Homework7

Q1.

Part a)

As proved in the homework PDF, $\eta = 4.5$ is the best choice to obtain the best estimator of the Lloyd's average done on the whole dataset. This is needed because we're using minibatches so we need to calculate the average by minimizing both bias and variance.

Part b)

By initializing the centers with images of the 10 digits we could think it's a smart choice since we are "helping" the model. To see if this is the case I ran the code selecting a random image for each digit as center. The final centers are shown in Figure 1 (unflattened to visualize them), while on the left we can see both the final flattened centers and the matrix containing the classifications for each image. The initialization choice seems to be a good one since we can see that the confusion matrix has high values on the main diagonal meaning that it correctly classifies most of the digits. The final loss on the testing set is around 0.087.

```
Epoch 10
-------------------------------
loss: 0.095885  [   64/60000]
loss: 0.085306  [ 6464/60000]
loss: 0.088986  [12864/60000]
loss: 0.080841  [19264/60000]
loss: 0.085828  [25664/60000]
loss: 0.086657  [32064/60000]
loss: 0.088227  [38464/60000]
loss: 0.089795  [44864/60000]
loss: 0.087382  [51264/60000]
loss: 0.082661  [57664/60000]
Test Error: Avg loss: 0.086956

Final Cluster Centers:
Parameter containing:
tensor([[0., 0., 0.,  ..., 0., 0., 0.],
        [0., 0., 0.,  ..., 0., 0., 0.],
        [0., 0., 0.,  ..., 0., 0., 0.],
        ...,
        [0., 0., 0.,  ..., 0., 0., 0.],
        [0., 0., 0.,  ..., 0., 0., 0.],
        [0., 0., 0.,  ..., 0., 0., 0.]], requires_grad=True)
Confusion Matrix:
[[4870    9   32   56   71  270  351   38   41  185]
 [   0 3055    0    0    0   72   16    4   11 3584]
 [ 133 1480 1793  389  378  564  476   62  154  529]
 [ 209  484   84 2939   16  118   91   13   79 2098]
 [   4  364   93   87 2833  138  366  623   19 1315]
 [ 454  107   51 1327  124  858  293  189  239 1779]
 [ 105  165   22    7  186  305 4157   26   41  904]
 [  45  160   53   13   52  153   21 3246   39 2483]
 [  14 1014   91 1088   83  106  150  170 1936 1199]
 [  41   77  358  269  250   84  223 1616   16 3015]]
```

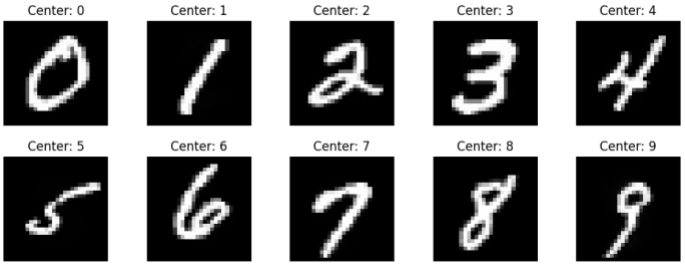

Figure 1

Part c)

From the theory we know that the weights initialization is a key point. Despite the good results obtained in b) we can try to stick with what the theory suggests and initialize the centers randomly. By doing that we can see that the final centers, this time, are random pixels and we can't visually distinguish any digit. We can see in Figure 2 that we achieve slightly worse results reaching a final loss of around 0.21.

By looking at the final matrix we can see that the classification this time is sparser. It reveals significant mode collapse, where all the digits classes are mapped into a single cluster. This suggests that the clustering model (with the current initialization and training settings) struggles to learn distinct representations for all digit classes. The observed mode collapse highlights the need for further refinement in the model's training dynamics and center initialization.

Overall, we can say that “suggesting” some starting centers is the best initialization method.

```
Epoch 10
-------------------------------
loss: 0.226713  [   64/60000]
loss: 0.226037  [ 6464/60000]
loss: 0.225702  [12864/60000]
loss: 0.225404  [19264/60000]
loss: 0.223480  [25664/60000]
loss: 0.222357  [32064/60000]
loss: 0.222349  [38464/60000]
loss: 0.221412  [44864/60000]
loss: 0.220358  [51264/60000]
loss: 0.220176  [57664/60000]
Test Error: Avg loss: 0.219964

Final Cluster Centers:
Parameter containing:
tensor([[0.0470, 0.1104, 0.5170,  ..., 0.2146, 0.1158, 0.8873],
        [0.8861, 0.3387, 0.7537,  ..., 0.9061, 0.8734, 0.6753],
        [0.7227, 0.0032, 0.2280,  ..., 0.6678, 0.0593, 0.3911],
        ...,
        [0.7394, 0.2795, 0.1123,  ..., 0.5796, 0.3086, 0.9032],
        [0.5264, 0.2373, 0.3323,  ..., 0.3944, 0.3243, 0.2820],
        [0.4097, 0.2512, 0.3232,  ..., 0.8147, 0.5709, 0.6412]],
       requires_grad=True)
Confusion Matrix:
[[   0    0    0    0    0    0    0 5923    0]
 [   0    0    0    0    0    0    0 6742    0]
 [   0    0    0    0    0    0    0 5958    0]
 [   0    0    0    0    0    0    0 6131    0]
 [   0    0    0    0    0    0    0 5842    0]
 [   0    0    0    0    0    0    0 5421    0]
 [   0    0    0    0    0    0    0 5918    0]
 [   0    0    0    0    0    0    0 6265    0]
 [   0    0    0    0    0    0    0 5851    0]
 [   0    0    0    0    0    0    0 5949    0]]
```

