PS TERM PROJECT

Course Instructor: Dr. Ranjana Sodhi
Teaching Assistants: Swati Agarwal , Manish

Submitted by:
Akarshi Roy Choudhury (2021EEB1149)
Ashwini Sahane (2021EEB1159)
Bhumika Chaudhari (2021EEB1160)
Lokesh Jassal (2021EEB1185)
Vishakha Agarwal (2021EEB1221)

Aim: Use of smart meter data for Load Forecasting.

The flow of the report:

(a) Approach towards forecasting (for both clustered and non-clustered data):

1) Convolutional Neural Networks (CNN)
2) Long Short Term Memory (LSTM)
3) Dense Neural Network (DNN)
4) Other ML models

(b) Our Engineering:

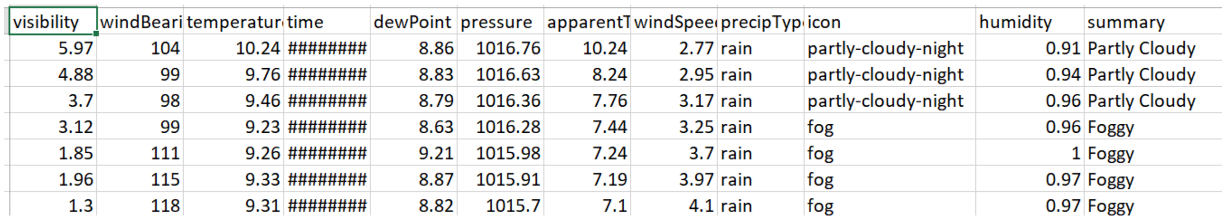
5) CNN + LSTM (in series)
6) CNN + LSTM (in parallel)

(c) Future Innovations

1) Parallel LSTM Architecture
2) Non-Intrusive Load Monitoring (NILM)

Dataset used:

We used "London Dataset" for our project. This had half-hourly power consumption and hourly weather data of different houses from 112 blocks for two years. However the timespan of all the houses were different, and were many missing data points in between.

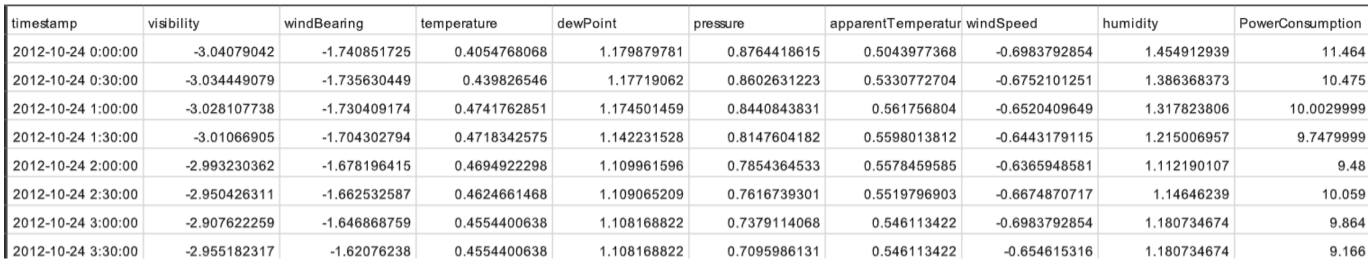


visibility	windBeari	temperatur	time	dewPoint	pressure	apparentT	windSpee	precipTyp	icon	humidity	summary
5.97	104	10.24	########	8.86	1016.76	10.24	2.77	rain	partly-cloudy-night	0.91	Partly Cloudy
4.88	99	9.76	########	8.83	1016.63	8.24	2.95	rain	partly-cloudy-night	0.94	Partly Cloudy
3.7	98	9.46	########	8.79	1016.36	7.76	3.17	rain	partly-cloudy-night	0.96	Partly Cloudy
3.12	99	9.23	########	8.63	1016.28	7.44	3.25	rain	fog	0.96	Foggy
1.85	111	9.26	########	9.21	1015.98	7.24	3.7	rain	fog	1	Foggy
1.96	115	9.33	########	8.87	1015.91	7.19	3.97	rain	fog	0.97	Foggy
1.3	118	9.31	########	8.82	1015.7	7.1	4.1	rain	fog	0.97	Foggy

Weather Data before Processing

While processing the data, we did the following changes:

- Replaced missing values with medians
- Selected a common timespan of 1 year and removed rest of the data
- Converted hourly weather data to half-hourly
- Calculated total power consumption for each block (by adding the power consumed by all the houses in it for each half hour)

