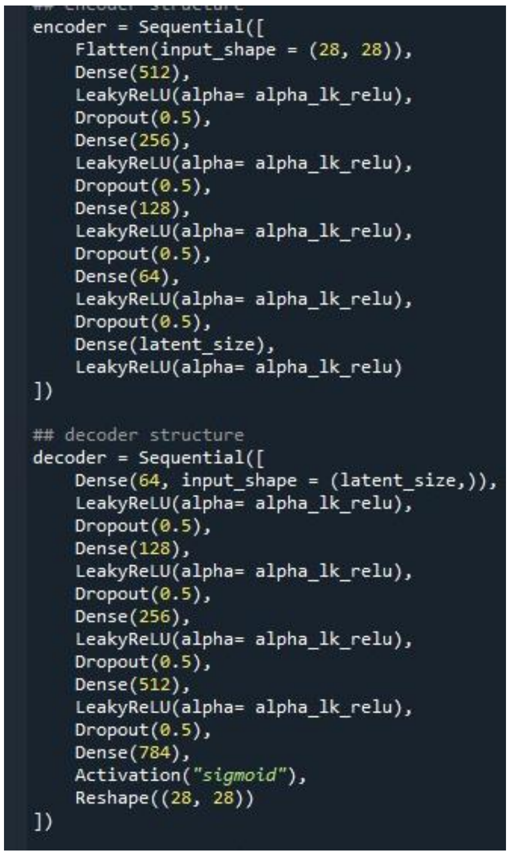
Reza Saadati Fard – 2nd Assignment

General points:

- I used MNIST data set for this assignment.
- I used tensorflow 2, and Sklearn as my major toolboxes
- The code is available in my Git: click
- In addition to the 2 main models (classic AutoEncoder (AE) and Convlolutional AutoEncoder (CAE), I have also implemented Convolutional Variational AutoEncoder (CVAE). However, I didn't have enough space and time to put it into report but the code and results are available in my Git.

In this project, I implemented 2 different architectures

1. Classic AutoEncoder (AE): in which at first all pixels are flattened then I use a regular NN -with dense layers- as the encoder. After getting the latent varaiable, I used the inverse of encoder to make decoder model.
   One can see used structure in the below code.

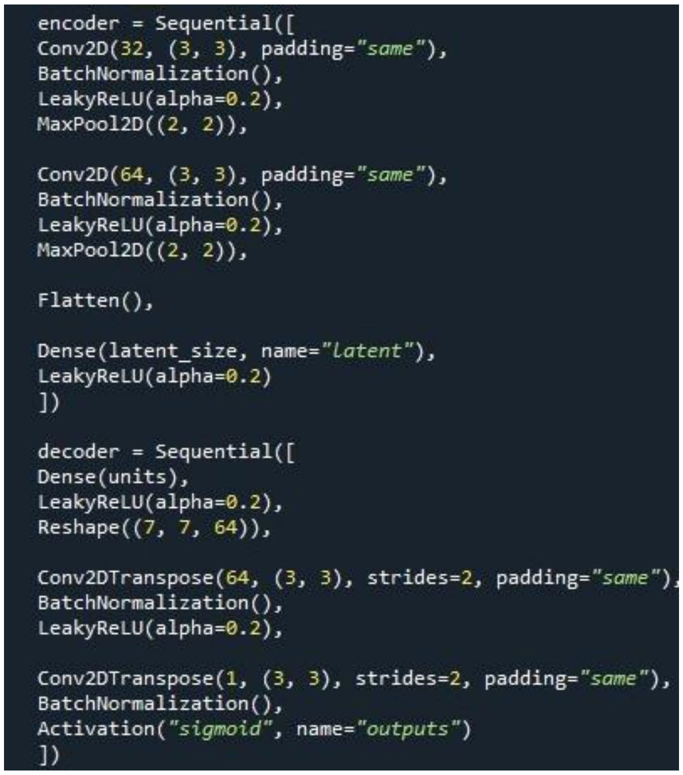


```
encoder = Sequential([
    Flatten(input_shape = (28, 28)),
    Dense(512),
    LeakyReLU(alpha= alpha_lk_relu),
    Dropout(0.5),
    Dense(256),
    LeakyReLU(alpha= alpha_lk_relu),
    Dropout(0.5),
    Dense(128),
    LeakyReLU(alpha= alpha_lk_relu),
    Dropout(0.5),
    Dense(64),
    LeakyReLU(alpha= alpha_lk_relu),
    Dropout(0.5),
    Dense(latent_size),
    LeakyReLU(alpha= alpha_lk_relu)
])

## decoder structure
decoder = Sequential([
    Dense(64, input_shape = (latent_size,)),
    LeakyReLU(alpha= alpha_lk_relu),
    Dropout(0.5),
    Dense(128),
    LeakyReLU(alpha= alpha_lk_relu),
    Dropout(0.5),
    Dense(256),
    LeakyReLU(alpha= alpha_lk_relu),
    Dropout(0.5),
    Dense(512),
    LeakyReLU(alpha= alpha_lk_relu),
    Dropout(0.5),
    Dense(784),
    Activation("sigmoid"),
    Reshape((28, 28))
])
```

