NEURAL NETWORKS IMPLEMENTATION

INITIAL CONSIDERATIONS

The goal of this assignment is to “Analyze how the structure of a network affects its prediction accuracy and how it depends on the size of training data”.

To make effective comparisons, I tried to select a “template neural network” and a “template convolutional neural network” and then modify different characteristics to evaluate if this changes affect positively or negatively the behavior of the net.

Moreover, I have decided to have a comparison, for every NN and CNN, between SGD and ADAM optimizers. In fact, SGD is a stochastic approximation of the Gradient Descent method: it replaces the actual gradient of the entire dataset with an estimated value derived from a randomly selected subset of the data. This avoids the risk of finding a local minimum, and also increases the speed of the minimization. However, even with SGD there is the risk to not correctly proceed towards the global minimum due to these random iterations. For that, the Momentum method has been introduced: ADAM is one of the most used optimizer to make SGD even more effective. Its aim is to speed up the optimization towards the minimum even with a relatively small number of training datasets automatically adapting the momentum, a parameter depending on previous iterations which is added to the gradient to regularized the movement of the parameters of the net.

Since the number of training data and the number of epochs strongly affect the results, I thought it was interesting to make the comparison between these two optimizers.

Another relevant consideration is that I have utilized the same number of epoch (4) for the networks trained on the 60000 dataset and those trained on the 1000 dataset. An epoch is a complete reiteration of the training procedure on the entire training dataset, storing and updating the parameters at every iteration, in order to find the optimal result. In a certain sense, the number of epochs can be seen as a “virtual enlargement of the dataset”. With a very high number of epochs, even the 1000 dataset can lead to better results with respect to the 60000 dataset, if this has a small number of epoch! Since our aim is to make effective comparisons, I decided to keep the same number of epochs for both categories, in order to see the differences between the nets performances.

RESULTS TABLE

MODEL	60000 samples		1000 samples	
	SGD	Adam	SGD	Adam
NN_2fcl_512	93.03 %	97.65 %	77.48 %	87.10 %
NN_3fcl_512_512	93.32 %	97.91 %	63.23 %	84.39 %
NN_3fcl_512_512_BATCHNORM	97.23 %	97.50 %	86.39 %	90.06 %
NN_3fcl_512_512_SIGMOID	71.25 %	97.67 %	20.70 %	84.01 %
NN_3fcl_256_128	93.74 %	98.11 %	60.29 %	86.90 %
NN_5fcl_512_512_512_512	92.94 %	96.57 %	15.47 %	84.39 %
NN_5fcl_512_512_512_512_DROPOUT	89.87 %	97.24 %	14.63 %	82.12 %
NN_2fcl_512_DROPOUT	93.29 %	98.03 %	75.92 %	87.02 %
CNN_2fltr_2fcl	98.11 %	98.97 %	71.73 %	93.50 %
CNN_2fltr_AVGpool_2fcl	96.27 %	99.05 %	62.23 %	93.15 %
CNN_2fltr_1/2chnnl_2fcl	97.85 %	98.84 %	70.91 %	93.22 %
CNN_2fltr_1/4chnnl_2fcl	97.74 %	98.73 %	54.85 %	93.05 %
CNN_2fltr_modKernelSize_2fcl	98.14 %	98.75 %	70.20 %	92.42 %
CNN_1fltr_2fcl	96.67 %	98.15 %	84.01 %	87.63 %
CNN_2fltr_2fcl_NOpool	96.30 %	98.97 %	68.61 %	90.16 %
CNN_2fltr_2fcl_BATCHNORM	98.79 %	99.20 %	92.84 %	94.49 %

RESULT ANALYSIS

Firstly, some general considerations. As aforementioned, I decided to evaluate each network with both optimizing method SGD and Adam. For all the tests, the parameters for SGD are:

`learning_rate=0.001` and `momentum=0.9`

for Adam, of course it is not necessary to define a momentum, but the learning rate is:

`learning_rate=0.001`

The used training method, whose procedure is the same for every network, is based on loss backpropagation. This means that for every iteration, starting from a random initialization of the weights, the algorithm updates the weights of the network through the Gradient Descent technique, towards the values which minimizes the overall error cost function.

Since we have at disposal a 60000 and 1000 samples training dataset, I decided to define minibatches for training with a dimension of, respectively, 100 and 50 samples, for a total of 60000/100 = 600 and 1000 / 50 =20 steps for epoch. Actually, the number of elements in the batches is a tradeoff between the size of training dataset to evaluate the gradient and the number of updating gradients at disposal.

