Project 1 – Classification, weight sharing, auxiliary losses

Maksim Kriukov
SCIPER: 313810
SV Section

Fedor Moiseev
SCIPER: 309259
SC Section

Simona Leserri
SCIPER: 312970
SV Section

January 11, 2022

Abstract

We compared the performance of different deep convolutional networks by assigning them the same task, that is the comparison of two digits visible in a two-channel image from MNIST dataset. The different performances obtained allow us to better appreciate the effect on accuracy of different techniques such as weight sharing and auxiliary losses. The latter was employable as the balanced dataset provides also the classes of the two digits represented. Exploiting at most the aforementioned techniques and Dropout layers the best network we trained reaches a final test accuracy of mean=0.966 with standard deviation std=0.009, weighted over ten rounds.

1 Methods

1.1 Hyperparameters

With the aim of running a fair comparison, we defined some hyperparameters common to all trials. First of all, we decide to use **Adam** optimizer, largely used and recommended for its efficiency and interpretability. We set the associated **learning rate** to $3 * 10^{-4}$, a common value that represents a good compromise between a too fast and unstable descent and a too slow and suboptimal one. All images are standartized using mean and std of the training set. Lastly, given the binary classification task, we selected the **Binary Cross Entropy with Logits Loss** that gains in numerical stability with respect to a simple BCELoss as it combines the Sigmoid function and the BCELoss into one layer. However, when we wanted our model to recognize the number present in the image, we used a **Cross Entropy Loss** since we also performed a multiclass classification of digits. Besides, we want to compare the models when they converge: therefore we train all of them for 100 epochs by default, a value high enough to avoid underfitting while reaching a good accuracy for most networks. We also divide initial training set into two tensors with sizes 800 and 200 - the first one is used for training, the second one is used for validation. At the same time, to hinder overfitting, we have introduced **EarlyStopping**, an util that blocks training ahead of time if it does not result into an improvement in validation accuracy for 20 epochs.

1.2 The Architectures

The networks differ substantially not only because of the inputs given but also because of their specific training techniques, described below. However, all networks largely rely on a common choice of architecture. The neural networks consist of two convolutional layers, followed by max-pooling layers and ReLUs, and several linear fully-connected layers, whose number and dimensions change according to the peculiarities of the model. The convolutional layers are consistent throughout the models, so that they share the **same feature extraction module** shown in Fig.1. Additionally, to prevent overfitting, **Dropout layers** were introduced after the linear ones.

The different performances, are summarized in Table 1 an visualized in Fig. 2. The figure makes also clear the effect of **Early Stopping**. Below, in order to grasp the slight differences amongst the models, there is a detailed description of all of them, in order of performance. The descriptions, however clear, are also supported by pictures added in Appendix.

TwoChannelsModel The first, naive approach predicts the final value having as input the image coupled as a two-channel tensor. Apart from the common architecture described above, the network is equipped with 3 linear layer. Over ten rounds it reaches an accuracy of 0.798 ± 0.011 but, even with the dropout, it seems to overfit.

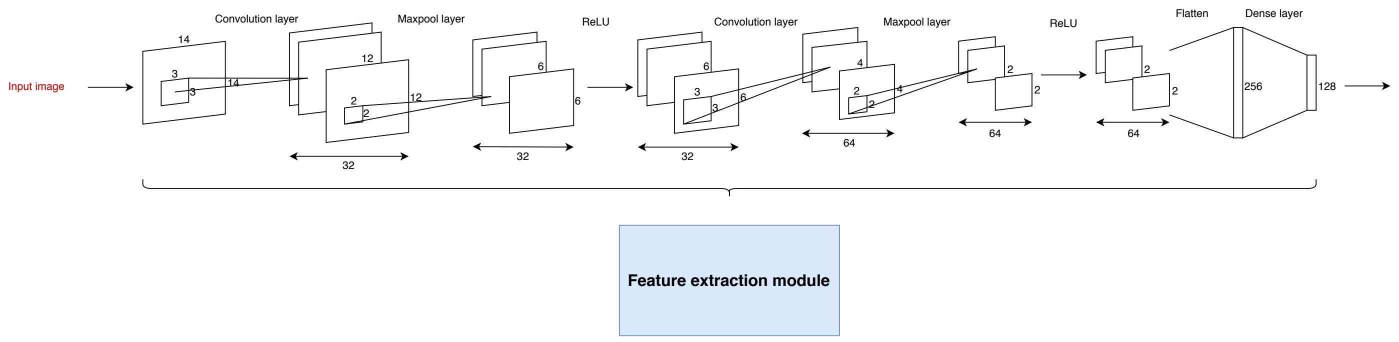


Figure 1: The feature extraction module architecture common for all networks.

