UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

Master in Innovation and Research in Informatics

Algorithm Data Mining

Apply CNN on MNIST data set

Second Assignment

Mohana Fathollahi

April 30, 2022

1 Introduction

In this assignment, convolutional neural network, CNN, has been applied on MNIST data set. In this dataset we have images that we want to detect class of each image based on 10 classes that have been defined. More detail of this dataset will be described in Dataset section. In the following, brief description about CNN has been provided, hyper parameters for building a network described and then dataset and training a model on that has been provided.

2 CNN

CNN is a form of artificial neural network but differentiate form multi layer perceptron, MLP. CNN has hidden layers called convolutional layers and CNN can have other non convolutional layers as well.In the figure 1 different layers and relation between them have been shown. Convolutional layers like other layer receive input and transform it and produce output. This transformation is called convolution operation. Every image has some details, such as shape, objects, multiple edges and so on that called pattern in image. Convolutional layers can help us to detect these pattern.
How can we detect patterns with these layers? When we add convolution layers to a model, we specified how many filters we want to have. Filter is relatively small matrix and values in this matrix is randomly initialized. For example in figure 1 two convolutional layers have been defined and size of each of them is 5*5. Therefore, dot product of first 5*5 pixels in the input with 5*5 filter will be computed and stored. This procedure should be done for all input and the result of dot products will be input for next layer.

Other parameters that are defined in convolutional layers are in_channels and out_channels, in_channels are number of channels in the input image, number of channels of the image. For RGB image, in_channels = 3 (red, green and blue); for a gray image, in_channels = 1. Out_channels are number of channels in the output image. Number of feature maps, which is often equivalent to the number of kernels that you apply to the input Larger number of out_channels, allow layer to learn more features.[1]

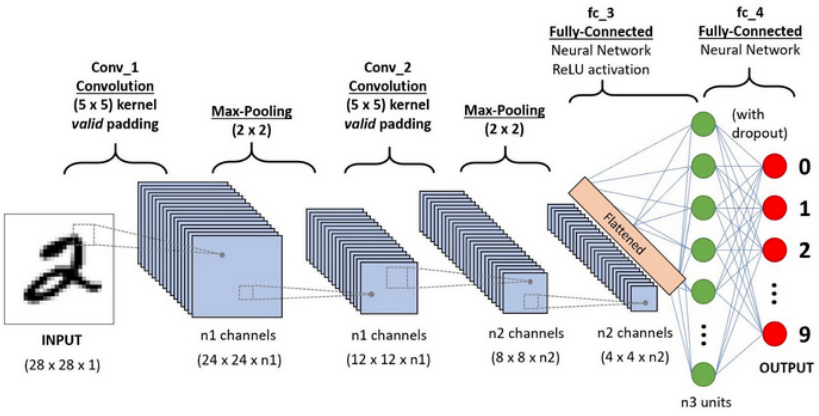

Figure 1: Structure of CNN

2.1 Hyper parameters

- **Batch size:**
  Our training set has 60,000 images and we cannot pass this amount of data to computer at once, therefore Mini-Batch Gradient Descent has been used. We need to define batches, in each batches we have same number of data, for example if our batch size is 100, we should have 600 batches. .
  We have 10,000 test set and we should define batch size for test set too.

- **Epoch:**
  In each epoch we train model with all training set, it means that we are using all batches in each epoch. Therefore, in one epoch, the number of iteration is equal to number of batches.
  Pass training data just one time is not enough, because we are using limited number of dataset and to optimise learning procedure, we should train model and update weights more than one time. In the figure 2 three cases have been shown, if we use one Epoch our model will be underfiting. As a result, we need to define more than one epoch. When more than one epoch has been used, our model update weights more than before and model moves to optimal and maybe overfitting.[2]

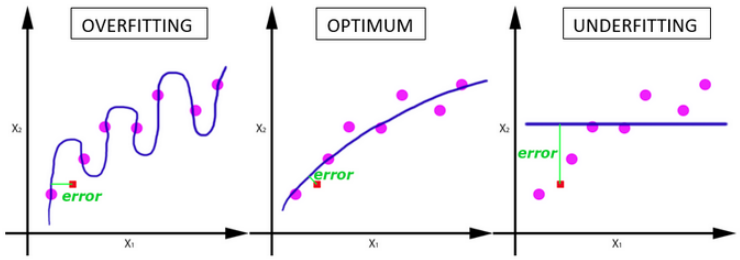

Figure 2: Different result of learning a model

How can we find an optimal value for Epoch to not have underfiting or overfitting model. This hyperparameter is depend on dataset, if we have more diverse dataset, we need more epochs.

