EPFL

May 22, 2020

Mini Project 1: Digit Classification

Vincent Bernaert, Sebastien Emery, Raphaël Reis Nunes

https://github.com/raphaelreis/EE559-DeepLearning-MiniProjects

1 Introduction

The objective of this mini-project is to implement different deep-learning network architectures for handwritten digit classification and to compare their performance. The input data consist of pairs of greyscale images of fixed size (2 x 14 x 14). The train and test datasets each contain 1000 pairs. Each element of a pair represents a digit from 0 to 9 and the goal is to predict for each pair whether the digit in the first image is lesser or equal to the digit in the second image. In this report, four different architectures which are convolutional neural networks (CNN) are presented and evaluated. We begin with a simple architecture and improve performance by adding weight sharing and auxiliary loss to more complex architectures. Data augmentation techniques are also used to improve performance.

2 Methods

2.1 Training and optimization

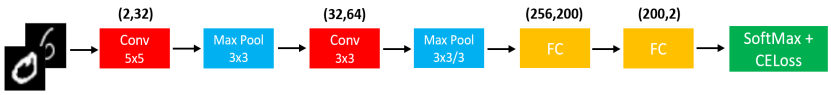
Training and testing data is taken from the MNIST database. Each pair sample is labeled with a boolean value: '0' if the first image is strictly greater than the latter and '1' otherwise. In addition,

each pair has a tag containing the class of each digit from 0 to 9. A total of 2000 pairs of images were generated: 1000 for training and 1000 for testing. Of the 1000 digit pairs contained in the train dataset, 200 randomly chosen pairs were kept to build a validation set. The remaining 800 were used to train the models. It can also be augmented by rotation (90,180 and 270 degrees), translation (one pixel upward,downward,to the left and to the right) and swapping the two channles for training. All models were trained on 40 epochs using Adaptive Moment Optimizer (Adam) as the optimizer with specific tuned learning rate for each model. In general, each model has two output values corresponding to the two possible outcomes. Softmax activation is applied to these two values and followed by calculation of the Cross Entropy Loss. The Cross Entropy Loss is optimized using mini-batch gradient descent with a batch size of 100. The train data is systematically shuffled prior to picking a new batch. Finally, to prevent overfitting, dropout is used and tuned for each model specifically.

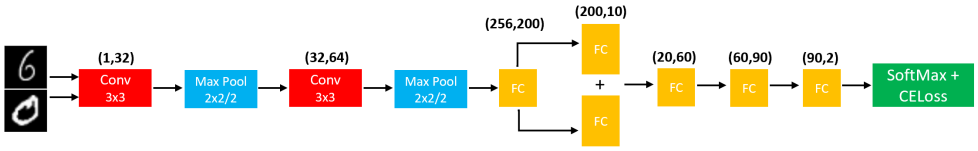
Model hyperparameters are tuned using a grid-search procedure. In brief, for each combination of hyperparameters, the model is trained and tested on the validation data for 10 selected seeds. Tuned hyperparameters include two dropout rates for the first and last fully connected layers, the learning rate and the number of units in the first and last fully connected layers (only implemented for the basic model). For each architecture, the combination of hyperparameters which gives the highest average validation accuracy are fixed and used to build the final model. Note that these are not global maximas since not all possible combinations have been tested. Optimal hyperparameters are shown in **Figure 1.** Due to its high computational cost, hyperparameter tuning was not performed for the GoogleNet architecture.

