Individual Report

Final Project

Professor/Class:

Amir Jafari/DATS 6203

Done by:

Maeshal Hijazi

Date:

12/08/2020

Introduction:

The goal of this project was to show the effective utilization of the Autoencoder (AE) model for online identification of the distribution systems (DS) topology based on a large amount of real-time phasor measurement units (PMUs) detection data. AEs are mainly used in unsupervised learning for dimensionality reduction purposes. However, in this project we used it to classify DS topologies with the help of fully connected layers. We first built an AE for each model (unsupervised) and then using the encoding layers and some fully connected layers, we were able to build the classifier. This project was divided int several tasks which were: generating the data, converting the data to heatmaps, building the models, testing the models on interfered datasets, and finally write the report and the presentation.

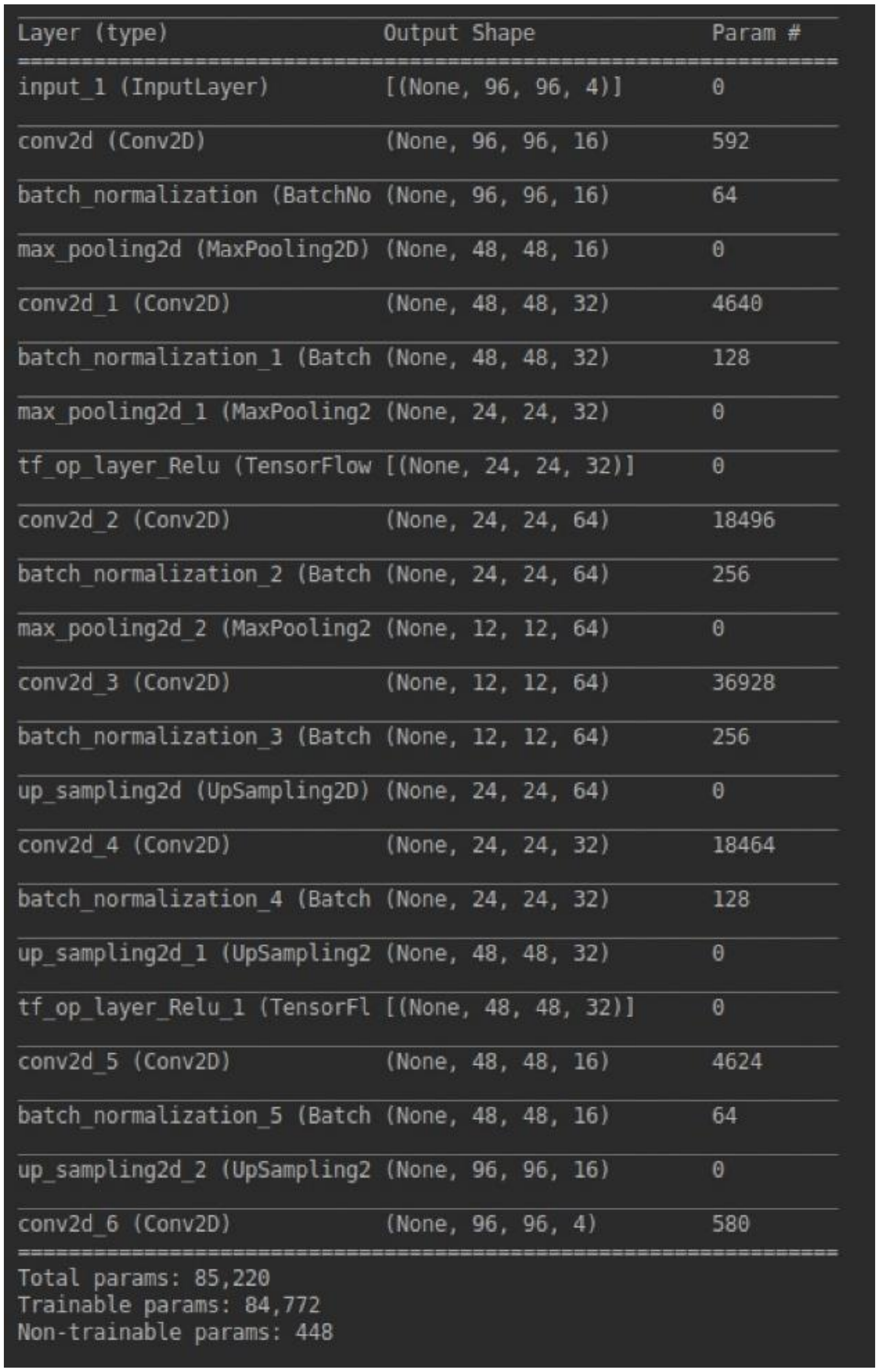
Description of Individual Work:

The project started with generating the data for the IEEE 34-Node Test Feeder for each model. These generated data were in a .csv format so we wanted to convert them to heatmaps so we can regard them as images. The process of converting the .csv files to heatmap images was very time consuming since our dataset for each model was around 20k files/images. Therefore, we divided this task between all group members. After the datasets were converted to heatmap images, we started building the models.

We divided the models into two parts: ideal and non-ideal. In the ideal model, the dataset was generated assuming that all PMUs were available across the whole bus system. Furthermore, we did not add white noise to the dataset when generated and kept all values in the matrices available. On the other hand, in the non-ideal models, one third of the PMUs were removed from the whole bus system (22PMUs), white noise was added to the dataset during generation, and removed a random value of the matrix during generation. All these changes would make these non-ideal models act as a real-world model.