2. Convolutional AutoEncoder (CAE): in this case, instead of flattening at the first layer, I use convolutional layer to extract feature, then I flatten extracted info and with a dense I reached a

latent variable with arbitrary size. After that, transpose convolution has been used to make decoder.



```
encoder = Sequential([
Conv2D(32, (3, 3), padding="same"),
BatchNormalization(),
LeakyReLU(alpha=0.2),
MaxPool2D((2, 2)),

Conv2D(64, (3, 3), padding="same"),
BatchNormalization(),
LeakyReLU(alpha=0.2),
MaxPool2D((2, 2)),

Flatten(),

Dense(latent_size, name="latent"),
LeakyReLU(alpha=0.2)
])

decoder = Sequential([
Dense(units),
LeakyReLU(alpha=0.2),
Reshape((7, 7, 64)),

Conv2DTranspose(64, (3, 3), strides=2, padding="same"),
BatchNormalization(),
LeakyReLU(alpha=0.2),

Conv2DTranspose(1, (3, 3), strides=2, padding="same"),
BatchNormalization(),
Activation("sigmoid", name="outputs")
])
```

In order to have a fair comparison between two models (AE and CAE); I used the same hyperparameters.

You can see the configuration of my code in the below table:

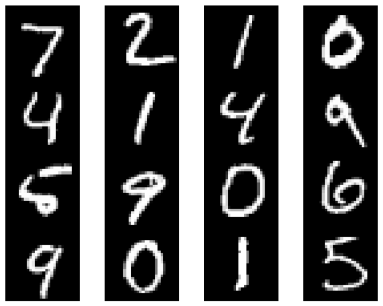
| #Train | #Test | #Epoch | Batch size |
|---|---|---|---|
| 20,000 | 4,000 | 25 | 256 |

