Introduction:

In neural networks, different kinds of networks have been developed. Previously, we studied Convolutional neural networks (CNN) which are an excellent choice when trying to accomplish any task dealing with images. However, it was noticed that CNN's underperformed when it comes to time series forecasts. Hence, it was necessary to study a different kind of neural network known as Recurrent neural network (RNN). These networks excel at forecasting based on time series data. In this project two neural networks will be developed: a 1-input predictor and a 2-input predictor in order to determine the energy demand. In addition both predictions will be tested on a 3 hour and 6 hour horizon in order to determine the best model. The values below represent information regarding the test set.

```
Highest value of test set:  [5224.]
Smallest value of test set:  [1979.]
Full range of test set:  [3245.]
```

Procedures:

In this project the data will be obtained from an excel zip folder. This data will then be organized depending on whether the model is 1-input predictor or a 2-input predictor. Once this has been accomplished the data will be split into train, val and test samples. Following this step, in order to obtain the best possible model the data will be normalized. After the data has been normalized it's time to create the datasets that will be used to train, validate and test the model. To create the model we must use an LSTM of at least 2-processing elements. Once the model(s) have been created then the training will start. In this kind of project the believed sweet spot for a good model is in the range of 750 to 1000 epochs. Hence, all models will undergo training under this range. Directly after the models have been trained they will go to an extensive process of evaluation in order to determine the effectiveness of the models, which will be covered in the results section of this report.

Results:

In this section the plots and final values pertaining to each model will be presented. In addition the structure of the network alongside its complete description (i.e number of layers, activation function, processing elements, etc.) will be encompassed in this section. The following information represents the structure of this section: *a) Sequence_length (in hours), b) compile method and the fit method code, c) model summary, d) Training and Validation loss plots, e) Training and Validation mae plots, f) Final Training and Validation (loss & mae), g) MAE for Test Set, h) Plot of values predicted, i) plots of targets, j) Overlay of predicted and targets plot from (6000 to 6500), g2) Unnormalized g, h2) Unnormalized h, i2) Unnormalized i, j2) Unnormalized j, and lastly k) PMAE*. In addition, a brief description will be provided regarding the changes that were made between from one model to the next at the end of each type of model

II.1 1-Input predictor (3 hour Horizon)

a) Sequence length = 12 hours
b) Compile and Fit method

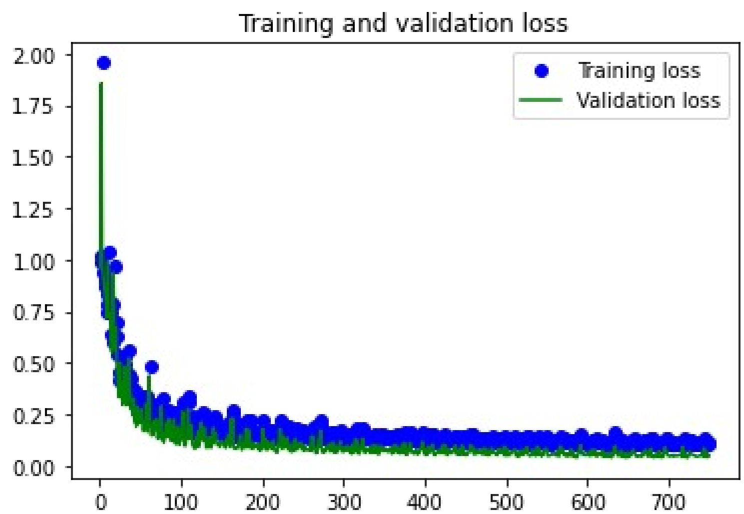
```
model.compile(optimizer=keras.optimizers.RMSprop(learning_ra
te=5e-3), loss="mse", metrics=["mae"])
history = model.fit(train_dataset,
epochs=750,
validation_data=val_dataset,
callbacks=callbacks)
```

c) Model.summary()

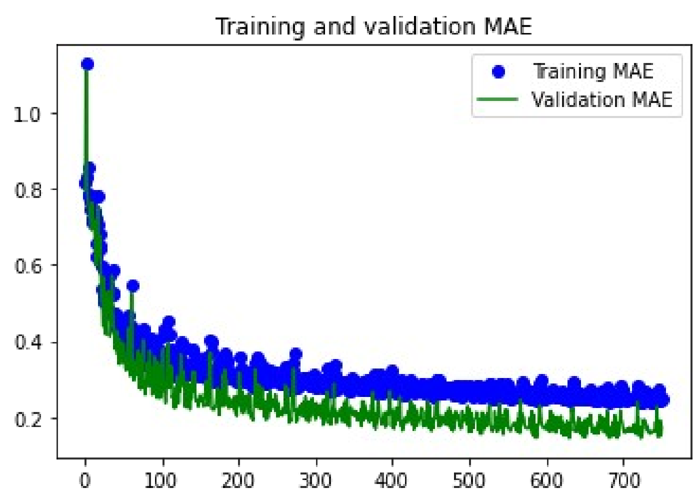
```
Model: "model"
______________________________________________________________
__
 Layer (type)                Output Shape              Param #

==============================================================
==
 input_1 (InputLayer)        [(None, 12, 1)]           0



 gru (GRU)                   (None, 12, 50)            7950



 gru_1 (GRU)                 (None, 12, 100)           45600



 lstm (LSTM)                 (None, 12, 32)            17024



 lstm_1 (LSTM)               (None, 16)                3136



 dropout (Dropout)           (None, 16)                0
```

```
 dense (Dense)                (None, 1)                 17

=================================================================
Total params: 73,727
Trainable params: 73,727
Non-trainable params: 0
```