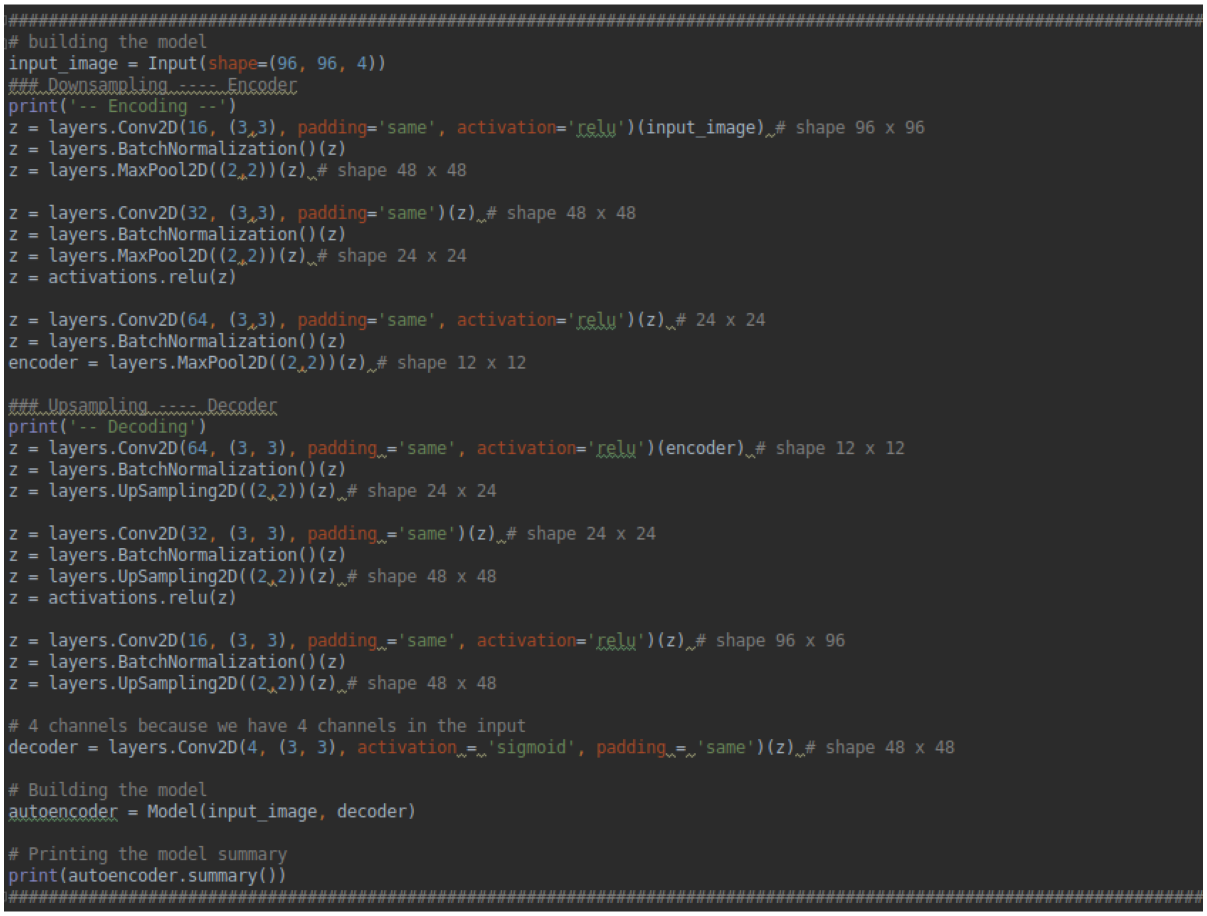
I was assigned to train these models. First, I applied the AE architecture on the ideal model to see if the idea is doable [1]. Before building the AE, I normalized each image by dividing them by 255 and resized them to 96 x 96. For the AE, I had one layer for the input, 5 hidden layers for the encoding and decoding, and one layer for the output as shown in Figure 1. Each layer had a max pooling layer with a filter of 2x2 and batch normalization layer attached to it. The code I wrote for the AE for all models (ideal and non-ideal) is shown in Figure 2. The flowchart of the proposed autoencoder used in all models is also shown in Figure 3. I have trained this AE for all models with mean squared error as loss metric. The hyper parameters for all AEs where the same except for the learning rate. The batch size was 512 and the number of epochs was 30. The learning rate for training the AE for the ideal model was 0.01 while for the non-ideal models was 0.001. This was because the non-ideal models are more complex, so they need a lower learning rate.

After training the AEs for each model, I used the encoding part to build the classifier to classify the topologies while keeping them untrainable and having the same weights they had on their last epoch. For the ideal model, I have used two fully connected layers with 128 and 8 neurons with the encoding layers of the AE, respectively as shown in Figure 4. However, for the non-ideal models, I used 512 and 8 neurons with the encoding layers of the AEs, respectively, for the fully connected layers used for classification. For the non-ideal classifier, I added a dropout of 0.5 to tackle over fitting. To build the classifiers for all models, I have used the categorical cross-entropy loss metric. All hyper parameters were kept the same except for the epochs where it decreased to 10 for model types.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 96, 96, 4)]       0
_________________________________________________________________
conv2d (Conv2D)              (None, 96, 96, 16)        592
_________________________________________________________________
batch_normalization (BatchNo (None, 96, 96, 16)        64
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 48, 48, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 48, 48, 32)        4640
_________________________________________________________________
batch_normalization_1 (Batch (None, 48, 48, 32)        128
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 24, 24, 32)        0
_________________________________________________________________
tf_op_layer_Relu (TensorFlow [(None, 24, 24, 32)]      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 24, 24, 64)        18496
_________________________________________________________________
batch_normalization_2 (Batch (None, 24, 24, 64)        256
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 12, 12, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 12, 12, 64)        36928
_________________________________________________________________
batch_normalization_3 (Batch (None, 12, 12, 64)        256
_________________________________________________________________
up_sampling2d (UpSampling2D) (None, 24, 24, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 24, 24, 32)        18464
_________________________________________________________________
batch_normalization_4 (Batch (None, 24, 24, 32)        128
_________________________________________________________________
up_sampling2d_1 (UpSampling2 (None, 48, 48, 32)        0
_________________________________________________________________
tf_op_layer_Relu_1 (TensorFl [(None, 48, 48, 32)]      0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 48, 48, 16)        4624
_________________________________________________________________
batch_normalization_5 (Batch (None, 48, 48, 16)        64
_________________________________________________________________
up_sampling2d_2 (UpSampling2 (None, 96, 96, 16)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 96, 96, 4)         580
=================================================================
Total params: 85,220
Trainable params: 84,772
Non-trainable params: 448
```

Figure 1 – Detailed layer description for the AE in all models



