Memory Technologies for Machine Learning - EITP25
Assignment 1 - Spring 2020

Playing around with CNNs

Hicham Mohamad, hi8826mo-s

6 April 2021

1 Introduction

In this assignment, we are going to deal with **convolutional** approach of **Neural Networks** CNN for image processing. In particular, we will train a CNN for a 10-class classification problem, that involves the `MNIST` image dataset of handwritten digits, consisting of data-target pairs with images $\mathbf{X}_i \in [0,1]^{28\times 28}$ and targets $t_i \in \{0, 1, \cdots, 9\}$. We will use an online interface [4] that allows us to build and test a convolutional network.

2 Background

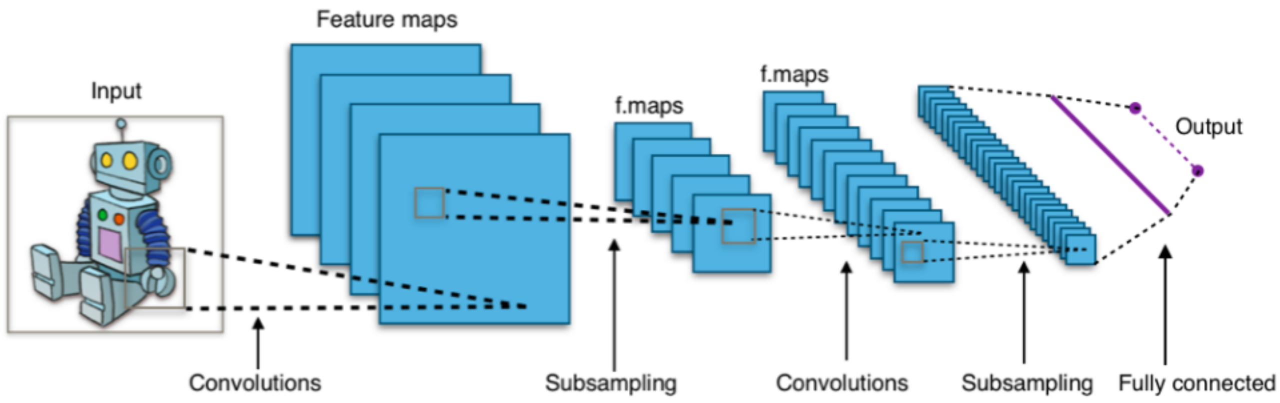


Figure 1: A CNN architecture. The term ***subsampling*** *used in the figure is the same thing as the pooling layer. In the figure, the network will learn four kernels in the first layer; in the second layer, it will subsample the images, keep one pixel every four pixels, for instance.*

Convolutional neural networks, CNN

In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of **deep neural networks**, most commonly applied to analysing visual imagery, and being used for image classification problem. In **fully connected networks**, each neuron in one layer is connected to all neurons in the next layer. The "fully-connectedness" of these networks make them prone to **overfitting** data. Typical ways of regularization includes adding some form of magnitude measurement of weights to the loss function. However, CNNs take a different approach towards regularization: they take advantage of the **hierarchical pattern** in data and assemble more complex patterns using smaller and simpler patterns.

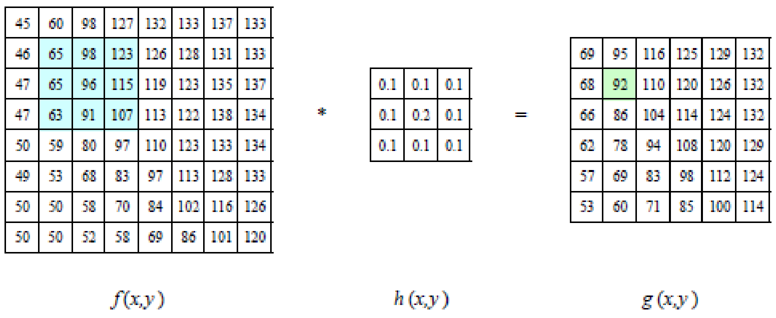
The big idea behind CNNs is that a local understanding of an image is good enough. The practical benefit is that having fewer parameters greatly improves the time it takes to learn as well as reduces the amount of data required to train the model. Instead of a fully connected network of weights from each pixel, a CNN has just enough weights to look at a small patch of the image. It's like reading a book by using a magnifying glass; eventually, you read the whole page, but you look at only a small patch of the page at any given time.

