Dexter Lab Facial Detection and Recognition

Anamaria Hodivoianu

January 21, 2025

1 Introduction

This project consists of detecting and recognizing faces from images from the Dexter Lab cartoon. The first task is to detect all the faces from an image, while the second taskt is to detect the faces of the four main characters: Dexter, Deedee, Dad and Mom. The solution must be based on the sliding window approach.

2 Approach

In my solution, I decided to train 5 different classifiers, one for the first task and 4 for the second task (one for each character). The classifiers were trained on positive and negative examples.

3 Sliding Window

In order to determine the optimal aspect ratio of the sliding window, I calculated the average aspect ratio of the annotated bounding boxes. The average aspect ratio of all the images is 1.14, so I decided to use a square sliding window of size 36x36. The sliding window dimensions for the second task are as follows: 42x36 for Dad, 36x54 for Deedee, 36x42 for Dexter and 36x36 for Mom (height x width).
The sliding window stride is 6 pixels.
In order to be able to detect faces of different sizes, I resize the images and run the sliding window on all the resized images. The resizing factors are 1, 0.75, 0.5, 0.25, 0.125 and 0.1.
After running the sliding window on all the resized images, I eliminate the overlapping detections using non-maximum suppression with a 0.3 threshold.

4 Positive Examples

Getting the positive examples on which to train a classifier is the first step. The images in the train folder have annotations as bounding boxes of varying sizes and aspect ratios. In order to bring them to the same aspect ratio, I increased the annotated bounding box to match the desired aspect ratio instead of directly resizing the patch as it would have distorted the face. I also flipped all the examples horizontally to increase the number of positive examples.

5 Negative Examples

The negative examples are generated by randomly selecting patches with the same aspect ratio as the positive examples from the train folder. In order for the patches to be more diverse, I used 4 different sizes for the patches. To make sure the patches are negative examples, I check if the intersection over union with the annotated bounding boxes is less than a 0.2 threshold. For the facial detection task, I considered all the annotated bounding boxes, while for the facial recognition task, I considered only the annotated bounding boxes of the character I was training the classifier for.

6 Classifier

For the first task, I tried 3 different classifiers: SVC with HOG descriptors, neural network with HOG descriptors and convolutional neural network. The best results were obtained with the CNN. For the second task, I only tried the CNN classifier.

6.1 SVC with HOG descriptors

The first classifier I tried is an SVC that takes as input the HOG descriptors of the patches. I calculated the HOG descriptors of the positive and negative examples using the hog function from skimage with the following parameters: orientations=9, pixels_per_cell=(6, 6), cells_per_block=(2, 2). I trained a linear SVC on the HOG descriptors and then tested it on the images from the validation folder. The results are presented in Figure 1. The results can probably be improved by tuning the parameters of the HOG function and the SVC and by generating better negative examples.

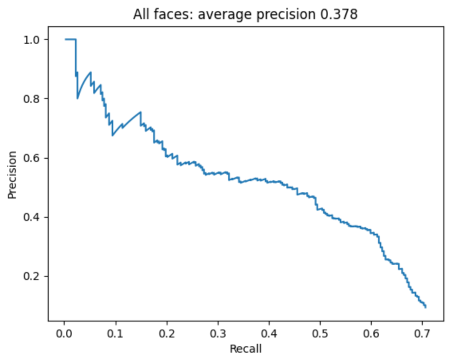

Figure 1: Average precision using SVC with HOG descriptors

6.2 Neural Network with HOG descriptors

The second classifier I tried is a neural network that takes as input the HOG descriptors of the patches. The architecture consists of two fully connected layers with 128 and 256 neurons, and an output layer with 1 neuron. The activation function is ReLU for the first two layers and sigmoid for the output layer. The loss function is binary crossentropy and the optimizer is Adam. The learning rate is 0.001 and the batch size is 32. I trained the neural network for 10 epochs. After training the classifier, I tested it on the images from the validation folder. The results are presented in Figure 2.

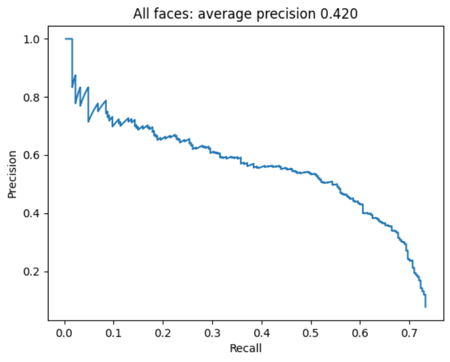

Figure 2: Average precision using neural network with HOG descriptors

6.3 Convolutional Neural Network

The CNN architecture consists of two convolutional layers, one with 32 filters and the other with 64 filters, each followed by a max pool layer, and two fully connected layers, and one output layer.