d) Training and Validation Loss



e) Training and Validation MAE



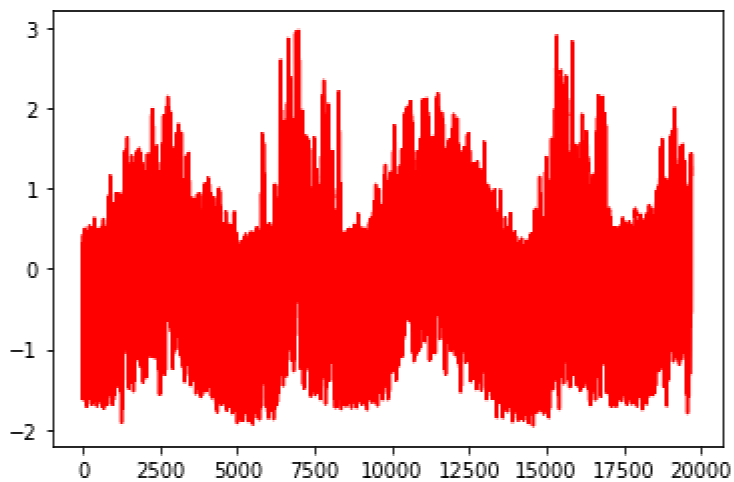
f) Final Training and Validation (loss & mae)

```
Final Training loss:  0.10616865009069443
Final Training MAE:  0.24745123088359833
Final Validation loss:  0.05192284286022186
Final Validation MAE:  0.17455925047397614
```

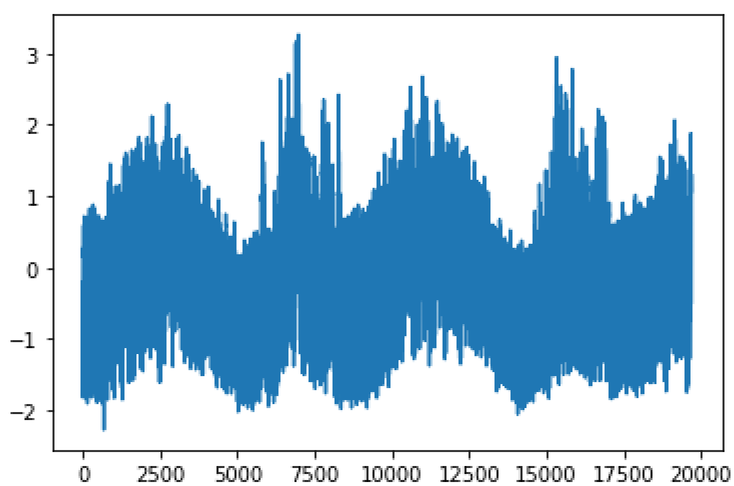
g) MAE of the Test Set

```
1/1 [==============================] - 2s 2s/step - loss:
0.0390 - mae: 0.1474 [0.03896249085664749, 0.14743258059024 81]
```

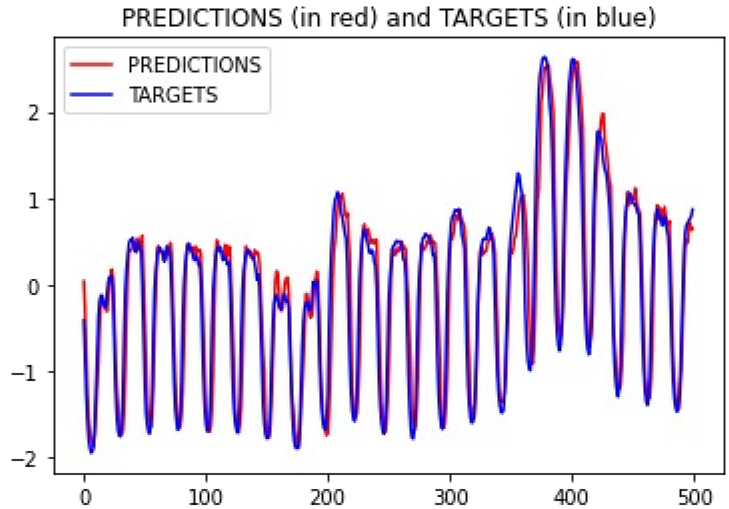
h) Time series plot of all the values predicted by the model on the test set