Filtering

To understand how the CNN architecture works we look at the concept of image filters. Assume the our image is given by $f(i,j)$, for simplicity we can assume that each pixel is a scalar. A "filtered" image is mathematically described by the process of **convolution**. The filter is represented by a **kernel**, $h(n,m)$ and the convolved (filtered) image $g(i,j)$ is given by,

$$g = f * h \implies g(i,j) = \sum_u \sum_v f(i-u, j-v)h(u,v) \tag{1}$$

Actually, the process of applying the filter is usually referred to as convolution. This can be seen clearly in Figure 2 using an example.



45	60	98	127	132	133	137	133
46	65	98	123	126	128	131	133
47	65	96	115	119	123	135	137
47	63	91	107	113	122	138	134
50	59	80	97	110	123	133	134
49	53	68	83	97	113	128	133
50	50	58	70	84	102	116	126
50	50	52	58	69	86	101	120

*

0.1	0.1	0.1
0.1	0.2	0.1
0.1	0.1	0.1

=

69	95	116	125	129	132
68	92	110	120	126	132
66	86	104	114	124	132
62	78	94	108	120	129
57	69	83	98	112	124
53	60	71	85	100	114

$f(x,y)$ $h(x,y)$ $g(x,y)$

Figure 2: Convolution: The image on the left is convolved with the filter in the middle to yield the image on the right.

Here is a small numerical example where a 4×4 image matrix is filtered by a 2×2 kernel.

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} \star \begin{bmatrix} -1 & 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} (-1+2+0+6) & (-2+3+0+7) & (-3+4+0+8) \\ (-5+6+0+10) & (-6+7+0+11) & (-7+8+0+12) \\ (-9+10+0+14) & (-10+11+0+15) & (-11+12+0+16) \end{bmatrix} = \begin{bmatrix} 7 & 8 & 9 \\ 11 & 12 & 13 \\ 15 & 16 & 17 \end{bmatrix}$$

We notice that the filtered image is smaller than the starting image. For CNNs there is also a choice of moving the filter more than one pixel when convolving the image. The amount of pixels the filter is moved is called **stride**. How can we make sure that the filtered image has the same dimension as the original image. The solution is called padding where **zero-padding** being the most common strategy. *Zero-padding consists of adding a boundary of zeros to the original image so that the resulting filtered images has the desired dimensions*, as shown in Figure 3.

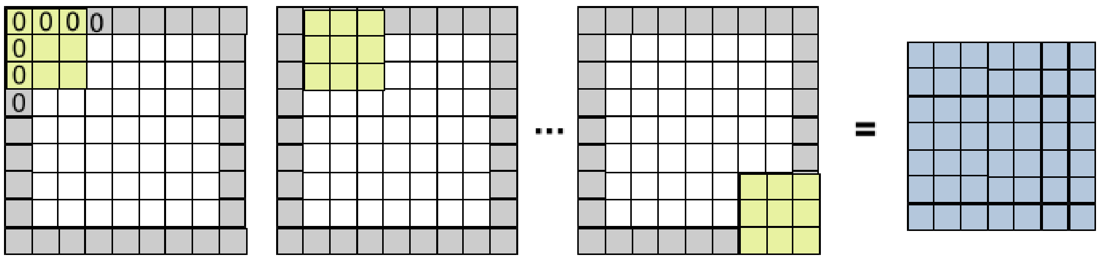


Figure 3: Zero-padding the images with a boundary of zeros will make sure that the filtered image has the same size as the original image.