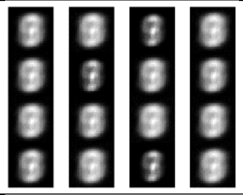
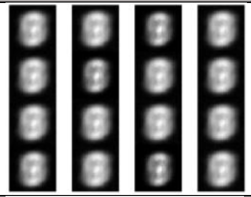
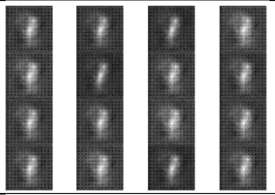
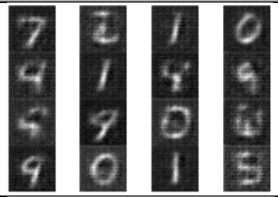
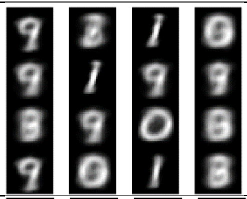
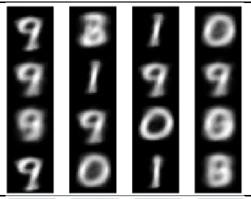
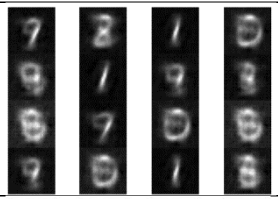
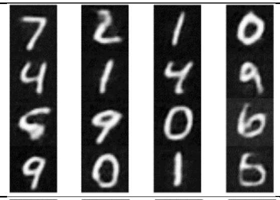
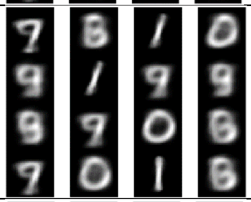
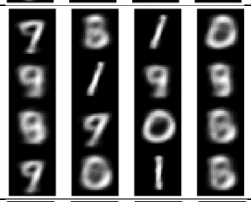
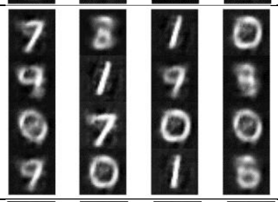
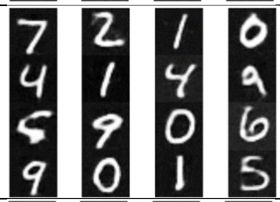
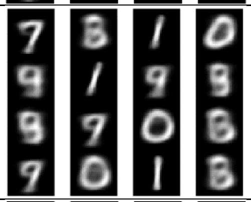
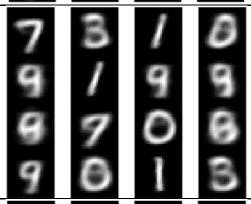
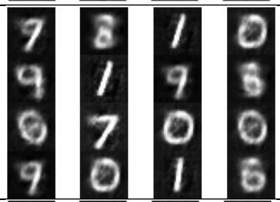
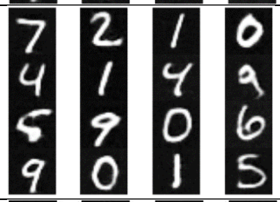
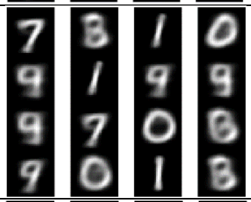
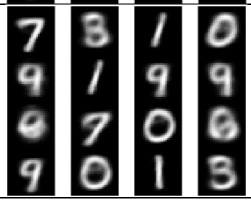
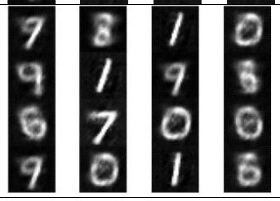
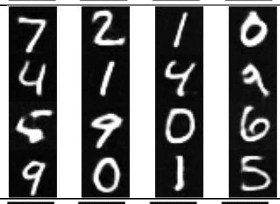
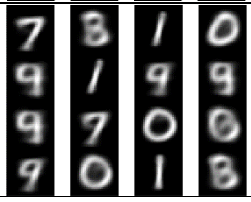
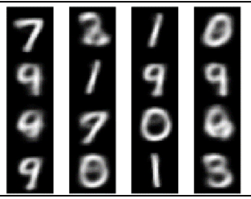
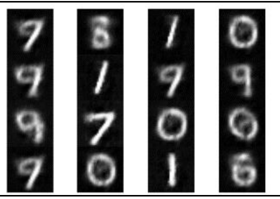
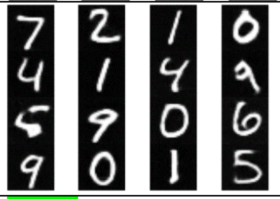
Both models (AE and CAE) have been implemented for two conditions of latent variables: 2, 32. This gives us a comparison of the latent size effect.

## Task 1

In this task, we should make a comparison between 2 main models for different configurations. I decided to assess the effect of latent size for two different models. It could give us a fair comparison because all other parameters were been unchanged.

In the below, I show the baseline figure for reconstructing. This is a figure of 16 numbers from the test. For all other models, I will show how models would be able to reconstruct this image.



| | AE | | CAE | |
|---|---|---|---|---|
| | Latent 2 | Latent 32 | Latent 2 | Latent 32 |
| Epoch 1 |  |  |  |  |
| Epoch 5 |  |  |  |  |
| Epoch 10 |  |  |  |  |
| Epoch 15 |  |  |  |  |
| Epoch 20 |  |  |  |  |
| Epoch 25 |  |  |  |  |
| # correct of readable digit | 9/16 | 9/16 | 8/16 | 16/16 |

If you check with eye, you can see CAE do better job than AE. Also greater latent produce better results that both of these results are predictable.

