Deep Learning - Mini Project 1 Report

Ravinithesh Annapureddy, Anmol Prasad, Kushagra Shah
École Polytechnique Fédérale de Lausanne (EPFL), Switzerland

I. Introduction

A. Problem Statement

The objective of this project is to test different architectures to compare two digits in a two-channel image and predict if the first number is lesser than or equal to the second number. We also demonstrate the impact of weight sharing, and the use of an auxiliary loss to help in training. The only constraint of the project is that the solution should be implemented using only the PyTorch library.

B. Dataset

The input consists of a series of $2 \times 14 \times 14$ tensors, corresponding to pairs of 14×14 grayscale images, each containing a digit from the MNIST dataset. There are two kinds of outputs - first is the target indicating if the first digit is less than or equal to the second digit for each pair, and the second kind is the class number i.e. a pair of integer digits represented by the input image pairs. The training and testing data is extracted from a pre-built function provided by the instructors of the course [1]. Figure 1 summarizes the data dimensions and content.

Name	Tensor dimension	Type	Content
`train_input`	$N \times 2 \times 14 \times 14$	`float32`	Images
`train_target`	N	`int64`	Class to predict $\in \{0, 1\}$
`train_classes`	$N \times 2$	`int64`	Classes of the two digits $\in \{0, \ldots, 9\}$
`test_input`	$N \times 2 \times 14 \times 14$	`float32`	Images
`test_target`	N	`int64`	Class to predict $\in \{0, 1\}$
`test_classes`	$N \times 2$	`int64`	Classes of the two digits $\in \{0, \ldots, 9\}$

Fig. 1: The data available for training and testing [1]

C. Approach

We have developed three models to compare the effects of weight sharing and auxiliary loss.

- Fully-connected network - For this model, the spatial dimension of the image is neglected and it is flattened into a tensor.
- Simple convolutional network - This model considers the naïve image input as it is, with its spatial dimension.
- Residual convolutional network - This model is an extension to the simple convolutional network with more convolutional layers and uses skip connections (in the form of residual blocks).

II. Methods

In this section, we briefly describe the methods and concepts used in the experiments for this project [1].

A. Batch Normalization

Batch normalization shifts and rescales the input to a layer in the deep learning network according to the mean and variance estimated on the batch during the training. During the inference/test, batch normalization performs a component-wise affine transformation.

B. Data Normalization

The input data tensors are normalized by subtracting the mean and dividing by the standard deviation. The output target and classes do not require any pre-processing.

C. Loss Function: Cross Entropy Loss

The cross-entropy loss is used to train the networks due to its convexity when using the softmax function, and hence, local minima can be potentially avoided. Additionally, unlike the mean square error (MSE), it increases in an *anisotropic* way, thereby, not penalizing all the errors equally.

D. Stochastic Gradient Descent

We use Stochastic Gradient Descent (SGD) to update the parameters as it helps in evading local minima since the computation involves multiple samples at a time.

E. Max-pooling

Pooling, in general, is used to reduce a high dimensional signal into a low dimensional one. Max-pooling computes the maximum of values over non-overlapping blocks in the given input. Max-pooling layers are used in models with convolutions to reduce the size of the activation maps.

F. Weight Sharing

Weight sharing between different layers of the network/networks is used to share parameters and it reduces the number of trainable parameters of the model. In this project, weight sharing is achieved by using the same sub-network on the two images provided as input to the model. Hence, the same transformations will be applied to both the images. Conversely, when there is no weight sharing, the input images might undergo different transformations.

G. Auxiliary Loss

Auxiliary loss helps in propagating the gradient to the early layers of a deep network and partially mitigates the vanishing gradient problem. In this project, we first try to infer the classes and then predict the target. Thus, instead of calculating the loss using only the target values of the classification, we can *additionally* calculate the losses at the inference of both the inputs' class labels, and then use all the three losses together, as a weighted sum, to train the network.

H. Dropout

The dropout mechanism is used to avoid over-fitting. It *drops out* some units of the model randomly by setting them to zero during training. This forces the model to learn the representation with lesser information, hence, distributing the representation among different units. It must be noted that the units are *not dropped out* during testing and deployment of the model for better consistency in results.