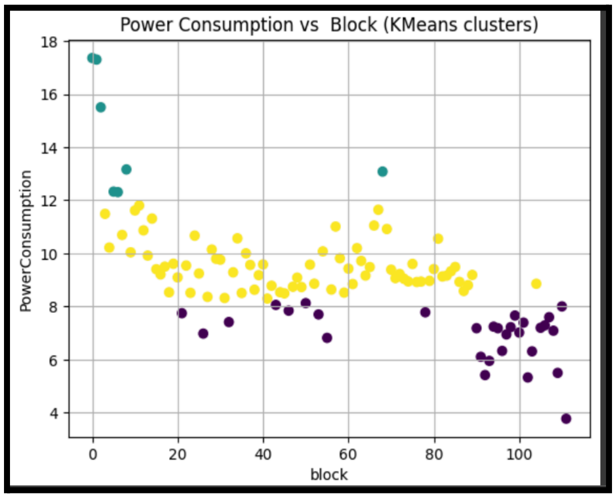


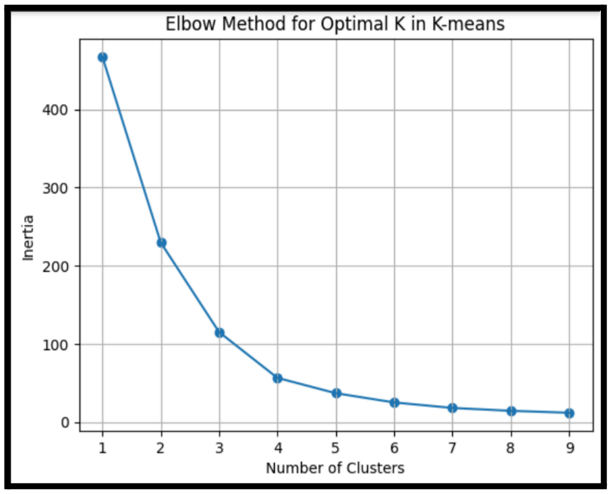
timestamp	visibility	windBearing	temperature	dewPoint	pressure	apparentTemperatur	windSpeed	humidity	PowerConsumption
2012-10-24 0:00:00	-3.04079042	-1.740851725	0.4054768068	1.179879781	0.8764418615	0.5043977368	-0.6983792854	1.454912939	11.464
2012-10-24 0:30:00	-3.034449079	-1.735630449	0.439826546	1.17719062	0.8602631223	0.5330772704	-0.6752101251	1.386368373	10.475
2012-10-24 1:00:00	-3.028107738	-1.730409174	0.4741762851	1.174501459	0.8440843831	0.561756804	-0.6520409649	1.317823806	10.0029999
2012-10-24 1:30:00	-3.01066905	-1.704302794	0.4718342575	1.142231528	0.8147604182	0.5598013812	-0.6443179115	1.215006957	9.7479999
2012-10-24 2:00:00	-2.993230362	-1.678196415	0.4694922298	1.109961596	0.7854364533	0.5578459585	-0.6365948581	1.112190107	9.48
2012-10-24 2:30:00	-2.950426311	-1.662532587	0.4624661468	1.109065209	0.7616739301	0.5519796903	-0.6674870717	1.14646239	10.059
2012-10-24 3:00:00	-2.907622259	-1.646868759	0.4554400638	1.108168822	0.7379114068	0.546113422	-0.6983792854	1.180734674	9.864
2012-10-24 3:30:00	-2.955182317	-1.62076238	0.4554400638	1.108168822	0.7095986131	0.546113422	-0.654615316	1.180734674	9.166

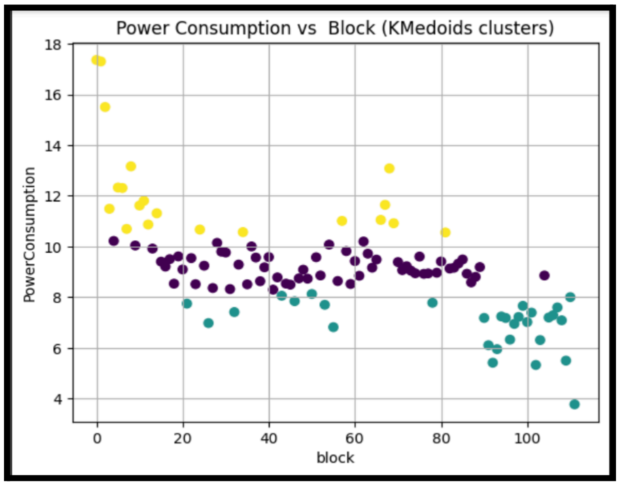
Data after Processing

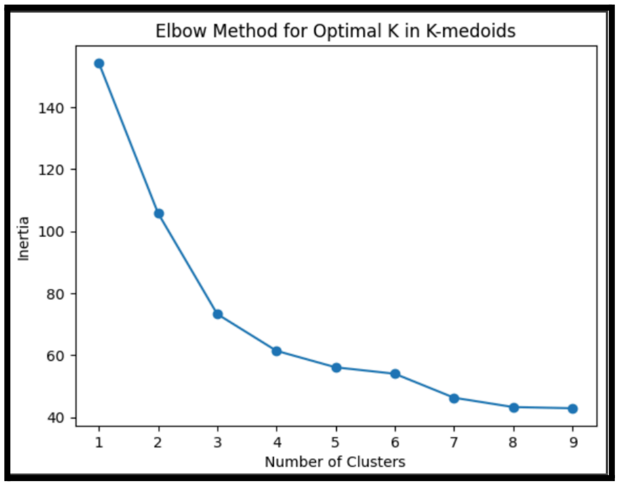
Clustering:

Clustering is used to simplify the processing of large datasets by grouping similar type of data. This reduces the training time of ML model. In our project, we have clustered together the blocks which have similar average daily consumption.









We used K-medoids clustering as it handles outliers better than K-means clustering. We chose K=3 clusters based on the elbow method.