The activation function is ReLU, except for the last layer, where the activation function is sigmoid in order to output a probability (1 for face, 0 for non-face). The loss function is binary crossentropy and the optimizer is Adam. The learning rate is 0.001 and the batch size is 32. I trained each CNN for 10 epochs.
Because the input size differs for each character, I had to calculate the shape after the convolutional layers in order to determine the input size for the first fully connected layer.

```
class ConvolutionalNeuralNetwork(nn.Module):
def __init__(self, input_size):
    super(ConvolutionalNeuralNetwork, self).__init__()
    self.conv1 = nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1)
    self.conv2 = nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1)
    self.pool = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)
    self.shape_after_conv = self.get_shape_after_conv(input_size)
    self.fc1 = nn.Linear(self.shape_after_conv, 128)
    self.fc2 = nn.Linear(128, 64)
    self.fc3 = nn.Linear(64, 1)

def forward(self, x):
    x = self.pool(F.relu(self.conv1(x)))
    x = self.pool(F.relu(self.conv2(x)))
    x = x.reshape(-1, self.shape_after_conv)
    x = F.relu(self.fc1(x))
    x = F.relu(self.fc2(x))
    x = F.sigmoid(self.fc3(x))
    return x

def get_shape_after_conv(self, input_size):
    dummy_input = torch.zeros(1, 3, input_size[0], input_size[1])
    dummy_output = self.pool(F.relu(self.conv1(dummy_input)))
    dummy_output = self.pool(F.relu(self.conv2(dummy_output)))
    return dummy_output.numel()
```

After training the classifiers, I tested them on the images from the validation folder. The results are presented in Figure 3.

7 Bonus Task: YOLO

For the bonus task, I fine-tuned a YOLO model (yolov9c) on the Dexter Lab images. I trained two models, one for the first task and one for the second task. I had to create a dataset and configuration file for each model, as YOLO requires a specific format for the annotations (for each image, a txt file with the following format for each annotation: object-class x-center y-center width height, all normalized to [0, 1]). I trained the facial detection model for 10 epochs and the facial recognition model for 20 epochs (note: do not try to fine-tune a YOLO model on your laptop; use GPU from Kaggle instead). The results are presented in Figure 4.