I. Activation Function: ReLU

The Rectified Linear unit activation function is used in all the models, as the derivative of ReLU is non-vanishing.

J. Skip Connections

A skip connection, as the name suggests, adds a connection between two points in a network which skips any transformations on that input. Such a technique is useful in retaining the original properties of the input as well as the learned transformations.

III. Models

The models used in this project follow a general architecture as follows: Process the two input images separately and extract feature vectors representing the images, and then, use a concatenation of these feature vectors to predict the output target. Further, each model takes in two parameters which determine if weight sharing and auxiliary losses are to be used or not, allowing for a total of 4 combinations per model. If weight sharing is enabled, then the input images are transformed using the same weights. This section elaborates on the model architectures.

A. Fully Connected Model

The first model, referred as *fullyconnected* (Figure 2), serves as the baseline model and contains only linear layers. Each image is flattened to a tensor of length 196 and then passed through two hidden layers (of sizes 50 and 10) to output a feature vector of size 10. Finally, the two feature vectors for the two images are concatenated to form a tensor of length 20. This concatenated feature vector is passed through a hidden layer of the same size to output a tensor of size 2. These two values are interpreted as the probabilities for the input to belong to the two possible classes of output - true or false.

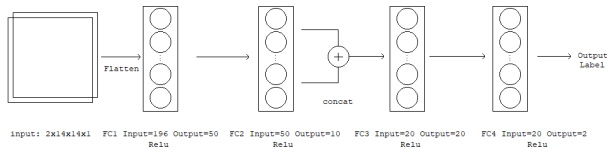

Fig. 2: Fully Connected Model

B. Simple Convolutional Model

The second model, referred as *simpleCNN* (Figure 3), uses convolutional layers instead of linear layers to transform the image. Each image is initially passed through 10 filters of size 3 with padding 1 and stride 1. These 10 activation maps are then passed through a set of another 10 filters with the same settings as earlier ones and the output is a $10 \times 6 \times 6$ tensor. This tensor is flattened and then passed through a linear layer to output a feature vector of size 10. Finally, the feature vectors are concatenated and processed similar to the previous model.

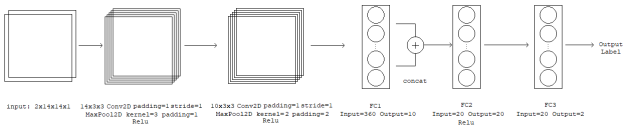

Fig. 3: Simple Convolutional Model

C. Residual Convolutional Model

The third model, referred as *skipCNN* (Figure 4), uses skip connections to learn a *residual* in the convolutional network. [1] Each image is passed through a set of 14 filters of kernel size 3 with no padding and stride 1 to output 14 activation maps of size 12×12. Then the 14 activation maps are passed through 6 residual blocks. For each residual block, the activation maps are passed through 14 filters of size 3 and padding 1, followed by batch normalization, ReLU activation and then again through 14 filters of size 3 and batch normalization. The input to the residual block is added to the output of batch normalization and later passed through ReLU activation. Then average pool filter of size 12 is applied to the output obtained after passing through the 6 residual blocks. The output of the average pool is then passed through a ReLU activation and flattened to a tensor of the length of 14. A linear layer is then used to reduce this flattened tensor to a feature vector of size 10. Finally, the feature vectors are concatenated and processed similar to the previous models.

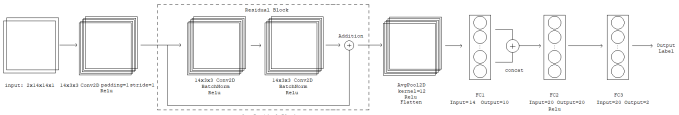

Fig. 4: Convolutional Model with the Residual Blocks

IV. Experiments

For the experiments, we train and test the aforementioned models with/without weight sharing and auxiliary loss. Hence, we explore 12 different scenarios, repeating each experiment 10 times to report the result statistics. It must be noted that the weights are re-initialized for each round. In this chapter, we describe the hyperparameters used across all the possible scenarios.

A. Input samples and Epochs

All the experiments are conducted using 1000 pairs of input images each, for training and testing. And each model is trained for 50 epochs.

