, we chose to maintain a ratio between the window size and the cell size, as this had two implications: on the one hand, the descriptors became more complex and increased in size, which theoretically would offer better precision per model, but, on the other hand, it significantly increased the required learning time, as well as the prediction time, since the model needed to be much more complex to find the correct separation. For linear kernel models, we could try, for example, values like $dim\_window = 5$

and $dim\_hog\_cell = 5$, but, for the RBF kernel, such a ratio of 15 made working with the models more difficult.

3.3 Models

The models used correspond to each type of window, so we have 4 models.

The type of model used is SVC with the RBF kernel, setting the C parameter to 1:

```
model = SVC(C=1, kernel='rbf', verbose=True)
```

Verbose was set to *True* to monitor the model's training progress. Of course, it required more training time than the linear kernel model, taking between 10 and 20 minutes. The C parameter was set to 1 to bring the model closer to convergence, this value being more convenient. The threshold kept was 0, observing that the average precision is quite sensitive to threshold changes, even small ones, leading to a decrease from 1-2% up to 5-6%.

Previously, we also tried SVC models with a linear kernel, but we observed a boost of up to 20% by switching the kernel.

The same model paradigm was used for both task 1 and task 2, the difference being in how they were trained, i.e., the selection of positive and negative examples.

3.4 Details on Prediction

For the prediction part, given that faces can appear at different sizes (smaller or larger, depending on the characters' positions in the frame), we use a multi-scale approach, resizing the image to better detect faces, thus creating an image pyramid. Initially, we slightly enlarge the face, then start shrinking it until its size becomes smaller than the window we are working with.

To create the image pyramid, we used the *pyramid_gaussian* function from scikit-image, which generates images starting from a given image, downscaling with a given factor and applying Gaussian filters for smoothing:

```
for resized_image in pyramid_gaussian(img, downscale=1.05)
```

The chosen factor is 1.05 to avoid reducing the image size too quickly, given that we use relatively large windows. When we obtain a detection, we make sure to rescale it to the corresponding coordinates in the initial image, taking into account the degree of downscaling so far, as well as any initial resizing (specifically, enlargements).

Of course, we follow these steps with each window used, then apply non-maximum suppression between them to filter the detections obtained on the image.

Now we will focus on the details of each task.

4 Task 1 - Detection

For the first task, our models needed to detect between face and non-face. The use of multiple models is used to detect multiple types of faces, combining the results (each model being trained with positive and negative descriptors for the type of window it represents). We chose to merge the results by applying non-maximal suppression to the detections obtained on an image to keep only the best ones.

4.1 Positive Descriptors

The positive descriptors were chosen from the annotated faces.

It can be seen that the faces are sometimes annotated more tightly, so we considered the face patch slightly extended, adding an extra 10% of the face size on both dimensions. Then, the HOG descriptor was calculated from the obtained patch and added to the list of positive descriptors.

However, the imbalance between the number of positive and negative examples remains, so it is important, of course, that the models have a large number of training examples to achieve better generalization capacity.

Thus, we tried to increase the number of positive examples using a method used in face prediction: building a Gaussian pyramid. Taking the initial patch but using a more aggressive downscale of 1.2, we resized the respective patch, then calculated the HOG descriptor from its resizing. In this way, we were able to increase the number of positive examples, adding some more robust examples to variations in images due to the applied filter.

This addition of positive examples brought an increase of 5-10% in the average precision obtained.

4.2 Negative Descriptors

For negative examples, we needed to choose random patches from the images, trying to avoid faces, leaving a small margin of 0.15 for calculating the intersection over union function between them.

Additionally, it was helpful to choose patches with variable sizes, where we kept the ratio for the window of interest and used *cv.resize* afterward, instead of always selecting the same sizes for patches. In this way, we introduced more variety into the chosen negative examples, reducing the possibility of overfitting and bringing an increase of 3-5% in the average precision obtained.