```
########################################################################################################################
# building the model
input_image = Input(shape=(96, 96, 4))
### Downsampling ----- Encoder
print('-- Encoding --')
z = layers.Conv2D(16, (3,3), padding='same', activation='relu')(input_image) # shape 96 x 96
z = layers.BatchNormalization()(z)
z = layers.MaxPool2D((2,2))(z) # shape 48 x 48

z = layers.Conv2D(32, (3,3), padding='same')(z) # shape 48 x 48
z = layers.BatchNormalization()(z)
z = layers.MaxPool2D((2,2))(z) # shape 24 x 24
z = activations.relu(z)

z = layers.Conv2D(64, (3,3), padding='same', activation='relu')(z) # 24 x 24
z = layers.BatchNormalization()(z)
encoder = layers.MaxPool2D((2,2))(z) # shape 12 x 12

### Upsampling ----- Decoder
print('-- Decoding')
z = layers.Conv2D(64, (3, 3), padding ='same', activation='relu')(encoder) # shape 12 x 12
z = layers.BatchNormalization()(z)
z = layers.UpSampling2D((2,2))(z) # shape 24 x 24

z = layers.Conv2D(32, (3, 3), padding ='same')(z) # shape 24 x 24
z = layers.BatchNormalization()(z)
z = layers.UpSampling2D((2,2))(z) # shape 48 x 48
z = activations.relu(z)

z = layers.Conv2D(16, (3, 3), padding ='same', activation='relu')(z) # shape 96 x 96
z = layers.BatchNormalization()(z)
z = layers.UpSampling2D((2,2))(z) # shape 48 x 48

# 4 channels because we have 4 channels in the input
decoder = layers.Conv2D(4, (3, 3), activation = 'sigmoid', padding = 'same')(z) # shape 48 x 48

# Building the model
autoencoder = Model(input_image, decoder)

# Printing the model summary
print(autoencoder.summary())
########################################################################################################################
```

Figure 2 – AE code for all models

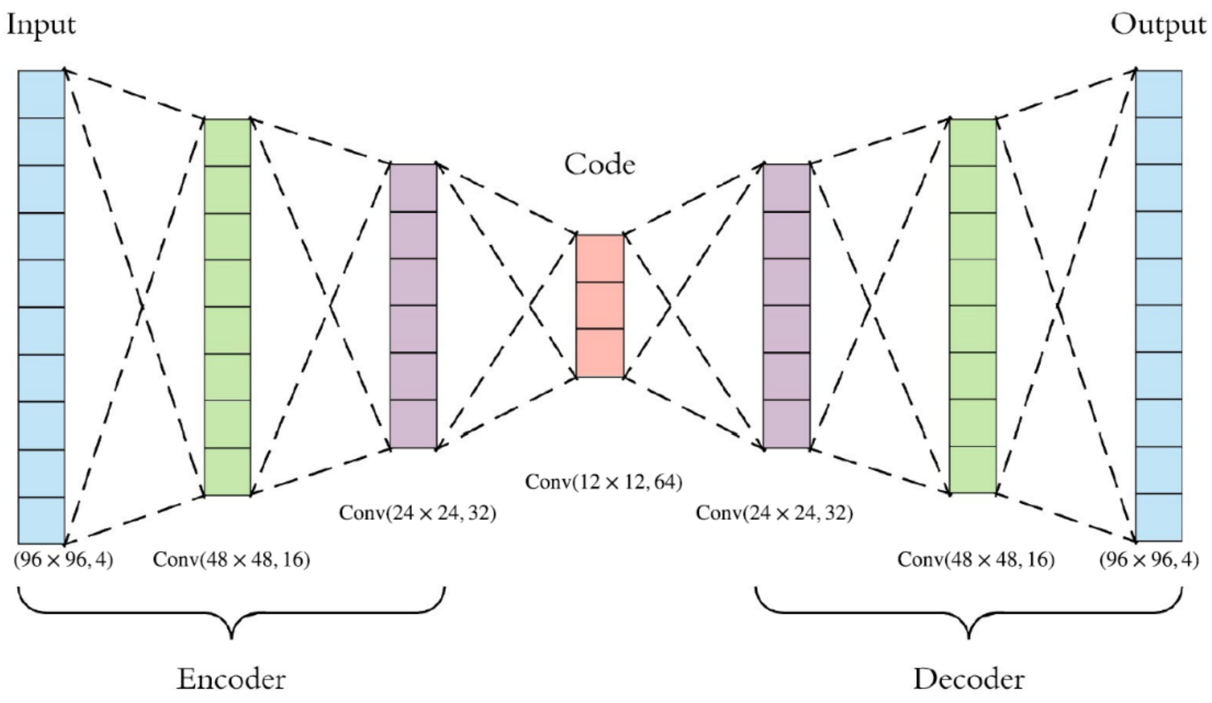
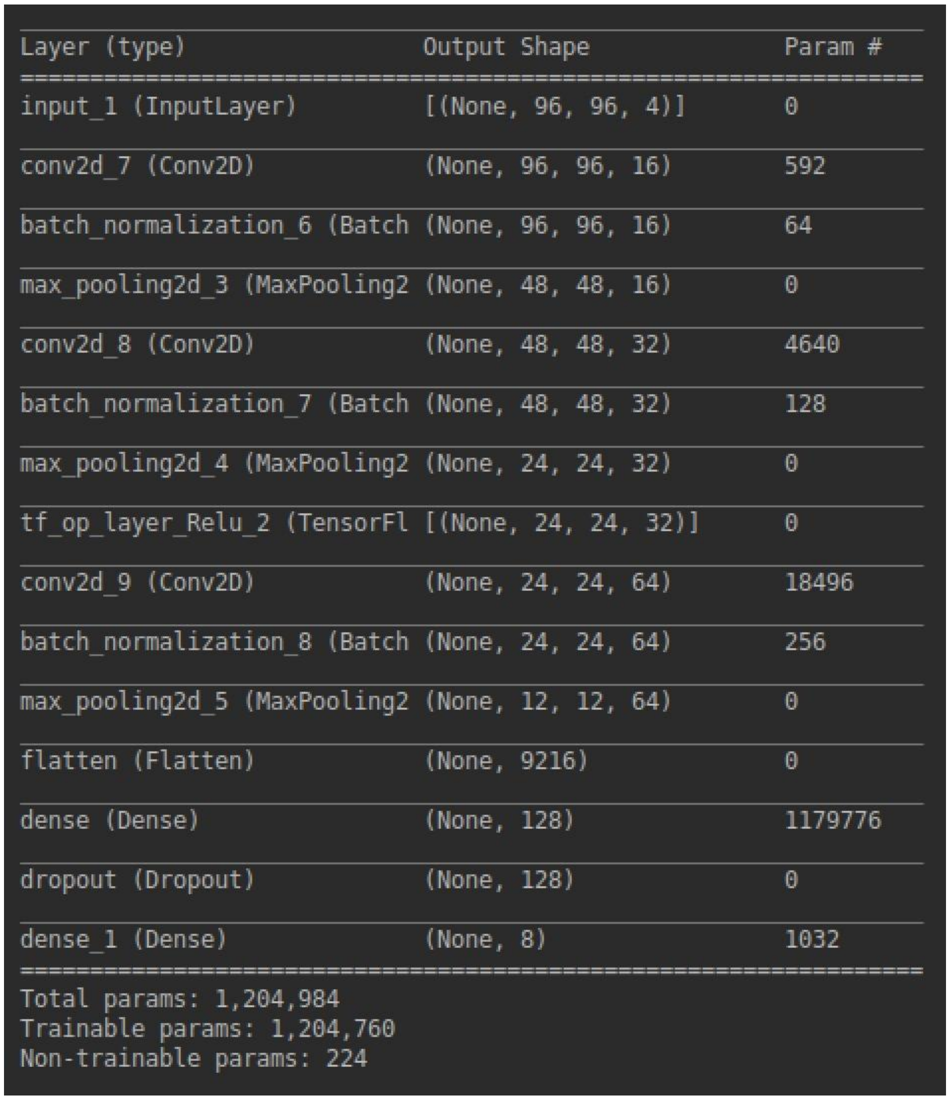


Figure 3 – The flowchart of the proposed autoencoder used in all models