Table 1: A comparison of performances evaluated over 10 rounds both with and without Dropout.

	No Dropout		With Dropout	
Model	Mean	STD	Mean	STD
TwoChannelsModel	0.798	0.011	0.798	0.011
TwoBranchesModel	0.812	0.013	0.826	0.018
WeightSharingBranchesModels	0.838	0.012	0.840	0.008
BranchesToVecModel	0.827	0.012	0.831	0.017
WeightSharingBranchesToVecModel	0.847	0.018	0.846	0.010
WeightSharingAuxiliaryModel	0.865	0.019	0.863	0.012
DirectClassificationModel	0.965	0.007	0.953	0.010

TwoBranchesModel This network receives as input the two images separately and applies the previously described routine and three linear layers individually to each of the two images. Therefore, each branch of the network has its own set of parameters to optimize. The final prediction is the difference between predictions for both branches. The initial separation of the input leads to an increase in accuracy, that reaches 0.812 ± 0.013. Although diminished, overfitting is still an issue.

WeightSharingBranchesModel What makes this network differ from the previous one is **weight sharing**. Since the two input images have the same nature, they undergo the same sequence of operations. As a consequence, during back-propagation, fewer parameters are optimized and the network learns faster and converges better, reaching a value of 0.838 ± 0.012.

BranchesToVecModel The network is a slight modification of the TwoBranchesModel one. The two images are processed independently (without weight sharing) and undergo only two linear transformations, each resulting in a linear tensor of dimension 10. After concatenation of the obtained arrays, the final linear transformations are applied. While it outperforms TwoBranchesModel, the model is still less accurate of WeightSharingBranchesModel, resulting in an accuracy of 0.827 ± 0.012.

WeightSharingBranchesToVecModel Combining the strategy of BranchesToVecModel and the weight sharing of WeightSharingBranchesModel, this network reaches an accuracy of 0.847 ± 0.018.

WeightSharingAuxiliaryModel Another weight sharing network that receives as input also the class of the digits represented in the images. The basic architecture is the same as for **WeightSharingBranchesToVecModel**. After the convolutional layers and a single linear layer, we obtain linear tensors of dimension 128. Once concatenated, we further reduce the dimension to 100 through a linear layer. The obtained linear tensor is reduced again to obtained the desired prediction value. At the same time, the obtained tensor of dimensions [mini_batch_size, 100] undergoes a reduction to a tensor of dimensions [mini_batch_size, 10] with the class probability distribution for the images. Therefore, the final loss is given by the sum of three terms: the difference between the prediction and the actual label, the difference between the first digit detected and its actual class and the difference between the second digit detected and its actual class. These two operate as **auxiliary losses**. By comparing this model with its equivalent WeightSharingBranchesToVecModel, that does not rely on auxiliary losses, we can appreciate that their use is highly beneficial. During backpropagation, the model simultaneously learns to recognize the digit of the two images, as well as predict the right final value. At the end we reach an accuracy of 0.865 ± 0.019.

DirectClassificationModel Our last and best model can be seen as an extreme and best exploitation of both Weight Sharing and Auxiliary losses. This model aims to predict only the digit values of both input images, without taking into account their difference. So the network just reshapes input tensor and classifies each image independently. Still the same convolutional layers are applied for each image, followed by two linear layers. The output of the model is a tensor of dimensions $[2 \times \text{mini\_batch\_size}, 10]$ with the class probability distribution for the two images. the loss is defined through a CrossEntropyLoss criterion, suitable for MultiClass Classification, resulting in a $[2 \times \text{mini\_batch\_size}]$ long one-dimensional tensor. Through reshaping to a two columns tensor, we can have a much easier interpretation of it: every column represents an image. Going row by row, we can directly compare the values of the digits represented in the initial couple. It comes not as a surprise that this is the best model, reaching an accuracy of 0.965 ± 0.007. Not only it does not overfit, but taking full advantage of the Early Stopping option, it results completely trained ahead of time.