i) Time series plot of the corresponding targets

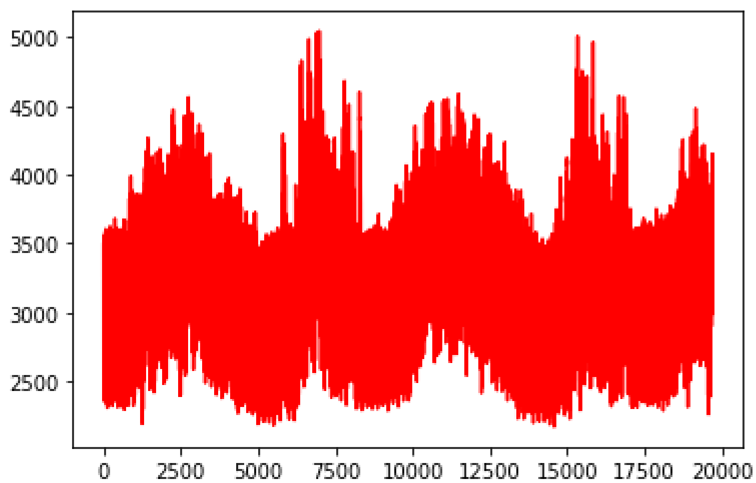


j) Overlay plot predictions and targets

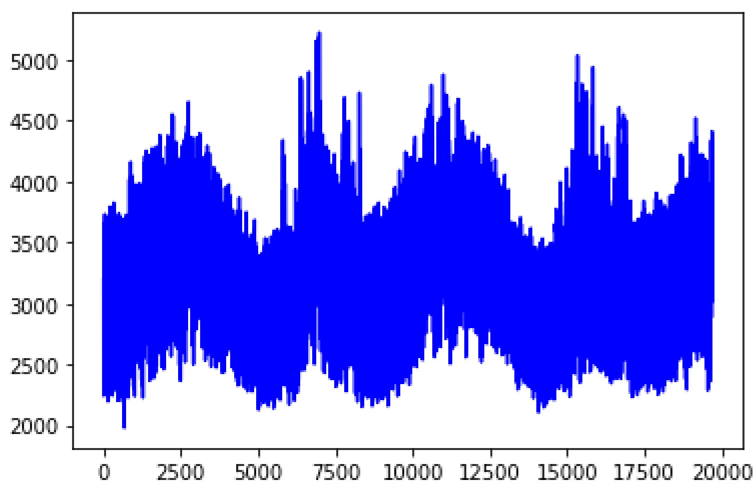


g2) EFFECTIVE real-scale MAE: 151.03

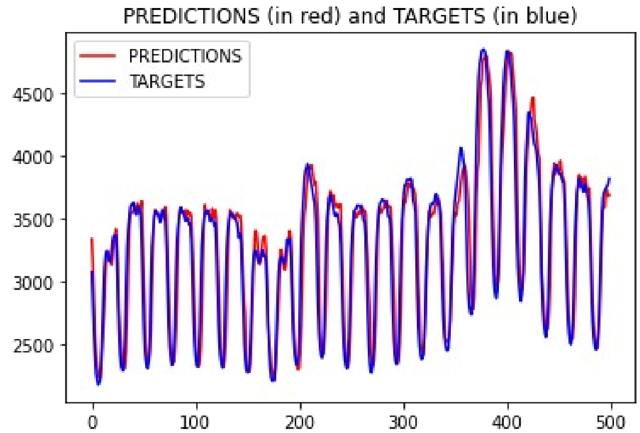
h2) Unnormalized time series plot of all the values predicted by the model on the test set



i2) Unnormalized Time series plot of the corresponding targets



j2) Unnormalized Overlay plot predictions and targets



k) PMAE: 4.654325158277641

II.2 1-Input predictor (6 hour Horizon)

a) Sequence length = 24 hours
b) Compile and Fit method

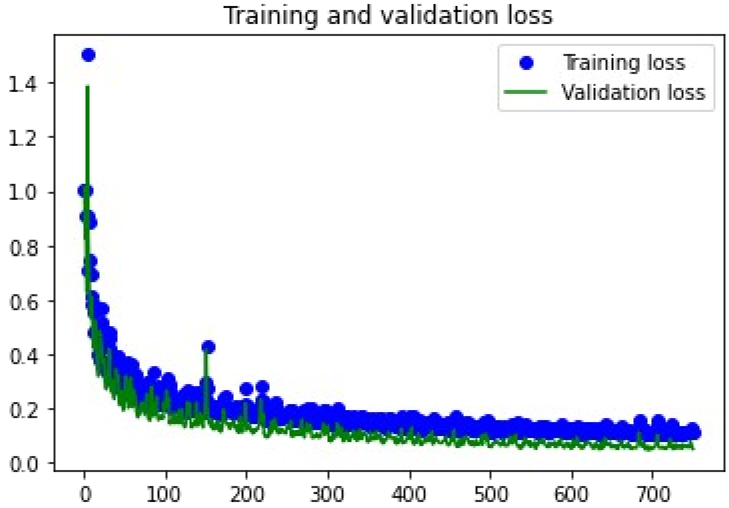
```
model.compile(optimizer=keras.optimizers.RMSprop(learning_ra
te=5e-3), loss="mse", metrics=["mae"])
history = model.fit(train_dataset,
epochs=750,
validation_data=val_dataset,
callbacks=callbacks)
model = keras.models.load_model("predictor.keras")
```

c) Model.summary()

