Deep Learning Project 1 - Classification, weight sharing, auxiliary losses -

Deep Learning (EE-559)
École Polytechnique Fédérale de Lausanne

Amir Ghali
amir.ghali@epfl.ch

Mahmoud Sellami
mahmoud.sellami@epfl.ch

Ahmed Ben Haj Yahia
ahmed.benhajyahia@epfl.ch

***Abstract*—This report details our approach to compare two digits of a pair of two-channel images from the MNIST dataset. The goal of this project is to compare the accuracy of different architectures and asses the performance improvement that can be achieved through models with shared weights and the use of an auxiliary loss . The performance of each of these different models will be estimated on a test dataset through 10 rounds where both data and weight initialization are randomized at each training.**

I. Introduction

A. Data

The data comes from the MNIST dataset but instead of the original size of 28x28, we work with a computationally less expensive version of size 14x14 in order to run our models faster. We did not perform any additional preprocessing.
The dataset consists of a randomly sampled pair of images put into two different channels, thus the shape of the data is 2x14x14. The dataset also provides labels for each channel's true digit and a Boolean value that indicates which channel's digit is greater than the other.

B. Approach

In order to decide whether the first channel is lesser or equal to the second channel, we tried different types of architectures but we will only present to you the most two that serve better the purpose of this project.
We started with a simple model that consists only of fully connected layers .
Then we constructed a deeper model consisting of a convectional neural network followed by a fully connected network. Using this model we were able to show the effect of weight sharing and the use of an auxiliary loss on the model's performance.

II. First Model : Baseline

This can be considered as our baseline model as it is rather a straightforward model.

A. Architecture Description

This model consists of three separate fully connected layers. The two channels are first flattened and passed to 14*14*2 neurons of the first layer. The last layer returns a binary output indicating whether first channel is lesser or equal than the second one. The loss is computed using CrossEntropy.

B. Activation function

For this model we used ReLU for the intermediate layers. Although this model is not so deep but the other activation functions like Tanh and Sigmoid performed worse with this model.
For the activation function of the last layer we tried both Sigmoid and Softmax and both gave similar results.

C. Hyperparameter tuning

Few hyperparameters were tuned using a simple grid search algorithm on Google Colab for faster results.
This gave us: nb_hidden = 40 , learning rate = 0.001 for SGD during training.

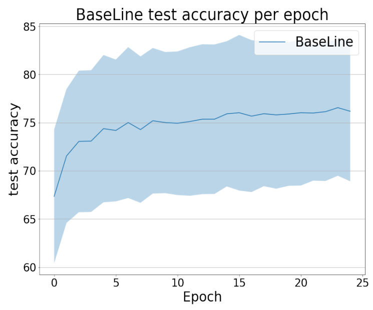
D. Model performance

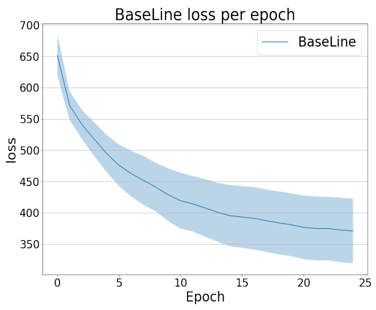
Considering the shallowness of our model, its performance in terms of test accuracy was a little bit surprising. See results in Table I

BaseLine	Accuracy	Standard deviation
Train	92.35%	± 11.45
Test	76.17%	± 7.26

TABLE I
BaseLine performance

Then we plot the model's mean value of the test accuracy and the training loss over the 10 rounds.





As we can see here on the graph this model is not good for our task as it's accuracy can can go from as high as 82% to go down to as low as 68% which indicates our model high variance.
Next we will present a model with a higher test accuracy and a smaller variance.

