Project One

Authors: Alec Flowers, Ali Falsafi, Rania Islambouli
Deep Learning EE-559, EPFL, Switzerland

May 28, 2021

I. Introduction

Among various machine-learning algorithms, deep learning has become one of the most prominent methodologies applied in various areas such as image processing, object detection, natural language processing, and speech recognition [A1]. Due to its ability to collect valuable knowledge from complex datasets deep learning is being used for different tasks like classification, regression, and clustering [A2]. However, training deep learning networks is not a trivial task and can suffer from many problems including vanishing gradient problems and overfitting [A3], [A4]. Different techniques have been studied to improve training including auxiliary loss, weight sharing, dropout, and batch normalization [A5]. In this project, we aim to explore and test different techniques to improve the training and accuracy of deep learning networks using multiple architectures.

The main task is to compare two digits represented as two-channel images [A6]. The dataset provided consists of $2 \times 14 \times 14$ grayscale images obtained via average pooling images of handwritten digits from MNIST dataset [A7]. The Boolean value outputted predicts whether the digit in the first image is less than or equal to the digit in the second. We implemented three different architectures. The first architecture is a single neural network doing all the task at once. Next we split the task into classification. comparing and adopt two neural networks for each them. Finally we applied the auxiliary loss idea by considering the losses resulting from classification sub-nets using the known digit labels.

II. Setup

Our implementation is organized as follows:

- *net.py*: defines the models and architectures we implemented.
  - **NeuralNet** class defines the fully connected neural network with arbitrary hidden layers addressed as multilayer perceptron (**MLP**) in this report.
  - **NeuralNetCalssifierComparer** is the weight sharing model addressed as classifier-comparer (**CC**) in this report.
  - **NeuralNetCalssifierComparerAuxLoss** defines the weight sharing model with auxiliary loss addressed as classifier-comparer with auxiliary loss (**CC_AUX**) in this report.
  - For each of the above we have a corresponding convolutional neural network. A fully connected convolutional neural network (**CNN**) a CNN-classifier-comparer (**CNN_CC**) and a CNN-classifier-comparer with auxiliary loss (**CNN_AUX**).
- *plot.py*: contains functions that runs models and plots outputs.
- *runner.py*: contains functions managing the training and testing of all the models. The specific classes for each network in this file handles any necessary data manipulating that the network needs. We also have utilized **Tensorboard** for visualization of our networks.
- *utils.py*: contains useful functions to evaluate the different models, plotting and reporting the outputs.
- *data.py*: functions used to load and generate the data.
- *test.py*: run file built to the project specifications. The hyper-parameters are set in this file, e.g. learning rate is set to $1e^{-4}$ and the networks' size. some verbosity enumerator for dumping train summary and **Tensorboard** output are also set here.

III. Architectures

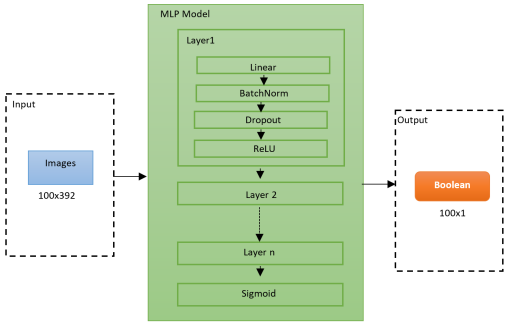

Fig. 1: Plain **MLP** architecture.

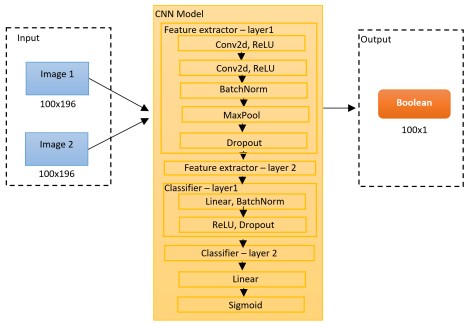

Fig. 2: Plain **CNN** architecture.