B. Optimizer and Learning Rate

The models are trained using SGD with a learning rate of 0.1, which was empirically chosen through trial and error.

C. Auxiliary Loss

For training a network with an auxiliary loss, a weighted sum of losses was used as follows: Let L_1 denote the cross-entropy loss for the predicted and actual class of the image pair. Let L_2 and L_3 denote the cross-entropy loss between the feature vector and the actual digit class for each image in the pair. Then the auxiliary loss is defined as

$$(1 - \alpha) * L_1 + \alpha * (L_2 + L_3) \tag{1}$$

Here, the α parameter controls the weight to be assigned to each type of loss. An optimal value of $\alpha = 0.2$ was empirically chosen after some trial and error.

D. Training Parameters

As the architectures of the models are different, the number of parameters trained also vary. When the networks share the parameters, the number of trainable parameters are nearly half without sharing the weights (see Table I). Unlike the fully connected layers where each input has as own set of weights, in convolutional layers, the weights are shared across inputs (different from weight sharing across networks). Thus, the number of parameters in a convolutional layer is less than that of a fully connected network.

Model	**Parameters**	
	without Weight Sharing	Weight Sharing
Fully connected	21,182	10,822
Simple Convolutional	9,702	5,082
Residual Convolutional	44,386	22,424

TABLE I: Trainable parameters of each model

V. Results

A. Loss

The loss after each epoch for each of the 10 iterations, along with the average loss across all the iterations, is shown in the Appendix (see Figures 6 - 8 for the four possible cases). We observe that the loss decreases slowly without any weight sharing and without using auxiliary loss, whereas it decreases faster with weight sharing and an auxiliary loss. Among the different models we tested, the convolutional model with skip connections produces the least loss. We hypothesize that the performance improvement over the simple convolutional network is achieved because the gradient is propagated to inner layers more effectively due to the skip connections.

B. Errors

The resulting test errors are reported in Tables II, III and illustrated in Figure 5. We observe that using weight sharing and an auxiliary loss provides the best performance. We can also infer that weight sharing improves performance significantly, even without the use of auxiliary loss.

Weight sharing works better because both the input images are extracted from similar distributions. Thus, the digit identification of the pair can be done with the same transformations. This also makes the training consistent and stable since the model *sees* images from both the channels of the input. Weight sharing also simplifies and speeds up the training as the number of trainable parameters are reduced. Likewise, the networks trained with an auxiliary loss have lower test errors. This improvement is due to the fact that the error in predicting the digit is propagated, although not explicitly, through the network. Thus, the model learns better on digit prediction which, in turn, helps in the prediction of the smaller number.

Next, the residual convolutional model achieved the least test errors. Drawing on similar lines as above, the reason for the better performance is that the gradient is propagated more effectively. Lastly, the fully connected model has the worst performance among the three architectures. Unlike the fully connected model, the convolutional networks use the spatial information available in the image, thus, helping the model learn the local properties. Additionally, the number of parameters in the simple convolutional model is nearly half of the parameters in the fully connected model, thereby reducing the training effort - both in terms of computation time and memory.

Model	**Test Errors (without Weight Sharing)**	
	without Auxiliary Loss	Auxiliary Loss
Fully connected	19.70 ± 1.58%	16.98 ± 0.74%
Simple Convolutional	19.32 ± 1.69%	14.58 ± 1.27%
Residual Convolutional	17.15 ± 0.84%	12.32 ± 1.04%

TABLE II: Average test errors of each model

Model	**Test Errors (with Weight Sharing)**	
	without Auxiliary Loss	Auxiliary Loss
Fully connected	16.09 ± 0.72%	*13.94 ± 0.75%*
Simple Convolutional	15.58 ± 1.59%	*11.57 ± 1.01%*
Residual Convolutional	14.87 ± 1.61%	*8.28 ± 0. 84%*