- Top layer (High daily power consumption) - Cluster 2
- Middle layer (Medium daily power Consumption) - Cluster 1
- Bottom layer (Low daily power Consumption) - Cluster 0

Cluster Distribution (Blockwise):

- Blocks in Cluster 2: [0, 1, 2, 3, 5, 6, 7, 8, 10, 11, 12, 14, 24, 34, 57, 66, 67, 68, 69, 81]
- Blocks in Cluster 0: [4, 9, 13, 15, 16, 17, 18, 19, 20, 22, 23, 25, 27, 28, 29, 30, 31, 33, 35, 36, 37, 38, 39, 40, 41, 42, 44, 45, 47, 48, 49, 51, 52, 54, 56, 58, 59, 60, 61, 62, 63, 64, 65, 70, 71, 72, 73, 74, 75, 76, 77, 79, 80, 82, 83, 84, 85, 86, 87, 88, 89, 104]
- Blocks in Cluster 1: [21, 26, 32, 43, 46, 50, 53, 55, 78, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 105, 106, 107, 108, 109, 110, 111]

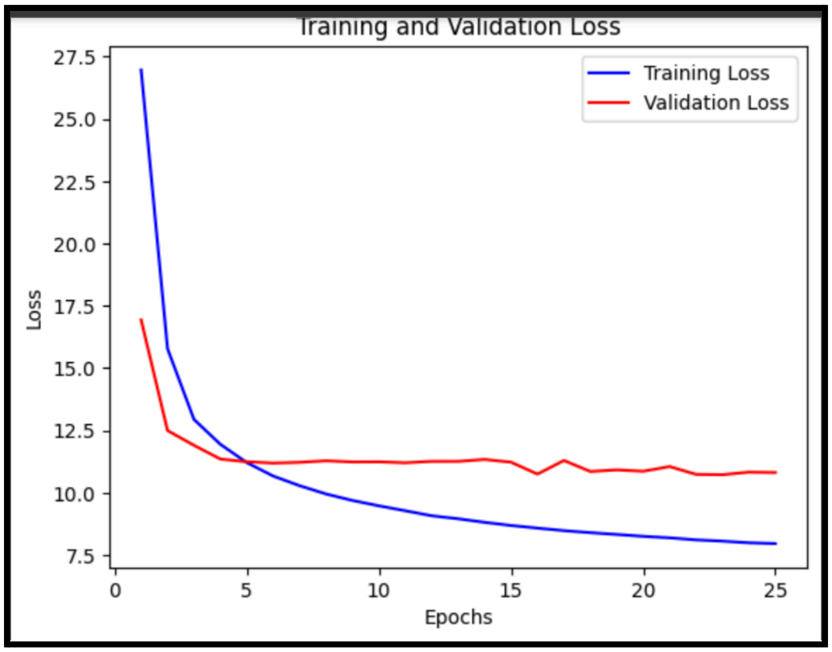
Implementations:

a1) Convolutional Neural Networks (CNN):

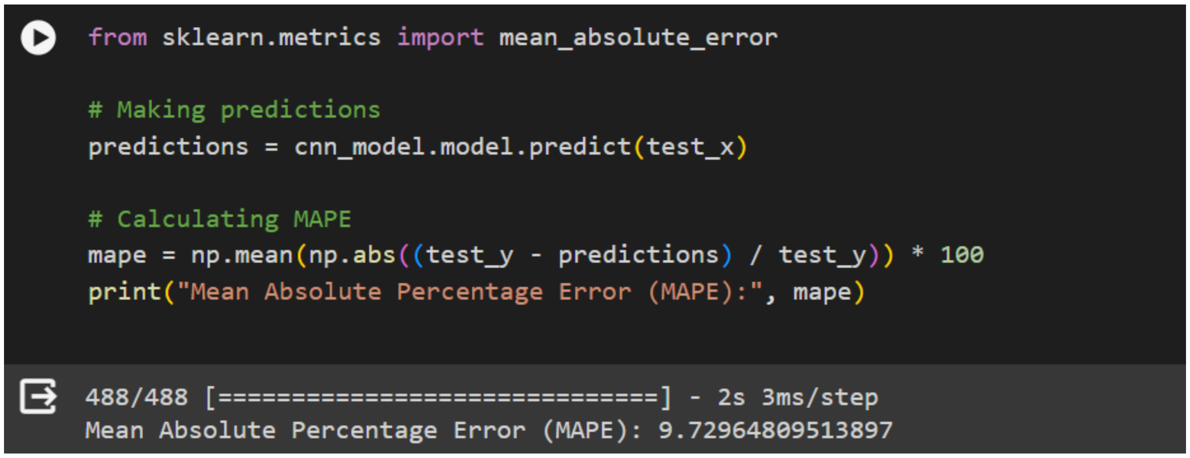
- Convolutional Neural Networks have a special ability for feature selection. Of all the multiple features available, the layers defined within this neural network choose the crucial features thereby increasing the efficiency of the forecasting system we are designing.
- CNN majorly comprise 3 layers, namely, the Convolutional layer itself which contains filters or the kernels responsible for the feature maps based on the weights and biases associated with the layer. After each of these layers, there is an activation function which decides whether the particular neuron must be activated or not.
- Nextly, we have the maxpooling layer responsible for reducing the data dimensionality of the input data thereby making CNN a very less complex and faster architecture.
- The final outputs associated are put through the dense layer which enables one to decide on the output size based on the number of neurons.