```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 96, 96, 4)]       0

conv2d_7 (Conv2D)            (None, 96, 96, 16)        592

batch_normalization_6 (Batch (None, 96, 96, 16)        64

max_pooling2d_3 (MaxPooling2 (None, 48, 48, 16)        0

conv2d_8 (Conv2D)            (None, 48, 48, 32)        4640

batch_normalization_7 (Batch (None, 48, 48, 32)        128

max_pooling2d_4 (MaxPooling2 (None, 24, 24, 32)        0

tf_op_layer_Relu_2 (TensorFl [(None, 24, 24, 32)]      0

conv2d_9 (Conv2D)            (None, 24, 24, 64)        18496

batch_normalization_8 (Batch (None, 24, 24, 64)        256

max_pooling2d_5 (MaxPooling2 (None, 12, 12, 64)        0

flatten (Flatten)            (None, 9216)              0

dense (Dense)                (None, 128)               1179776

dropout (Dropout)            (None, 128)               0

dense_1 (Dense)              (None, 8)                 1032
=================================================================
Total params: 1,204,984
Trainable params: 1,204,760
Non-trainable params: 224
```

Figure 4 – Detailed layer description for the classifier in the ideal model (no dropout)

Results:

For training the AE for the ideal model, the training loss vs the validation loss plot is shown in Figure 5. It indicates that there is no sign of over fitting since both errors are decreasing. The AE of the ideal model is then tested on some unseen test images as shown in Figure 6. It shows that the AE for the ideal model has been well trained. After training the ideal model AE, I trained the classifier using the encoding layers of the AE. The training loss vs the validation loss for the classification of the topologies for the ideal model is shown in Figure 7. This figure also indicates that there is no sign of over fitting in my model. Lastly, I have tested the Ideal model classifier on the test percent and got a 100% accuracy as shown in Figure 8.

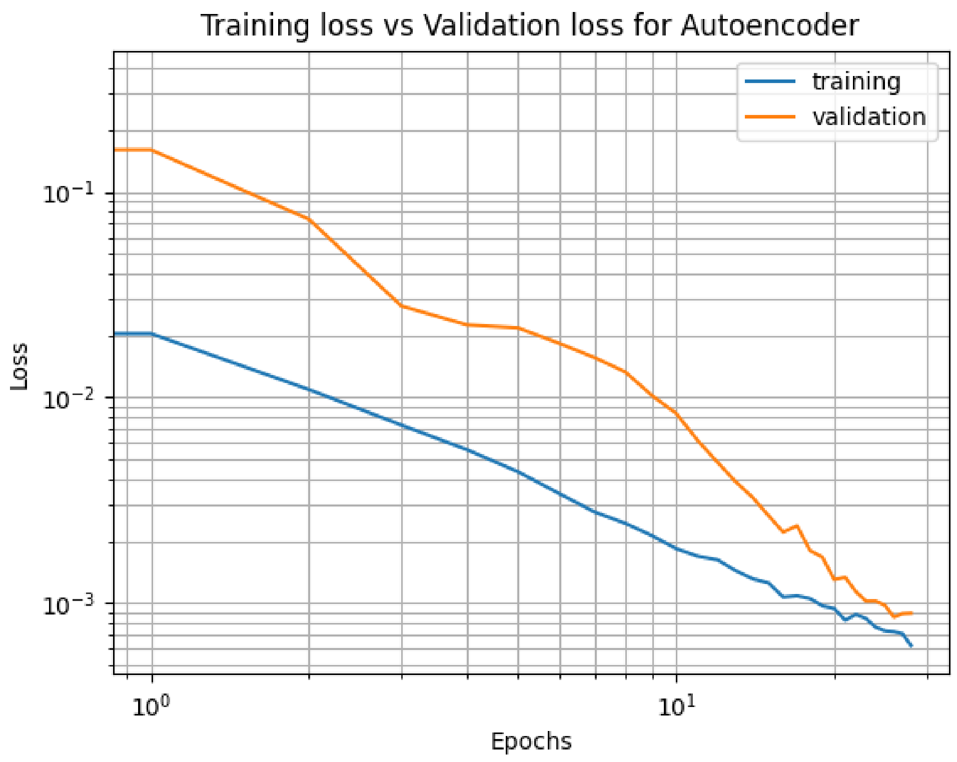


Figure 5 - Training loss vs validation loss for AE in the ideal model

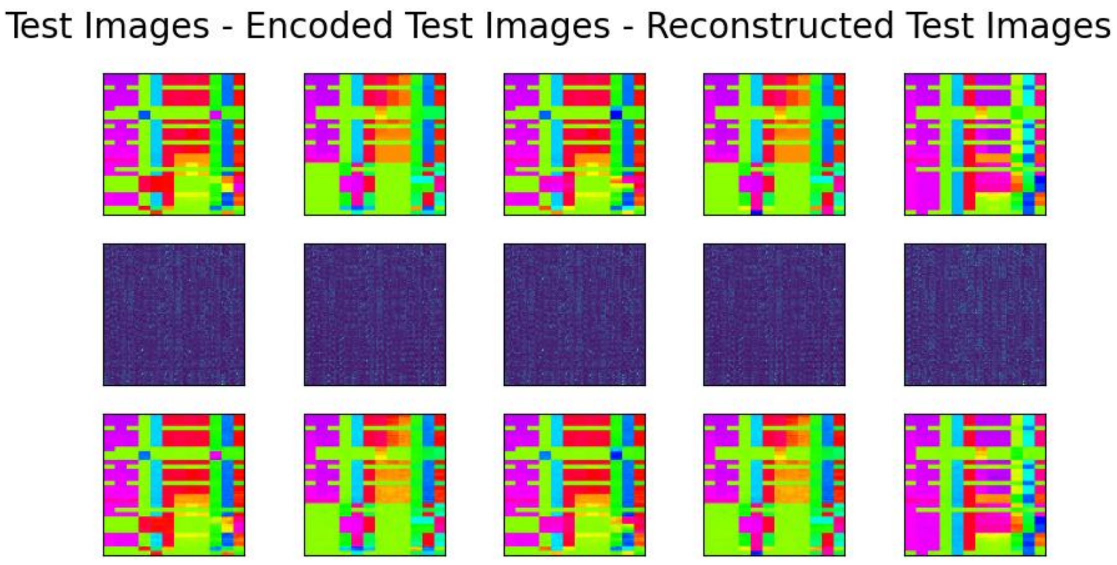

Test Images - Encoded Test Images - Reconstructed Test Images

Figure 6 – AE prediction on some test set images (Image – Encoding – Decoding)

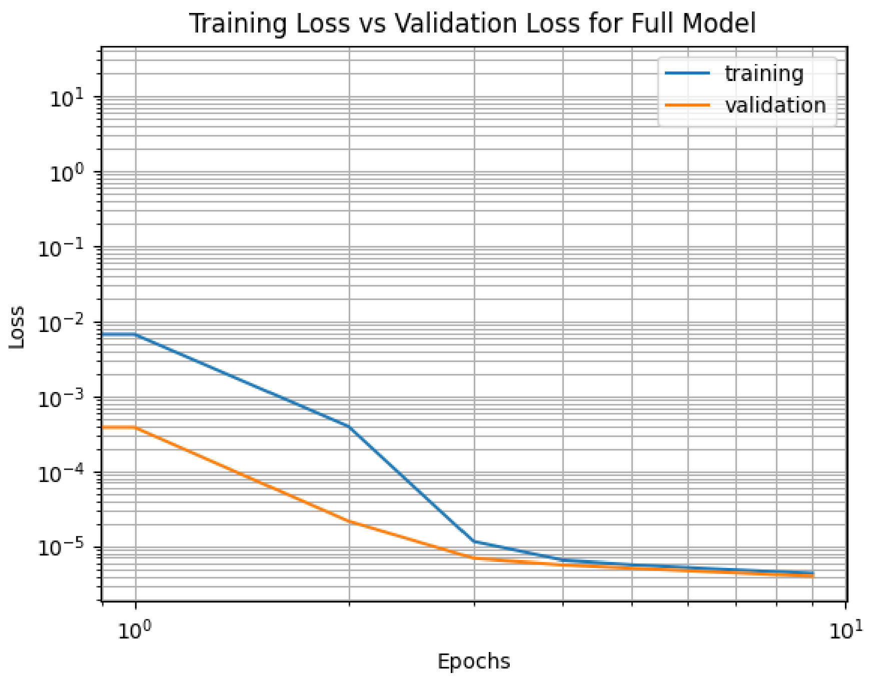
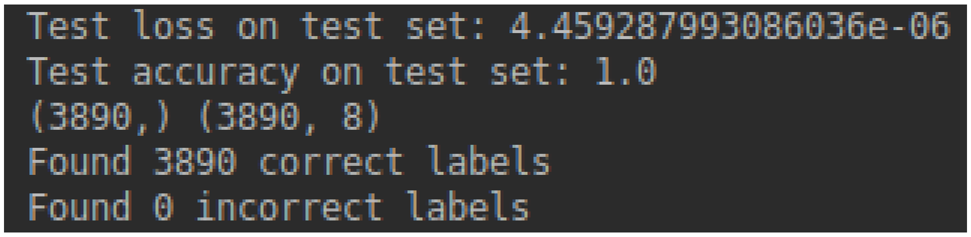


Figure 7 - Training loss vs validation loss for the classifier in the ideal model