Let’s explain this with an example: selecting a batch of 30000 for the overall 60000 training dataset means that the algorithm evaluates the gradient of the error function only 60000/30000 = 2 times! This is of course not a good approach for a method like SGD, whose principle is entirely based on the stochastic evaluation of many minibatches to reduce the time to find global minimum. On the other hands, choosing a very small size for the batch is no good either: choosing, for example, a size of 10, the number of times the algorithm will evaluate gradients is 60000 / 10 = 6000, increasing too much the computational effort and also the possibility to have a wrong gradient for single minibatches. In fact, since data in minibatches are randomly chosen, there is the risk that with a very small number of elements in the batch, gradient direction will be completely wrong, slowing down the process of finding the minimum. This, on the other hand, is something that does not occur with a sufficient high number of stochastically distributed data.

For this reason, for 60000 dataset a 100-size batch has been chosen: the elements of a single batch are enough to be sure that gradient will stochastically go every time in the right direction, and the number of iterations (600) is quite high, guaranteeing the finding of the minimum.

For the 1000 dataset, things are more complex: the chosen size (50) is good for the correct working of the algorithm, while the low number of iterations (20) does not always permit to sufficiently get close to the absolute minimum as some results will show!! However, I tried to reduce the size of the batch to

have more steps, but the number of data becomes too low to have a correct evaluation of the direction of the gradient and overall results tends to get worse. Also, a batch of at least 50 is required to apply Batch Normalization.

Note also that since the given datasets for training and testing are datasets of images, each image has dimension 28x28 = 784, which is also the number of elements per inputs in the networks! In fact, every network is fed with 1 input image, formed by 784 elements (nodes).

Regarding the testing phase of the network, also the testing dataset has been divided into batches. With a similar reasoning as before, the reason is quite intuitive: the network must be able to correctly classify not only a particular dataset, but every dataset which is fed into the network itself. Even in this case, let’s explain this with an example: consider a dataset of solely handwritten ones (1). The number 1 is (usually) very easy to identify as number one (it is difficult to write 1 very bad!). The network, if well designed, will probably match the prediction with the actual value almost every time, giving an accuracy of almost 100%. On the other hand, if a set if composed only by very bad hand-written 2 and 7, it might be very hard for the net to correctly assign the label to the input images!

Since the batches of the overall testing dataset are chosen with random values (`shuffle='True'`), the most appropriate way to evaluate the accuracy of the network is to mean the many accuracies which are found testing different batches!

With similar reasoning as before, a batch with size equal to 100 seems to be a good tradeoff.

Network 1) NEURAL NETWORK 2 full connected layers (512 HL)

```
simple_network(
  (fc1): Linear(in_features=784, out_features=512, bias=True)
  (fc2): Linear(in_features=512, out_features=10, bias=True)
)
```

NN_2fcl_512	93.03%	97.65%	77.48%	87.10%

This is a very basic neural network, and starting from this net, adding or modifying layers, number of nodes, functions etc., I will try to get some considerations of general validity about the performance of different networks.

As you can see from the result table, for the 60000 dataset results are not bad! An accuracy above 90%, depending on the application, is considered good. This high accuracy is made possible by the use of fully connected layers. In fact, in principle it is possible to define layers that do not have connection between each node connected to them. However, both for performance and for simplicity of the code, in this assignment only fc layers have been used. In this case, for sure the network is performing well, but since the classification problem taken into consideration is relatively easy, there is extensive room for improvement.

Notice also the different in precision between 60000 and 1000 training dataset: precision is reduced of more than 10%, and in particular the reached accuracy for the 1000 SGD network is only of 77 %. Main problem is that the network has not enough training samples to iterate the algorithm enough times to effectively reach the minimum. However, while there are difference in performance between 60000 SGD and Adam but they are not so big, the same difference in accuracy between 1000 SGD and Adam is of 10%: this is because Adam is way faster to reach the minimum and it needs less samples to obtain the same result

Network 2) NEURAL NETWORK 3 full connected layers (512, 512 HL)

```
simple_network(
  (fc1): Linear(in_features=784, out_features=512, bias=True)
  (fc2): Linear(in_features=512, out_features=512, bias=True)
  (fc3): Linear(in_features=512, out_features=10, bias=True)
)
```