- **learning rate:**
  Based on below formula, we should update weights in each step, therefore it is important to know which value is good for learning rate.
  $new\_weight = existing\_weight - learning\_rate * gradient$
  Four different scenarios can happen when we choose different values for learning rate, figure 3. If we choose very high value for learning rate, our step will be so large and we may not converge to local minima, yellow curve. On the other side when we pick small value for learning rate, our steps will be so small to converge to local minima and take more time especially when we stuck in a plateau region. Based on below figure, we have more chances to converge when we pick small value for learning rate.[3]

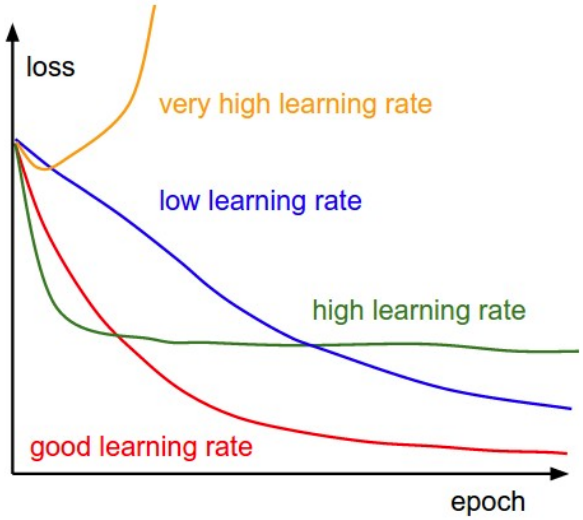

Figure 3: Effect of Learning rate

- **momentum:**
  To improve performance of gradient descent, we need to differentiate between learning rate in horizontal and vertical axis. Therefore, it reduces high variance in Stochastic Gradient Descent, SGD, and softens the convergence. It improves performance by accelerating the convergence towards the relevant direction and reduces the fluctuation to the irrelevant direction.

2.2 Dataset

This data set contains ten digits from 0 to 9. Dimension of each image is 28×28 and represented as a form of matrix, additionally because the images are grey, depth or the number of channel is 1. If images are colourful, such as RGB, the number of channels would be three. As mentioned before our train set has 60,000 images and test set has 10.000 images. These

images are different handwritten for numbers from 0 to 10, In the figure 4 different kind of 7, 4, 8 and 3 have been shown.
To import data set and build a network, PyTorch has been used as a deep learning framework.

With data loader, we can randomly select from training and test set batch size for each of them. In the first step, we should normalize data in train and test separately. The values 0.1307 and 0.3081 are global mean and standard deviation for MNIST dataset and these values used for normalization.

