Computer Vision Project Code Documentation

Prudhvi Raj Dachapally

Module 1: Representational Autoencoder Units

1.1. Training the Autoencoder Network:

1.1.1. Shallow Model:

To train the shallow model, first, the images should be resized to a particular size, 64 x 64 in this case. We already provided the data in the folder *resized_JAFFE_data_64_by_64*. All the files should contain the emotion shortcut in their filename. For example, anger is represented by KA.**AN**2.40.

To train the shallow model with 300/500 units and some n-number of iterations, the following parameters should be passed to the program *one_layer_autoencoder_train.py*.

Arguments:
First: Emotion - [AN, SA, SU, NE, FE, DI, NE]
Second: Number of Iterations
Third: Number of Hidden Units - 300 (or) 500

Example: python one_layer_autoencoder_train.py AN 100 300 trains the construct for Anger in a 300-dimensional space and runs for 100 epochs.

Once all the representations for the emotions are computed, we proceed to the appender step.

1.1.2. Dense Model:

The same first step for data preprocessing is done.

To train the dense model with 300/500 units and some n-number of iterations, the following parameters should be passed to the program *two_layer_autoencoder_train.py*.

Arguments:
First: Emotion - [AN, SA, SU, NE, FE, DI, NE]
Second: Number of Iterations
Third: Number of Hidden Units - 300 (or) 500

Example: python *two_layer_autoencoder_train.py* AN 100 300
-> trains the construct for Anger in a 300-dimensional space and runs for 100 epochs.

Once all the representations for the emotions are computed, we proceed to the appender step.

1.2. Appending Learned Representations:

This step should be done before testing the model. This step, briefly put, takes all the learned representations and saves them in a pickle dictionary to make things easier while testing.

To run this method, the program named *autoencoder_train_appender.py* is used.

Arguments:

first: Number of layers (one/two)
second: Number of hidden units (300/500)

For example, in the above step, if we learned a shallow model with 500 hidden units, we combine all of them; we execute this program in the following way.

Example: python *autoencoder_train_appender.py* one 500
-> Combines all the learned 500 unit representations into one dictionary.

The argument value of *one* is used for the shallow network, while *two* is used for the dense network.

1.3. Testing the Autoencoder Network:

The autoencoder networks can be tested on two different datasets, namely JAFFE test set, and the LFW test set.

1.3.1. Shallow Model:

To use the test data on the shallow model, the program *one_layer_autoencoder_test.py* is used.

This takes three arguments.
first: dataset (jaffe for JAFFE test set, and lfw for LFW data set)
second: Number of Hidden Layers (300/500)
third: Number of iterations for each example (100 was giving the best results)

Once it takes all the images, it converts them into 300/500 dimensional space and uses cosine distance as a metric to find the distance between two emotions.

Example: To test on the JAFFE data set with 500 unit learned representations, the program should be executed as follows.

➔ python *one_layer_autoencoder_test.py* jaffe 500 100

1.3.2. Dense Model:

To use the test data on the dense model, the program *two_layer_autoencoder_test.py* is used.

This takes three arguments.
first: dataset (jaffe for JAFFE test set, and lfw for LFW data set)
second: Number of Hidden Layers (300/500)
third: Number of iterations for each example (100 was giving the best results)

Once it takes all the images, it converts them into 300/500 dimensional space and uses cosine distance as a metric to find the distance between two emotions.

Example: To test on the LFW data set with 300 unit learned representations, the program should be executed as follows.

➔ python *two_layer_autoencoder_test.py* lfw 300 100

Module 2: Convolutional Neural Network

2. Data Augmentation

Since we need more data to make our CNN model work better, we augment our data by capturing smaller patches from the original image.

The program *augment_jaffe_data.py* is used to do this.

Arguments:

First: train_files.txt (or) test_files.txt

2.2. **Training the CNN**

Training the CNN is fairly simple. Since model is already built, it takes the train data, and further splits it into 80 and 20 percentage, where 20% of the data is used for validation.

Each epoch takes 75 seconds (approximately) on a CPU. We trained this model for 420 iterations, 20 at a time. The best validation accuracy was after 360 iterations, therefore that was the model we used for testing, and that model was included in the folder.

Arguments:

first : Number of epochs

Example:

- ➔ python CNN_train.py 10
  - This trains the model for 10 epochs and saves the model file.

2.3. Testing the CNN Model

The program *CNN_Jaffe_test.py* is used for testing the JAFFE test data.

The program loads the trained CNN model and tests it on the JAFFE test set. This should print the number of correct predictions, total number of test files, the accuracy percentage, and a confusion matrix.

The program *CNN_lfw_test.py* is used for testing the LFW test set.

This does the same as the jaffe_test program, but addition to the result, this program also creates a file called *predictions_lfw.txt*, which consists of the image name followed by the model's top three predictions.

2.4. Single Image Prediction

If you want to get a prediction of the emotion on a single image, the program *CNN_single_image_test.py* can be used.

Arguments:

First: Image file.

Before running the code, please make sure that your machine has Keras running with Theano or Tensorflow backend.