To know the size of the resulting filtered image we can use the following formula for both width and height

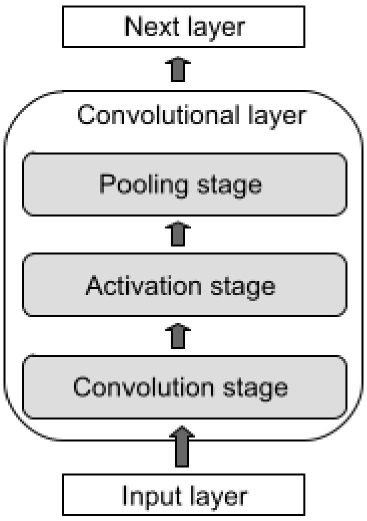
$$\text{output width} = \frac{W - K_w + 2P}{S_w} + 1, \qquad \text{output height} = \frac{H - K_h + 2P}{S_h} + 1 \tag{2}$$

where P = padding, S = stride and K = kernel/filter size.

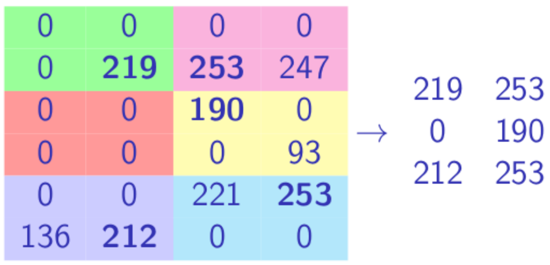
CNN building blocks

Following the terminology used in the Deep Learning Book [5], a **convolutional layer** consists of three blocks, as shown in Figure 4a, the convolution stage, the activation stage and the pooling stage. The first two are vital, but the pooling stage can many times be omitted.

The **activation** stage typically consists of using the rectified linear (ReLU) activation for the hidden nodes. The **pooling** stage consists of replacing the hidden layer with a new layer that can be said to summarize the neighborhood of a given hidden node. A very common pooling operation is **max pooling**. *Max pooling consists of replacing the hidden node value by the maximum value of a small neighborhood around the hidden node.* The term **Subsampling** is generally used and carried out using the max-pooling operation.



(a)



(b)

Figure 4: a) The three building blocks of a convolutional layer in a CNN. The convolutional stage, the activation stage and the pooling stage. b) An example of maxpooling by using a tile of size (2, 2), we have reduced the image size from (6, 4) to (3, 2).

3 Demo - Default network configuration and Training

In this assignment, we use **ConvNetJS MNIST demo** for training a Convolutional Neural Network on the MNIST digits dataset in browser. To improve the generalization performance of this network, **Data Augmentation** technique is performed by taking a 28x28 MNIST image and crops a random 24x24 window before training on it.

Default configuration and training of this network is shown in List 1. First the definition of the layers is done by pushing layer descriptions into a list `layer_defs`. At the end, after building the default convolution network, the training is implemented using the Stochastic Gradient Descent SGD with **Adadelta** optimizer, which is one of **per-parameter adaptive step size** methods. In this way, we don't have to worry about changing learning rates or momentum over time.

This MNIST demo also provides a **Network Visualisation**, where it is possible to see a graphical representation of the layers and the data as it flows through the layers. These views can be useful to evaluate how the network relates to the **abstract features** of the data.

It is possible to make changes to the network architecture by just changing the lines in the "Instantiate a Network and Trainer" window, pressing "change network", and then restarting the training in the "Training Stats" window. This is illustrated in Listing 1.

Listing 1: CNN default configuration and Trainer.

```
layer_defs = [];
layer_defs.push({type:'input', out_sx:24, out_sy:24, out_depth:1});
layer_defs.push({type:'conv', sx:5, filters:8, stride:1, pad:2,
                 activation:'relu'});
layer_defs.push({type:'pool', sx:2, stride:2});
layer_defs.push({type:'conv', sx:5, filters:16, stride:1, pad:2,
                 activation:'relu'});
layer_defs.push({type:'pool', sx:3, stride:3});
layer_defs.push({type:'softmax', num_classes:10});

net = new convnetjs.Net();
net.makeLayers(layer_defs);

trainer = new convnetjs.SGDTrainer(net, {method:'adadelta',
                                         batch_size:20,
                                         l2_decay:0.001});
```