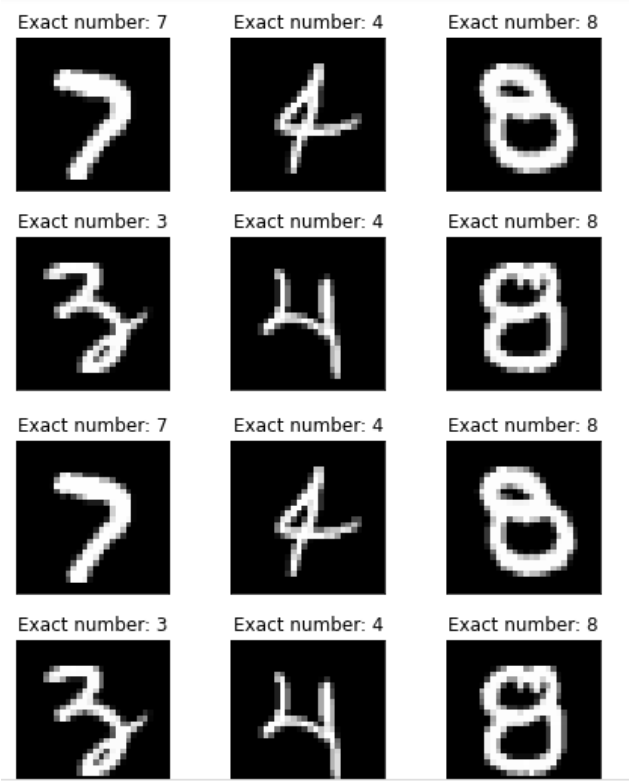

Figure 4: Example of test set

2.3 Network

In PyTorch a nice way to build a network is by creating a new class for the network. To build network, two convolution layers have been used, and in_channel for first layer is one because our picture colour is grey, and 10 for out_channel. For the next convolutional layer in_channel is equal to out_channel. One of problems that we may have in network is overfiting, to overcome to this problem we can use Dropout. There are two types of dropout; dropout and dropout2d.
The problem of regular dropout in CNN is that adjacent pixels are highly correlated and may not help to reduce dependency as much as in other cases except images. Therefore, it was a motivation to use dropout2d that proposed Spatial Dropout, which drops channels instead of individual units.

2.3.1 Define size of input and output

As we saw in figure 1 size of each layer changed, these changes should be applied when we are building network. There is a formula that help us to calculate size of output for each layer. After applying convolutional layer, size of input change based on below formula:

Table 1: parameters for calculating output size

Input size	filter	padding	stride	Output size
n*n	f*f	p	s	o

$$O = (\frac{n - f + 2p}{s}) + 1$$

In table 1 there are padding and stride, in below these two parameters will be described.

- Padding: In padding we add empty pixels around the edges of an image. The purpose of padding is to preserve the original size of an image when applying a convolutional filter and enable the filter to perform full convolutions on the edge pixels.
- Stride: This parameter is related to Max pooling layer. This layer will be added after convolutional layer and the goal of this layer is reducing dimension and extract necessary features, based on figure 5 when our input is 4*4 and we used stride=2, we get maximum between 4 items like in the figure 5 and output size will be half of input size.

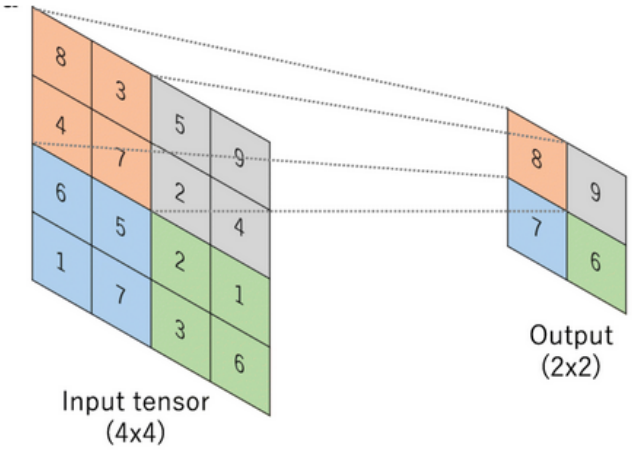

Figure 5: Max pooling

For example, output size for first convolutional layer will be:

$$O = \frac{28 - 5 + 0}{1} + 1 = 24$$

For other output size we can replace values like above and calculate output size.

2.3.2 optimization algorithm

Stochastic Gradient Descent, SGD has been used as an optimization algorithm. It's a variant of Gradient Descent and tries to update the model's parameters more frequently so that, on each training example, parameters for model are altered after computation of loss.[4] As mentioned before Momentum has been used to improve performance of SGD too.

2.4 Train the model

First the gradients should set to zero by; optimizer.zero_grad(), because PyTorch by default accumulates gradients.
Then we produce the output of our network (forward pass) and compute a negative log-likelihodd loss between the output and the ground truth label by nll_loss.
Then loss.backward() calculates gradients and updates weights with optimizer.step().[5]

3 Result of model

After training and running model on test set, table below shows how accuracy could improves and loss decrease in each Epoch, in the figure 6 this comparison has been shown.

