MAI-OR Practical Block 2: **Object Classification using CNN**

Gonzalo Recio and Jana Reventós— 19th May 2020

1 Introduction

Computer vision field has a wide range of applications and one of the most studied is object classification problem, which consists in classifying objects in different categories. Traditionally, this was done by extracting descriptors or features (i.e. HOG, SIFT, Haar-like, SURF, etc.) and then use them as input for classifiers. Nevertheless, the recent appearance of deep learning with convolutional neural networks (CNN) has revolutionized the object recognition task obtaining state-of-the-art results.

In this practical work we aim to learn how to perform object classification using deep learning methods and understand the role of the main tuning parameters available such as the simulation of lack of data, batch normalization and data augmentation. By using different parameter configurations we will analyze the performance of the CNN in the training, validation and test sets.

1.1 Dataset

The dataset selected for studying the perfomance and parameter impact of convolutional neural networks is the well-known dataset MNIST [4]. As shown in figure 1, it consists in a labelled collection of handwritten digits samples from 3600 writers. The goal is to classify each handwritten digit to its corresponding number label ”0”-”9”.

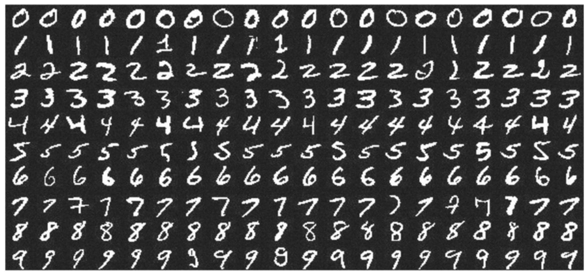


Figure 1: Synthetic images of handwritten digits dataset (MNIST)

2 Convolutional Neural Networks

Convolutional Neural Networks (CNN) have shown a great performance when it comes to image classification tasks, beating the all the previous proposed machine learning approaches and becoming the state-of-the-art models for solving computer vision problems [1]. In particular, they shifted the problem of feature extraction (previously tackled with handcrafted descriptors) to the CNNs, which now they are also in charge of extracting the appropriate feature maps using convolutional layers.

For this work, we are going to consider two CNN versions:

- Version 1 (`CNN-v1`): consists of a single convolutional layer and one fully connected layer.
- Version 2 (`CNN-v2`): contains 4 convolutional layers and one fully connected layer.

In both network versions, convolutional layers have ReLu activation function and a max-pooling layer after them. Also, they use batch normalization in-between convolutional layers, but since we want to study the effect of using batch norm. layers, they are going to be removed in some experiments.

3 Experiments

3.1 Baseline training

Firstly, we start by computing the baseline training to compare quantitatively with other executions. To do so, we trained `CNN-v1` with 4000 data samples. In figure 2, the training and validation curves of accuracy and loss are plotted.

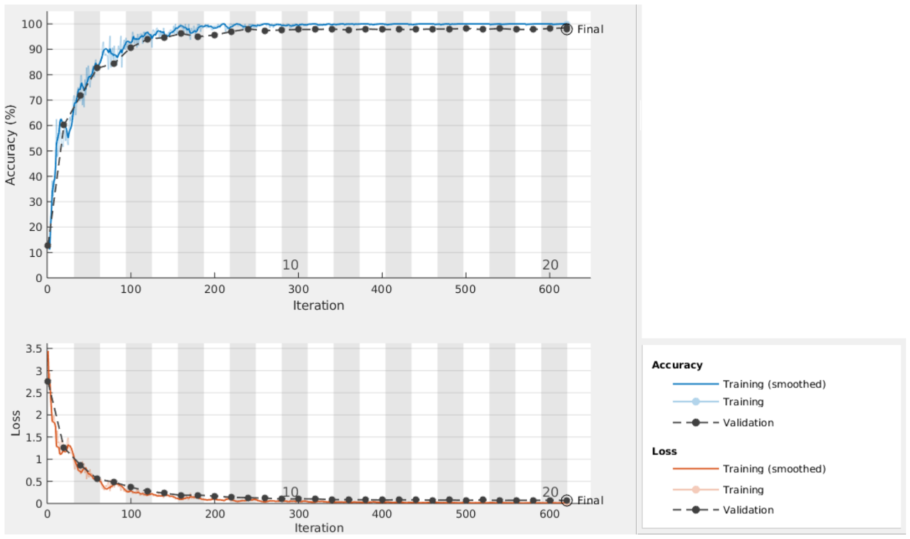


Figure 2: Accuracy and loss curves of the baseline training with `CNN-v1` model.