- We have applied the above model for both clustered and non-clustered data. We performed both k-mean and k-medoid clustering, however, we'll be going ahead with k-medoid as it is highly robust to noise and outliers and thus gives us better predictions.

  CNN Results (without clustering):



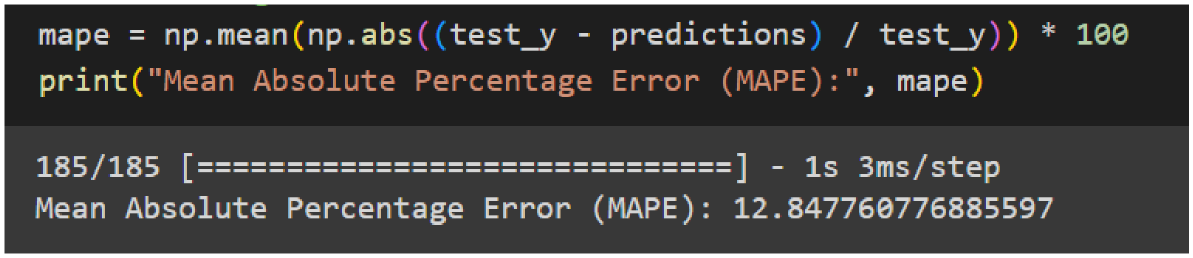
  The validation loss started to increase after 25 epochs, thus we chose number of epochs as 25.



```
from sklearn.metrics import mean_absolute_error

# Making predictions
predictions = cnn_model.model.predict(test_x)

# Calculating MAPE
mape = np.mean(np.abs((test_y - predictions) / test_y)) * 100
print("Mean Absolute Percentage Error (MAPE):", mape)
```

```
488/488 [==============================] - 2s 3ms/step
Mean Absolute Percentage Error (MAPE): 9.72964809513897
```

CNN Results (with clustering-Cluster 2):

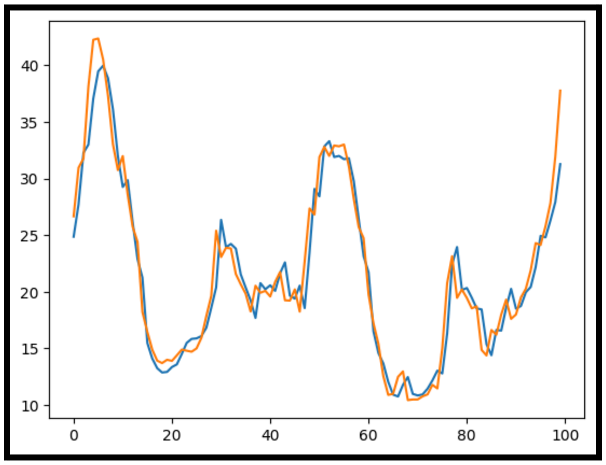


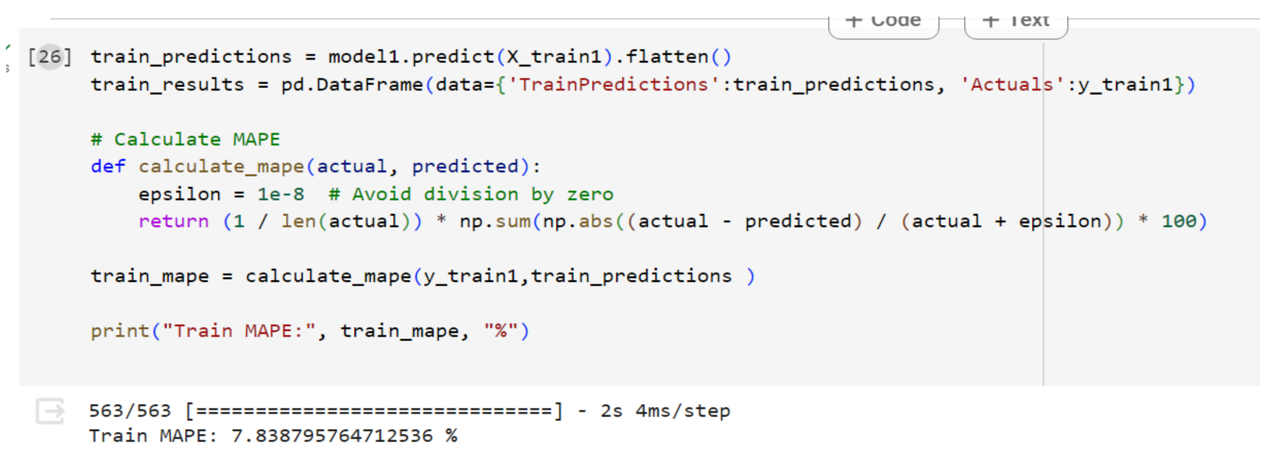
```
mape = np.mean(np.abs((test_y - predictions) / test_y)) * 100
print("Mean Absolute Percentage Error (MAPE):", mape)

185/185 [==============================] - 1s 3ms/step
Mean Absolute Percentage Error (MAPE): 12.847760776885597
```

a2) Long Short-Term Memory (LSTM):