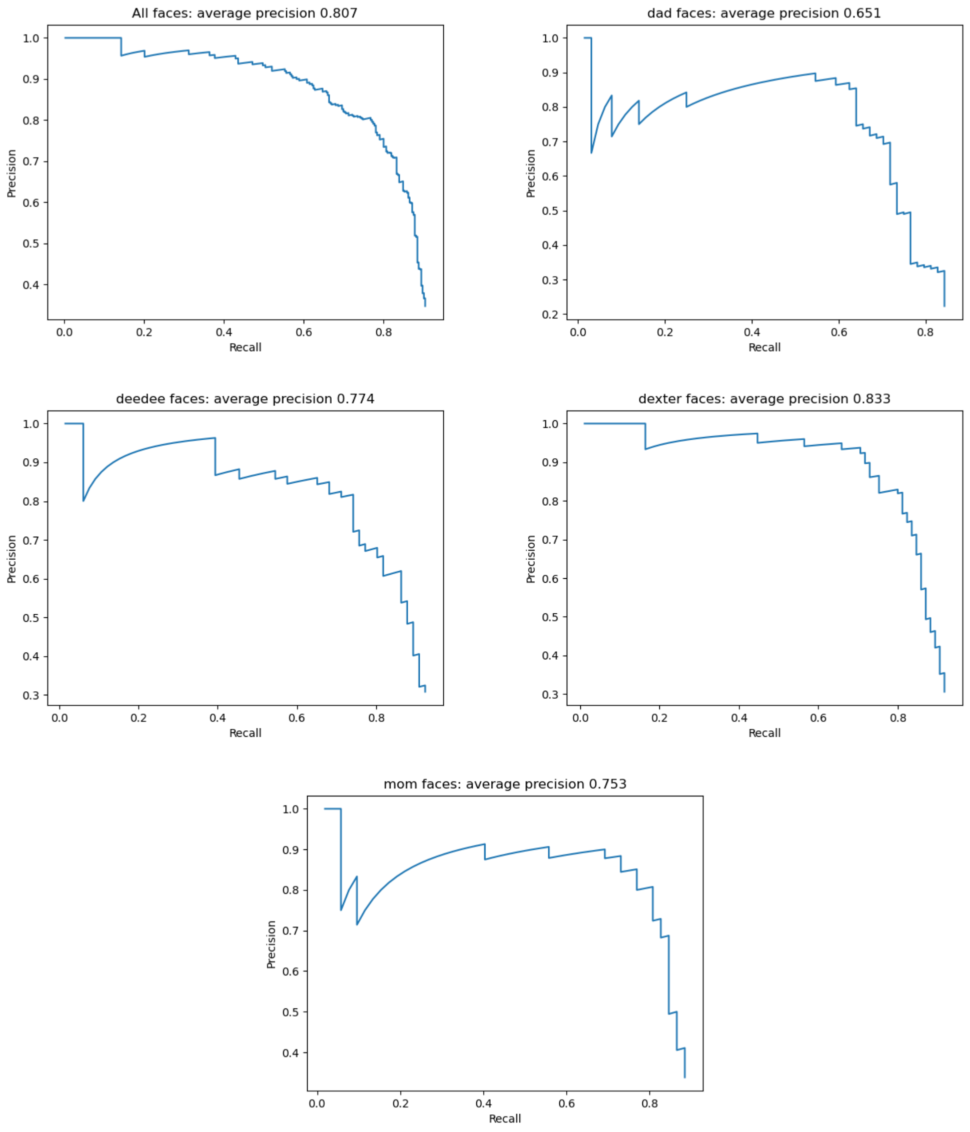

Figure 3: Average precision using CNN

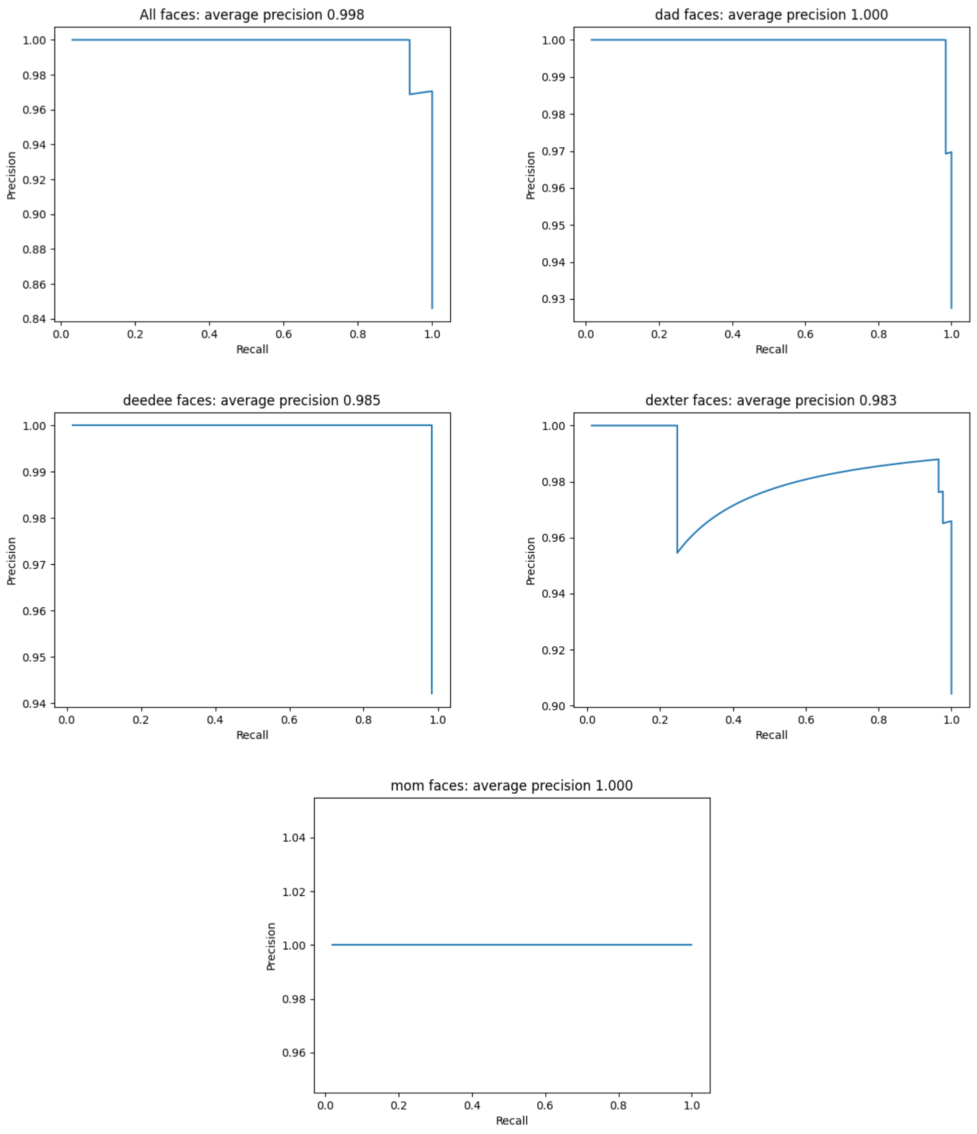

Figure 4: Average precision using YOLO