MSc - Media and Textmining

Homework results

Daniel Mark Kiss

2023

Contents

Chapter 1

Task description

The data set contains small black and white images labelled with character identifier. The task is to recognize the character (0-9, a-z, A-Z) of unknown images in the test set by deep learning classifier without any human activity (human's help is equivalent to cheating). Work out a deep learning classifier model, and train this model by the training dataset! Use cross-validation in order to measure the goodness indicators (accuracy, AUC) of your model! Please pay attention to the parameter optimization! Please investigate the model, the result and describe the details of your solution!

Chapter 2

Ellaborated work

In the chapter I will present the ellaborated work that I have done. For my work I have created separate classes and files for the different subtasks of the Homework. In every section I will present and explain what the different classses are used for.

2.1 program.py

Program.py is the core of the program. It is the main file that is used to run the program. It is responsible for the following tasks:

1. Initializing the FileReader class
2. Running the FileReader class functions
   - read_training_files()
   - create_train_set()
3. Initializing the ConvolutionalNeuralNetwork class
4. Running the ConvolutionalNeuralNetwork class functions
   - split()
   - compile()
   - train()
   - evaluate()
5. Reading the test files
6. Creating the test set

7. Exporting results to output.txt file

When we would like to run the program we should also import the FileReader and ConvolutionalNeuralNetwork classes. For running we should type python program.py to the terminal and the program should start running.

2.2 cnn.py

This file contains the fundamental code for the neural network. I used many other libraries for my homework like *numpy*, *tensorflow*, and *sklearn* also.

2.2.1 init

In the __init__ function I initialize the arrays which I used for traing and spliting. These are the different variables that I used in my class:

- self.images: contains the raw images
- self.label: contains the labels for the images
- self.img_train: contains the train set of images
- self.img_test: contains the test set of images.
- self.label_train: contains the labels of the train set
- self.label_test: contains the labels of the test set
- self.model: is a model used from tensorflow.keras.It has five layers, three 2D convolutional layers and two 2D MaxPooling layers. The first convolutional layer has 32 filters/kernels. Each filter is 3x3 matrix. I choose ReLU as the activation function. The input shape is 128x128, 1 which is set because the given images are this size and they are grey scaled. The MaxPooling layers are used for down-sampling and reducing the spatial dimensions of the input. It uses a 2x2 pooling window/filter. It also helps reducing the chnace of overfitting.It only retains only the most important informaton from it.

After the Initialization of the model I add a Flatten layer to it.It is used for converting and outputing a 1D array. It is necessary before passing to a dense, fully connected layer. The next line adds a layer with 64 units (neurons) and a ReLU activation function. A dense layer means that each neuron in this layer is connected to every neuron in the previous layer. The last line in the init function adds another dense layer with 62 units, representing the output layer. The activation function used in the output layer is "softmax." Softmax is often used for multi-class classification problems. It

converts the raw output scores into probabilities, where each value represents the probability of the input belonging to a particular class. The sum of all probabilities for a given input is 1, making it suitable for classification tasks like this task also.

2.2.2 split

The split function is used for spliting the images and the labels into traing and testing set, with the test set is 33% of the input. I also add a random_state seed so it will result alawys the same output.

2.2.3 compile

The compile function is used for compiling the model. I used the Adam optimizer for the compilation. It is an adaptive learning rate optimization algorithm that's been designed specifically for training deep neural networks.

2.2.4 train

The train function is used for training the model. I used 15 epochs for the training. I also added a LearningRateScheduler to the training. It is used for changing the learning rate during the training. I used 64 as the batch size. The batch size is a number of samples processed before the model is updated. The validation split is set to 0.2. It means that 20% of the training data is used for validation.

2.2.5 evaluate

The evaluate function is used for evaluating the model. It returns the loss and the accuracy of the model.

2.2.6 predict

The predict function is used for predicting the labels of the test set. It returns the predicted labels. It only processes one image and gives a prediction for it. It is used for the output.txt file.

2.3 labeler.py

The labeler.py file is used for labeling the images. It is only a huge dictionary with the labels for the images.

2.4 filereader.py

The filereader.py file is used for reading the images and the labels from the given files.