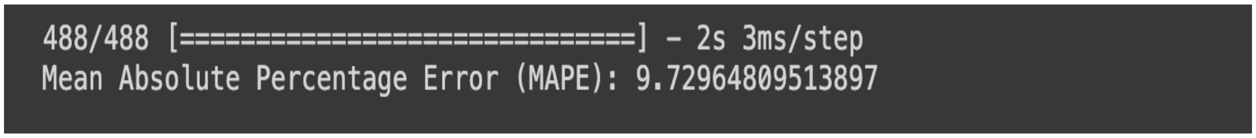
- LSTM is an improvised version of the Recurrent Neural Networks (RNN). The issue associated with RNN is that of a vanishing gradient which thus obstructs long-term dependencies of the data and the architecture for the prediction.
- Specialized neural networks called LSTM networks are designed to process data sequences. LSTMs are perfect where context over time is important because, unlike normal feedforward neural networks, they feature loops that allow information to endure.
- An input gate, a forget gate, and an output gate are the three components that make up an LSTM cell. These gates control information flow, enabling the network to forget or keep certain pieces of information gradually.

LSTM Results (without clustering):




```
[26] train_predictions = model1.predict(X_train1).flatten()
     train_results = pd.DataFrame(data={'TrainPredictions':train_predictions, 'Actuals':y_train1})

     # Calculate MAPE
     def calculate_mape(actual, predicted):
         epsilon = 1e-8  # Avoid division by zero
         return (1 / len(actual)) * np.sum(np.abs((actual - predicted) / (actual + epsilon)) * 100)

     train_mape = calculate_mape(y_train1,train_predictions )

     print("Train MAPE:", train_mape, "%")

     563/563 [==============================] - 2s 4ms/step
     Train MAPE: 7.838795764712536 %
```

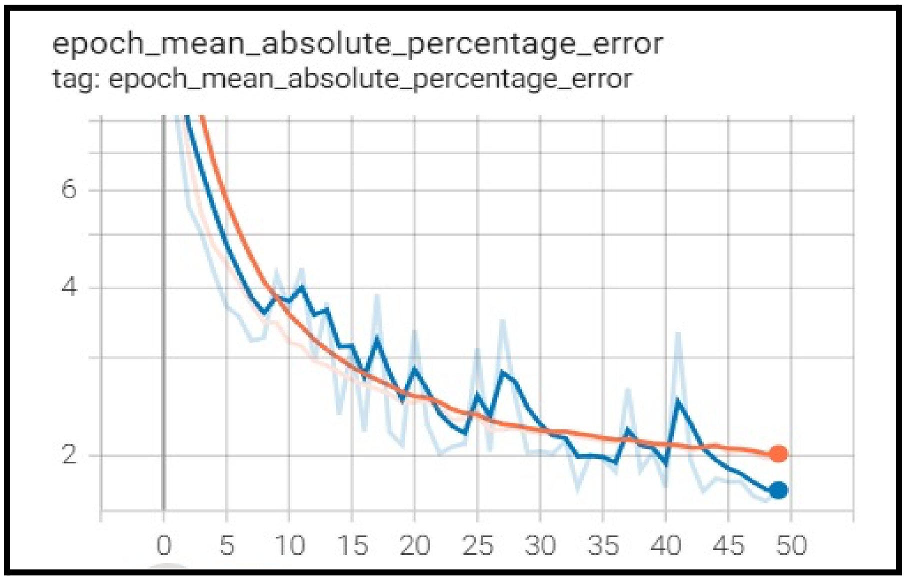
LSTM (with clustering):

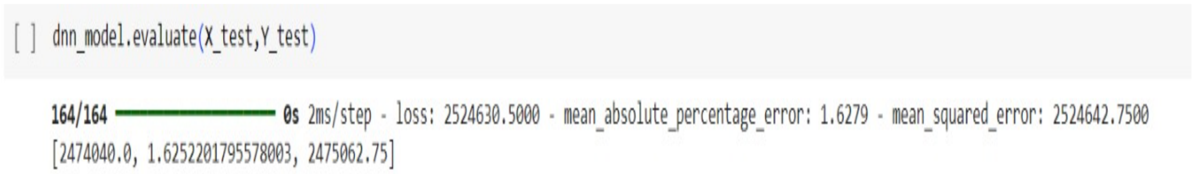

```
488/488 [==============================] - 2s 3ms/step
Mean Absolute Percentage Error (MAPE): 9.72964809513897
```

a3) Dense Neural Networks (DNN):

- Dense layers are used in every architecture for a favorable number of outputs as per the user.
- It was also observed that DNN retains the least error of all the individual architectures.
- However, unlike CNN, DNN layers require all the features/ labeled data thereby increasing its complexity, the interpretability of this architecture is also quite ambiguous and due to its massive data hunger feature, it’s considered to be quite complex.