NN_3fcl_512_512	93.32%	97.91%	63.23%	84.39%

In this case, a layer as been added to compare improvement or decrement of performances of the net. As you can see from results, having more layers does not necessarily means better performances: it is true that a layer with more connection, in principle, is more complex and so can detects and correctly classified a bigger amount of inputs. However, having more layers means having more parameters to optimize,

and so complexity of the objective function increases. This is not a problem when we have a large amount of training samples (many samples → many available iterations → more probability to reach minimum and more capability of correctly classify different inputs), but it becomes a deal when training data are few! In particular, the net has not sufficiently high samples to correctly determine the value of the weights of the net. This is confirmed from the results: 60000 samples are enough also in this case where we have more parameters to properly train the network (results are totally comparable with the 2fcl net), but performances badly decreases in the 1000 SGD network for the same reason mentioned above.

Network 3) NEURAL NETWORK 3 full connected layers (512, 512 HL) with Batch Normalization

```
simple_network(
  (fc1): Linear(in_features=784, out_features=512, bias=True)
  (bn1): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (fc2): Linear(in_features=512, out_features=512, bias=True)
  (bn2): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (fc3): Linear(in_features=512, out_features=10, bias=True)
)
```

NN_3fcl_512_512_BATCHNORM	97.23%	97.50%	86.39%	90.06%

Batch Normalization is a technique used to improve precision of the network without its structural modification. At the moment, it is still a topic of discussion why this technique is so effective in improving overall performances, but basically the two additional bn layers operate through a normalization step that fixes the means and variances of each next layer's inputs. As results show, increment of accuracy is relevant, especially for SGD based networks. The idea to understand this improvement is simple: normalizing the training mini batches, training samples are “better” because relevant features of the batch are extracted and network can be trained more effectively to classify other inputs.

Network 4) NEURAL NETWORK 3 full connected layers (512, 512 HL) with Sigmoid Act. Function

```
simple_network(
  (fc1): Linear(in_features=784, out_features=512, bias=True)
  (fc2): Linear(in_features=512, out_features=512, bias=True)
  (fc3): Linear(in_features=512, out_features=10, bias=True)
)
```

NN_3fcl_512_512_SIGMOID	71.25%	97.67%	20.70%	84.01%

The activation function of a node defines the output of that node given an input. In particular, the most used activation function is ReLU (Rectified Linear Unit), because it effectively enables gradient propagation, avoiding the problem of vanishing gradient. In this network, I wanted to use another famous activation function, non-robust against vanishing gradient problem. This problem is related to the fact that weights are updated at every iterations based on activation functions that have gradient with values between (0,1). Since gradients are multiplied with chain rule, approaching 0 value, the gradient esponentially descreases, vanishes. In principle then, this network should have worse performance that the net 2) based on ReLU. Also in this case, results confirm this fact: in particular, with SGD results are way worse than before! Instead, with Adam optimizator, results remain the same due to its inner adaptability at every iteration.

Network 5) NEURAL NETWORK 3 full connected layers (256, 128 HL)

```
simple_network(
  (fc1): Linear(in_features=784, out_features=256, bias=True)
  (fc2): Linear(in_features=256, out_features=128, bias=True)
  (fc3): Linear(in_features=128, out_features=10, bias=True)
)
```

NN_3fcl_256_128	93.74%	98.11%	60.29%	86.90%

In this case, I wanted to modify not the number of layers, but only the number of connections, which means, the parameters to optimize. Directly analyzing the results, you can see that both 60000 and 1000 Adam network improve their performances of some points, while for SGD networks performances, since accuracy is aleatory, we can say that they did not change. This means that even with less parameters the network can obtain the same results in terms of classification.

Network 6) NEURAL NETWORK 5 full connected layers (512, 512, 512, 512 HL)

```
simple_network(
  (fc1): Linear(in_features=784, out_features=512, bias=True)
  (fc2): Linear(in_features=512, out_features=512, bias=True)
  (fc3): Linear(in_features=512, out_features=512, bias=True)
  (fc4): Linear(in_features=512, out_features=512, bias=True)
  (fc5): Linear(in_features=512, out_features=10, bias=True)
)
```

NN_5fcl_512_512_512_512	92.94%	96.57%	15.47%	84.39%

This 5 layer neural network as been created to see how a big increment of the number of parameters affects the overall performance. As already explained above, many layers and many connections means more precision, provided that there exist a sufficient number of samples in the training dataset. In this case, on purpose this network has a huge number of parameters. Results show that overall performance with respect to the 3 layer network decreases. In particular, for 1000 SGD net, dataset is absolutely not big enough for the correct working of the network itself, leading to a very poor 15 % of accuracy. 1000 Adam still keeps to work, not with excellent performances but with acceptable ones. Regarding 60000 dataset, performances have slightly decreased: this means that even if the size of the batch is still sufficiently big for this amount of parameters, further increasing of the number of layers can affect the overall behavior of the network.