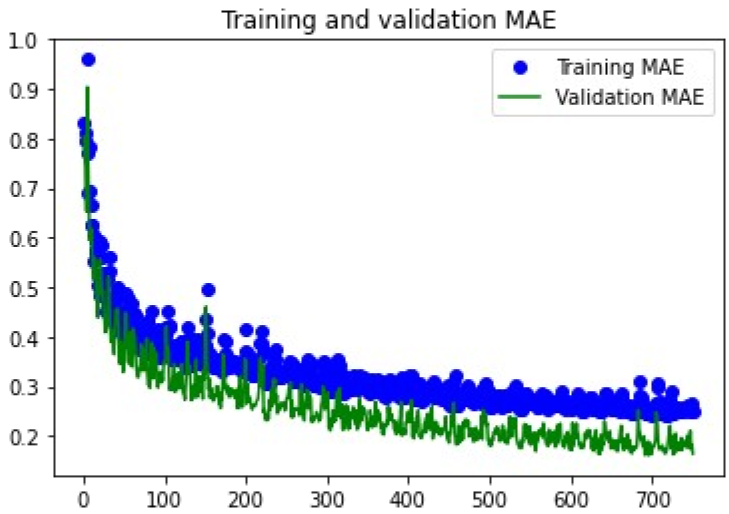
```
Model: "model"

_____________________________________________________________
__
 Layer (type)                Output Shape              Param #

=============================================================
==
 input_1 (InputLayer)        [(None, 24, 1)]           0



 gru (GRU)                   (None, 24, 50)            7950
```

```
 gru_1 (GRU)                 (None, 24, 100)           45600

 lstm (LSTM)                 (None, 24, 32)            17024

 lstm_1 (LSTM)               (None, 16)                3136

 dropout (Dropout)           (None, 16)                0

 dense (Dense)               (None, 1)                 17

================================================================
==
Total params: 73,727
Trainable params: 73,727
Non-trainable params: 0
```

d) Training and Validation Loss



e) Training and Validation MAE



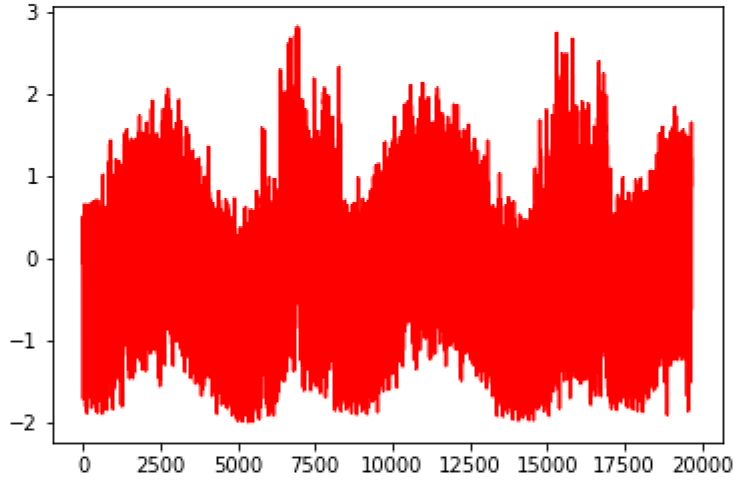
f) Final Training and Validation (loss & mae)

```
Final Training loss:  0.11105220764875412
Final Training MAE:  0.2512136399745941
Final Validation loss:  0.05078193545341492
Final Validation MAE:  0.16434980928897858
```

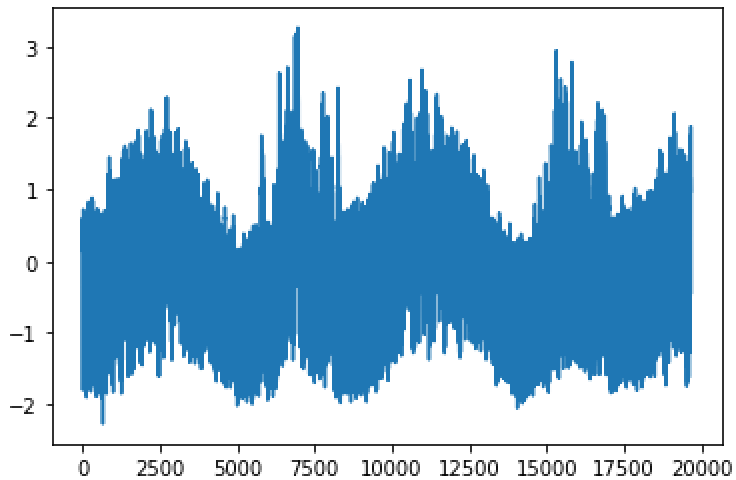
g) MAE of the Test Set

```
1/1 [==============================] - 2s 2s/step - loss:
0.0529 - mae: 0.1718 [0.05294589698314667, 0.17182381451129913]
```