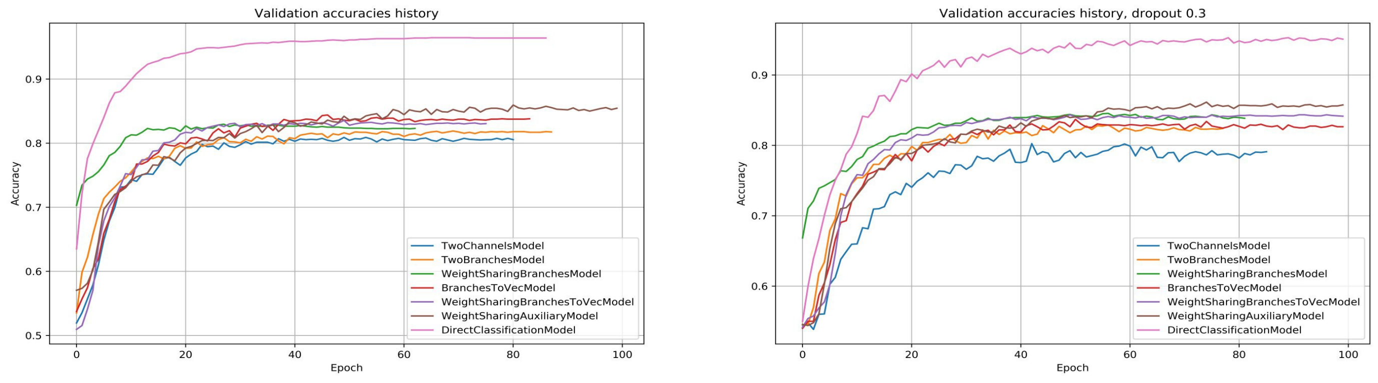


Figure 2: Validation accuracies history with and without Dropout.

To reproduce our results you can use `test.py` file. Without any arguments, it runs 10 rounds of **DirectClassification** model. If you want to train it only once, pass argument `--nb_rounds {1}`. For other models, pass argument `--model {model name (as in this report)}`. When we tested our code on the VM we found that it can produce slightly different results (difference could be approximately 0.003). Our local setup: `Python 3.6.9, torch 1.3.0, torchvision 0.4.1 (used only in dlc_practical_prologue.py)`.

2 Conclusions

In this work we have built, trained and compared seven slightly different networks, and analyzed their performances in terms of accuracy using MNIST dataset. Overall, our comparison has served to prove that even slight architecture changes can result in major differences in accuracy. Even if we added Dropout layers, the improvements were only marginal and did not occurr in all networks. As the networks are not so deep Dropout seems to disturb the training, without having a significant positive impact on overfitting. Both weight sharing and auxiliary loss help to improve the performance of the network. All of our models go beyond 80 % accuracy but DirectClassificationModel significantly outperforms the others, making us appreciate the advantage introduced by weight sharing and auxiliary losses. While the former technique reduces the number of parameters the model has to update, the latter improves the efficiency of the main task by creating useful representation of the images.

3 Appendix

In this appendix we show the architectures of all described models.

