Facial Detection and Recognition of Characters from the Animated Series Dexter's Laboratory

A. Introduction

Objective

The purpose of this project is to implement an automated system for detecting and recognizing the faces of characters from the animated series *Dexter's Laboratory*. This will be achieved using Computer Vision algorithms discussed during lectures and partially implemented in laboratory sessions.

Dexter's Laboratory

Dexter's Laboratory is an animated series created by Genndy Tartakovsky and produced by Cartoon Network. It premiered in 1996. The show follows the adventures of a boy-genius named Dexter, who has a secret, ultra-high-tech laboratory hidden behind the bookshelf in his room. The main characters are Dexter, his sister, Deedee, his mom and his dad.

B. Tasks

Task 1

The first problem to solve is the detection of faces for all characters appearing in the images. For each input image, your algorithm must return a set of detections (rectangular bounding boxes and confidence scores) that localize all the faces in the image. Figure 1 illustrates some examples from the training dataset along with their corresponding annotations, which consist of red rectangular bounding boxes perfectly enclosing each face.

Task 2

The second problem involves facial recognition for specific characters only. Alongside Dexter, the main character of the series, there are three other prominent characters: DeeDee (Dexter's sister), Mom (Dexter's mother), and Dad (Dexter's father). Facial recognition will be limited to these four characters: Dexter, DeeDee, Mom, and Dad. For each input image, your algorithm must return a set of detections (character name, rectangular bounding box, and confidence score) that localize the faces of the four characters of interest in the image. Figure 2 provides examples from the training dataset along with their corresponding annotations. These annotations consist of rectangular bounding boxes that perfectly enclose the faces of interest. Each detection is assigned a color specific to the character class:

- Blue for Dexter
- Yellow for DeeDee
- Green for Dad
- Violet for Mom

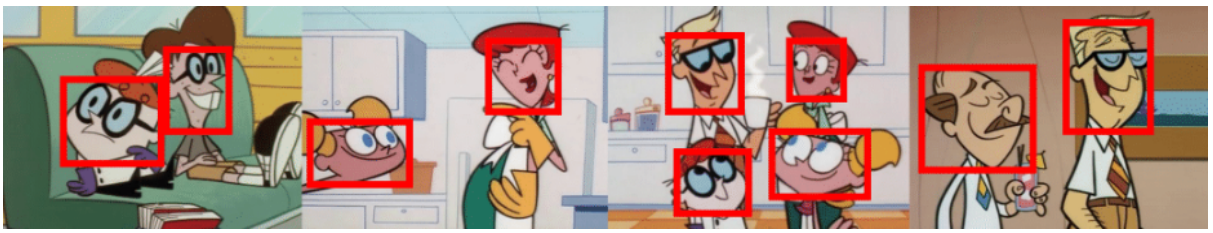


Figure 1. Facial Detection of Characters from Dexter's Laboratory

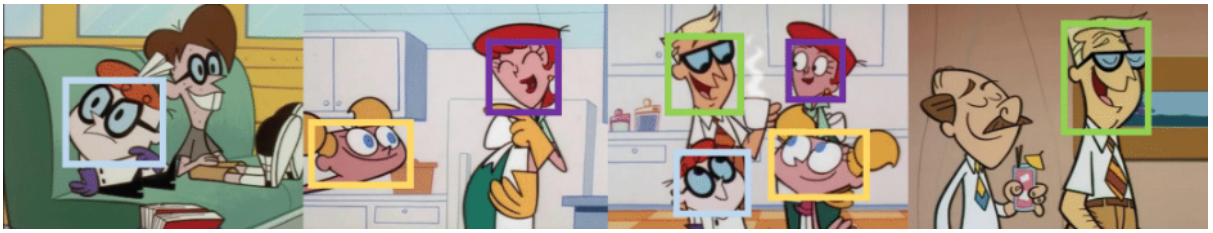


Figure 2. Facial Recognition of Characters from Dexter's Laboratory:
Each face of interest is annotated with a rectangular bounding box in a color specific to the character class: blue for Dexter, yellow for DeeDee, green for Dad, and violet for Mom.

C. Tasks 1

An initial approach to this task was to use the sliding-window paradigm with HOG (Histogram of Oriented Gradients), which was then classified using a simple SVM. With this method, I was able to achieve scores around 0.4. These scores prompted me to try a completely different approach. Instead of improving the classifier or the HOG features, I decided to use classification based on a CNN (Convolutional Neural Network). Using a CNN simplified the work a lot because it extracted features and classified photos at the same time. This approach, even with a simplistic model, achieved scores above 0.8 on the first attempt. For this to work, I had to code some functions to have every element ready to use.

1. Training data

I have 2 functions that generate me the training data. The first one saves every positive detection from the annotations folder in a special folder only with faces. Those pictures all have 64x64 pixels so I can further use them in the CNN. This step was done using cv.resize when reading all the pictures. The second function generates negative examples from the pictures provided. In order to have random examples I cropped 13 pictures of different dimensions from each initial photo, each with random coordinates, then resized them to 64x64 pixels. To assure that the negative examples are not actually faces from the photos I used an intersection_over_union function with the random coordinates and each pair of coordinates of faces from that picture. These 2 functions generated the training data for the CNN. For all the training data, I used also their flipped image to have more examples for a better classification.