```
Test loss on test set: 4.459287993086036e-06
Test accuracy on test set: 1.0
(3890,) (3890, 8)
Found 3890 correct labels
Found 0 incorrect labels
```

Figure 8 – Test accuracy on test set for the ideal model classifier

After building the ideal model, it was the time to the build the non-ideal models. The training loss vs the validation loss plot for the AE with 22 PMU and 10dB white noise is shown in Figure 8. Furthermore, the same plot is shown in Figure 9 for the AE with 22PMU, 10 dB white noise, and missing one data. Both plots show that all errors are decreasing which is a good sign of not over fitting our AEs. Both AEs for each non-ideal model has been tested on unseen test images as shown in Figures 11 and 12. Both Figures indicate that AEs were trained perfectly since they reconstructed the unseen test images almost correctly. After I made sure that I built the AEs for each model correctly, I moved into building the classifiers. For both models, I have used the encoding layers trained in the previous step to help in classifying the 8 topologies. I freeze all encoding layers and added some fully connected layers. The training loss vs the validation loss for the classifier with 22PMU & 10dB snr white noise is shown in Figure 13 while the plot for the classifier with 22PMU, missing one Data, and 10 dB snr white noise is shown in Figure 14. Both classifiers have been tested on unseen test images, as shown in Figures 15 and 16, and got a 100% accuracy indicating the effectiveness of the classifiers I built.