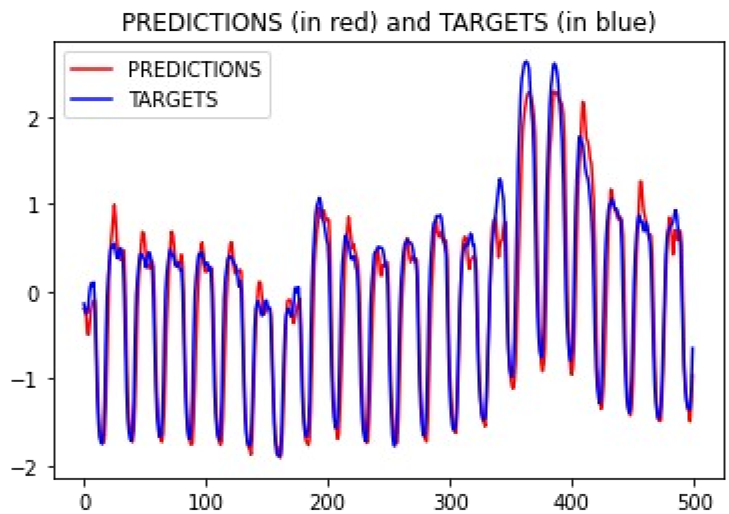
h) Time series plot of all the values predicted by the model on the test set



i) Time series plot of the corresponding targets

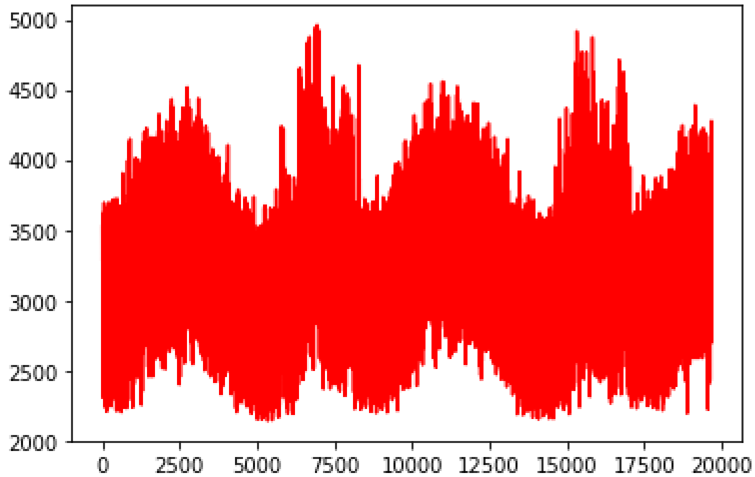


j) Overlay plot predictions and targets

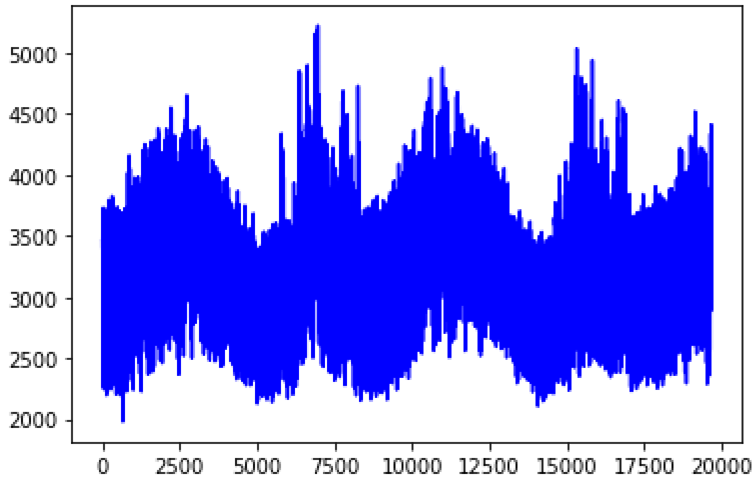


g2) EFFECTIVE real-scale MAE: 169.55

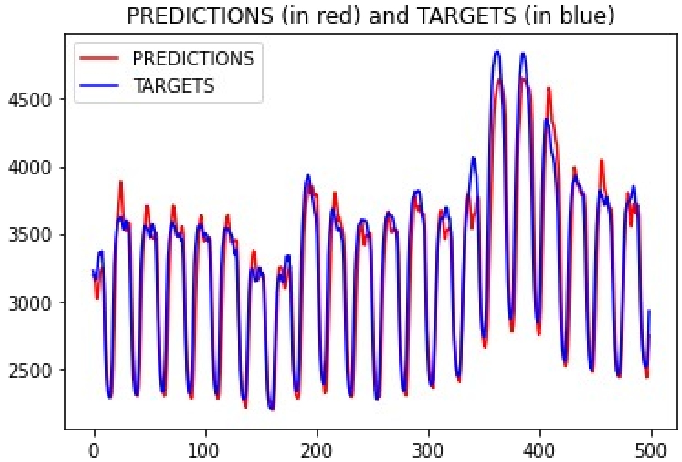
h2) Unnormalized time series plot of all the values predicted by the model on the test set



i2) Unnormalized Time series plot of the corresponding targets



j2) Unnormalized Overlay plot predictions and targets



k) PMAE = 5.224982928952527

Discussion/Observations of 1-Input Predictor (3 hours and 6 hours) :