2.2 Network architectures

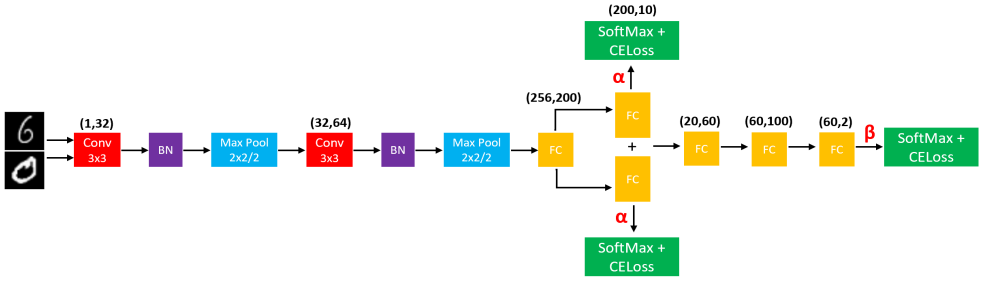
For all architectures described below, the Rectified Linear Unit (ReLU) activation functions was used. The first proposed architecture (**Figure 1.a**) is a simple convolutional neural network (CNN) with a single two-unit output corresponding to the two classes. It takes a pair of 14x14 images (2x14x14) and contains 53280 parameters. The second architecture takes a single image of size 14x14 (**Figure 1.b**) . In this weight sharing implementation, the two images in the pair are sequentially passed through two convolutional/max pool layers and a fully connected layer containing the same shared weights. Then, each image goes through a fully connected, reducing the number of units to 10. These units can be interpreted as the different classes of digits (from 0 to 9). In this part, the model learns two sets of weights (one for each image). These two 10-unit vectors are then concatenated into a 1x20 vector, which passes through 3 successive fully connected layers of shared weights resulting in the final output. This implementation has 63318 parameters in total. The third architecture (**Figure 1.c**) is almost identical to the previous one except that an auxiliary loss equal to the sum of the two losses computed on the outputs of the (200x10) fully connected layers is added to the final loss. Thus, the final aggregate loss is a weighted sum of three losses. Importantly, this aggregate loss is only minimized during model training. By minimizing this aggregate loss, the model is trained to simultaneously compare the images and to predict the class in each. Given that the task of recognizing the digits should help to compare them, a significant boost in performance is expected with this architecture compared to the Weight Sharing one. This implementation has 64100 parameters. In the last architecture, the first two convolutional layers of the Auxiliary architecture are replaced by an inception layer composed of four different filters based on the GoogleNet [1] architecture, allowing the model to extract more features simultaneously. To reduce the higher computational cost of this method, the convolutional filters in the inception layer are factorized as shown in **Figure 1.d**. This architecture contains 2128652 parameters.



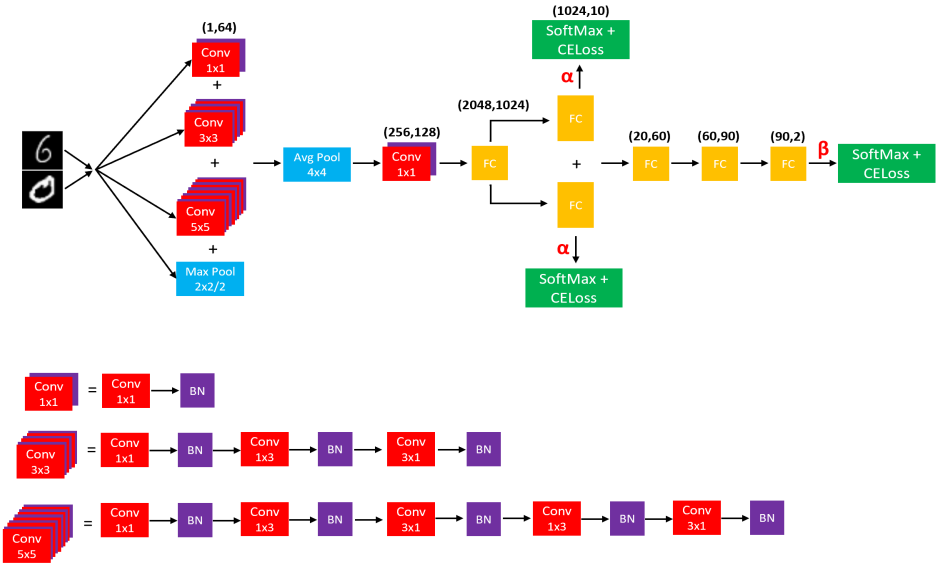
(a) Net2c - Basic CNN.



(b) LeNet-sharing - CNN with Weight Sharing.



(c) LeNet-sharing-aux - CNN with Weight Sharing and Auxiliary Loss.



(d) Google Net - CNN with Weight Sharing, Auxiliary Loss and an Inception block.

Figure 1: Network architectures
a/b/c = height/width/stride. A '+' indicates a concatenation of feature maps. Conv: Convolutional layer. FC: Fully connected layer. CELoss: Cross entropy loss. BN: Batch normalization. Changes in the number of channels are indicated in parentheses: (x,y) where x is the number of input channels and y is the number of output channels. In (a), the input size is (2x14x14). In (b), (c), and (d), the input size is (1x14x14). For all architectures, there are two units in the output layer. All parameter values shown are default values and may change after hyperparameter tuning. For simplicity, regularization procedures such as dropout regularization are not shown here. $\alpha = 0.5$ /$\beta = 1$: weights for each computed loss.