TABLE III: Average test errors of each model

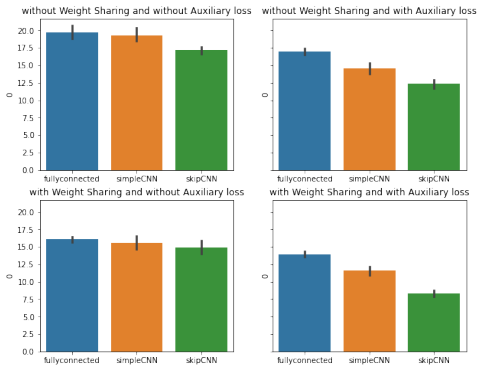

Fig. 5: Test Errors of the models in four different settings

VI. Conclusion

In summary, we have tested three network architectures to identify the smaller digit from a pair of input images. We have experimented with four different combinations of using (or not using) weight sharing and auxiliary loss for each model. Using a convolutional filter improves the performance when compared to a linear model, and using a skip connection along with convolutional filters gives the best performance among all architectures. Furthermore, across all the networks, the models that use the same weights to transform the input (i.e., weight sharing) and those that also include the loss related to the recognition of digit (i.e., auxiliary loss) provide lower test error rates compared to their counterparts.

Thus, in this specific case of training models for image classification, we conclude that using a convolutional model instead of a linear model not only reduces the number of parameters but also increases the accuracy. Further, the use of weight sharing when the inputs are sampled from the same distribution helps in stabilizing the result and reduces the number of trainable parameters. These techniques of using convolutional layers, weight sharing and auxiliary are especially useful when training deeper and wider neural networks as they potentially help in reducing the number of parameters, and help in propagating the error to the inner layers of the network.

References

[1] "UNIGE 14x050 – EPFL EE-559 – Deep Learning." [Online]. Available: https://fleuret.org/dlc/

VII. Appendix

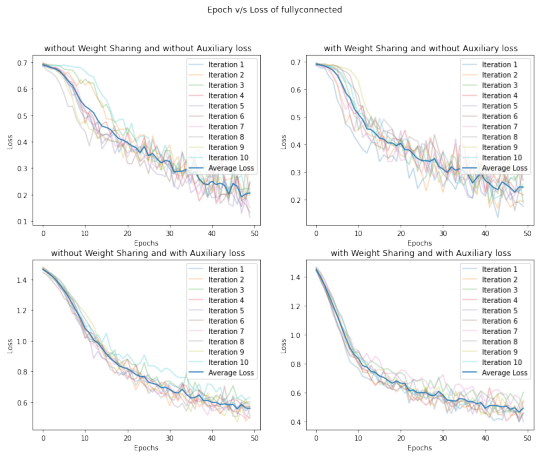

Fig. 6: Loss of the Fully Connected Model

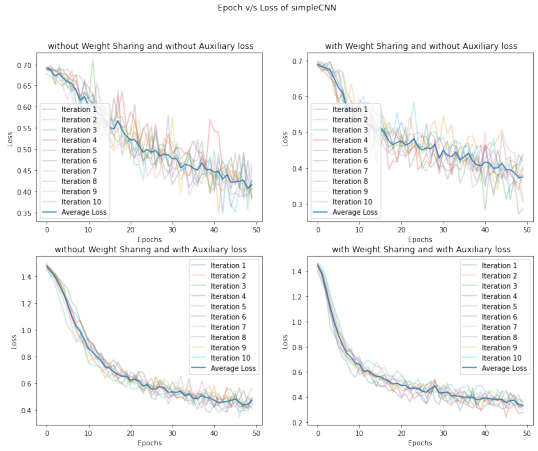

Fig. 7: Loss of the Simple Convolutional Model

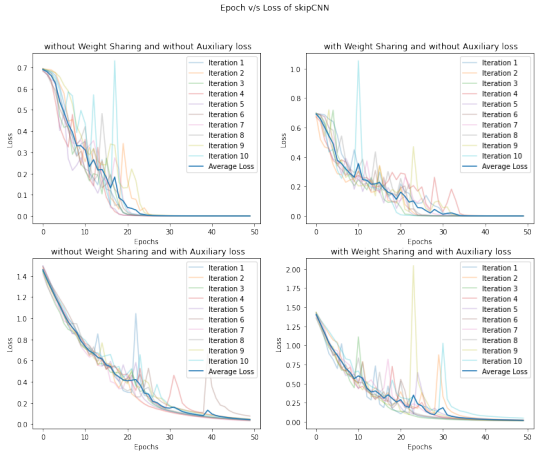

Fig. 8: Loss of the Residual Convolutional Model