Network 7) NEURAL NETWORK 5 full connected layers (512, 512, 512, 512 HL) with Dropout

```
simple_network(
  (fc1): Linear(in_features=784, out_features=512, bias=True)
  (dropout1): Dropout(p=0.5, inplace=False)
  (fc2): Linear(in_features=512, out_features=512, bias=True)
  (dropout2): Dropout(p=0.5, inplace=False)
  (fc3): Linear(in_features=512, out_features=512, bias=True)
  (dropout3): Dropout(p=0.5, inplace=False)
  (fc4): Linear(in_features=512, out_features=512, bias=True)
  (dropout4): Dropout(p=0.5, inplace=False)
  (fc5): Linear(in_features=512, out_features=10, bias=True)
)
```

NN_5fcl_512_512_512_512_DROPOUT	89.87 %	97.24 %	14.63 %	82.12 %

Dropout is a regularization technique for reducing overfitting. Usually, overfitting occurs when the network is trained to match a very high number of training samples, but when it comes to evaluate a new input (different from the training inputs) its classification performances decrease or fail. In our case, performances with respect to the previous 5 layers network have remained almost the same: probably, no overfitting has occurred in Network 6) and so in this case dropout is not useful

Network 8) NEURAL NETWORK 2 full connected layers (512 HL) with Dropout

```
simple_network(
  (fc1): Linear(in_features=784, out_features=512, bias=True)
  (dropout1): Dropout(p=0.5, inplace=False)
  (fc2): Linear(in_features=512, out_features=10, bias=True)
)
```

NN_2fcl_512_DROPOUT	93.29%	98.03%	75.92%	87.02%

This network has been created to verify if, with a much smaller network, overfitting has occurred. From results, you see that they are totally comparable with the one obtained with the 2fcl without dropout. We can affirm that no overfitting has occurred, and in this case dropout can be considered as a useless addition.

GENERAL CONSIDERATIONS ABOUT CONVOLUTIONAL NEURAL NETWORKS

Fully connected networks are often very good for classification, especially in the case of a database compound by simple hand written numbers. However, for more complex images and for more accuracy, there is the need to extract fundamental characteristics of an image to correctly classify it with a minor computational effort (to achieve same results with full connected networks, an incredible number of layers is necessary, with an unacceptable computational effort and an easy risk of overfitting).

CNNs solve this problem exploiting correlations between adjacent inputs in images → not every node in the network needs to be connected to every other node!
This idea is implemented through convolutional filters, which are exactly like a layer in the classical neural network, with the difference that every channel of a filter is optimized to extract a particular characteristic of the picture, without the need of having all the nodes connected!
Reason for channels is that weights of individual filters are held constant and trained to extract a particular feature. If I want to extract different feature of an image, I need to have a number of channels equal to the number of desired extracted features.

Another typical tool used in CNN is pooling, similar to a filter, but it applies a statistical function instead of weights (so they cannot be trained) in order to reduce number of parameters and make feature detection more robust.

In this report, I tried to compare efficiency of CNNs with respect to fc NNs, and starting from a basic (but very accurate) Convolutional Neural Network, add and remove channels, filters, pooling etc. to see how performances vary.

Network 9) CONVOLUTIONAL NEURAL NETWORK 2 filters (32, 64), 2 fully connected layers

```
ConvNet(
  (layer1): Sequential(
    (0): Conv2d(1, 32, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (layer2): Sequential(
    (0): Conv2d(32, 64, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (drop_out): Dropout(p=0.5, inplace=False)
  (fc1): Linear(in_features=3136, out_features=1000, bias=True)
  (fc2): Linear(in_features=1000, out_features=10, bias=True)
)
```

CNN_2fltr_2fcl	98.11%	98.97%	71.73%	93.50%

This is the convolutional neural network that I will use as a reference to explore different modifications that can be implemented in a CNN. The network is composed by 2 filters that maintain the same size of the input at the output, 2 Maxpooling filters that, with stride > 1, reduce the size of the output and by 2 fully connected layer as they are usually added to have a proper behavior of a convolutional net. Also, dropout is added because in this case parameters are many and risk of overfitting is higher with respect to simple neural networks case.