3 Results

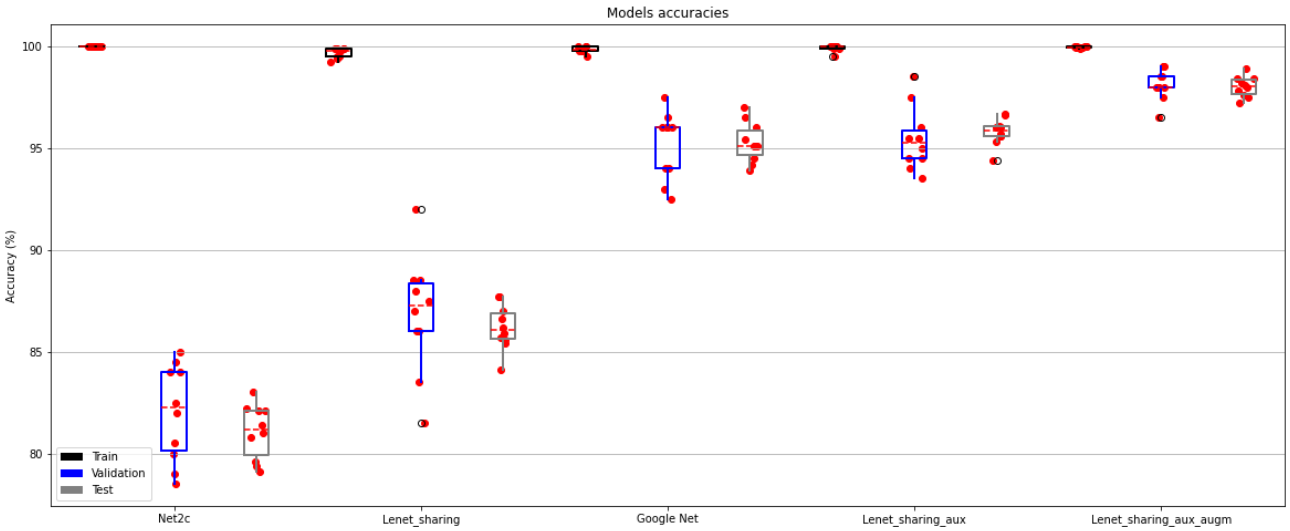


Figure 2: Model performance results. Performance is measured in terms of of the prediction accuracy on the train, validation and test data. Statistics are calculated based on 10 different random seeds using th optimal hyperparameters found during grid-search. Test accuracies : Net2c: mean = 81.7, std = 1.27. Lenet-sharing: mean = 86.2, std = 1.05. Lenet-sharing-aux: mean = 95.8, std = 0.62. GoogleNet: mean = 95.28, std = 0.93. Lenet-sharing-aux with data augmentation: mean = 98.01, std = 0.47.
Optimal hyperparameters: Lenet-sharing-aux: drop out rate in the first fully connected layer = 0.3; dropout rate in last fully connected layer = 0.1; ADAM learning rate = 0.01; number of units in the first fully connected layer = 200; number of hidden units in the last fully connected layer = 60. Lenet-sharing-aux with data augmentation: drop out rate in the first fully connected layer = 0.5; dropout rate in last fully connected layer = 0.1; ADAM learning rate = 0.001.

4 Discussion

The basic network architecture (Net2c) has the lowest test accuracy on the test data (81.7%), which was expected. Although it could be possible to observe a slight increase in performance by through regulatization techniques, this network is clearly limited by its simplistic architecture. With weight sharing alone, the performance is significantly improved (85.2%) as the parameters in the first layers are more efficiently learned on both images. Moreover, by splitting the input into two images and adding the two (200,10) fully connected layers, parameters for single digit classification task are also learned during training. Adding auxiliary loss to weight sharing during model training allows to reinforce the single digit classification task, resulting in a higher classification accuracy (95.8%). This result is in line with our initial idea that a model forced to classify the digits in each image through a separate loss dedicated to this subtask will perform better on the main classification task. The GoogleNet architecture has a similar accuracy (95.28%), suggesting that replacing the first two convolutional layers by an inception block does not improve performance. This may be because the GoogleNet network was not optimized through grid-search or simply that our architecture is too shallow for the inception block to have an significant effect. The data augmentation performed allows us to yield the best accuracy overall (98.01%) suggesting that our model is able to learn the digit in different configuration.

References

[1] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–9, 2015.