DNN Results (without clustering):

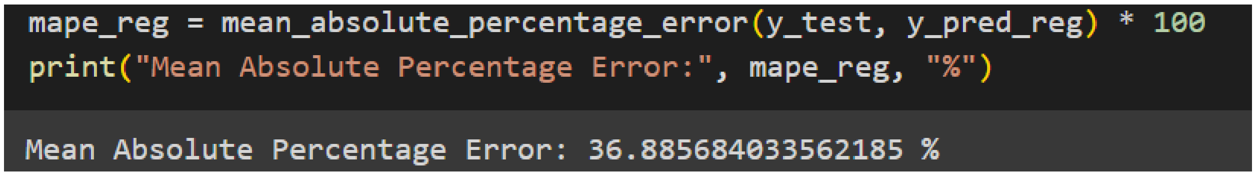




```
[ ] dnn_model.evaluate(X_test,Y_test)

164/164 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 2524630.5000 - mean_absolute_percentage_error: 1.6279 - mean_squared_error: 2524642.7500
[2474040.0, 1.6252201795578003, 2475062.75]
```

a4) Other ML models:

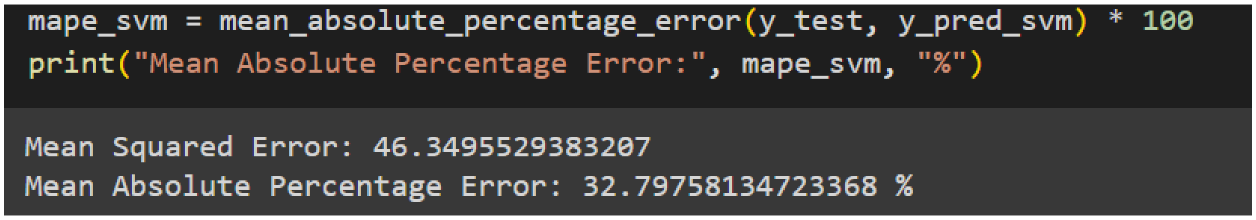
- Linear Regression:
  This ML model is quite popular for prediction purposes as we use to deduce a dependent variable's value from the independent ones.



```
mape_reg = mean_absolute_percentage_error(y_test, y_pred_reg) * 100
print("Mean Absolute Percentage Error:", mape_reg, "%")

Mean Absolute Percentage Error: 36.885684033562185 %
```

- Support Vector Machine
  This ML model is usually used in classification, one can thus makes "classes". Eg: the class of transformers working at same frequency. This helps us having an organized data set and thus a faster organized output. Thus, we can also see relatively less error here compared to linear

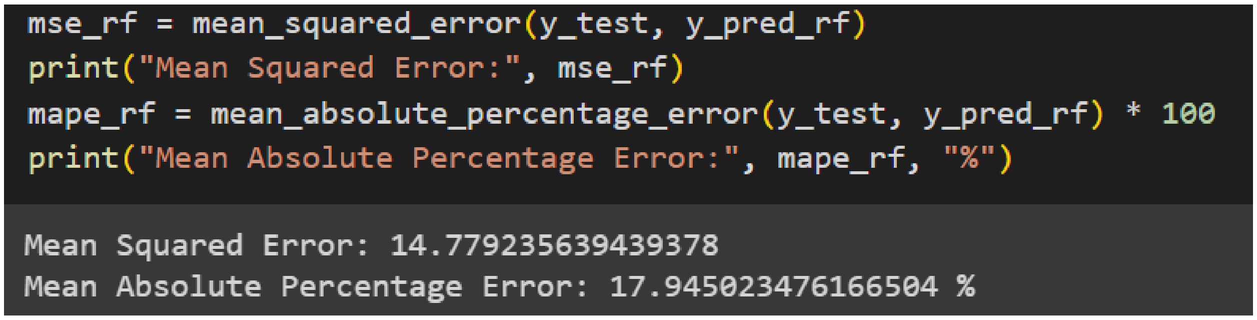
regression.



```
mape_svm = mean_absolute_percentage_error(y_test, y_pred_svm) * 100
print("Mean Absolute Percentage Error:", mape_svm, "%")
```

```
Mean Squared Error: 46.3495529383207
Mean Absolute Percentage Error: 32.79758134723368 %
```

- Random Forest
  Random forests rely on decision trees, a type of machine learning model that uses a tree-like structure to make predictions. The tree splits the data based on features (attributes) at each node, ultimately reaching a leaf node that represents a class (in classification) or a continuous value (in regression).



```
mse_rf = mean_squared_error(y_test, y_pred_rf)
print("Mean Squared Error:", mse_rf)
mape_rf = mean_absolute_percentage_error(y_test, y_pred_rf) * 100
print("Mean Absolute Percentage Error:", mape_rf, "%")
```