Table 2: Accuracy and loss in test set

	Initial	Epoch1	Epoch2	Epoch3	Epoch4
Accuracy_Average	9%	93%	95%	97%	97%
Loss _Average	2.3096	0.2596	0.1540	0.1137	0.0968

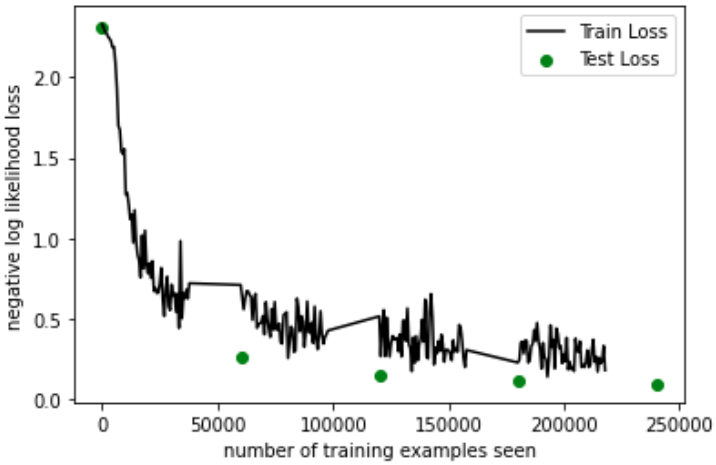

Figure 6: Compare between train and test loss

In the figure 7 result of applying model on example, that used in first part, has been provided and as we can see model predict numbers correctly.

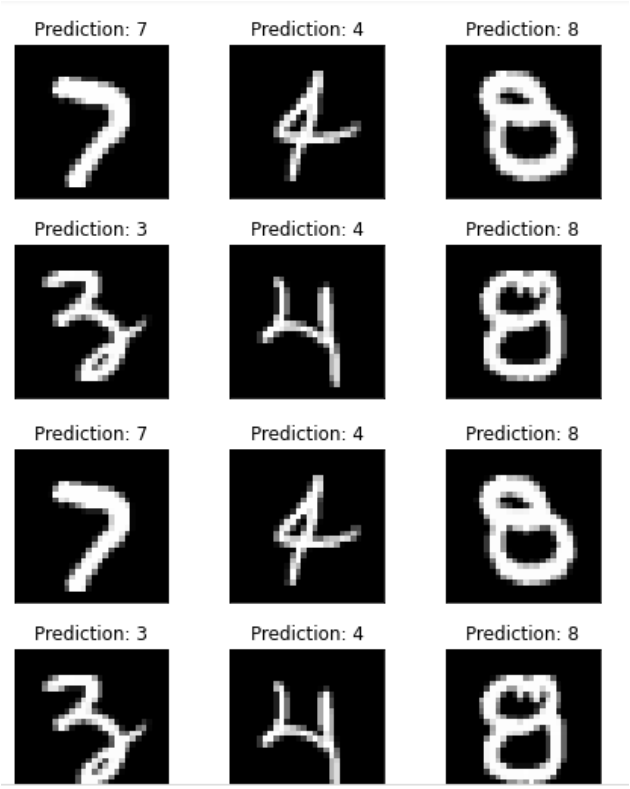

Figure 7: Result of prediction on example set

References

[1] *CNN*. URL: https://deeplizard.com/learn/video/YRhxdVk_sIs.

[2] *Epoch and Batch size*. URL: https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9.

[3] *Learning rate*. URL: https://towardsdatascience.com/understanding-learning-rates-and-how-it-improves-performance-in-deep-learning-d0d4059c1c10.

[4] *SGD*. URL: https://courses.smp.uq.edu.au/MATH7502/2019/projects/Summary_group5.pdf.

[5] *Training model*. URL: https://medium.com/analytics-vidhya/complete-guide-to-build-cnn-in-pytorch-and-keras-abc9ed8b8160.