2.4.1 read_training_files

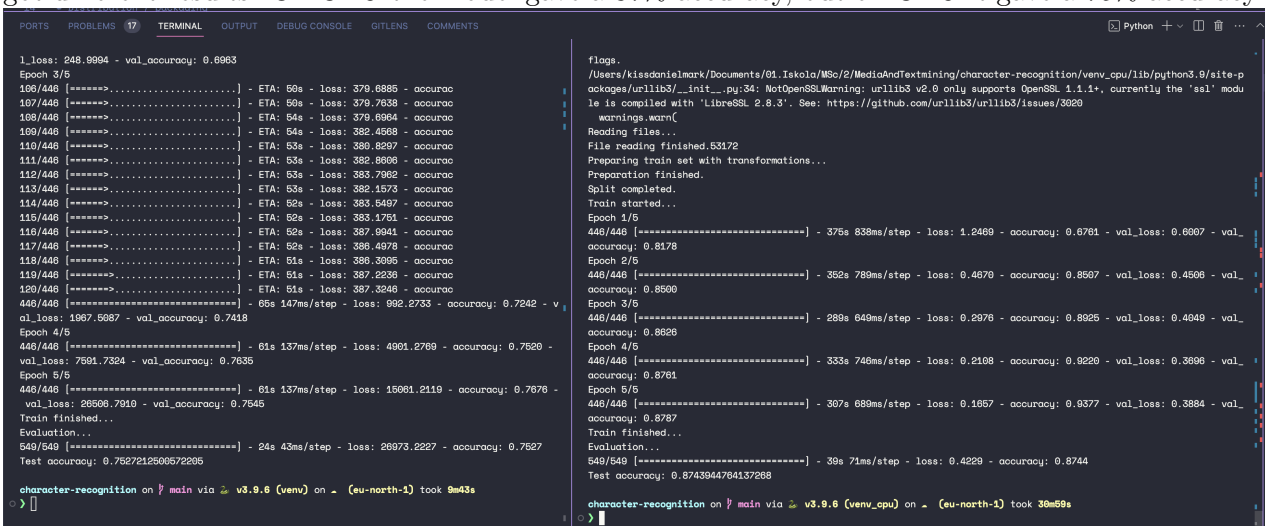
It reads the folders and images recursively. These functions also used for reading not only the training folders but the test folder as well.

2.4.2 create_train_set

For every image that has been read it is processed by a private function called _process_image(). These processed images are added to the images array and the labels are added to the labels array. The _process_image() function is used for creating the labels for the images extracted from the filename. Also the images are converted to numpy arrays. After that the function is normalizing the images.

2.5 Tensorflow using GPU

I used tensorflow-gpu for my homework. It is a version of tensorflow that uses the GPU for the calculations. I had an interesting problem with it.When I ran the same code on CPU and GPU I got different results. On CPU the model gave a 87% accuracy, but on GPU it gave a 75% accuracy.



I tried to solve this problem but I could not find the solution for it.

Chapter 3

Results and parameter optimization

In this chapter I will showcase the results of my homework and the parameter optimization that I have done. As I already mentioned I tried playing with the GPU execution of the CNN which resulted in a lower accuracy but it gave me a faster execution time. On CPU one epoch calculation took nearly 3 minutes in contrast on GPU it only took 40 sec. On the other hand I tried to play with the batch size and the epochs.

For the first run on the CPU it gave me a 87% accuracy with 5 epochs and 64 batch size. After this I started the transition to GPU. On GPU I tried to run the same code with the same parameters but it gave me a 75% accuracy. I tried to play with the batch size and the epochs. The third try I have changed and increased the epoch size from 5 to 10. It resulted an increase in the accuracy to 80%. The fourth try I have changed the epoch size from 10 to 15. It resulted an increase in the accuracy to 81%. The next I tried adding a LearningRateScheduler to the training. It resulted an increase in the accuracy to 83,6%. I used a logic where if the epoch is smaller than 5 then it uses the normal learning rate and after that using a lr * tf.math.exp(-0.1). I also tried using differnt learning rate. Increasing it to 0,002 resulted in a 83,7% accuracy. Decreasing it to 0,0005 resulted in a 82,1% accuracy. An other bigger leap was when I adjusted the learning rate with treshold to 3 for the epoch and using a lr * tf.math.exp(-0.5) calculation. It resulted in a 84,2% accuracy. Decreasing the batch size to 32 resulted in a 84,7% accuracy. I also tried increasing the batch size to 128 but it resulted in only a 82,5% accuracy.