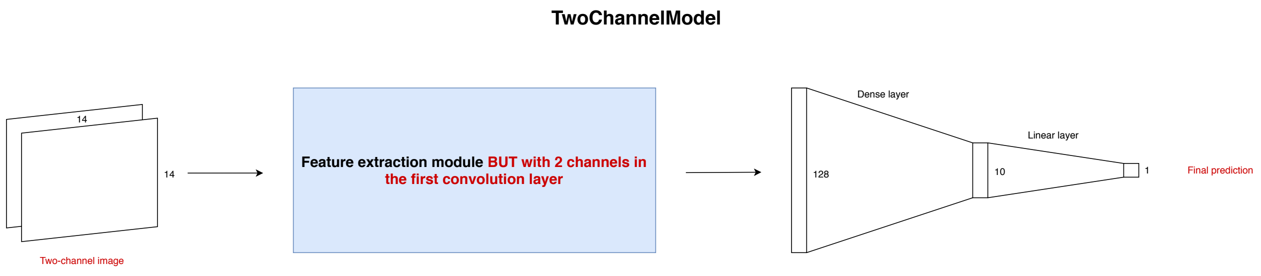


Figure 3: TwoChannelModel

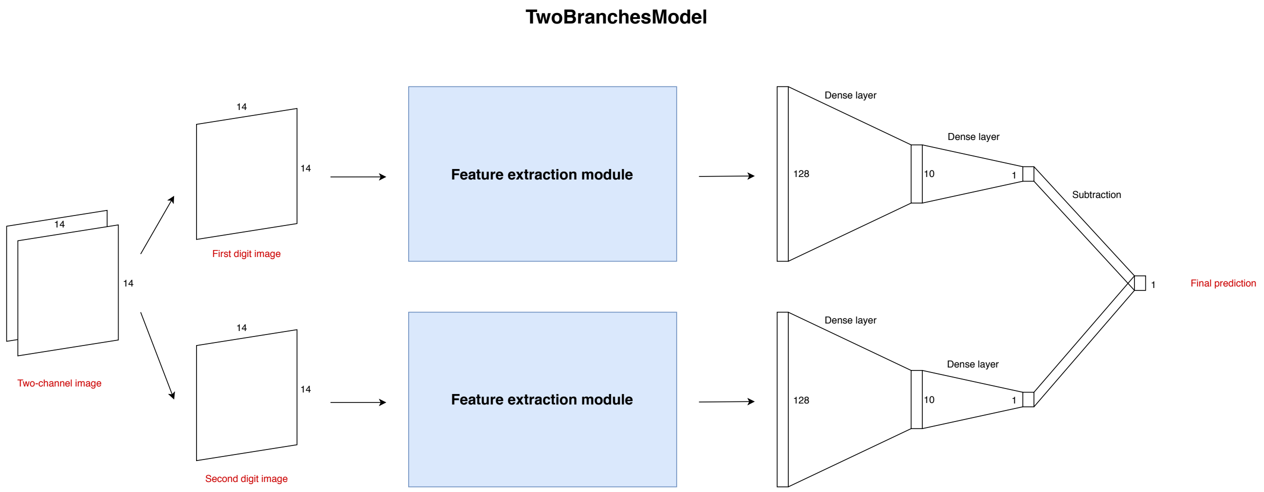


Figure 4: TwoBranchesModel

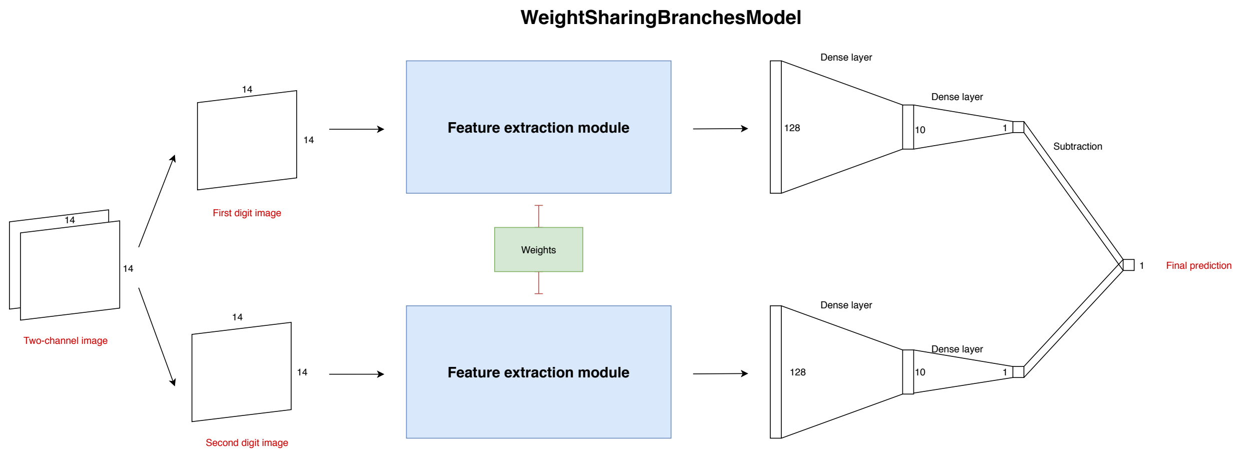


Figure 5: WeightSharingBranchesModel

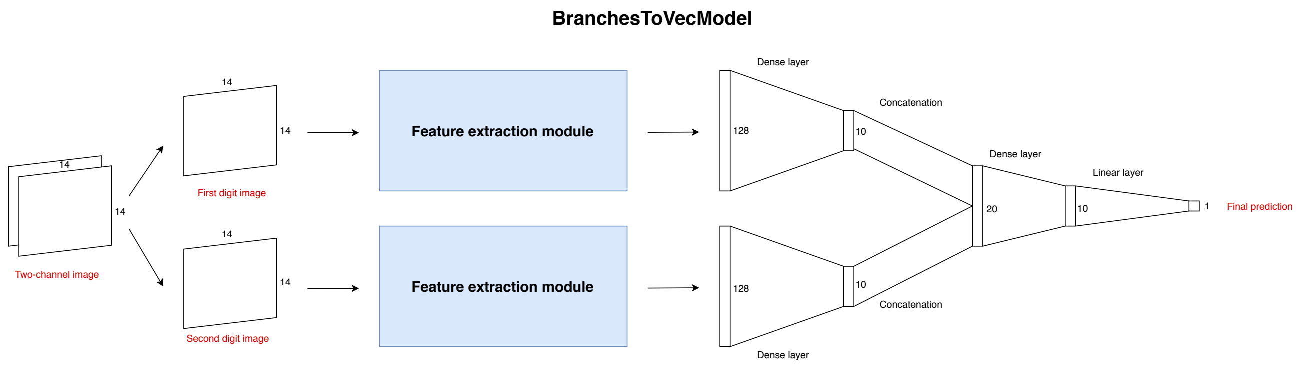