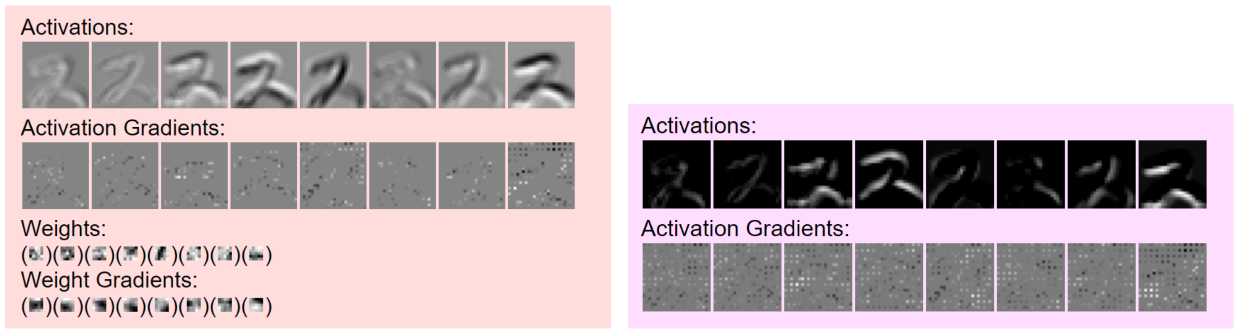
4 Task 1 - Default network

As a model, we start with the proposed CNN provided to solve this classification problem. It consists of the following:

- The **input layer** of size $(24 \times 24 \times 1)$, it is a placeholder for the input we give the network.
- **First convolutional layer** consisting of 8 kernels of size $(5 \times 5 \times 1)$, with ReLU activation of size $(24 \times 24 \times 8)$ and Maxpooling of size $(12 \times 12 \times 8)$.
- **Second convolutional layer** of 16 kernels of size $5 \times 5 \times 1$, with ReLU activation of size $(12 \times 12 \times 16)$ and Maxpooling size $(4 \times 4 \times 16)$.
- **Fully connected layer** $(1 \times 1 \times 10)$, with Softmax activation function of size $(1 \times 1 \times 10)$ because we have a 10-classification problem.

Question A

After training the default CNN model on the MNIST digits dataset in browser and running to 30000 examples, we need to study the accuracy and analyse the resulting loss function. Graphical results are shown in Figure 5.



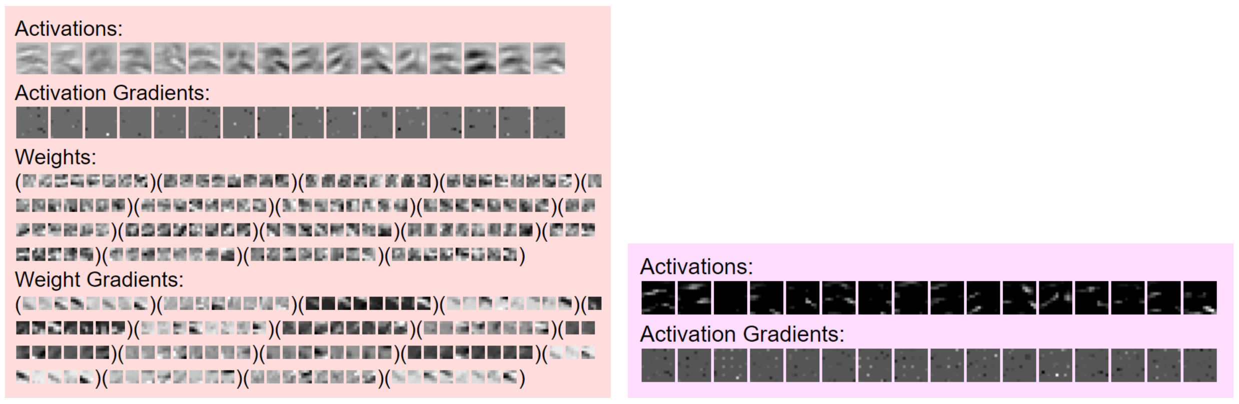
(a) (b)