We can observe that the validation curves (black with dashed lines) are very close to both train accuracy and train loss curves. This shows that there is not any sign of overfitting during the training process and that the network robustly learned the classification task. Furthermore, with `CNN-v1` network we achieved a **0.9820** test accuracy. Regarding the training time, the network took 31s to train, hence, each epoch took 1.55 seconds, and converged rapidly to almost 0 loss (we remark that this CNN uses batch normalization in our baseline training).

3.2 Simulating lack of data

In this section we are going to simulate the lack of data by reducing the number of images in the training set to 1000, to 500, and then to 300. Note that the `epochs_factor` is computed to compensate the limited number of samples, but we also wanted to study the behaviour when `epochs_factor` is set to 1. In table 1, we can see the baseline execution in the first row and then the executions with 1000, 500 and 300 training samples with `epochs_factor` computed or set to 1.

# Train. samples	Epoch factor	BatchNorm	Epochs	Interations	Train. time (s)	Valid. acc.	Test acc.
4000	1	True	20	620	31	97.80	98.20
1000	5	True	100	700	36	87.05	86.50
1000	1	True	20	140	8	86.45	87.75
500	10	True	200	600	32	78.80	75.78
500	1	True	20	60	3	72.84	71.96
300	17	True	340	680	36	66.21	64.43
300	1	True	20	40	2	61.45	62.81

Table 1: Executions reducing the number of instances of train data with `CNN-v1`

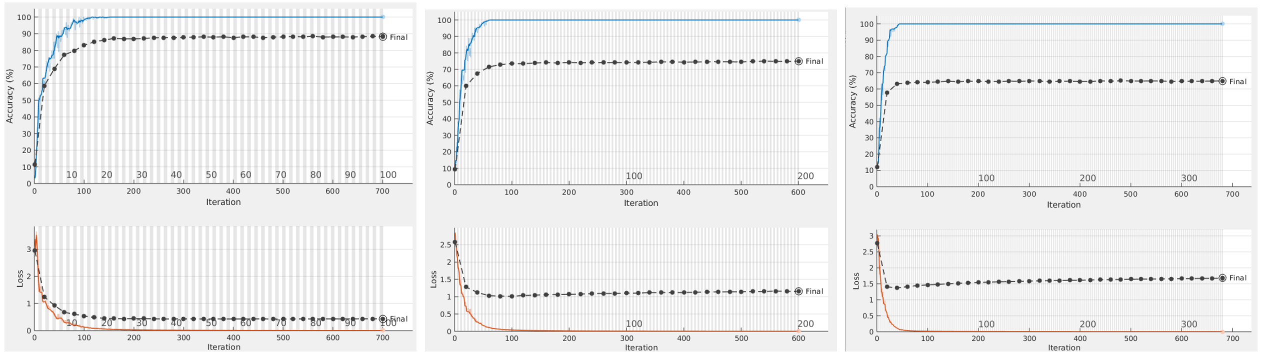

(a) 1000 train instances (b) 500 train samples (c) 300 train samples

Figure 3: Accuracy and loss curves for train (blue and orange, respectively) and validation (black dashed lines).

First thing we can notice from table 1 is that the computation of `epoch_factor` helps to achieve better results because it allows the model to train for longer and compensate the lack of data. In addition, we can see that test accuracy significantly drops proportionally with the lack of training samples. As it can be observed in figure 3, we show the accuracy and loss curves for each experiment, and we can notice that the gap between train and validation curves increases with the reduction of train size. More concretely, as shown in figure 4, accuracy seems to decay following a logarithmic tendency when decreasing the training samples, not showing a linear relation.