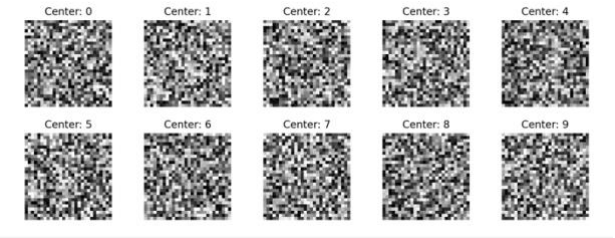

Figure 2

Q2.

Part a)

A sketch of the tensor shape transformation during the encoding and decoding parts is shown in Figure 3. Stages on the same line are related to each other, meaning that the stage on the right-hand side (decoder) inverts the stage on the left-hand size (encoder).

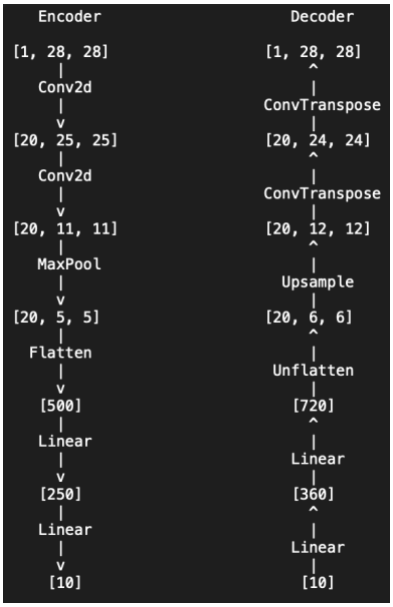

Figure 3

Part b)

I trained the model for 10 epochs and the obtained performance are shown in Figure 4, where the final testing loss is 0.069. Now that the model is trained, we can exploit it to generate 20 random digits starting from a random z and exploiting the decoder part.

The generated images are shown in Figure 5: as we can see the model is not performing very well, since it wasn't able to capture all the important information and details to generate proper digits images. In fact, we can't distinguish any digit since the images have a sort of white fog in the middle.