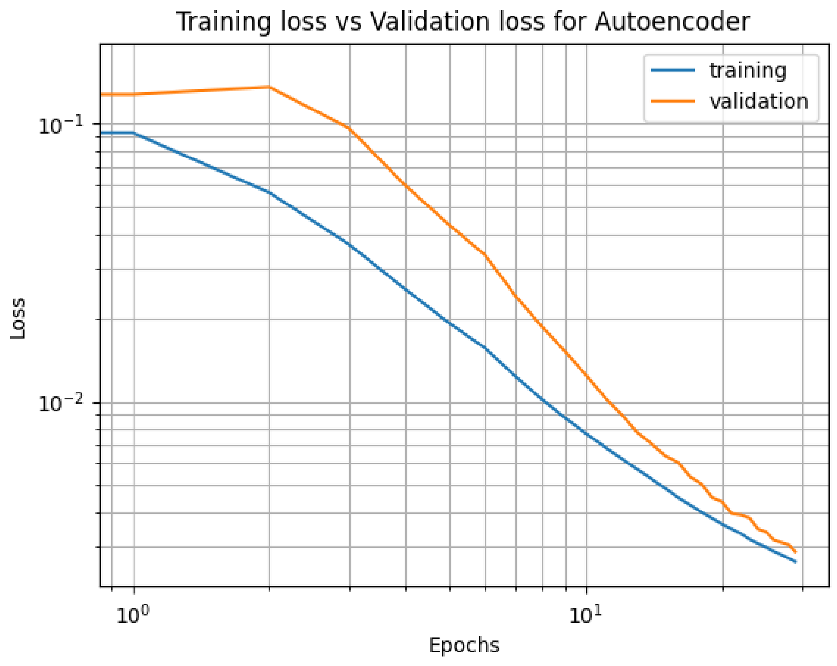


Figure 9 - Training loss vs validation loss for the AE with 22PMU & 10dB snr white noise model

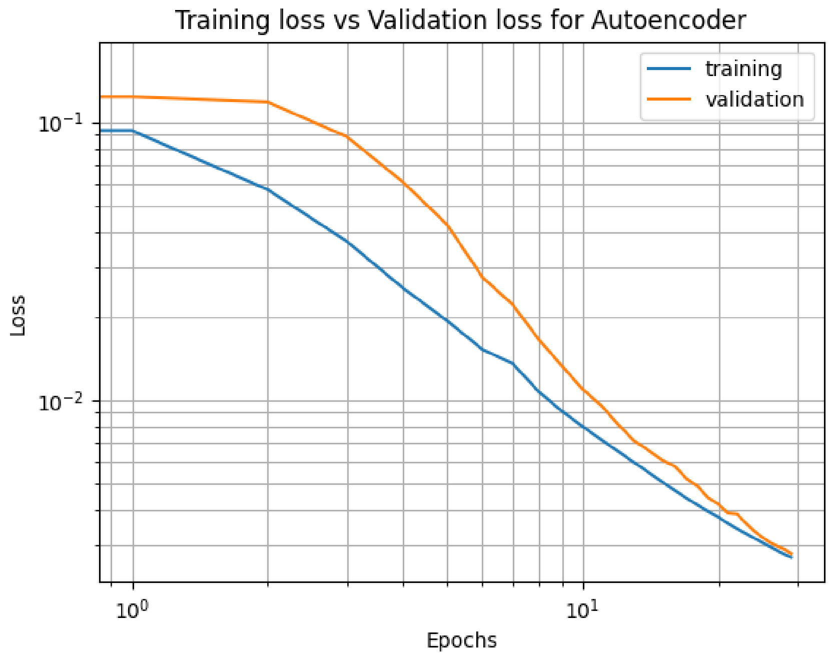


Figure 10 - Training loss vs validation loss for the AE with 22PMU, missing one data, and 10 dB snr white noise model

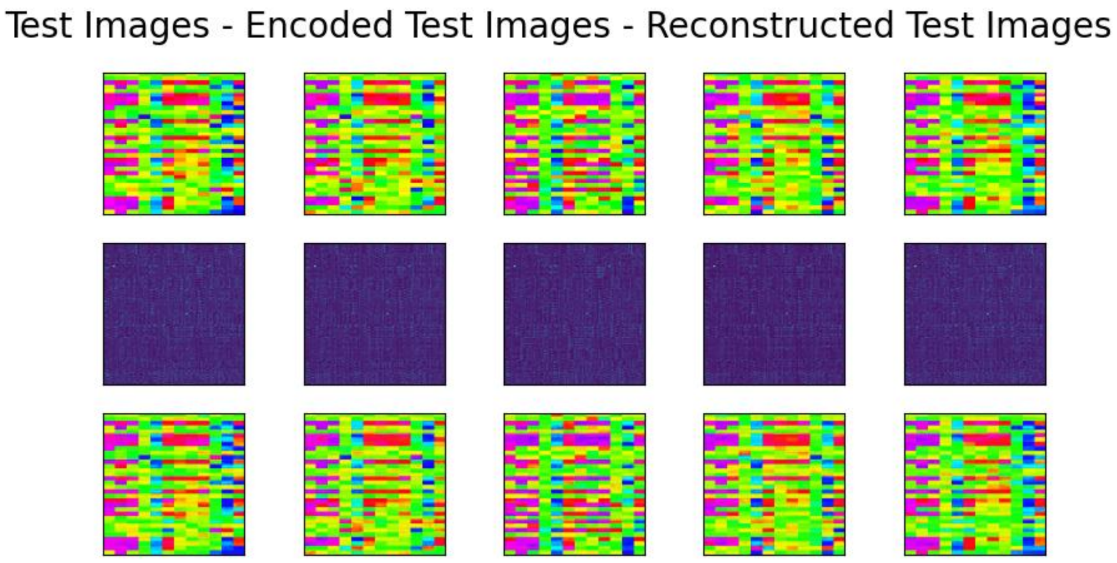

Test Images - Encoded Test Images - Reconstructed Test Images

Figure 11 – AE with 22PMU & 10dB snr white noise model prediction on some test set images (Image – Encoding – Decoding)

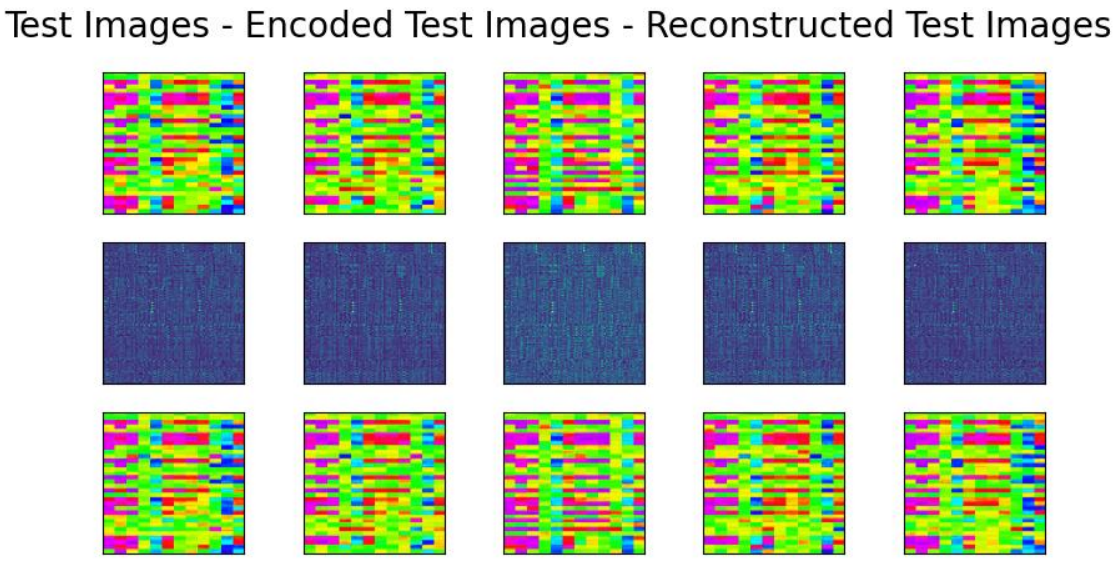

Test Images - Encoded Test Images - Reconstructed Test Images

Figure 12 – AE with 22PMU, missing one data, and 10 dB snr white noise model prediction on some test set images (Image – Encoding – Decoding)

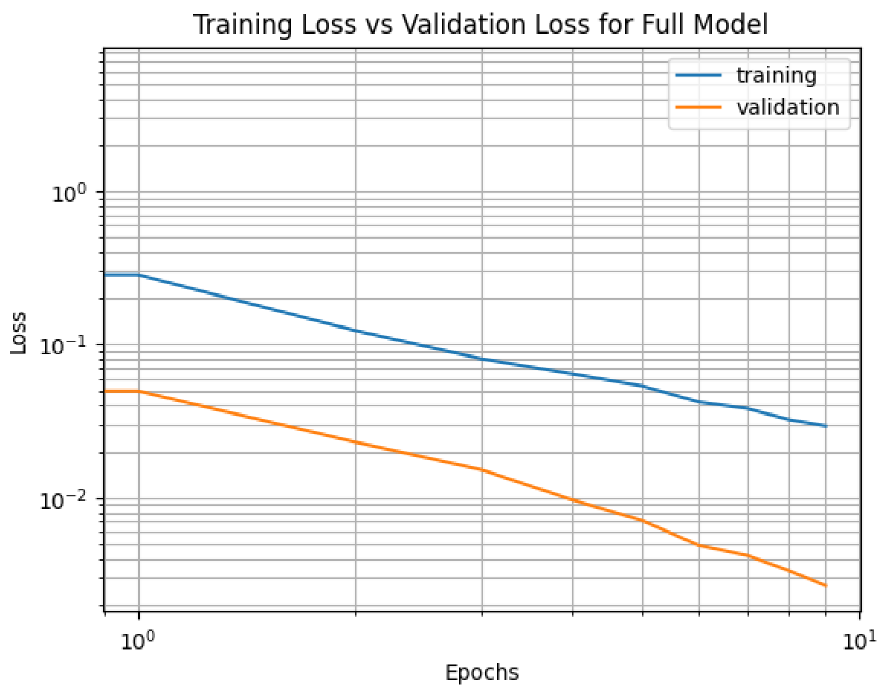


Figure 13 - Training loss vs validation loss for the 22PMU & 10dB snr white noise classifier

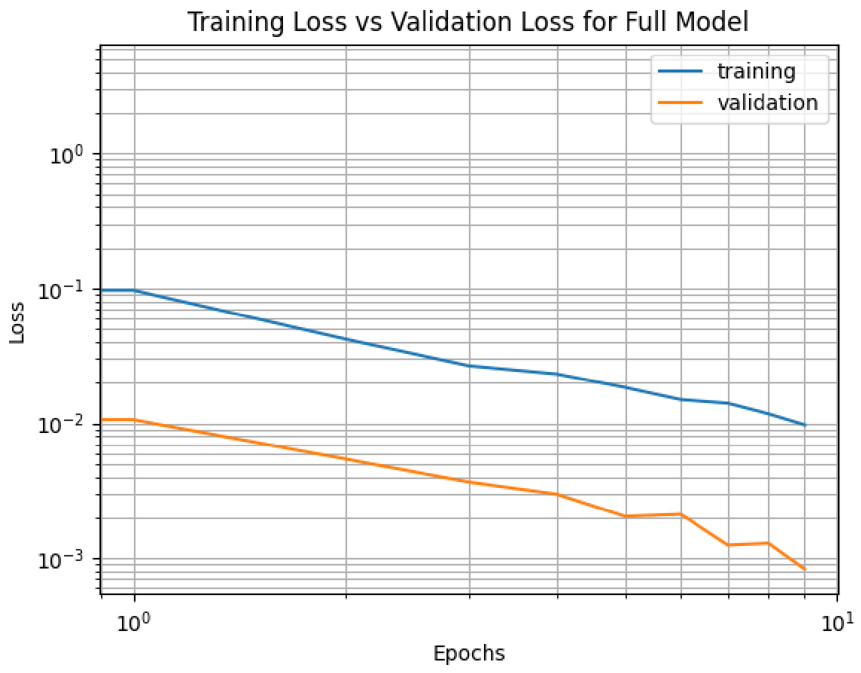
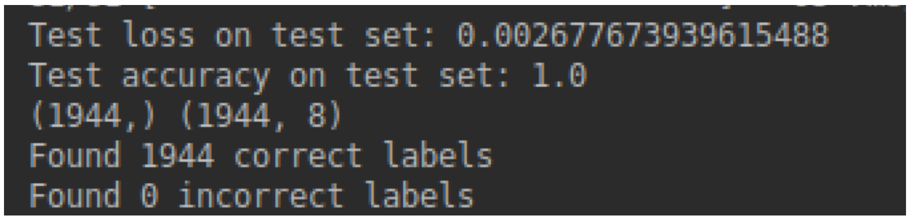
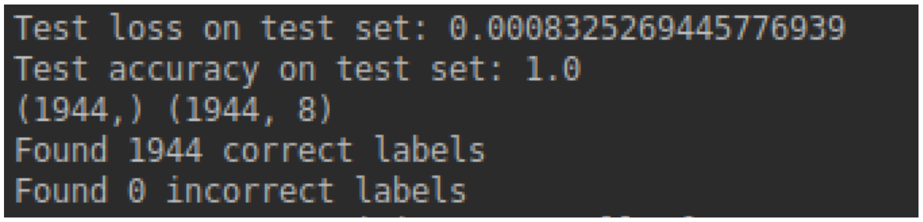


Figure 14 - Training loss vs validation loss for the 22PMU, missing one data, and 10 dB snr white noise model classifier