Figure 6: BranchesToVecModel

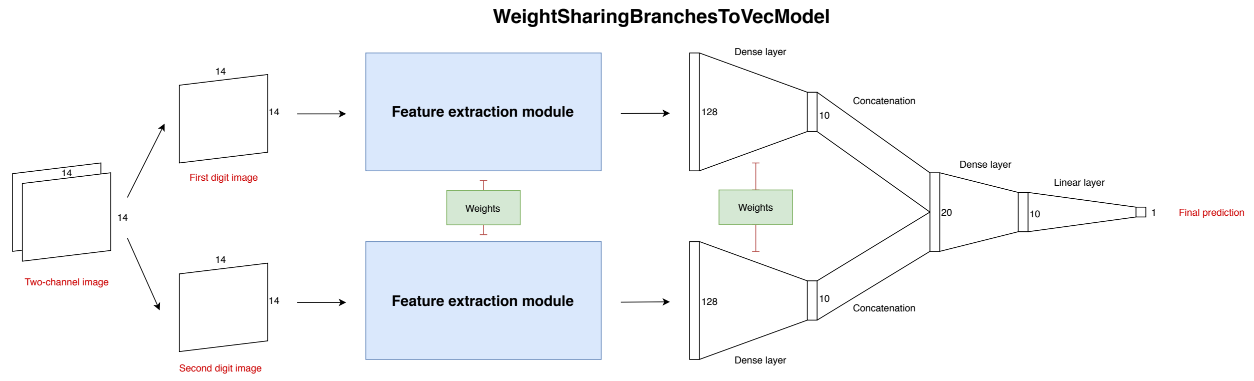


Figure 7: WeightSharingBranchesToVecModel

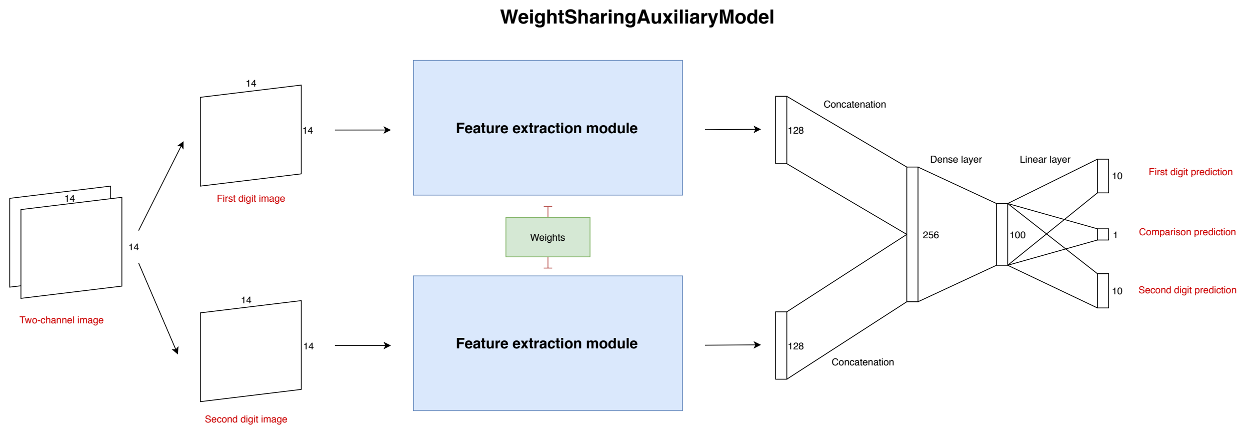


Figure 8: WeightSharingAuxiliaryModel

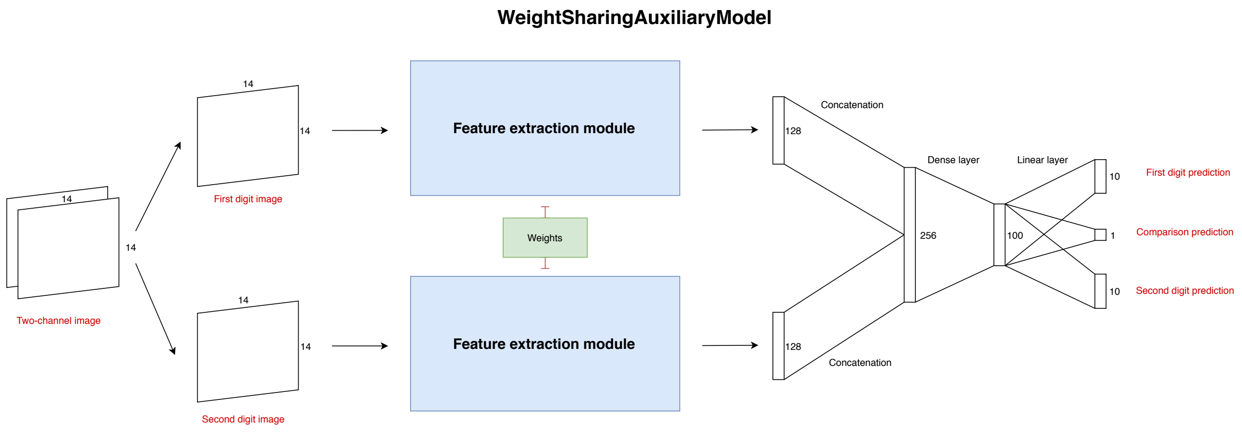