Figure 5: Graphical representation of first conv layer. a) conv (24x24x8), filter size 5x5x1, stride 1. Max activation: 3.4125, min: -4.46508. Max gradient: 0.00001, min: -0.00002. Parameters: 8x5x5x1+8 = 208. b) ReLU (24x24x8), max activation: 3.4125, min: 0. Max gradient: 0.00001, min: -0.00002.

In List 2, we note down the obtained training and validation accuracy at this point. Training accuracy means how well the network classified on the training data set. Validation accuracy means how well it performed on the validation data set.

Listing 2: Training results in Question 1A.

```
Classification loss: 0.21679
L2 Weight decay loss: 0.00464
Training accuracy: 0.96
Validation accuracy: 0.94
```



(a) (b)

Figure 6: Graphical representation of second conv layer. a) conv (12x12x16), filter size 5x5x8, stride 1. Max activation: 6.53334, min: -18.55814. Max gradient: 0.00001, min: -0.00002. Parameters: 16x5x5x8+16 = 3216. b) ReLU (12x12x16). Max activation: 6.53334, min: 0. Max gradient: 0.00002, min: -0.00002.

Looking at the graph of the Loss (cost function) obtained in Figure 7, we can see that the shape of the curve converge rapidly at the beginning. It seems that it has saturated. But the gradient seems to fluctuate at the end causing a large variance in stochastic gradients and thus having a slower convergence rate. Thus, this noisy behaviour of the gradient slow down the convergence.

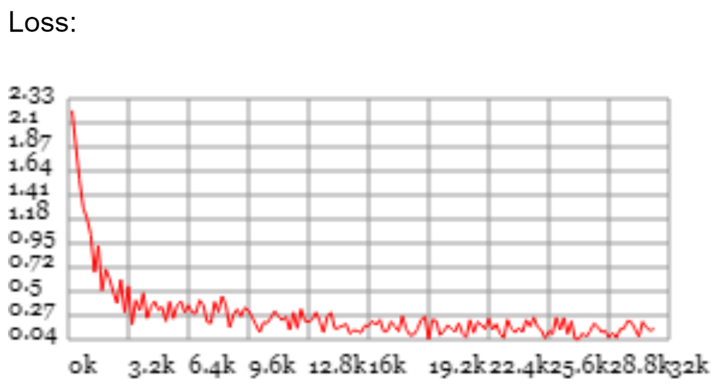


Figure 7: The resulting loss function after running 30000 examples.

Question B

In this question, we change the size of the convolutional kernel in the first conv layer and the second conv layer to 3, instead of 5 in the original size. Now the CNN model consists of the following:

- The **input layer** of size $(24 \times 24 \times 1)$, it is a placeholder for the input we give the network.
- **First convolutional layer** consisting of 8 kernels of size $(3 \times 3 \times 1)$, with ReLU activation of size $(26 \times 26 \times 8)$ and Maxpooling of size $(13 \times 13 \times 8)$.
- **Second convolutional layer** of 16 kernels of size $3 \times 3 \times 1$, with ReLU activation of size $(15 \times 15 \times 16)$ and Maxpooling size $(5 \times 5 \times 16)$.

- **Fully connected layer** $(1 \times 1 \times 10)$, with Softmax activation function of size $(1 \times 1 \times 10)$ because we have a 10-classification problem.

Then, we restart the training and let it again run to 30000 examples. In this way, when we have now a 3x3 kernel we produce a 26x26 filtered image, i.e. feature map, which is now larger than the original image. This decreasing of the kernel size affect also the performance getting worse accuracy, as shown in Listing 3. This may be explained due to the choice of having the kernel operating outside the bounds set by the image.