**CAE vs AE:**

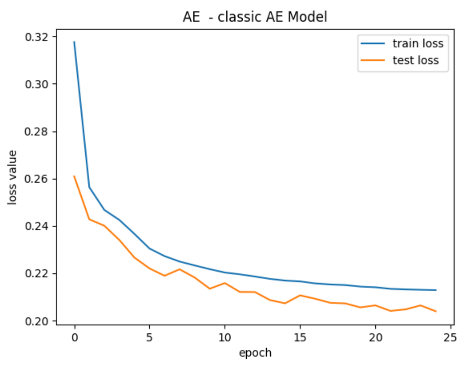
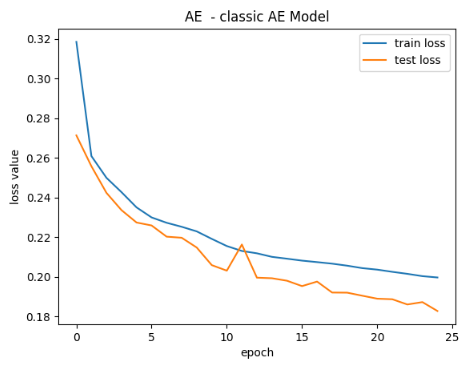
CAE use better architecture to extract information of image, so it works better than using a flatten at first layer and this is very obvious.

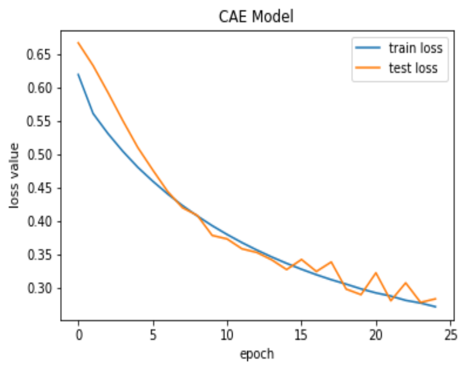
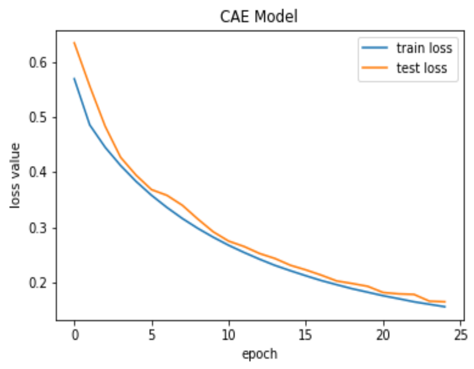
**Latent 2 vs 32:**

When we use larger latent, we could save more information and we expect larger latent contain more information and therefore better result. However, it is interesting in CAE, with using just 2 dimension we got very good results.

In the below, you can see loss function for both methods. However, CAE could still be trained but in order to have a fair comparison, I use 25 epoch for all methods.

All results for task2, task3 are for 25 epochs.

| AE | |
|---|---|
| Latent 2 | Latent 32 |
|  |  |

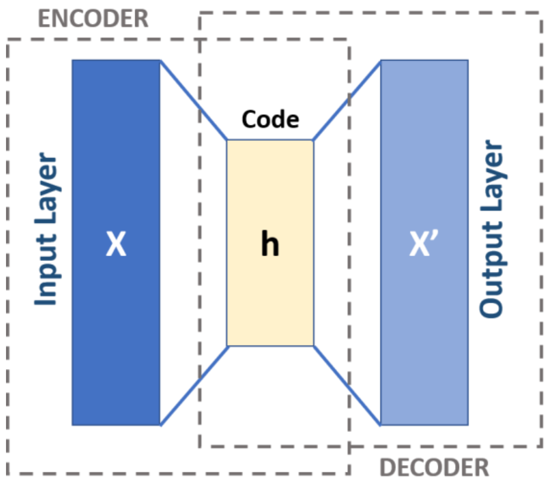
| CAE | |
|---|---|
| Latent 2 | Latent 32 |
|  |  |

- I need to mention that the runtime of the CAE is much more than AE (about 10 times greater)

**Task 2**

In this task, we are asked to use Kmeans (an unsupervised method) to group latent variable of the model. However, in the task we are asked to show the best performance of task 1 (CAE with 32 latent) but I decided to show information of all models to have a good comparison.