- **Plain Neural Network** As a starting point, we implement plain neural network as a single block which receives the images as a two-channel tensor and returns the Boolean comparing the two digits. Both **MLP** -Figure 1- or **CNN** -Figure 2- can be utilized for carrying out the task.
- **Weight sharing**: Afterwards, we tried to enhance our model by separating classifying and comparing tasks between two neural networks acting on series to yield the final Boolean output desired (**CC**) depicted in Figure 3. The idea of implemented weight sharing is rooted in the fact that because a main part of the process performed on two channels is essentially identical (recognizing the value i.e. classifying), the part of the network doing this process can be shared.
- **Auxiliary loss**: Finally, based on the **CC** architecture, we tried to also include the auxiliary loss by incorporating the error in classifying the digits done with the classifier sub-network (**CC_AUX**). The loss function is a weighted summation of the following losses
  1) loss of the predicted Boolean comparing the digits ($loss_B$),
  2) loss of the first image classification ($loss_{C1}$)
  3) loss of the second image classification ($loss_{C2}$),

  and the total loss can be calculated according to equation 1, where W_1, W_2, and W_3 are the weights of the losses. The red dotted arrows in Figure 3 represent the auxiliary loss computation.

$$Loss = W_1 \times loss_{C1} + W_2 \times loss_{C2} + W_3 \times loss_B \quad (1)$$

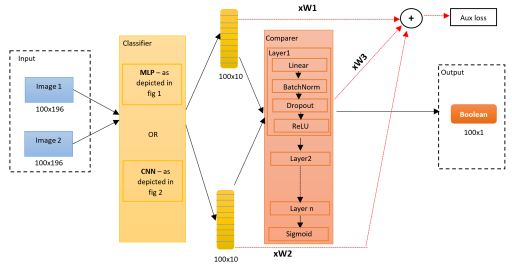

Fig. 3: Weight sharing & Auxiliary loss.

It is also notable that as the architecture offers the flexibility to choose the sub-nets as desired, we have examined both **MLP** and **CNN** as the network used in the classifier sub-net of the two other architectures as shown in Figure 3. We expect **CNN**s to show better performance since they are specifically effective in image classification due to their exceptional ability to extract local features. In addition, dropout with ($p = 0.5$) and batch normalization are also considered in the intermediate layers of the neural networks used as sub-nets and can be either activated and deactivated easily, The results in this report are with both dropout and batch-normalization activated.

IV. Results

All models are compared under similar conditions and while using the 1000 pair of images.

- Epoch: The number of the epochs is set to 200 in this file. However, it is modified in the training of the networks whose outputs are reported in the following in order to achieve training accuracy of higher than 95% i.e. exploiting the learning capability of the network. For example, it is 200, and 500 respectively for networks with depth of 2, and 3. For the CNN networks we use 100 epochs to conserve compute resources.
- Optimizer: We use the Adam optimizer with a learning rate of $1e^{-4}$ in all models and binary cross entropy loss in the **MLP**, **CC**, **CNN**, and **CNN_CC** model and cross entropy for classification loss and binary cross entropy for Boolean loss in the **CC_AUX** and **CNN_AUX** model. In auxiliary loss models we define $W_1 = 0.2$, $W_2 = 0.2$ and $W_3 = 0.6$.
- Batch size: For training the module parameters, a batch size of 100 in a stochastic gradient decent solver is used.

In **CC** and **CC_AUX**, the classifier sub-network has a constant architectures for all the results reported and a neural network of depth 3 and sizes $[80, 80, 20]$ is adopted as the comparer part of the neural network. Similarly in **CNN_CC** and **CNN_AUX** the classifier sub-network has a consistent architecture for all results reported, which is the same architecture as the **CNN** Figure 2. This architecture was arrived at with inspiration starting with an architecture similar to LeNet [A8] and moving to something that resembled VGGnet [A9]. Iterations included adding dropout, batch normalization, and increasing the number of convolutions.