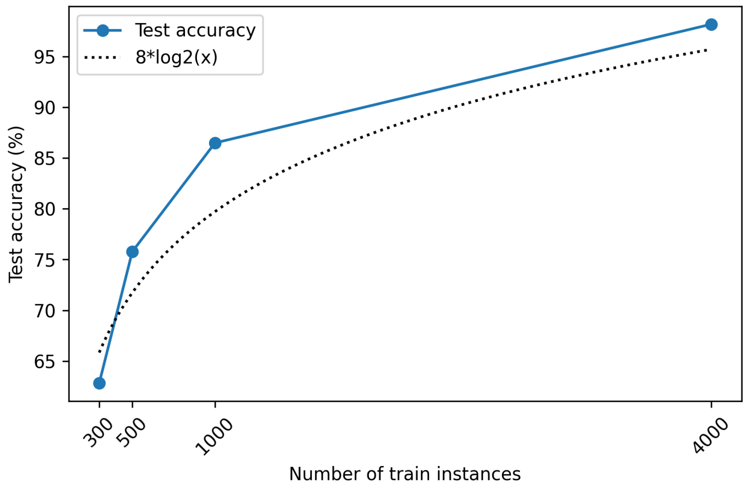


Figure 4: Test accuracy achieved along the number of training instances showing a logarithmic improvement tendency with respect to the number of used samples

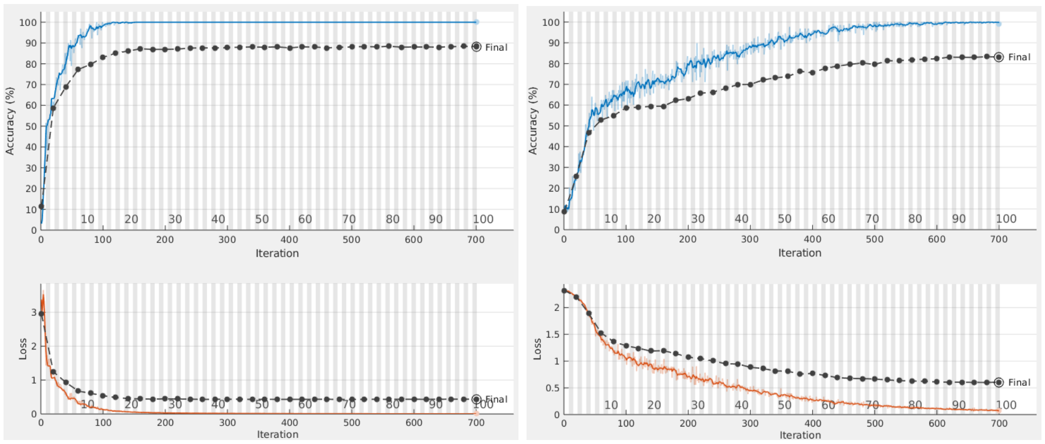
3.3 Influence of Batch Normalization

Batch normalization (BN) is used to normalize all features extracted from convolutional layers between 0 and 1, and it has been proven to speed up learning by improving model training convergence and also to reduce internal covariance shift between hidden units [3].

To analyze their behaviour, we are going to execute `CNN-v1` with 1000 and 300 train samples **without** the batch normalization layer and compare the results with the previous executions. Also, we will execute with `epoch_factor` computed and set to 1. In table 2 the results of this execution are shown.

# Train. samples	Epoch factor	Batch Norm.	Train. time (s)	Valid. acc.	Test acc.
1000	5	True	36	87.05	86.50
1000	5	False	32	82.55	83.00
1000	1	True	8	86.45	87.75
1000	1	False	6	61.55	60.75
300	17	True	36	66.21	64.43
300	17	False	33	58.72	59.62
300	1	True	2	61.45	62.81
300	1	False	2	38.55	37.32

Table 2: Comparison between using batch norm. (cyan rows) and not using batch norm (white rows) on `CNN-v1`



(a) 1000 train samples **with** batch norm. (b) 1000 train samples **without** batch norm.

