Homework 6

a) The autoencoder has the aim of reproducing the input that is given and is used for unsupervised learning. Starting for example by an input 30x30, the first phase is to reduce the dimensionality of the data for example going to a layer of 100 neurons (encoder phase). After this, the decoder phase tries to reconstruct the initial image, giving an output of the same dimension of the input (30x30 in this example). The encoder phase is implemented in order to avoid the curse of dimensionality of the data, so starting from a space X we go to a new space Z characterized by a lower dimension.

The denoising autoencoder is a particular type of autoencoder, that has the purpose of removing noise. Given a set of input data, it will be trained in order to remove noise from similar images. The main difference is that in this case the inputs are the original images, but with some noise added (rotation,crop,resize,blur…), while the outputs are the original images. The loss in this network is calculated measuring the similarity between the different images. The similarity between images belonging to the same original one should me increased with respect to the other.

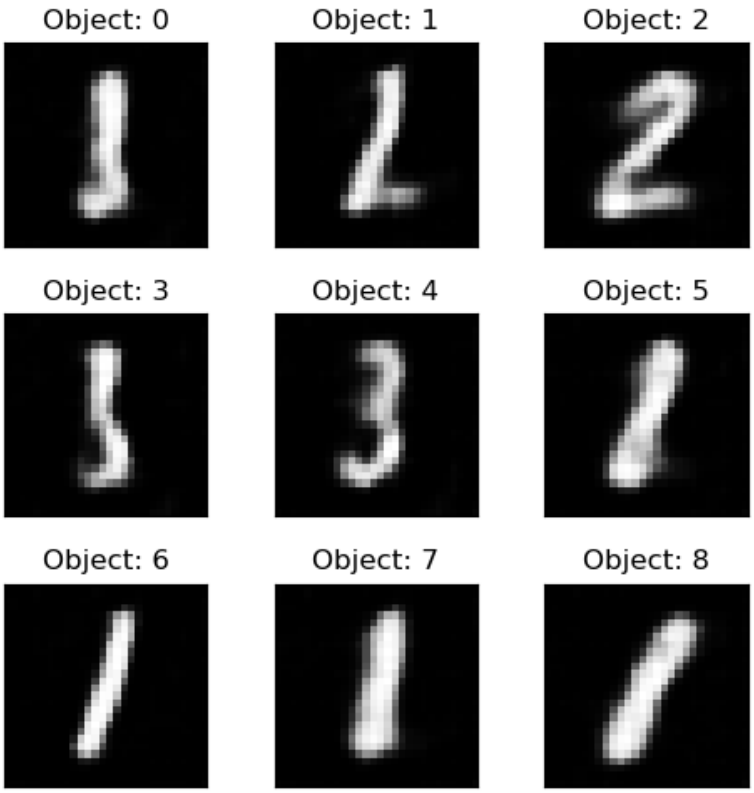
b) Batch normalization is a technique that is used in order to reduce the internal covariate shift, problem due to the fact that the distribution of each layer's input changes during training, as the parameters of the previous layers change . It is based in the normalization of layer inputs. In this case is implemented in the encoder

$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta$$

phase thanks the command *torch.nn.BatchNorm2d(16)* and applies the following formula:

This normalization is done on the mini-batch passed to the network and the parameters gamma and beta are learnable parameter vectors of size equal to the input size. During training this layer continue to estimate the mean and variance that are then implemented in the normalization during the evaluation phase.

c)



This is the output of the trained decoder for 9 different random tensor. The functionality of the decoder is to reconstruct the input image, according to the labels on which was trained (written digits in this case).

d) The accuracy reached on the training set is 73.47%.

In my implementation, for each cluster I create a list of labels associated to that cluster number. From each single list I find the value that appear most frequently in the list; in this way I can understand the correspondence between the cluster number and the label. Thanks a dictionary I can now assign the cluster number to the label and, for each point, see if the label corresponds to the true one. If this happens the number of corrects is incremented by one, otherwise the number of errors is incremented. At the end I can calculate the accuracy.

This is the pseudocode:

```
dict={}
for i in range(10):
    label_list=[]
    for j in range(len(clusters)):
        if clusters[j] == i:
            label_list.append(labels[j])
    dict[i] = Counter(label_list).most_common()[0][0]

correct=0
wrong=0
for cl,lb in zip(clusters,labels):
    if dict[cl] - lb ==0:
        correct+=1
    else:
        wrong+=1
accuracy = ((correct/(correct+wrong))*100)
print(accuracy)
```