Listing 3: Training results in Question 1B.

```
Classification loss: 0.08078
L2 Weight decay loss: 0.00309
Training accuracy: 0.97
Validation accuracy: 0.92
```

Question C

For CNNs there is also a choice of moving the filter more than one pixel when convolving the image. The amount of pixels the filter is moved is called strid. In this question, we alter the stride in the two conv layers from 1 to 2. Now we get the following network configuration

- The **input layer** of size $(24 \times 24 \times 1)$, it is a placeholder for the input we give the network.
- **First convolutional layer** consisting of 8 kernels of size $(5 \times 5 \times 1)$, with ReLU activation of size $(12 \times 12 \times 8)$ and Maxpooling of size $(6 \times 6 \times 8)$.
- **Second convolutional layer** of 16 kernels of size $5 \times 5 \times 1$, with ReLU activation of size $(3 \times 3 \times 16)$ and Maxpooling size $(1 \times 1 \times 16)$.
- **Fully connected layer** $(1 \times 1 \times 10)$, with Softmax activation function of size $(1 \times 1 \times 10)$ because we have a 10-classification problem.

This increasing of the stride affects the accuracy as well, as shown in Listing 4.

Listing 4: Training results in Question 1C.

```
Classification loss: 0.24539
L2 Weight decay loss: 0.0037
Training accuracy: 0.92
Validation accuracy: 0.88
```

Question D

By increasing the number of filters in the first convolutional layer to 16 we get the following configuration

- The **input layer** of size $(24 \times 24 \times 1)$, it is a placeholder for the input we give the network.
- **First convolutional layer** consisting of 16 kernels of size $(5 \times 5 \times 1)$, with ReLU activation of size $(24 \times 24 \times 16)$ and Maxpooling of size $(12 \times 12 \times 16)$.
- **Second convolutional layer** of 16 kernels of size $5 \times 5 \times 1$, with ReLU activation of size $(12 \times 12 \times 16)$ and Maxpooling size $(4 \times 4 \times 16)$.
- **Fully connected layer** $(1 \times 1 \times 10)$, with Softmax activation function of size $(1 \times 1 \times 10)$ because we have a 10-classification problem.

This increasing of the number of filters in the first conv layer improves the accuracy and it increases the computational time as well, as shown in Listing 5.

Listing 5: Training results in Question 1D.

```
Classification loss: 0.12
L2 Weight decay loss: 0.00368
Training accuracy: 0.96
Validation accuracy: 0.97
Examples seen: 30187
```

5 Task 2 - Play around with the number of convolutional/pooling layers

Question A

A convolutional neural network uses trainable filters/kernels as the basic tool for analysing images, where the kernel is represented by the weight values of a given hidden node. An important property of the filters is that they only see a small part of the image.

As we know in an image analysis task, each of the filters will learn different **feactures** that must be useful for the task. By observing the features that the feature maps learn, it seems that the kernels learn to detect horizontal and vertical edges (differences), or other kind of features in the image, like circles, curves...

Question B

In this question, we add a third convolutional layer with pooling layer sx = 2 and stride = 1, to avoid decrease the data size. The new CNN model consists of the following:

- The **input layer** of size $(24 \times 24 \times 1)$, it is a placeholder for the input we give the network.
- **First convolutional layer** consisting of 8 kernels of size $(5 \times 5 \times 1)$, with ReLU activation of size $(24 \times 24 \times 8)$ and Maxpooling of size $(12 \times 12 \times 8)$.
- **Second convolutional layer** of 16 kernels of size $5 \times 5 \times 1$, with ReLU activation of size $(12 \times 12 \times 16)$ and Maxpooling size $(4 \times 4 \times 16)$.
- **Third convolutional layer** of 16 kernels of size $5 \times 5 \times 1$, with ReLU activation of size $(4 \times 4 \times 16)$ and Maxpooling size $(3 \times 3 \times 16)$.
- **Fully connected layer** $(1 \times 1 \times 10)$, with Softmax activation function of size $(1 \times 1 \times 10)$ because we have a 10-classification problem.