4.3 Results

With the approach described above, we obtained the following average precision:

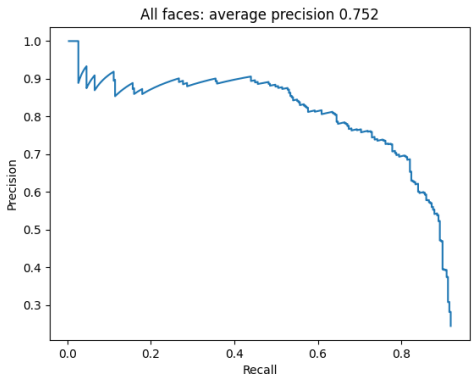


Figure 2: Average precision for detections

5 Task 2 - Classification

For the second task, the models no longer represented only the window size but also the detected character. Thus, each of the 4 models was a character detector, assigned to the character from whose histogram the window's aspect ratio was chosen.

The prediction method remains the same as in the first task, using multi-scale. A small adjustment is that, through repeated trials, we observed that for DeeDee, enlarging the image before creating the Gaussian pyramid did not help, unlike the other characters (losing 2% of the average precision). This may be due to the window size, which is the largest (aspect ratio of 1.5).

The only major difference is that, in this case, each brought results independently, no longer needing to combine the results, since we were interested in the detections of each character separately.

5.1 Positive Descriptors

The positive examples were chosen similarly to those in the first task, slightly enlarging the character's patch with a factor of 1.2, the difference being that each classifier used as positive examples only the faces of its character, not all of them.

5.2 Negative Descriptors

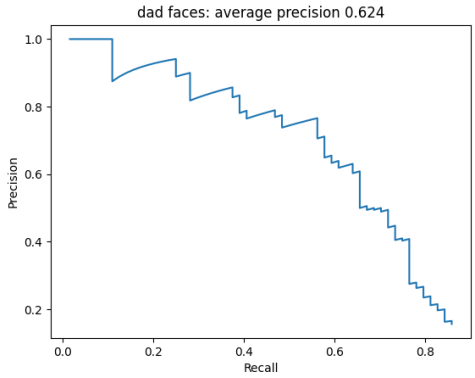
Needing a Dexter vs. non-Dexter classification, the model used had to distinguish between these classes.

We divided this capability into two: distinguishing between a character and the other characters and distinguishing between a face and a non-face.

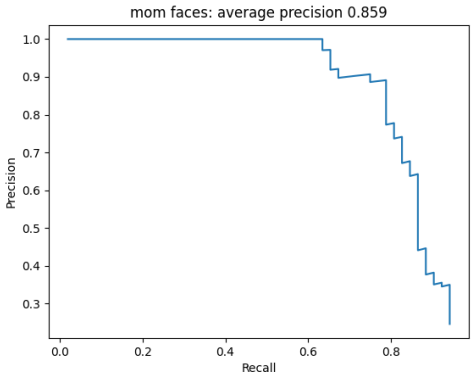
Thus, the number of negative examples was divided into two: half of them were given by the HOG descriptors of the other characters' patches, and the other half were given by random patches that did not have too large an intersection over union with the faces.

5.3 Results

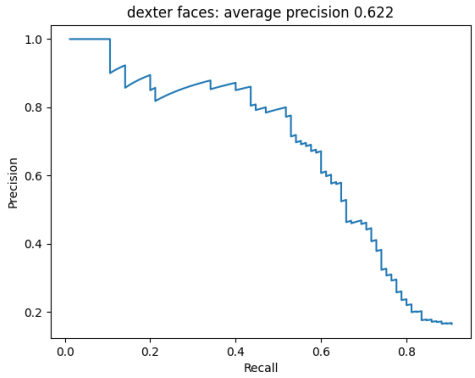
Using the approach described above, we obtained the following average precisions:



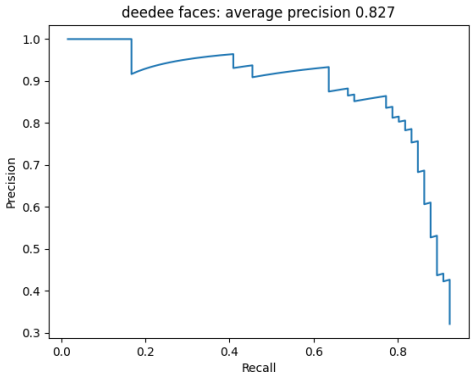
(a) Dad



(b) Mom



(c) Dexter



(d) DeeDee

Figure 3: Average precision for each character

6 Bonus Task - YOLO

For the bonus task, we chose to use the YOLOv8 model.

The way it was used is as follows:

- processing the training data to match the required format of the model
- training the model with our training data
- taking the obtained model and testing it on images
- processing the output and converting it back to our format

6.1 Input Processing

In the model training part, we need:

- files organized in the */datasets/train* folder, which contains the */images* and */labels* folders

- a .yaml file where we specify the paths to the training, validation, and testing folders, as well as the number of classes and their labels

The annotations must also respect the format imposed by the model, in the form: *class_id x_center y_center width height*, normalized data, compared to the previous

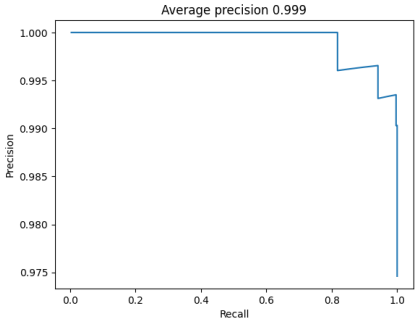
annotation format, where we used the character's name and the coordinates x_min y_min x_max y_max.

6.2 Output Processing

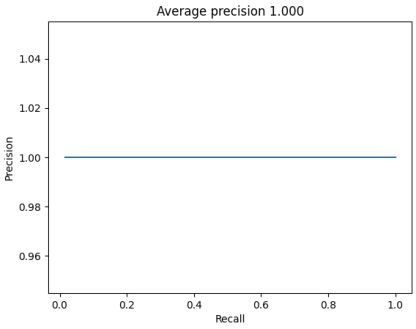
After obtaining the predictions, they must be processed again, in reverse from the input, to return to the initial format. The obtained annotations are in the */labels* folder in the specified folder. Finally, we can save the obtained predictions as in the other tasks.

6.3 Results

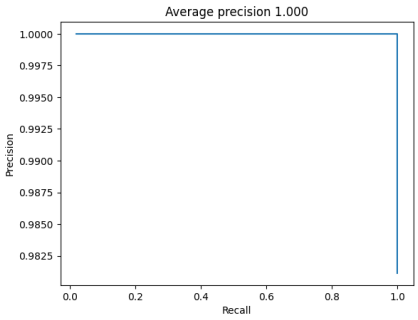
The results obtained by this model are as follows:



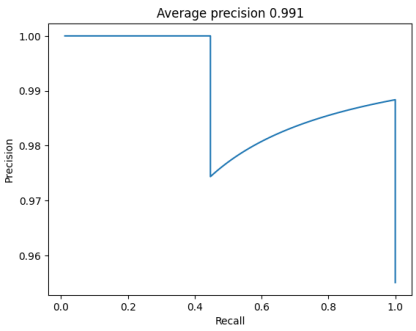
(a) Detectare de fete



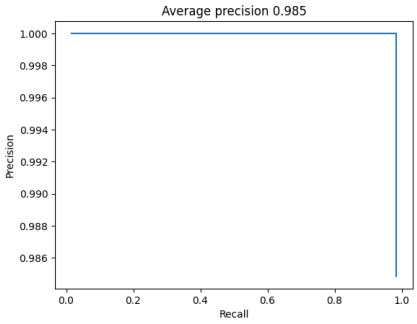
(b) Dad



(c) Mom



(d) Dexter



(e) DeeDee

Figure 4: Precizii medii YOLO