As result show, there is a very big improvement in the accuracy of both this four networks. In particular, now accuracy of 60000 training dataset reaches almost the 99%, which is a very good result (and also, accuracy of SGD and Adam are almost the same now). Even with the small size-1000 dataset and Adam accuracy of the network surpass the 93%! A very high precision with a small training dataset is a notable thing. You can however notice that the 1000 SGD network, even if has improved too, has not yet very good results. Also in this case, since the values in the filters can be seen as additional parameters, the size and so the number of iterations is not sufficient to properly set the values and obtain a very high accuracy, but with some changes even in this case, we will see, it is possible to have accuracy above 90%!

Network 10) CONVOLUTIONAL NEURAL NETWORK 2 filters (32, 64), AVGpooling, 2 fully connected layers

```
ConvNet(
  (layer1): Sequential(
    (0): Conv2d(1, 32, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU()
    (2): AvgPool2d(kernel_size=2, stride=2, padding=0)
  )
  (layer2): Sequential(
    (0): Conv2d(32, 64, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU()
    (2): AvgPool2d(kernel_size=2, stride=2, padding=0)
  )
  (drop_out): Dropout(p=0.5, inplace=False)
  (fc1): Linear(in_features=3136, out_features=1000, bias=True)
  (fc2): Linear(in_features=1000, out_features=10, bias=True)
)
```

CNN_2fltr_AVGpool_2fcl	96.27%	99.05%	62.23%	93.15%

Pooling, as aforementioned, can be of different types. In this assignment, max pooling is usually adopted. This means that, for example, that the output of a window of dimension 4, will be the maximum value between these four. In our case, however, Average pooling has been used: the output value is the mean between the four values. You can see that very few has changed in the results. The worst result is given by the 1000 SGD net. Probably, since the pooling not only is used to reduce the computational effort but also for extracting characteristics of closest pixel, in this case

avg pooling is not so effective as maxpooling because at the edges of a letter, instead of decide weather the pixel is black or white, it proceed with meaning them.

Network 11) CONVOLUTIONAL NEURAL NETWORK 2 filters (16, 32) 2 fully connected layers

```
ConvNet(
  (layer1): Sequential(
    (0): Conv2d(1, 16, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (layer2): Sequential(
    (0): Conv2d(16, 32, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (drop_out): Dropout(p=0.5, inplace=False)
  (fc1): Linear(in_features=1568, out_features=1000, bias=True)
  (fc2): Linear(in_features=1000, out_features=10, bias=True)
)
```

CNN_2fltr_1/2chnnl_2fcl	97.85%	98.84%	70.91%	93.22%

So far we have seen that incrementing the number of channels correspond to incrementing the parameter. Basically, each channel is used to extract a particular feature of the image (ex: horizontal, vertical lines, etc..). Question is: is half the number of the channels sufficient to extract a number of features enough for our classification problem? Answer is evidently yes. From results, you can see that they are totally compatible with the results of the 32, 64 channels network (maybe a little improvement). That means that for our problem we don’t need such a high number of channels.

Network 12) CONVOLUTIONAL NEURAL NETWORK 2 filters (8, 16) 2 fully connected layers

```
ConvNet(
  (layer1): Sequential(
    (0): Conv2d(1, 8, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (layer2): Sequential(
    (0): Conv2d(8, 16, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU()
```

```
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (drop_out): Dropout(p=0.5, inplace=False)
  (fc1): Linear(in_features=784, out_features=1000, bias=True)
  (fc2): Linear(in_features=1000, out_features=10, bias=True)
)
```

CNN_2fltr_1/4chnnl_2fcl	97.74%	98.73%	54.85%	93.05%

With exactly the same reasoning done for network 10, are a quarter of channels still enough for our problem? Answer is yes, but you can see that there is a decrease of accuracy of 1000 SGD network.

Network 13) CONVOLUTIONAL NEURAL NETWORK 2 filters (8, 16) 2 fully connected layers, Kernel size = 3

```
ConvNet(
  (layer1): Sequential(
    (0): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=4, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (layer2): Sequential(
    (0): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (drop_out): Dropout(p=0.5, inplace=False)
  (fc1): Linear(in_features=4096, out_features=1000, bias=True)
  (fc2): Linear(in_features=1000, out_features=10, bias=True)
)
```

CNN_2fltr_modKernelSize_2fcl	98.14%	98.75%	70.20%	92.42%

In this network, the number of units of the filters have been modified and reduced from a Kernel size of 5 to 3. Also, parameters of pooling filter have been adjust to have a correct number of outputs (now the output of the second pooling filter is 8x8x64). From the results, you can notice that performance has remained quite the same (it is only slightly worse) with respect to the 5 kernel filters one.