The comparison results for models with 2 hidden layers trained for 500 epochs are noted in Table I. The values are the mean of 10 test rounds. The obtained results depict that the weight sharing has increased the accuracy by 3.58% and another 8.42% test accuracy increase was obtained by considering the auxiliary loss. The number of parameters in **CC** is actually lower than **MLP** network because the input size is, in practice, 392 for **MLP** and 196 for **CC**. It shows that the performance improvement was achieved using a model with less parameters yet with smarter design. The number of parameters are the same for **CC** and **CC_AUX** networks since we have used the same architecture for them and augmented the loss considering the intermediate output of the network (the classifier output) for the **CC_AUX** network.

In Table I we can see the results for our 3 CNN models trained for 100 epochs over 10 rounds. Even trained for fewer epochs and with less than half the parameters the **CNN** and **CNN_CC** are comprabale to their MLP equivalent while the **CNN_AUX** does by far the best of all our networks.

Table I: Accuracies in percentage (%) obtained with running the train-test procedure 10 times for all of the networks. Numbers are reported of fully connected networks with 2 hidden layers with width of 500.

	# of params	train acc.	test acc.
MLP	547787	99.70 ± 0.22	81.04 ± 0.14
CC	461741	98.91 ± 0.22	84.62 ± 0.12
CC_AUX	461741	99.06 ± 0.25	93.04 ± 0.11
CNN	274273	99.68 ± 0.17	81.65 ± 0.12
CNN_CC	285453	98.16 ± 0.04	84.53 ± 0.25
CNN_AUX	285453	99.58 ± 0.25	98.00 ± 0.11

We plot the train and test loss/accuracy for our various architectures to get a better understanding of the effect of the model changes, weight sharing, and auxiliary loss. Figure 4 shows the outputs of 3 architectures with fully connected classifier sub-net with 1 hidden layers. We can see that the **CC_AUX** model outperforms other models after 200 learning epochs as it reaches a test accuracy of around 0.89 compared to 0.82 for **CC** and around 0.81 **MLP**. As we increase the number of hidden layers of the classifier to 2, as shown in Figure 5, the test accuracy of **CC_AUX** increases to around 0.91 which is still greater than the other models where **MLP** stayed at 0.81 and 0.88 for **CC**. The fact that only accuracy for both networks with weight sharing increased and not for the plain network emphasizes that brute force increase of network depth or width does not necessarily result in performance improvement while having a smarter design for the network based on the given data (digits in the images) can help much more with the performance. Figure 6 depicts the performance of 3 networks with fully connected sub-nets with different architectures. The width of the intermediate layer of the networks are respectively 600, 400, and 300 respectively for **MLP**, **CC**, and **CC_AUX**. However after 500 training epochs, their test accuracy are respectively 0.92, 0.84, and 0.81 which shows that; even though; the **MLP** network has significantly wider layers it cannot compete with the other two networks with narrower layers. Figure 7 depicts the performance of 3 cnn networks with fully connected sub-nets and same width. We can clearly see that the **CNN_AUX** model performs by far the best and quickly reaches a high test accuracy in under 60 epochs. We think the reason being that the CNN is very good at capturing the structure in the data and predicting the proper class for the image. This means that when we add in the auxiliary loss, the model quickly learns to predict the image classes with high accuracy in the sub tasks and therefore becomes very proficient in the main task of saying whether the image is greater or less than.

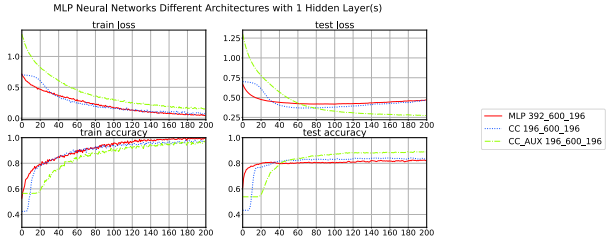

Fig. 4: Comparison of the 3 models with 1 hidden layers.