In order to apply Kmeans, we should extract information from latent layer (**h** in the below figure) and use it as input of our Kmeans model.



K-means clustering is an unsupervised machine learning method; consequently, the labels assigned by our KMeans algorithm refer to the cluster each array was assigned to, not the actual target integer. To fix this, let's define a few functions that will predict which integer corresponds to each cluster.

Because we have 10 digits, I used 10 clusters.

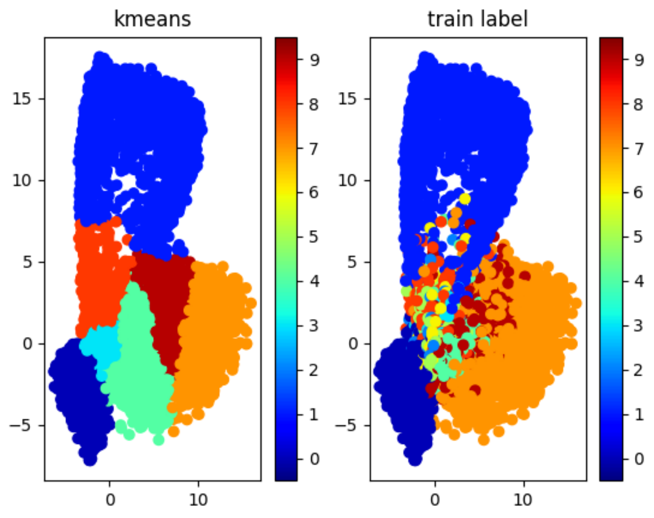
To do this, I check cluster numbers and labels and find out which cluster in the most cases is assigned to a certain digit. Then, I assigned that cluster to that digit. I have done this for all clusters. ( in the code you can see this function in ae.py > check def infer_cluster_labels function)

After assigning a digit to each cluster, now we are able to check the accuracy of this unsupervised clustering. As we expected CAE with 32 latent dimension works the best!

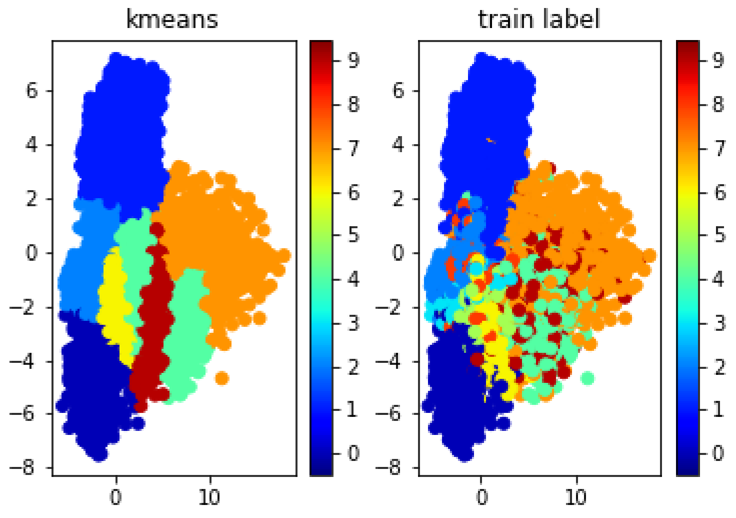
| | AE | | CAE | |
|---|---|---|---|---|
| Latent | 2 | 32 | 2 | 32 |
| Accuracy | 41.74 | 43.34 | 45.19 | 54.57 |

In addition to this evaluation, I also used 2 dimension latent variables and their assigned clusters to have a visualization.