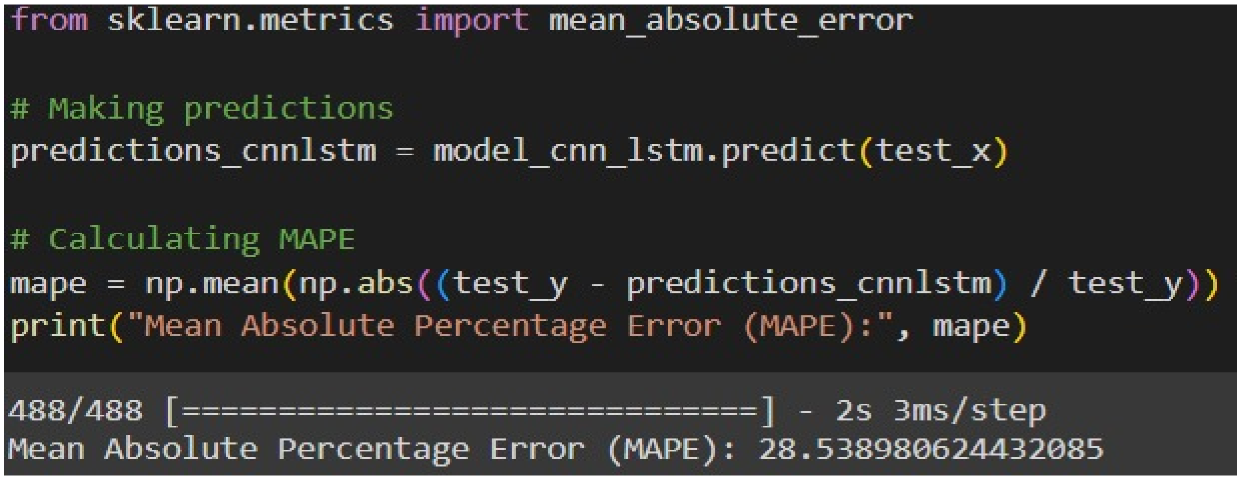
```
Mean Squared Error: 14.779235639439378
Mean Absolute Percentage Error: 17.945023476166504 %
```

Thus, it can be concluded that **of all architectures individually, LSTM is a preferable model for predictions/ load forecasting.**

Our Engineering:

Since we saw that LSTM gave the least error, we tried various combinations of LSTM with our 2nd best model (CNN). With CNNs feature extraction and LSTMs long term dependencies.

b1) CNN + LSTM (in series)

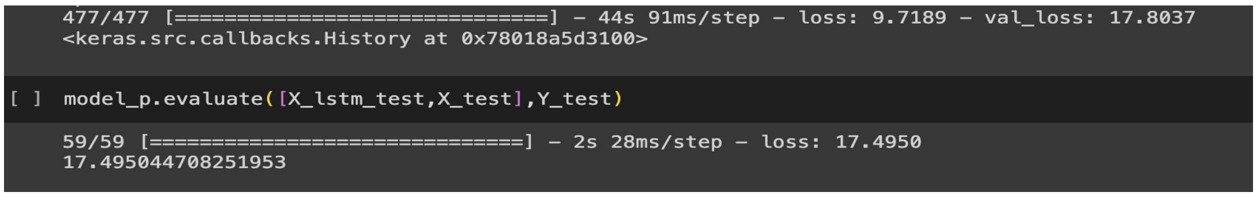


```
from sklearn.metrics import mean_absolute_error

# Making predictions
predictions_cnnlstm = model_cnn_lstm.predict(test_x)

# Calculating MAPE
mape = np.mean(np.abs((test_y - predictions_cnnlstm) / test_y))
print("Mean Absolute Percentage Error (MAPE):", mape)
```

```
488/488 [==============================] - 2s 3ms/step
Mean Absolute Percentage Error (MAPE): 28.538980624432085
```

b2) CNN + LSTM (in parallel)

Parallel CNN+LSTM Results (without clustering):

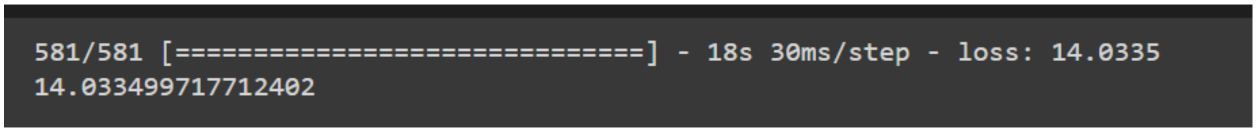


```
477/477 [==============================] - 44s 91ms/step - loss: 9.7189 - val_loss: 17.8037
<keras.src.callbacks.History at 0x78018a5d3100>
```

```
[ ] model_p.evaluate([X_lstm_test,X_test],Y_test)
```

```
59/59 [==============================] - 2s 28ms/step - loss: 17.4950
17.495044708251953
```

Parallel CNN+LSTM Results (with clustering- On cluster 0):



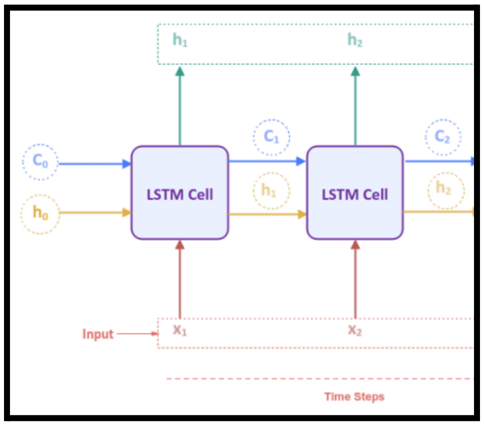
```
581/581 [==============================] - 18s 30ms/step - loss: 14.0335
14.033499717712402
```

We can see less error in the clustering data which doesn't happen generally. This is because the cluster 0 dataset have many outliers and k-medoid clustering is robust to the same.

