

# Classification, weight sharing, auxiliary losses

Austin Zadoks  
(Dated: May 28, 2021)

Handwriting recognition and image comparison are posterchildren for the success of deep convolutional neural networks. In this report, I summarize a twin neural network implementation for predicting which digit is the greater in a pair of subsampled MNIST images. I also discuss the effects of weight sharing and auxiliary loss on the performance of the model. A twin neural network with  $\sim 20,000$  parameters trained on 1000 image pairs over 25 epochs with weight sharing and auxiliary loss achieves an average test accuracy of  $94.97 \pm 0.90$  % in  $3.51 \pm 0.28$  seconds on the course VM ( $N = 32$ ).

## INTRODUCTION

The goal of this miniproject is to design a deep neural network which takes pairs of  $14 \times 14$  grayscale images of handwritten digits from the MNIST database and predicts if the first digit is lesser or equal to the second digit. Corresponding to each pair of images, we are given both a binary target which corresponds to whether the first digit is truly less than or equal to the second digit and the classes of the two digits in the image pair.

One can conceptually approach this problem in two steps: (1) construct a network which predicts the class of each of the images, and (2) construct a network which takes two outputs from the first network and predicts whether the digit in the first image is lesser or equal to that in the second.

To solve the first problem, a LeNet-5 style convolutional neural network is a good approach. For the second, a simple multilayer perceptron or fully-connected network should be both efficient and effective. If a convolutional network that uses the same weights for both images in each pair is used for the first step, the overall network is called a twin or siamese network.

## MODEL ARCHITECTURE

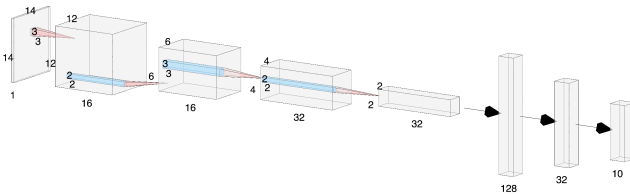


FIG. 1: Convolutional subnet architecture

The convolutional subnet architecture, shown in Fig. 1, takes in one image of shape  $1 \times 14 \times 14$  and returns a vector of length 10 containing the predicted categorization of the image as a digit 0-9. Written as an equivalent `torch.nn.Sequential` model, the convolutional subnet is:

```
import torch.nn as nn

nn.Sequential(
    nn.Conv2d(1, 16, 3), # 3x3 to avoid aliasing
    nn.ReLU(),
    nn.AvgPool2d(2, 2),
    nn.Conv2d(16, 32, 3), # "above"
    nn.ReLU(),
    nn.AvgPool2d(2, 2),
    nn.Flatten(),
    nn.Linear(128, 32),
    nn.BatchNorm1d(32),
    nn.Dropout(0.25),
    nn.Linear(32, 10)
)
```

Testing showed that ReLU activation to be more stable and to give higher accuracy than tanh activation. Dropout is used after the output hidden layer to reduce overfitting by the model. Batch normalization is used to improve training stability by reducing the risk of vanishing gradients. `torchsummary` reports that this convolutional network has 9,322 parameters.

The fully-connected subnet takes in two concatenated outputs from the convolutional subnet and is equivalent to the following `torch.nn.Sequential` model:

```
import torch.nn as nn

nn.Sequential(
    nn.ReLU(),
    nn.Linear(20, 10),
    nn.BatchNorm1d(10),
    nn.ReLU(),
    nn.Dropout(0.25),
    nn.Linear(10, 2)
)
```

The model begins with an activation because it is taking input directly from the final linear layer of the convolutional model above. Just as in the final fully-connected layers of the convolutional model, batch normalization is used for stability and dropout is used to avoid overfitting. `torchsummary` reports that this fully-connected model has 252 parameters.

The goal of this project is to predict whether the first image in a pair contains a digit of lesser or equal value than the second image. In order to achieve this goal,

the two subnets above must be combined, but there is a choice to make: should two separate convolutional subnets (with separate weights) be used for the two images, or should they share the same weights? Additionally, we have access to both the intermediate outputs of the convolutional subnet(s) and the corresponding ground truths; can we use those to our benefit?

## WEIGHT SHARING AND AUXILIARY LOSS

My `TwinNet` implementation allows for weight sharing and/or the use of auxiliary losses. This is achieved by two options in its constructor. The training function then supports calculating loss from multiple predictions:

```
import torch.nn as nn

class TwinNet(nn.Module):
    def __init__(self, share_weight, aux_loss):
        self.aux_loss = aux_loss
        if share_weight:
            self.lenet1 = LeNet()
            self.lenet2 = self.lenet1
        else:
            self.lenet1 = LeNet()
            self.lenet2 = LeNet()
        # ...
    def forward(self, input):
        # ...
        if self.aux_loss:
            return (output, output1, output2)
        else:
            return (output,)

def train_twin_net(*args):
    # ...
    for pred, target in zip(preds, targets):
        loss += criterion(pred, target)
    # ...
```

In practice, this means that if weight sharing is disabled, the `TwinNet` is composed of two separate convolutional subnets with two separate sets of weights and biases, one for each image in the pair. If it is enabled, only one convolutional subnet with one set of weights and biases is used for both images in each pair.