The results of a 1-input predictor for a horizon of 3 hours and a horizon of 6 hours were very interesting. For starters, it was noticed that both of these models did not overfit even when it was running at 750 - 1000 epoch range. This potentially suggests that this model can potentially be always improved by just training for a longer period of time. Naturally, training for a longer period of time is not always the best solution when it comes to improving a model but the fact that the model can be trained for long periods of time without overfitting is definitely an important aspect. Advancing towards the difference found when the horizon was changed from 3 hours to 6 hours, it was noticed that the 3 hour model performed slightly better. This can be observed not only in the overlay plots, but also in the fact that the PMAE for the 3 hours predictor is *4.654325158277641*, while the 6 hours predictor is *5.224982928952527*. This suggests that when using a 1-input predictor to forecast eload, it is best to predict on a 3-hour horizon rather than the 6 hours.

<u>III.1 2-Input predictor (3 hour Horizon)</u>

a) Sequence Length = 12 hours
b) Compile and Fit method

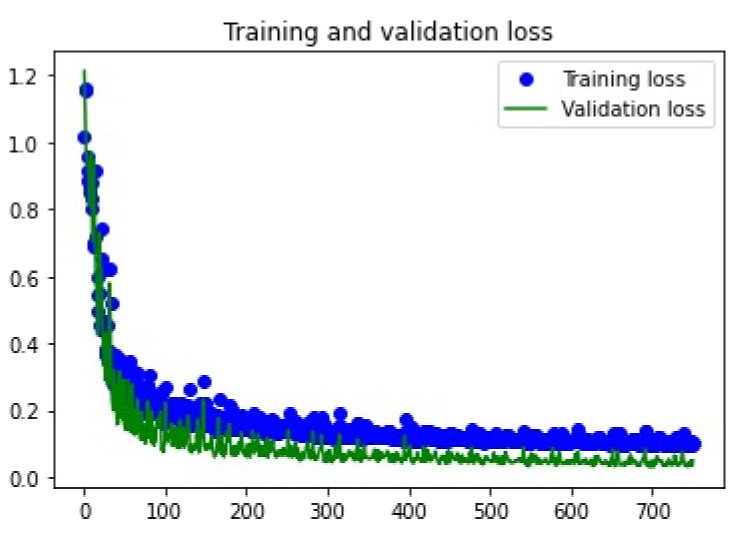
```
model.compile(optimizer=keras.optimizers.RMSprop(learning_ra
te=5e-3), loss="mse", metrics=["mae"])
history = model.fit(train_dataset,
epochs=750,
validation_data=val_dataset,
callbacks=callbacks)
```

c) Model.summary()

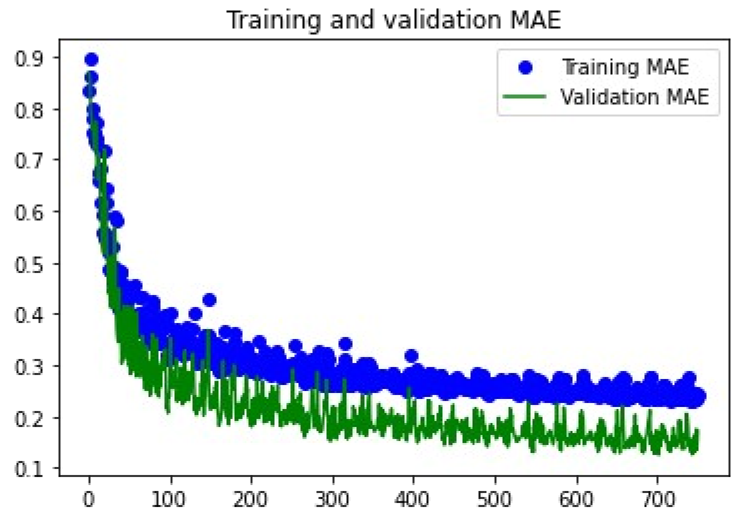
```
Model: "model_1"
_______________________________________________________________
__
 Layer (type)                Output Shape              Param #

===============================================================
==
 input_2 (InputLayer)        [(None, 12, 2)]           0



 gru_2 (GRU)                 (None, 12, 50)            8100
```

```
 gru_3 (GRU)                 (None, 12, 100)           45600


 lstm_2 (LSTM)               (None, 12, 32)            17024


 lstm_3 (LSTM)               (None, 16)                3136


 dropout_1 (Dropout)         (None, 16)                0


 dense_1 (Dense)             (None, 1)                 17


===============================================================
==
Total params: 73,877
Trainable params: 73,877
Non-trainable params: 0
```