Future Innovations:

c1)Parallel LSTM:

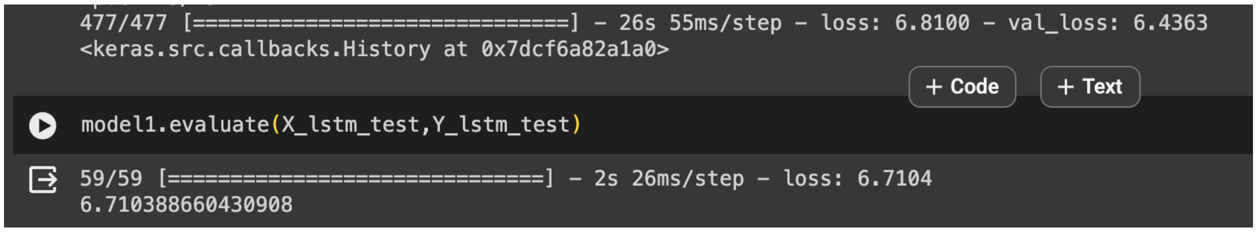
Since LSTM was found to be quite favourable, we tried designing the model in parallel with each other. We can clearly understand the same from the block diagram given below:



We got a validation loss of only 6.43% and a MAPE of 6.81% which is less than the actual LSTM MAPE (around 7.2%). The error has decreased by 0.4%. The circuit although becomes complex now, increasing the probability of dropout. However, with advance python one can manage this dropout for large datasets hence a parallel LSTM can be used.

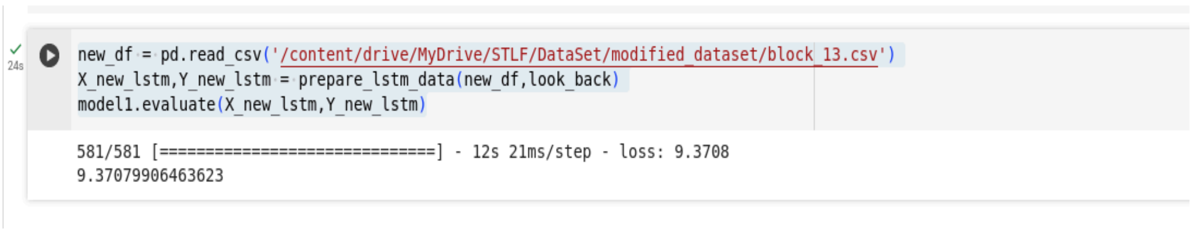
We will further try integrating more of the LSTM architectures in parallel and look for optimal forecasting networks which gives minimum error for both of our training and testing datasets.

Parallel LSTM Results (without clustering):



```
477/477 [==============================] - 26s 55ms/step - loss: 6.8100 - val_loss: 6.4363
<keras.src.callbacks.History at 0x7dcf6a82a1a0>

model1.evaluate(X_lstm_test,Y_lstm_test)

59/59 [==============================] - 2s 26ms/step - loss: 6.7104
6.710388660430908
```

Parallel LSTM Results (with clustering- Cluster 0):



```
new_df = pd.read_csv('/content/drive/MyDrive/STLF/DataSet/modified_dataset/block_13.csv')
X_new_lstm,Y_new_lstm = prepare_lstm_data(new_df,look_back)
model1.evaluate(X_new_lstm,Y_new_lstm)

581/581 [==============================] - 12s 21ms/step - loss: 9.3708
9.37079906463623
```

c2) Non-Intrusive Load Monitoring (NILM):

NILM is a method where we analyze the power consumption of individual devices/ distinct type of devices based on the house consumption overall. NILM can be implemented by either CNN or LSTM architectures. In case of NILM, we generate a window sequence of defined number of samples that transverse over the power consumption data of the houses. We can generate the output in 2 ways:

1) Sequence-to-sequence:
   Here we take all the data point under the window and parallely produce equivalently same size output.
2) Sequence-to-Point:
   Here we take the median of the data points in the window and produce a single output. The complexity is easier here with a faster output. However, you'll have to compromise with the accuracy.

One can further modify NILM and also predict things like appliance status (on/off), Appliance Energy consumption with higher precision and appliance behaviour (normal/ abnormal). These models create a feature by themselves and based on the pattern of the appliances. The predictions given by the dense network is further de-normalized for accurate results. NILM can also be implemented using hardware likes Arduino and Raspberry Pi etc.

Conclusion:

- We can see that generally the error after applying clustering on the data set has increased. However, the opposite was observed for cluster-0 data due to the presence of many outliers therein and in such cases clustering is preferable due to robust to high value noise nature.
- In case of ML models, we saw the least error in the random forest method due to its optimal feature selection strategies thereby giving accurate results of the rest of the ML Models.
- Amongst the individual architectures, we saw least MAPE in the LSTM architecture and thus tried improvising the same by connecting it in series as well as in parallel with other neural networks.
- For future innovations, we have decided to work more on the different combinations of the LSTM models and Non-Intrusive Load monitoring (NILM) implementation via hardware for power consumption on appliance level.

Bibliography:

1] https://wonmin-byeon.github.io/files/2-15-nips.pdf
2]https://www.simplilearn.com/tutorials/artificial-intelligence-tutorial/lstm#:~:text=LSTMs%20Long%20Short%2DTerm%20Memory,series%2C%20text%2C%20and%20speech.
3]https://www.geeksforgeeks.org/convolutional-neural-network-cnn-in-machine-learning/
4]https://www.databricks.com/glossary/machine-learning-models