Optionally, both the target output and the auxiliary outputs of the convolutional subnet(s) are returned by `TwinNet.forward` and used to calculate the loss.

Leveraging this implementation, four different training methods / architectures have been tested corresponding to the possible combinations of weight sharing and use of auxiliary loss.

## TRAINING

Training is performed for each combination of weight sharing and auxiliary loss in 32 rounds for statistical sam-

pling. Each round consists of providing a new random seed to `PyTorch`, generating 1000 pairs each of training images and test images, and training for 25 epochs with a batch size of 128. Optimization is performed using ADAM ( $\eta = 1 \times 10^{-2}$ ). Loss is calculated using cross-entropy, as is conventional for classification problems. Error is calculated using a “winner-take-all” method applied directly to the `TwinNet` outputs.

The optimal  $\eta$  was determined using a 1D grid search over values from  $1 \times 10^{-3}$  to  $1 \times 10^{-1}$ . Visual inspection of the training curves suggested that  $1 \times 10^{-2}$  yields a good balance of stability and performance. Similarly, batch sizes from 8 to 128 were tested, and 128 was selected.

Training curves averaged over the 32 rounds for the different combinations of weight sharing and auxiliary loss are shown in Fig. 2.

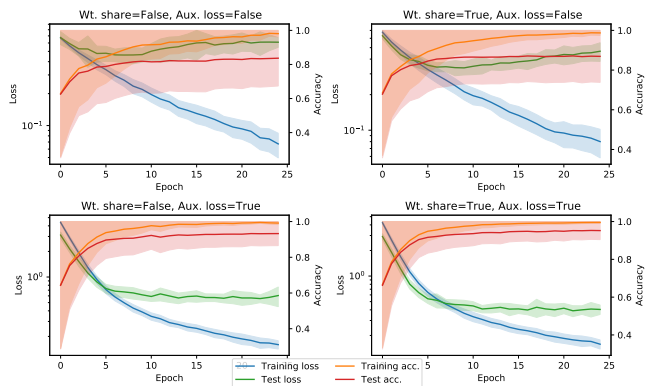


FIG. 2: Mean training curves for combinations of weight sharing and auxiliary loss with standard deviation uncertainty shading ( $N = 32$ ).

## RESULTS AND DISCUSSION

In the miniproject assignment document, it is stated that a simple convolutional network with  $\sim 70,000$  parameters can reach an error rate of  $\sim 15\%$  in 25 epochs and  $\sim 2$  seconds. In a sense, a twin network with no shared weights or auxiliary loss can be seen as a convolutional network where each convolution has 2 groups. This is not strictly true as other layers in the network may mix the two input groups, however it is a useful analogy for comparing the performance reported in the first line of Table I. As expected from this analogy, the performance of the non-weight-shared twin network is similar, if slightly worse, than a standard convolutional network.

However, with the addition of weight sharing, the error falls to 12%; with auxiliary loss it falls to 7%; and with both weight sharing and auxiliary loss, the twin network achieves 5% training error.

Both weight sharing and auxiliary loss independently improve the final error, however as shown by the test

TABLE I: Training results ( $N = 32$ )

Wt Share	Aux. Loss	Final loss		Final error (%)		Time (s)
		Training	Test	Training	Test	
		0.08±0.06	0.6±0.1	3.1±2.4	16.7±1.7	3.7±0.2
X		0.04±0.02	0.4±0.07	1.2±0.7	12.7±1.6	3.7±0.3
	X	0.07±0.02	0.6±0.07	0.6±0.3	6.6±0.8	3.9±0.3
X	X	0.06±0.03	0.4±0.08	0.4±0.3	5.0±0.9	3.6±0.3

loss and test accuracy curves in Fig. 2, auxiliary loss is the more powerful method for improving model performance. With weight sharing but no auxiliary loss, the model overfits the input, shown by the positive curvature in the corresponding test loss curve. The auxiliary loss seems to lessen this overfitting.

This may not be a true fault of weight sharing. Because twice as many data points are used to train the convolutional subnet, it is inherently much more likely to overfit. One way to avoid this could be to increase the dropout rate or add more dropout layers to the model in

the case of weight sharing without auxiliary loss.

More rigorous study of model implementation details and training methods, including but not limited to comparisons with other model architectures, the optimizer and its parameters, and better or more advanced data treatment practices could be performed to improve the model performance.

## CONCLUSIONS

Here, I have presented an implementation of a twin neural network to the problem of classifying and comparing images. Without weight sharing or auxiliary loss, this model performs slightly worse than a simple convolutional network, however it does so with less than half the model parameters. The addition of weight sharing and particularly of auxiliary loss drastically improve the model performance, achieving down to  $\sim 5\%$  training error in this implementation.