```
Test loss on test set: 0.002677673939615488
Test accuracy on test set: 1.0
(1944,) (1944, 8)
Found 1944 correct labels
Found 0 incorrect labels
```

Figure 15 – Test accuracy on test set for the 10dB snr white noise model classifier

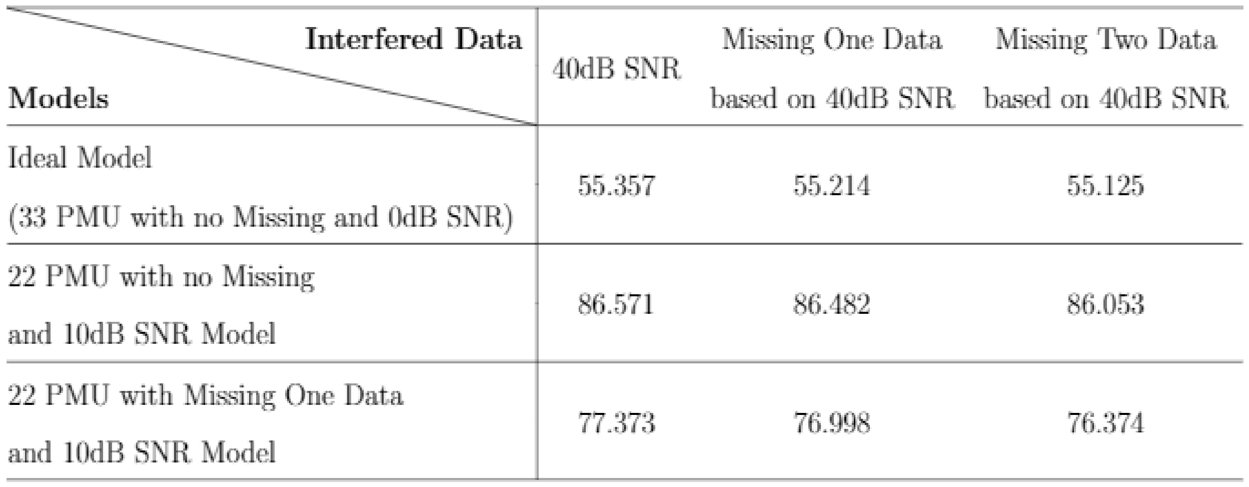