III. Second Model : SiasemeNet

A. Architecture Description

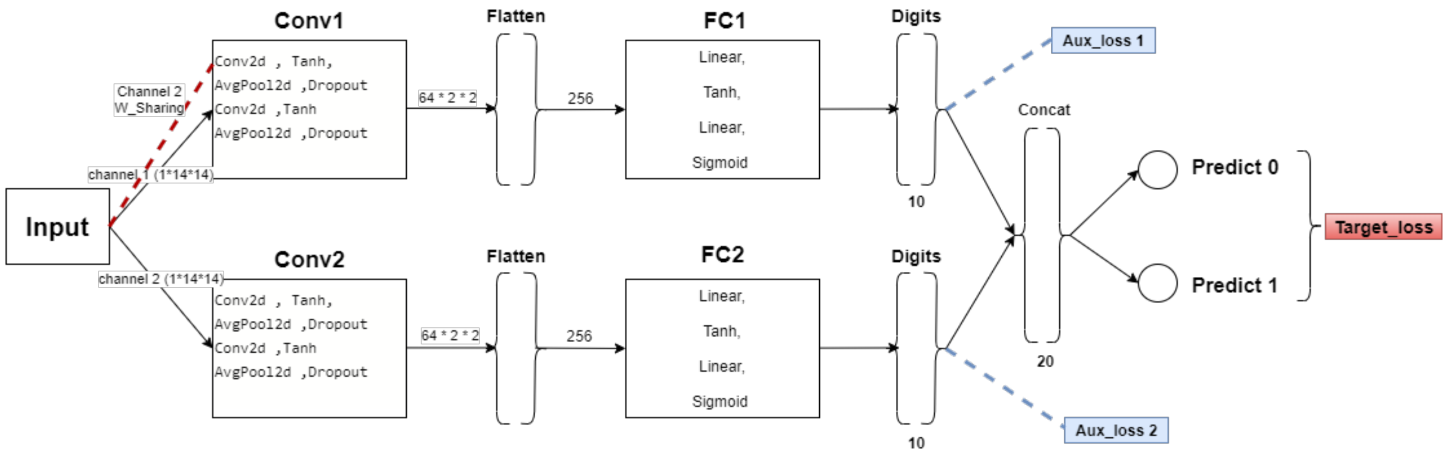


Fig. 1. SiasemeNet Architecture

Our second model is a "Siamese network" a network where two or more similar sub-networks learn to differentiate between two inputs(channels in our case) [1] .
As it is shown on the architecture Fig.1, it consists of two similar convolutional networks Conv1 and Conv2 .
These two networks recive one of the input channels , do the filtering and then the output is fed to the two similar fully connected layers Fc1 and Fc2 . This model also offers two boolean parameters to customize it:

w_sharing: If set to True only Conv1 and Fc1 will be used during the training/testing I.e. the second channel of the input will be fed to Conv1 instead of Conv2.

aux_loss: If set to True the loss of the digit recognition of the two channels will also be included in the final loss.

1) *Convolution Layer* Convolutional network are very know in the field of image recognition [2]. They use filters to extract important features from the images that will help later in the classification. We used two convolutions the first having 32 channels_out that will capture the most important features in our handwritten digits and the second one having 64 channels_out to get even more detailed features. A kernel of size 3 is used as the resolution of the image is already small.
2) *AvgPool Layer* Pooling layers reduce the dimensions of the data, by merging the outputs of neuron clusters at one layer into a single neuron in the following layer.AvgPool was preferred against MaxPool because AvgPool because it takes everything into account which encourages the network to identify the complete extent of the object. [3]
3) *DropOut Layer* In this layer, individual nodes are 'dropped out' of the net at random, with probability p. This is necessary, as it prevents the problem of over-fitting.

B. Activation function:

There are many know non-linear function that serve as good activation functions like ReLU , Sigmoid and Softmax etc.. . Considering the depth of our network, we first started with ReLU as it avoids the vanishing gradient. And to our surprise the model performed even worse than the BaseLine model with an average accuracy of 40% . So we checked the model parameters specifically using param.grad and quickly we saw that starting from the first epochs most of the parameters gradients were equal to zero which indicates that these neurons "died" due to negative values.
Using Tanh function we got a way better model performance. Even though our model is deep, the gradient of the earliest layers did not vanish during the whole training.
As for the last layer Sigmoid performed better than softmax for both : the digit and the target prediction.