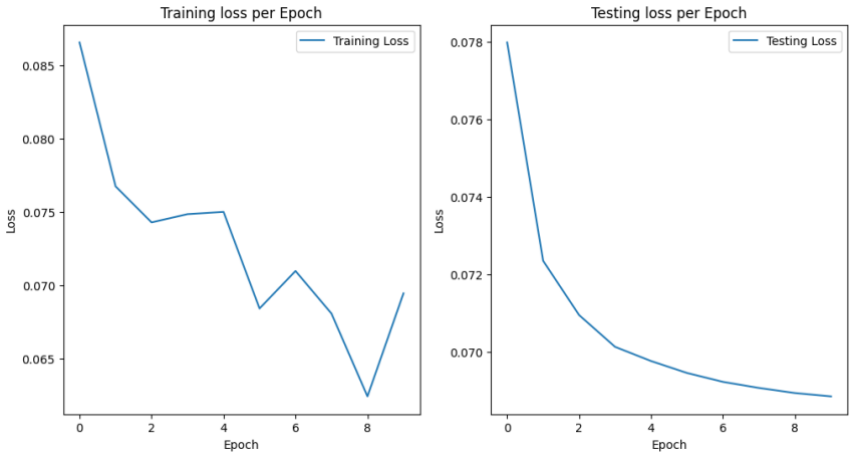

Figure 4

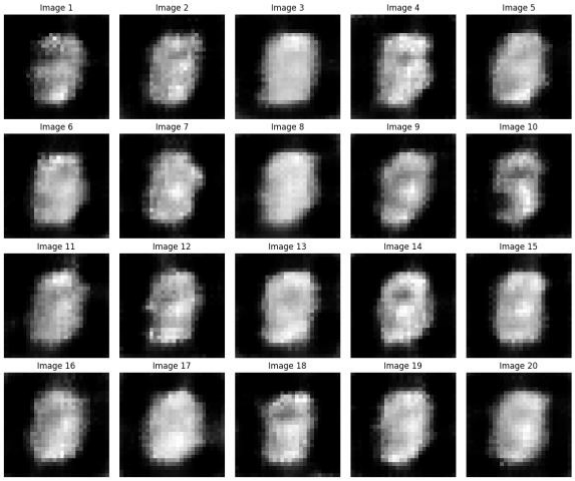

Figure 5

Part c)
Given the poor results of the previous model, we can try to improve it.
I tried to change the architecture by adding more channels so that the network can extract more information.

Concerning the training process, I added some optimization to the hyperparameters:

- I tried to make the learning rate smaller so that it could better converge without overshooting (and giving more epochs to train better), but I didn't get any better results. I then tried to slightly decrease it to 0.05 which seemed the best value.
- I increased the number of epochs to 20, 30 and even 50. I tried 30 and 50 with a batch size of 128 to make the training time feasible, but I didn't get any improvements. I then decided to keep 20 which seemed to perform better than just 10 epochs.
- Concerning the batch size, I increased it to 128 (exploiting the GPU capabilities) when training with 30 and 50 epochs without getting any better results, so I tried to reduce it to 32 getting slightly better results.
- I added some regularization techniques by adding a weight decay of 1e-4. I also tried other values around it, like 1e-3 and 1e-5, but the best results were achieved with 1e-4.

Finally, I tested the new architecture in the same way as before getting the results shown in Figure 6, reaching a final testing loss of 0.068.

The 20 generated images shown in Figure 7 look a bit better than before since, this time, we can distinguish a bit more some digits similar to 3, 9, 5 and 2. The images are, though, far from perfect since there is still a lot of noise.

To achieve better results, I tried many different configurations of architectures changing the latent space size value, the number of intermediate channels, the number of intermediate layers. I also changed the training parameters like the learning rate the batch size and the number of epochs.

Despite all the changes the best configuration I found is the presented one, that is still an improvement with respect to the basic one tried in point b).

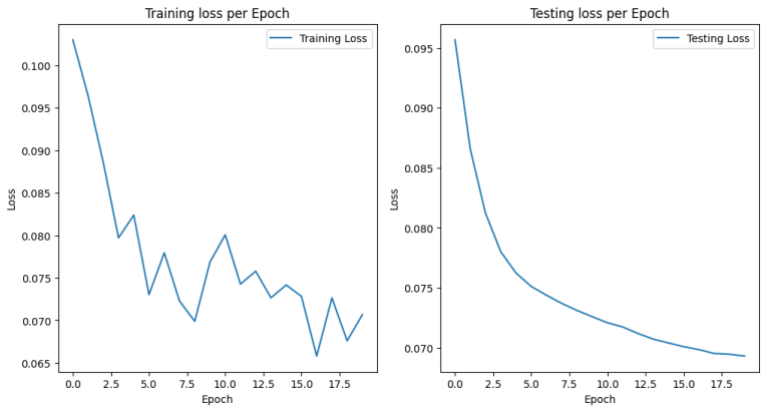

Figure 6

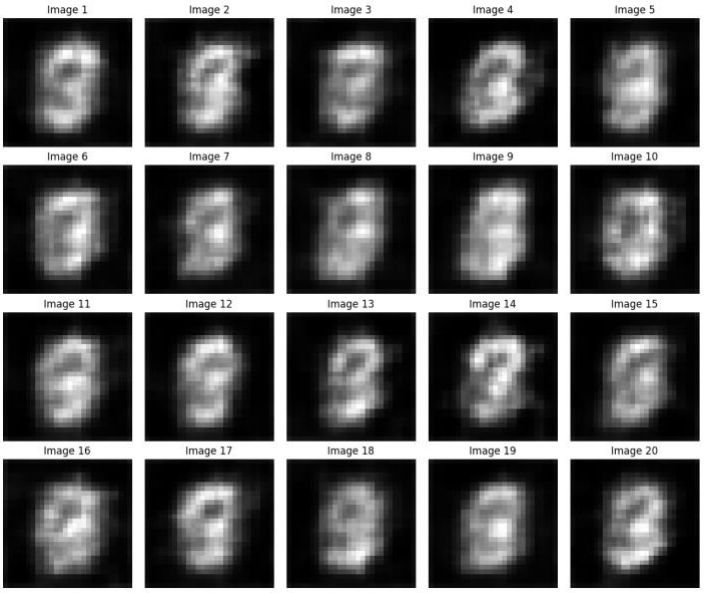

Figure 7