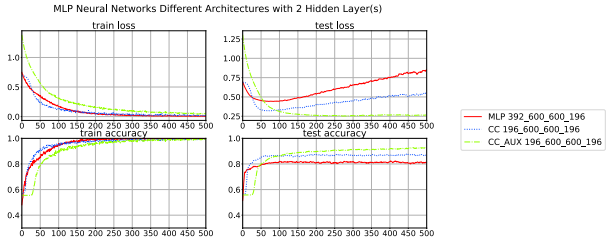

Fig. 5: Comparison of the 3 models with 2 hidden layers.

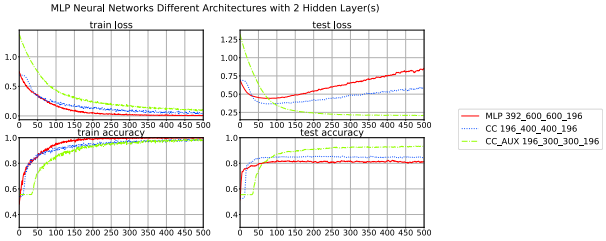

Fig. 6: Comparison of the 3 models with 2 hidden layers and different width.

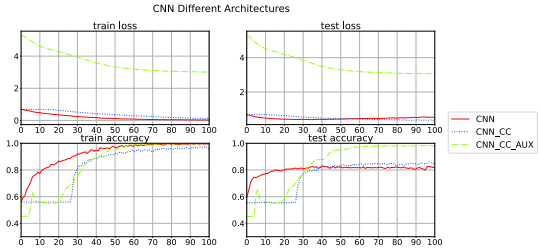

Fig. 7: Comparison of CNN with different architectures.

V. Conclusion

By utilizing weight sharing and auxiliary loss, we have been able to improve the performance of the neural network aimed to compare handwritten digit two images. Had we not known that we could categorize images according to their contained digits, and then compare those digits, the network would have been limited to the plain neural networks. The results obtained illustrate how important it is to understand a neural network's objective in order to design an effective and practical network. Compared to the changes that resulted in better performance of neural networks, domain knowledge is a real-world analogy. In order to design more effective neural networks, it is therefore often necessary to utilize domain experts' insight. As immediate possible improvements for the auxiliary loss architecture, we think one could consider the weights of the different losses as learning parameters. One other possible improvement could be employing pre-trained existing digit recognition neural networks for the classifier sub-net.

References

[A1] Q. Zhang, L. T. Yang, Z. Chen, P. Li, A survey on deep learning for big data, Information Fusion 42 (2018) 146–157.

[A2] S. Pouyanfar, S. Sadiq, Y. Yan, H. Tian, Y. Tao, M. P. Reyes, M.-L. Shyu, S.-C. Chen, S. Iyengar, A survey on deep learning: Algorithms, techniques, and applications, ACM Computing Surveys (CSUR) 51 (5) (2018) 1–36.

[A3] M. M. Najafabadi, F. Villanustre, T. M. Khoshgoftaar, N. Seliya, R. Wald, E. Muharemagic, Deep learning applications and challenges in big data analytics, Journal of big data 2 (1) (2015) 1–21.

[A4] M. A. Nielsen, Neural networks and deep learning, Vol. 25, Determination press San Francisco, CA, 2015.

[A5] I. Goodfellow, Y. Bengio, A. Courville, Y. Bengio, Deep learning, Vol. 1, MIT press Cambridge, 2016.

[A6] F. Fleuret, Deep learning mini-projects, https://fleuret.org/dlc/materials/dlc-miniprojects.pdf, accessed: 2021–05-27 (2021).

[A7] Y. LeCun, C. Cortes.

[A8] Y. Lecun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE 86 (11) (1998) 2278–2324. doi:10.1109/5.726791.

[A9] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition (2015). arXiv:1409.1556.