Figure 5: Comparison of the convergence between using and not using batch normalization with `CNN-v1` model.

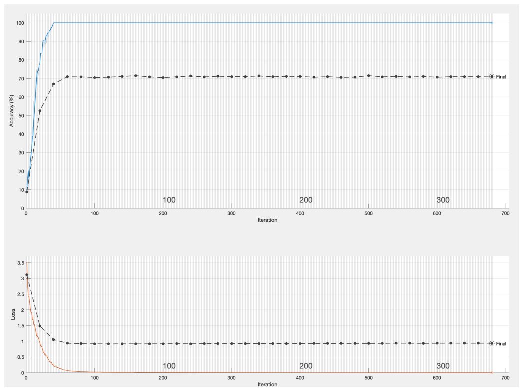
From table 2, we can see that test accuracy drops significantly when not using batch normalization: table rows in cyan color (batch norm.) have better test accuracy results than white rows (without batch norm.). In addition, in figure 5 we can clearly notice that the model converged much faster when batch norm. is applied since loss and accuracy curves rapidly converged to optimal values within the first iterations (figure 5(a) vs. 5(b)).

Now, we are going to test batch normalization with a more complex network (`CNN-v2`) only using 300 training instances.

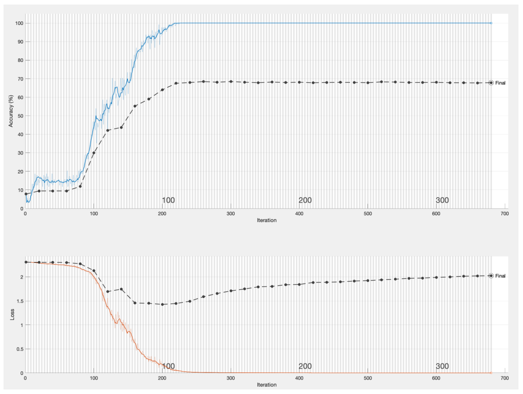
# Train. samples	Epoch factor	Batch Norm.	Train. time (s)	Valid. acc.	Test acc.
300	17	True	55	71.06	**72.17**
300	17	False	49	67.83	69.32
300	1	True	4	66.89	64.30
300	1	False	3	16.09	14.85

Table 3: Results using batch normalization on `CNN-v2`

Table 3 shows that BN definitely helps with model convergence. When `epochs_factor` is set to 1, which means that only 20 epochs are executed, using BN the model reaches 0.64 test accuracy while without BN it gets stuck at 0.14 test accuracy. This improvement of the model convergence can also be clearly visualized in figure 6, where we can see that the model struggles to reach and stabilize at the optimal configuration when BN is not used (figure 6(a) vs. 6(b)).



(a) 300 train samples **with** batch norm.



(b) 300 train samples **without** batch norm.

Figure 6: Comparison of the convergence between using and not using batch normalization with `CNN-v2` model.

3.4 Data augmentation

Until now, we have seen that using only 300 data instances to train gives poor accuracy results. However, we observed that batch normalization helps to improve the CNN performance and that `CNN-v2` managed to beat `CNN-v1` with a test accuracy of **0.7217** (using the computed `epoch_factor` that allows to train for some more epochs).

Nontheless, we know that our model is able to learn the classification task with accuracy of almost 1.0, as observed with the baseline training results (section 3.1), but it gets stuck at 0.72 of test accuracy because of the few data samples used to train it. Regarding that the bad performance is due to the lack of data (as mentioned in section 3.2), we can use data augmentation techniques to see if we can improve the results.

Data augmentation is a strategy that enables to significantly increase the diversity of data available for training by applying some small deformations to the dataset samples. In our study we considered the following augmentation techniques:

- **Rotation**: Randomly rotate the images with an angle up to 20 or -20 degrees.
- **(X, Y, XY)-Translations**: Randomly translate the images up to three pixels horizontally (X), vertically (Y) or both (XY).
- **(X, Y, XY)-Reflections**: Randomly flip the images with respect to x-axis (X), y-axis (Y) or both (XY).