2. CNN architecture

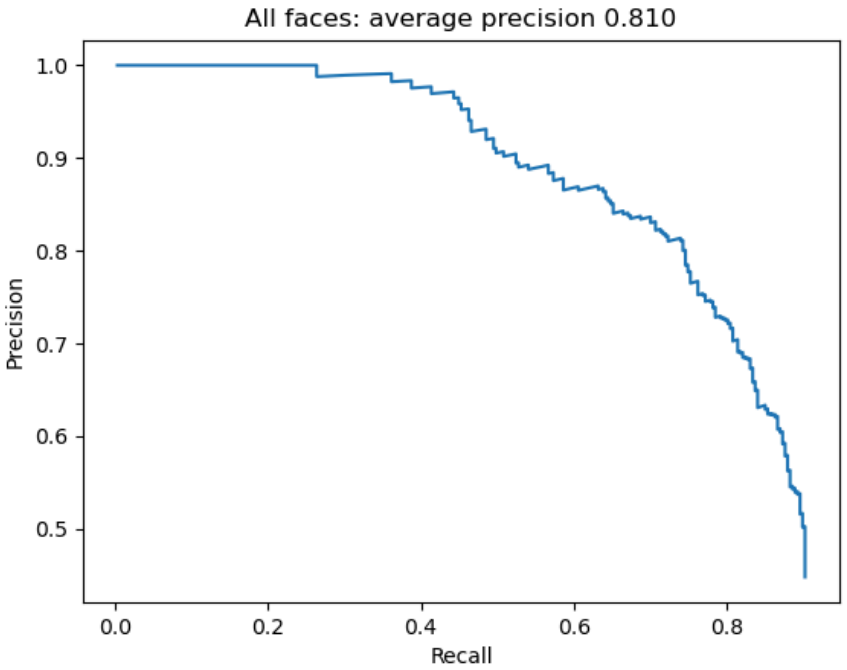
I used a basic CNN architecture designed for image classification tasks:

- **Convolutional Layers:** These layers automatically learn to detect patterns in images (like edges, textures, and shapes) by applying filters (kernels) that slide over the input image. Each convolutional layer extracts more complex features as the data passes through the network.
- **MaxPooling:** After each convolutional layer, a pooling layer is applied to reduce the size of the feature maps, which helps in making the model computationally more efficient and less prone to overfitting. It essentially retains only the most important information.
- **Fully Connected Layers:** Once the convolutional and pooling layers have processed the image, the data is "flattened" into a single vector and passed through fully connected layers, which help the model make final decisions based on the learned features.
- **Activation Functions (ReLU):** These functions introduce non-linearity, allowing the network to learn more complex patterns beyond simple linear relationships.

I used a simple architecture using only these concepts to train a model for 30 epochs, which overfits the model but works just fine. I saved the model so I don't have to train it every time for the detection.

3. Facial detection

Having the model trained, the only thing left to do is to have cropped pictures to classify using the model. In the run function I load the model, then for each test photo I crop a lot of pieces to classify. For that I use the paradigm of sliding_window on many dimensions. Starting from a scale of 0.7 to a scale of 0.1, I multiply the original photo's dimensions with 0.96 each step. On each intermediate picture I use a sliding window and classify each of them (resized to 64x64) with the trained model. Each crop gets a score from the model, which is the difference between the 2 output neurons. Crops closer to faces get higher scores, and in order not to have too many guesses stacked one on top of the other I use a non-maximal suppression function to get only the best guess out of the stacked ones. After this I use a threshold (5, this works good for my model) to detect the faces, not parts of the background. After this I just save the detections for each photo.