Network 14) CONVOLUTIONAL NEURAL NETWORK 1 filters (32) 2 fully connected layers

```
ConvNet(
  (layer1): Sequential(
    (0): Conv2d(1, 32, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (drop_out): Dropout(p=0.5, inplace=False)
  (fc1): Linear(in_features=6272, out_features=1000, bias=True)
  (fc2): Linear(in_features=1000, out_features=10, bias=True)
)
```

CNN_1fltr_2fcl	96.67%	98.15%	84.01%	87.63%

What does it happen, instead, uif we use one filter instead of two? In principle, the accuracy increases if we have complex images to classify, because the network can extract more particular features. On the other end, if the task of the net is relatively easy, we can only add further complexity to the net and further parameters, that are difficult to correctly train if we have a small training dataset. This is exactly what happens in our case: precision of 60000 dataset networks is very good and perfectly comparable with previous results. However there is a big improvement for our SGD 1000 network: a less number of parameters means that training is more simple, and for SGD which is not fast as Adam to find the minimum, these is an important advantage because objective function is less complex and minimum can be find with fewer steps. Accuracy increases.

Network 15) CONVOLUTIONAL NEURAL NETWORK 2 filters (32, 64) without Pooling, 2 fully connected layers

```
ConvNet(
  (layer1): Sequential(
    (0): Conv2d(1, 32, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2))
    (1): ReLU()
  )
  (layer2): Sequential(
    (0): Conv2d(32, 64, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2))
    (1): ReLU()
  )
  (drop_out): Dropout(p=0.5, inplace=False)
  (fc1): Linear(in_features=3136, out_features=1000, bias=True)
  (fc2): Linear(in_features=1000, out_features=10, bias=True)
)
```

CNN_2fltr_2fcl_NOpool	96.30%	98.97%	68.61%	90.16%

We have seen what the purpose of pooling is. If we remove pooling of our net we can see from the results that for 60000 data-training net almost nothing has changed, while accuracy has decreased in 1000 training dataset network. With fewer training sample, the importance of feature extraction is more relevant.

Network 16) CONVOLUTIONAL NEURAL NETWORK 2 filters (32, 64), with Batch Normalization, 2 fully connected layers

```
ConvNet(
  (layer1): Sequential(
    (0): Conv2d(1, 32, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (bn1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (layer2): Sequential(
    (0): Conv2d(32, 64, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (drop_out): Dropout(p=0.5, inplace=False)
  (fc1): Linear(in_features=3136, out_features=1000, bias=True)
  (bn3): BatchNorm1d(1000, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (fc2): Linear(in_features=1000, out_features=10, bias=True)
)
```

CNN_2fltr_2fcl_BATCHNORM	98.79%	99.20%	92.84%	94.49%

This final network, which has the best performance among all the networks created in this assignment, confirms the efficiency of Batch Normalization. Now, every network has accuracy above 90 % and the best accuracy, is of 99.20 % really close to our ideal objective of 100%. Same reasoning done for batch normalization in full connected neural networks can be used to explain this great results.

CONCLUSIONS

In addition to the specific analysis that has been done for every network created, some general conclusion can be derived from the overall testing of these networks.

First of all, it is evident how much, in general, Convolutional Neural Network are more accurate, more robust and more reliable with respect to simple fully connected Neural Networks. Results confirm the theory without any doubt.

Secondly, we cannot say in absolute that Adam is a better optimizer than simple SGD. For our case of study, yes, we can say that networks where Adam was implemented achieved better results. However, we have to keep in mind that Adam is only “faster” in approaching the optimal, minimum value for the parameter, but it can reveal itself less robust than simple SGD for some applications. SGD is then slower because it requires more iterations to achieve accuracy of Adam, but if well implemented it will find the minimum with a very high probability.

Third, importance of the size of the training dataset is fundamental. The bigger the dataset at disposal, the better the network can be trained and better accuracy can achieve. This simple reasoning is confirmed by results, where 60000 accuracy is always better than 1000 dataset networks. In addition, with 1000 samples SGD sometimes completely fails.

To conclude, in this assignment I have tried to explore the most common very basic tools at disposal of a neural network designer to implement an efficient net. However, obtained result cannot be taken for general validity because NN behavior is strongly affected by its application, which in this case was based on MNIST dataset at disposal.