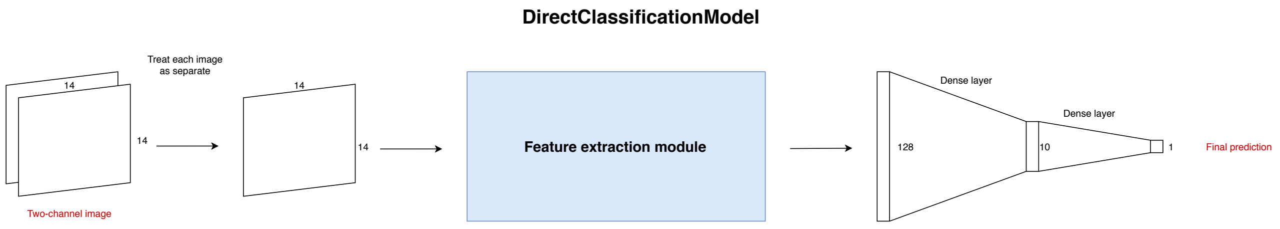


Figure 9: WeightSharingAuxiliaryModel



DirectClassificationModel
14
14
Two-channel image
Treat each image as separate
14
14
Feature extraction module
128
Dense layer
10
Dense layer
1
Final prediction

Figure 10: DirectClassificationModel