d) Training and Validation Loss



e) Training and Validation MAE



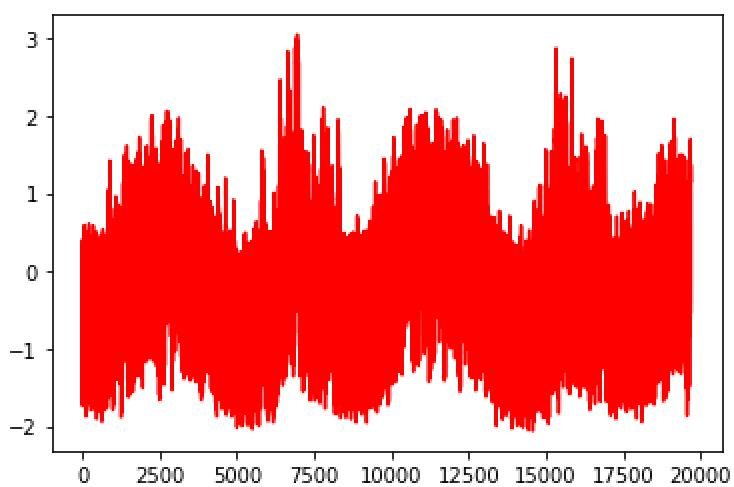
f) Final Training and Validation (loss & mae)

```
Final Training loss:  0.10276700556278229
Final Training MAE:  0.24040523171424866
Final Validation loss:  0.04883880168199539
Final Validation MAE:  0.17187492549419403
```

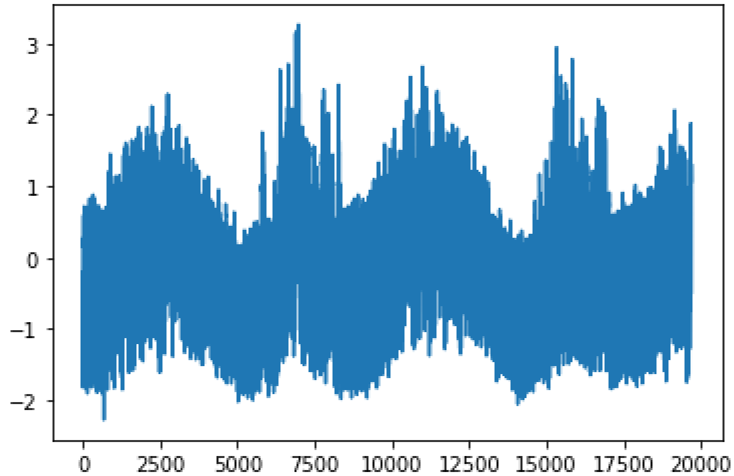
g) MAE of the Test Set

```
1/1 [==============================] - 2s 2s/step - loss:
0.0348 - mae: 0.1406
```

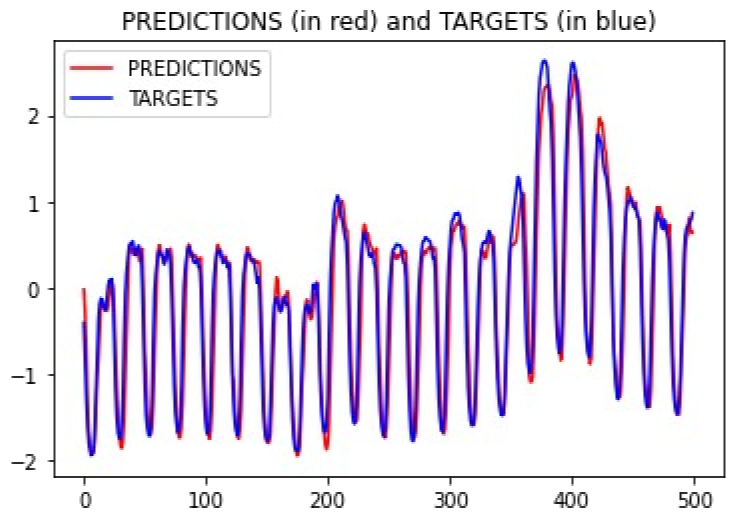
h) Time series plot of all the values predicted by the model on the test set