```
Test loss on test set: 0.0008325269445776939
Test accuracy on test set: 1.0
(1944,) (1944, 8)
Found 1944 correct labels
Found 0 incorrect labels
```

Figure 16 – Test accuracy on test set for the missing one Data and 10 dB snr white noise model classifier

To test the classifiers I built, ideal and non-ideal ones, we have tested these classifiers on interfered datasets. The models I have trained were built on with each load having the ability to change between 95% to 105%. However, these interfered datasets were generated with each load having the ability to change between 93% to 95% and 105% to 107%. Thus, it indicates that these datasets were not seen in the classifiers I built. The interfered datasets are composed of three different datasets which are: 40dB SNR, 40dB SNR with one data missing, and 40dB with two data missing. Since the classifier I trained were not reproducible, I had to train each classifier 5 to 10 times and then test that classifier on each interfered dataset. Then, we averaged all the results and recorded them in Table 1. Looking at the Table, since the ideal model is less related to real-world, it had the lowest accuracy. Furthermore, as we move from left to right, the interfered dataset gets more complex yielding to a lower accuracy which is satisfied below.

Table 1 – Testing each classifier on different interfered datasets



Interfered Data / **Models**	40dB SNR	Missing One Data based on 40dB SNR	Missing Two Data based on 40dB SNR
Ideal Model (33 PMU with no Missing and 0dB SNR)	55.357	55.214	55.125
22 PMU with no Missing and 10dB SNR Model	86.571	86.482	86.053
22 PMU with Missing One Data and 10dB SNR Model	77.373	76.998	76.374

Conclusion:

The objective of this final project was to classify the topology of the IEEE 34-Test Node Feeder. This has been done using AEs. The AEs were first trained and then tested to make sure they are working effectively. Then, with the help of the encoding layers of each AE, classifiers were trained with some fully connected layers. 3 classifiers were trained in this final project; one was ideal and two were non-ideal. Then all trained classifiers were tested on interfered datasets that these classifiers did not see before. Some improvements I suggest using cross validation or assembling techniques to increase the accuracy of the interfered datasets. 20% of my code that I found or copied from the internet and then I modify it to correspond to our models.

References:

[1] https://www.datacamp.com/community/tutorials/autoencoder-classifier-python