C. Loss calculation:

To compute the loss during training we refer the model settings: If aux_loss is activated then the total loss will be a linear function of aux_loss$_1$, aux_loss$_2$ (See Fig.1) and the target loss. The coefficients should sum up to 1.

$$loss = \alpha * target\_loss + \frac{1-\alpha}{2}\,(aux\_loss_1 + aux\_loss_2)$$

If aux_loss is not activated α equals 1.
We used the BCELoss as a criterion because it gave better results than CrossEntropy especially for the auxiliary loss. That's why we used a modified version of the function convert_to_one_hot_labels available in the provided file dlc_practical_prologue.py to convert the train_classes and train_target to one hot encoding.

D. *Hyperparameter tuning*

Using a simple grid search algorithm on Google Colab we tuned the following hyperparameters :
drop_rate = 0.3 , learning_rate = 0.001 , α = 0.2
The α value might be surprising but indeed it makes sence as it tells the model to focus more on the task of recognizing the digits for its huge importance for a successful later comparison.

E. *Model Performance:*

In TABLE II you can see our model performance with four different settings. These estimation are performed over 10 rounds where both data and weight initialization are randomized.

Siamese	**W_Sharing**	**Aux_loss**	**Accuracy**	**Std**
No_Ws & No_Aux	False	False	68.06%	± 7.01
Ws & No_Aux	True	False	85.00%	± 1.00
No_Ws & Aux	False	True	92.21%	± 1.16
Ws & Aux	True	True	**93.76%**	**± 0.85**

TABLE II
DIFFERENT SIAMESE SETTINGS PERFORMANCES

From the TABLE II, you can see that we managed to reach a test accuracy as high as 93.76% with a very small standard deviation using a Siamese network with weight sharing and auxiliary loss enabled.
We can also notice that enabling one of the two settings makes the model somehow stable and have a small variance.
We can clearly see that Aux_loss has a higher impact on the model performance and that once it is enabled the effect of w_sharing is almost insignificant.
However, when disabling these two settings our model performed a lot worse than our BaseLine model !
This was quite a surprise for us as we expected just a convolutional network would perform better than our BaseLine model.

IV. SUMMARY:

As a summary and From looking at the graph in Fig. 2, we can understand the effect of the weight sharing and auxiliary loss settings.
To start, both of these settings improved the model accuracy and decreased its variance which is very important to have stable results.
In one hand, the Auxiliary loss that compares the recognized digit from each channel with its corresponding true class is very important before carrying out another classification task such as comparison. In fact the model should focus more on minimizing this loss.
On the other hand, the effect of weight sharing might not be obvious at a first glance but remember that our first Siamese model with both settings disabled performed worse than our BaseLine model. And with enabling weight sharing , the model gained 18% in terms of accuracy !
In fact, our Siamese model with both settings disabled was under-fit as we only train using 1000 pairs of images and the use of weight sharing provides it with double train size. So this setting is important to train deep models with a relatively small train size.

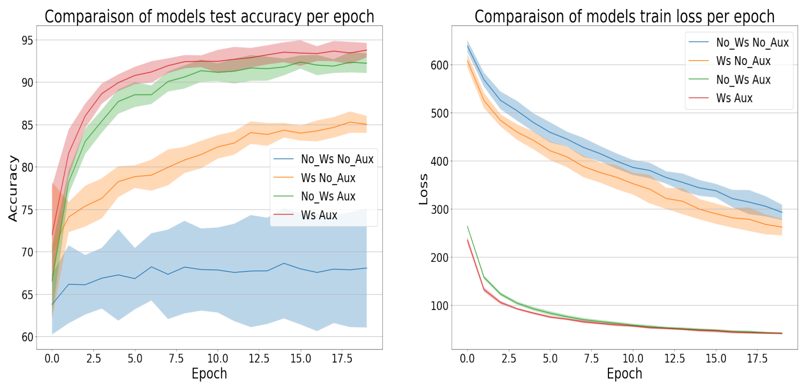


Fig. 2. Performances of different models

REFERENCES

[1] Signature Verification using a "Siamese" Time Delay Neural Network *Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Sackinger, and Roopak Shah.*
[2] *Convolutional Neural Networks and Image Classification* [online] Available at: https://towardsdatascience.com/wtf-is-image-classification-8e78a8235acb
[3] *MaxPool vs AvgPool* [online] Available at: https://iq.opengenus.org/maxpool-vs-avgpool/