Computer Vision - Simpsons Family - Character Recognition

ANDREI-CRISTIAN RUSU

January 7, 2022

1 Introduction

In this document you will find implementation details about the Simpsons Family Character Recognition application.

2 Reading the training data

For training, we have the following directory structure:

```
data_task1
    [positive/negative] // type of image
        [character name]_[index].jpg // image
data_task2
    [bart/homer/lisa/marge/unknown] // character name
        [character name]_[index].jpg // image
```

The fact that we're duplicating the images is not the optimal way to do things. The only reason we do this is to simplify things with the *PyTorch data loader*. This will be updated in the future.

3 Generating the data

The generators for positive / negative images can be found in *ExamplesGenerator.py*. It contains two functions:

- *generate_positive_examples*()
- *generate_negative_examples*(*count_per_image*)

The first function:

- iterates through every image in the train directory
- finds the corresponding box for each detection
- crops that box
- resizes it to $(image\_height, image\_width)$ (i.e. to $(72, 72)$)
- saves it to the corresponding directory

The second one:

- iterates through every image in the train directory
- finds the corresponding boxes for each detection

- for *count_per_image* times, it tries to crop random patches which do not intersect with the real detections *or* have at most 50% yellow pixels.
- for the other third of *count_per_image*, it crops random patches that might or might not intersect with the real detection, but that contain at least 1% and at most 3% yellow pixels
- saves each patch to the corresponding directory

4 Training the Neural Networks

We mainly have two tasks:

- Given a image of $n * m$ size, check if there is a face of a character.
- Given a image of $n * m$ size which contains a character, determine the name of the character.

In order to solve this, we will train two neural networks:

The first CNN will be trained on the data from *data_task*1 directory - and will learn how a *positive* image looks like (an image which contains a character).

The second CNN will be trained on data from *data_task*2 - learning the name of the characters.

For both of our networks, we'll be using *resnet18*, with a *SGD* optimizer and *Early Stopping ReduceLROnPlateau*. The implementation resides in *NeuralNetwork.py*.

The hyperparameters that we chose are the following:

- batch_size: 64
- momentum: 0.9
- learning_rate: $1e-2$
- early_stopping_patience: 3
- reduce_lr_on_plateau_patience: 1

The number of epochs is *at most* 50.

When training the network, we are *augmenting* the images, using the following transformations:

```
transform: Compose = Compose([
    Resize((image_height, image_width)),

    RandomHorizontalFlip(),
    RandomRotation(degrees=10),
    RandomAffine(degrees=0.2),

    ToTensor(),
])
```

Here are some logs from the training session:

```
2022-01-07 07:38:46,401/INFO: Training CNN #1...
2022-01-07 07:39:29,345/INFO: Epoch 1 | loss: 0.09360 | acc: 0.96752 | val_loss: 0.05067 | val_acc: 0.98376
2022-01-07 07:40:10,040/INFO: Epoch 2 | loss: 0.04641 | acc: 0.98489 | val_loss: 0.03776 | val_acc: 0.98790
2022-01-07 07:40:51,098/INFO: Epoch 3 | loss: 0.04118 | acc: 0.98641 | val_loss: 0.03156 | val_acc: 0.98986
```

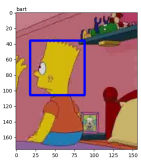
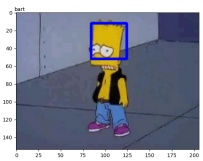
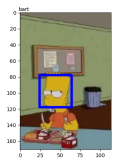
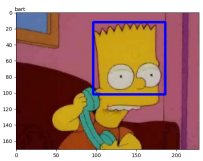
```
2022-01-07 07:41:32,412/INFO: Epoch 4 | loss: 0.03114 | acc: 0.98997 | val_loss: 0.03196 | val_acc: 0.98923
EarlyStopping counter: 1 out of 3
2022-01-07 07:41:32,476/INFO: Done.
2022-01-07 07:41:32,476/INFO: Training CNN #2...
2022-01-07 07:41:48,172/INFO: Epoch 1 | loss: 1.05340 | acc: 0.59094 | val_loss: 0.55163 | val_acc: 0.81775
2022-01-07 07:42:03,595/INFO: Epoch 2 | loss: 0.42271 | acc: 0.85790 | val_loss: 0.33255 | val_acc: 0.89311
2022-01-07 07:42:18,857/INFO: Epoch 3 | loss: 0.28836 | acc: 0.90759 | val_loss: 0.23276 | val_acc: 0.92629
2022-01-07 07:42:34,724/INFO: Epoch 4 | loss: 0.22213 | acc: 0.92978 | val_loss: 0.17778 | val_acc: 0.94811
2022-01-07 07:42:50,113/INFO: Epoch 5 | loss: 0.17843 | acc: 0.94591 | val_loss: 0.14874 | val_acc: 0.95728
2022-01-07 07:43:05,548/INFO: Epoch 6 | loss: 0.16297 | acc: 0.95160 | val_loss: 0.11749 | val_acc: 0.96700
2022-01-07 07:43:21,109/INFO: Epoch 7 | loss: 0.11551 | acc: 0.96956 | val_loss: 0.09829 | val_acc: 0.97158
2022-01-07 07:43:36,593/INFO: Epoch 8 | loss: 0.09721 | acc: 0.97140 | val_loss: 0.07810 | val_acc: 0.97910
2022-01-07 07:43:36,654/INFO: Done.
```

5 Recognizing faces

For this, we're using the $run_face_detector function from FaceDetector.py$.

The logic behind this function is the following:

- *iterate through every image in the test directory*
- *for every image, create an array of detections and scores*
- *iterate through a number of different $n * n$ window_sizes*
- *for each window, apply the sliding window technique, and get the (x, y) coordinates of the top left corner*
- *if we can't create a $(window\_size, window\_size)$ window, continue*
- *get the percentage of yellow pixels from the window, and, if it's not in $[0.3, 0.7]$, continue*
- *run the detection CNN on the window, and get the score class*
- *if the window is a positive detection, add the coordinates of the window the score to the arrays*
- *after iterating with each window, with all window sizes, we continue if there were no detections. otherwise,*
- *we apply non maximum suppresion over all detections, using the associated scores, and, for each detection,*
- *we run the second CNN on the corresponding window and get the name of the character that is in the image*
- *we add all the needed data to the corresponding arrays, and, at the end, return the filenames of the images, with the detections, scores and names.*

6 End notes

All the hyper parameters that were used in the application were either tested manually, or via a Grid Search technique.

Everything can be found in the implementation.