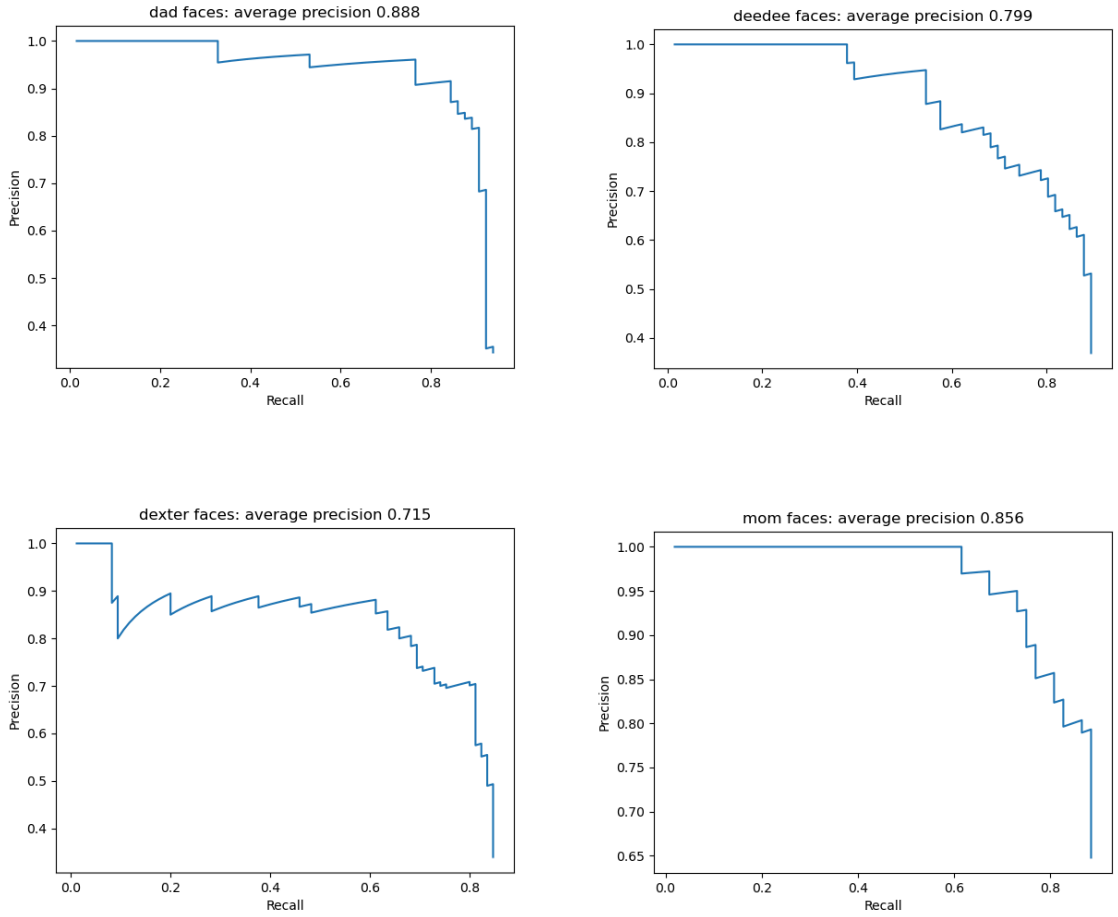
Average precision for task 1 on validation data

D. Tasks 2

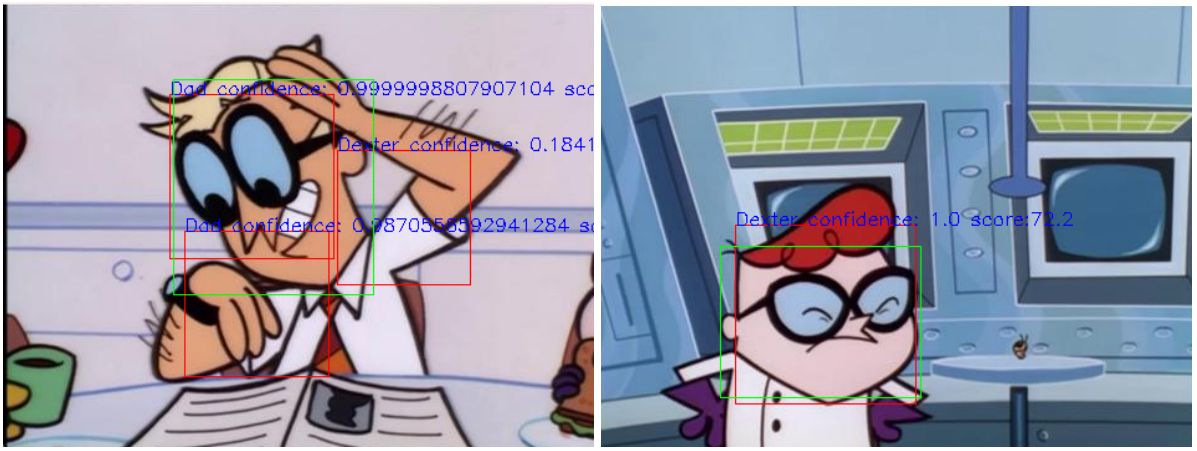
The approach for this task is very similar to the first one. I used a CNN architecture as well, the only difference was the training data, which had 4 characters instead of every face, and other negative examples, which were the same as for the first task.

The new CNN architecture (which is the same actually) was made using the same concepts as the previous one but was made to classify between 5 classes, one for each main character and one for other parts of the pictures. I also trained this model for 30 epochs to get a good result.

I loaded this model in the same function I loaded the previous model. For each detection made by the first one I used the recognition model to classify the detection and make it a recognition. Each recognition had a confidence score for each possible class, and I selected the biggest score. After this, I saved each detection with the most probable class for a recognition and its confidence score.



Average precision for each main character recognition



Detections and Recognitions on some photos

E. Running the code

Running the project does all the steps described previously. Generates or loads all the training data from the annotation folder, trains or loads the models, detects faces and then classifies all the detections, then saves all the results in numpy files for testing. A lot of improvements can be done, starting with a better architecture, better image processing,

improving the sliding_window to have more precise examples to classify and even adding more training data.