AE



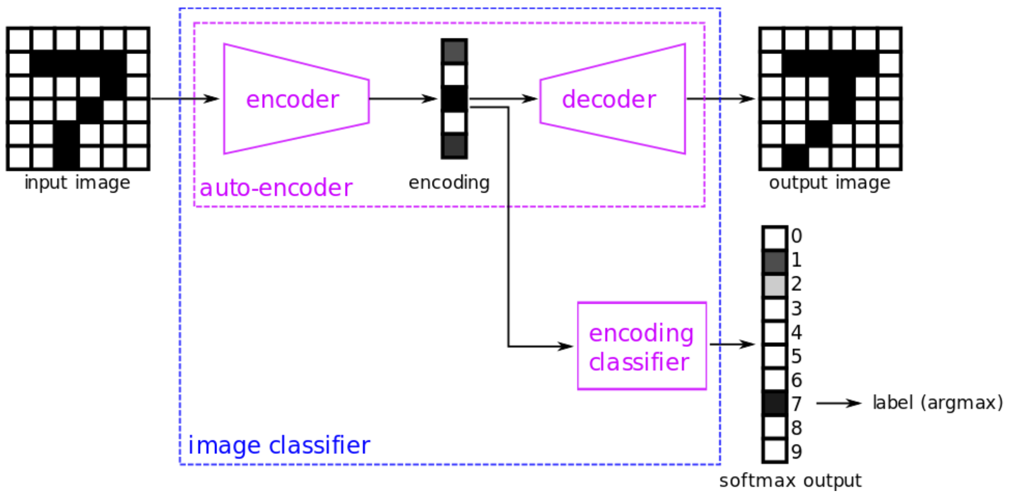
CAE



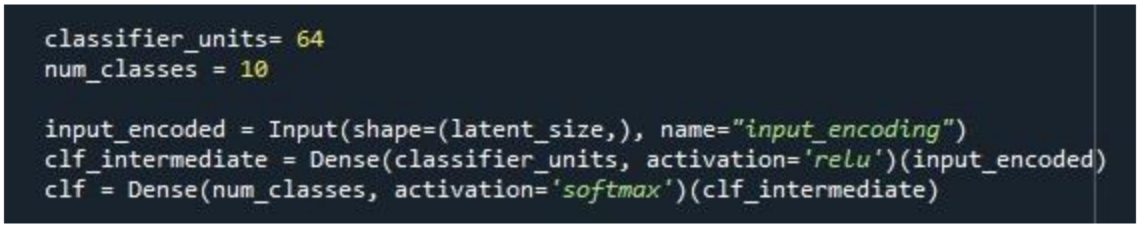
In the left, one can see assigned cluster to 2-dim latent variables by clustering. In the right, one can see real labels for 2-dim latent. Obviously, you can see lots of similarity between assigning clusters and real labels. It shows the power of Encoder to putting information in just 2 dimensions.

## Task 3

In this task we are asked to do classification.



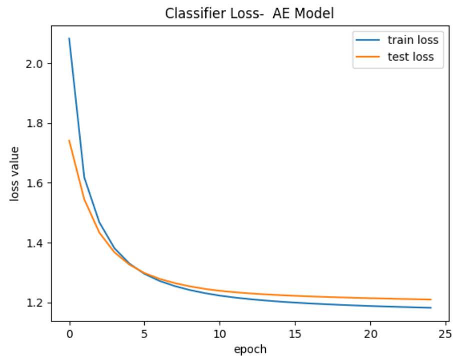
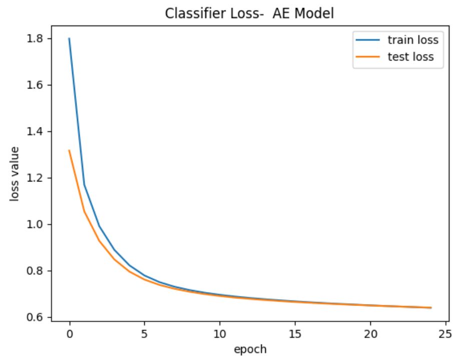
In order to do classification, we again need to use the pre-trained (in the task1) latent variable. The encoding classifier is simple MLP with an ouput with 10 neurons (for each number). The strcuture of the classifier is:

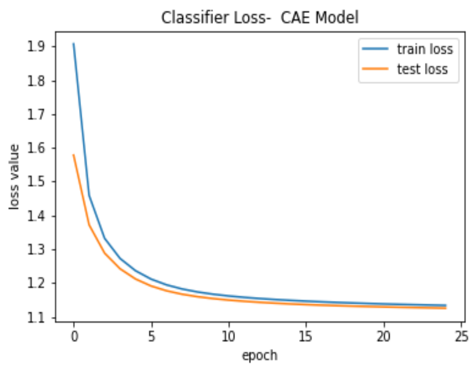


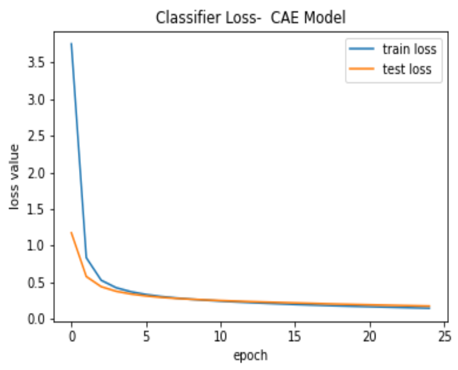
```
classifier_units= 64
num_classes = 10

input_encoded = Input(shape=(latent_size,), name="input_encoding")
clf_intermediate = Dense(classifier_units, activation='relu')(input_encoded)
clf = Dense(num_classes, activation='softmax')(clf_intermediate)
```

Loss values:

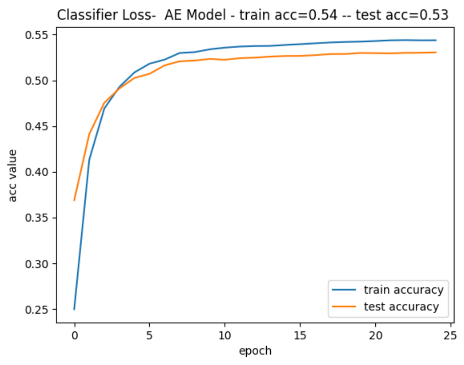
| AE | |
|---|---|
| Latent 2 | Latent 32 |
|  |  |
| CAE | |
| Latent 2 | Latent 32 |

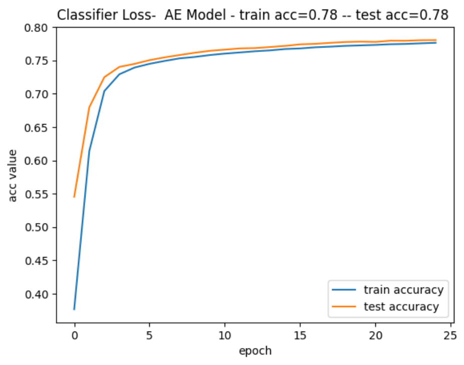




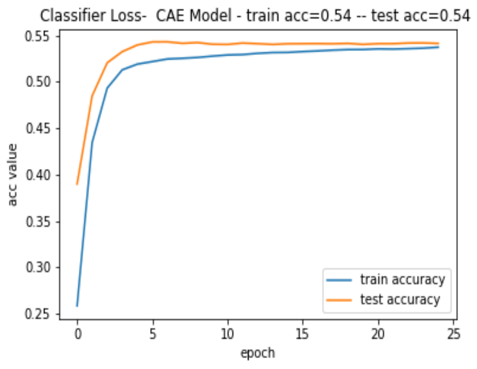
Accuracy of classification:

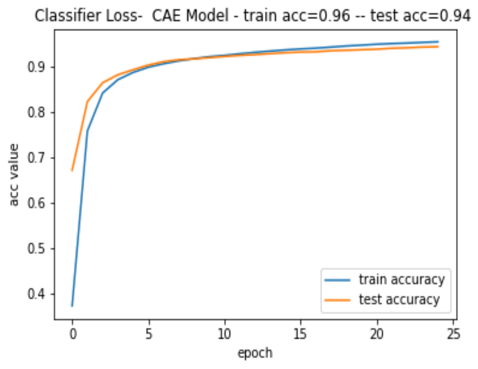
| AE | |
|---|---|
| Latent 2 | Latent 32 |





| CAE | |
|---|---|
| Latent 2 | Latent 32 |





As you see the best information is for CAE with 94% on test (please pay attention to title for each figure) and we expected this. The second one is AE with 32 latent with accuracy of 78%. For 2 latent dim we got 54% that is also acceptable.