# Train. samples	Layer	Data Augmentation	Train. time	Valid. acc.	Test acc.
300	v2	Rotation	1min 21s	86.09	85.36
300	v2	Rot + XTrans	1min 2s	91.91	91.23
300	v2	Rot + YTrans	1min 2s	92.09	92.13
300	v2	Rot + XYTrans	1min 1s	91.40	**92.77**
300	v2	Rot + XYTrans + XRef	55s	85.70	86.34
300	v2	Rot + XYTrans + YRef	1min 3s	92.30	91.79
300	v2	Rot + XYTrans + XYRef	1min 2s	79.96	77.49

Table 4: Results of using data augmentation with `CNN-v2`

Table 4 presents the accuracy achieved with each combination of data augmentation techniques. We can observe that in general using these techniques helped to significally improve model performance. However, using Reflections does not seem to improve the performance at all and, in the case of XY-Reflection definitely makes predictions worse. It make sense because reflecting digit images produce a change the meaning of the numbers. For instance, Y-Reflecting "6" makes the digit look like a "9", and X-Reflecting "2" may convert the number to a "5", and vice-versa (although numbers such as "0", "1" and "8" could benefit from these transformations). Therefore, reflections might difficult the task for the CNN, which coincides with the results in table 4, which shows that best results are achieved with Rotations + XY-Translations (without using Reflections).

4 Conclusions

Throughout this work, we have seen that CNN models excel in object recognition tasks. However, in challenging situations such as lack of data, we need to come up with appropriate techniques and parameter configurations in order to make the most of CNNs.

Lack of data showed to produce a significant drop in model performance and test accuracy. We noticed that this decrease is proportional to the number of training samples following a logarithmic relation (not linear), as shown in figure 4. This situation is common in computer vision problems since having a large annotated image dataset can sometimes be a laborious task, although we have proven that there are existing techniques, like data augmentation, that can help to solve this problem.

Regarding batch normalization, we have observed that it helped the models to converge faster: training accuracy and loss curves converged much faster (figures 5 and 6) and, hence, learning times are much lower if *early stopping* is used. Specially when working with more complex networks, such as `CNN-v2` in our case, BN achieved a really good model convergence in contrast with not using BN (figure 6) − remark that training heavy networks (e.g. VGG16 [5], ResNet50 [2]) may consume a lot of resources, and BN can be a crucial factor since we have seen that it can speed up training times a lot. This acceleration of convergence can be explained because in CNNs, features maps are extracted by convolutional layers and if their outputs are not normalized, it is similar as feeding a model with unscaled features, which can add difficulty to the gradient descent process [3]. Furthermore, batch normalization also improved the test accuracy significantly, helping to reach up to +5% of acc. improvement.

As for data augmentation, we observed that it is one of the most useful approaches when dealing with lack of data. We managed to improve the test accuracy of `CNN-v2` from 0.72 up to **0.92** (+20% of acc. improvement!) showing a great learning robustness. Nevertheless, we have to be careful when applying data augmentation techniques and consider which is the domain of the dataset samples. Reflection deformations did not work positively for our model, since digits may lose their meaning when being flipped. For instance, similar thing happens with images for face recognition: horizontal flipping make sense to obtain a symmetric face, but flipping faces vertically may just only complicate the task for the model. Therefore, we first need to think of which data augmentation techniques are suitable for our data, and reflections/flips make no sense in MNIST dataset, as it can quantitatively be observed in table 4.

References

[1] A. Canziani, A. Paszke, and E. Culurciello, *An analysis of deep neural network models for practical applications*, 2016.

[2] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), (2016).

[3] S. Ioffe and C. Szegedy, *Batch normalization: Accelerating deep network training by reducing internal covariate shift*, 2015.

[4] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, *Gradient-based learning applied to document recognition*, Proceedings of the IEEE, 86 (1998), pp. 2278–2324.

[5] K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, arXiv preprint arXiv:1409.1556, (2014).