After running 30000 examples, this CNN configuration improves the learning accuracy best but it increases the computational time slightly, as shown in Listing 6.

Listing 6: Training results in Question 2B.

```
Classification loss: 0.01575
L2 Weight decay loss: 0.00405
Training accuracy: 1
Validation accuracy: 1
Examples seen: 30338
```

Question C

Now we remove all but the first convolutional / pooling layer. The new CNN model consists of the following:

- The **input layer** of size $(24 \times 24 \times 1)$, it is a placeholder for the input we give the network.
- **First convolutional layer** consisting of 8 kernels of size $(5 \times 5 \times 1)$, with ReLU activation of size $(24 \times 24 \times 8)$ and Maxpooling of size $(12 \times 12 \times 8)$.
- **Fully connected layer** $(1 \times 1 \times 10)$, with Softmax activation function of size $(1 \times 1 \times 10)$ because we have a 10-classification problem.

After running 30000 examples, this CNN configuration decreases the learning accuracy and the computational time, as shown in Listing 7.

Listing 7: Training results in Question 2C.

```
Classification loss: 0.12581
L2 Weight decay loss: 0.00394
```

```
Training accuracy: 0.97
Validation accuracy: 0.96
Examples seen: 30198
```

6 Task 3 - Finding the best possible network structure

Question A

Without tuning the hyper-parameters, learning rate and momentum, because we use the optimizer Adadelta, and using the implemented data augmentation technique, it is possible to optimize the network.

Data augmentation is a strategy that enables practitioners to significantly increase the **diversity of data** available for training models, without actually collecting new data. Data augmentation techniques such as cropping, padding, and horizontal flipping are commonly used to train large neural networks.

At the end, we can conclude the assignment having the following final configuration:

- The **input layer** of size $(24 \times 24 \times 1)$, it is a placeholder for the input we give the network.
- **First convolutional layer** consisting of 8 kernels of size $(5 \times 5 \times 1)$, with ReLU activation of size $(24 \times 24 \times 8)$ and Maxpooling of size $(12 \times 12 \times 8)$.
- **Second convolutional layer** of 16 kernels of size $5 \times 5 \times 1$, with ReLU activation of size $(12 \times 12 \times 16)$ and Maxpooling size $(4 \times 4 \times 16)$.
- **Third convolutional layer** of 16 kernels of size $5 \times 5 \times 1$, with ReLU activation of size $(4 \times 4 \times 16)$ and Maxpooling size $(3 \times 3 \times 16)$.
- **Fully connected layer** $(1 \times 1 \times 10)$, with Softmax activation function of size $(1 \times 1 \times 10)$ because we have a 10-classification problem.

Listing 8: Training results in Question 3A.

```
Classification loss: 0.06547
L2 Weight decay loss: 0.0041
Training accuracy: 0.97
Validation accuracy: 0.99
Examples seen: 30185
```

7 Appendix

References

[1] Mattias Borg, Memory Technologies For Machine Learning - EITP25, Lectures notes: `https://canvas.education.lu.se/courses/2101`

[2] Chen, Hutchby, Zhirnov and Bourianoff, Emerging Nanoelectronic Devices, Wiley 2015.

[3] Manan Suri: Advances in Neuromorphic Hardware Exploiting Emerging Nanoscale Devices. Springer, 2017, ISBN: 978-81-322-3701-3.

[4] ConvNetJS MNIST demo: `https://cs.stanford.edu/people/karpathy/convnetjs/demo/mnist.html`

[5] Ian Goodfellow, Yoshua Bengio, Aaron Courville. *Deep Learning.* MIT Press, 2016. An HTLM version of the book is available at `https://www.deeplearningbook.org/`.

[6] Introduction to convolutional neural networks (CNN) for classification: `http://www.robots.ox.ac.uk/~vgg/practicals/cnn/`

[7] Adit Deshpande, The 9 Deep Learning Papers You Need To Know About: `https://adeshpande3.github.io/The-9-Deep-Learning-Papers-You-Need-To-Know-About.html`