i) Time series plot of the corresponding targets

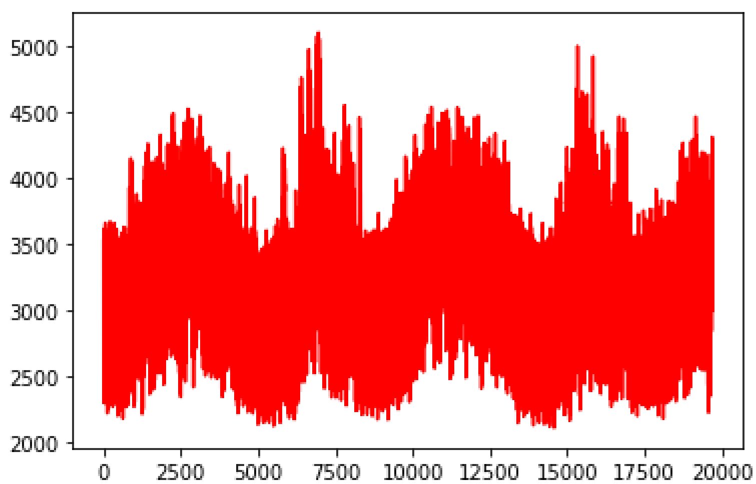


j) Overlay plot predictions and targets

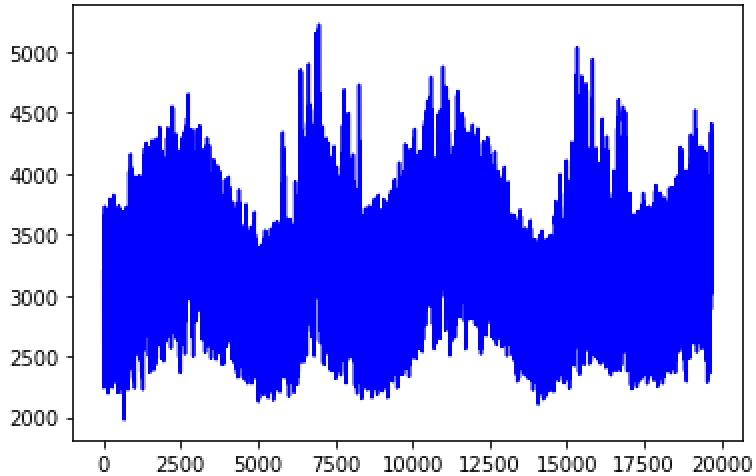


g2) EFFECTIVE real-scale MAE: 144.76

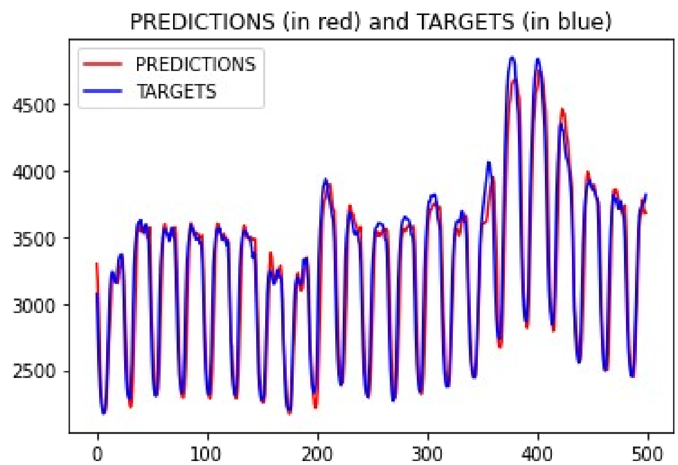
h2) Unnormalized time series plot of all the values predicted by the model on the test set



i2) Unnormalized Time series plot of the corresponding targets



j2) Unnormalized Overlay plot predictions and targets



k) PMAE = 4.461068025316224

<u>III.2 2-Input predictor (6 hour Horizon)</u>

a) Sequence Length = 24 hours

b) Compile and Fit method

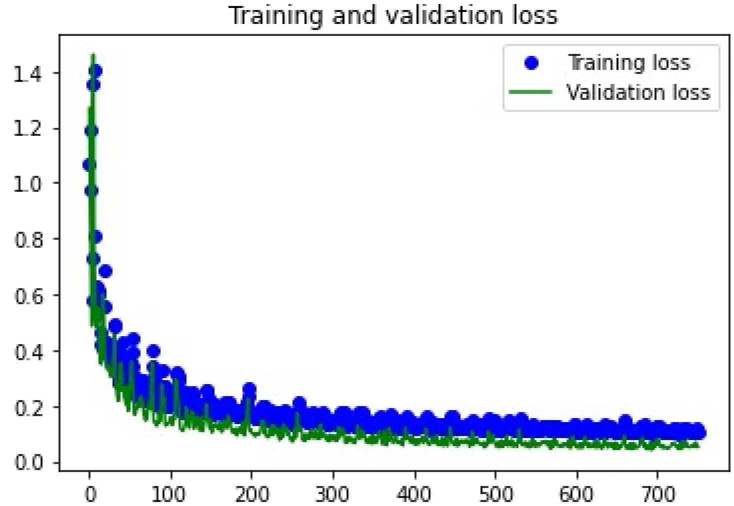
```
model.compile(optimizer=keras.optimizers.RMSprop(learning_ra
te=5e-3), loss="mse", metrics=["mae"])
history = model.fit(train_dataset,
epochs=750,
validation_data=val_dataset,
callbacks=callbacks)
```

c) Model.summary()

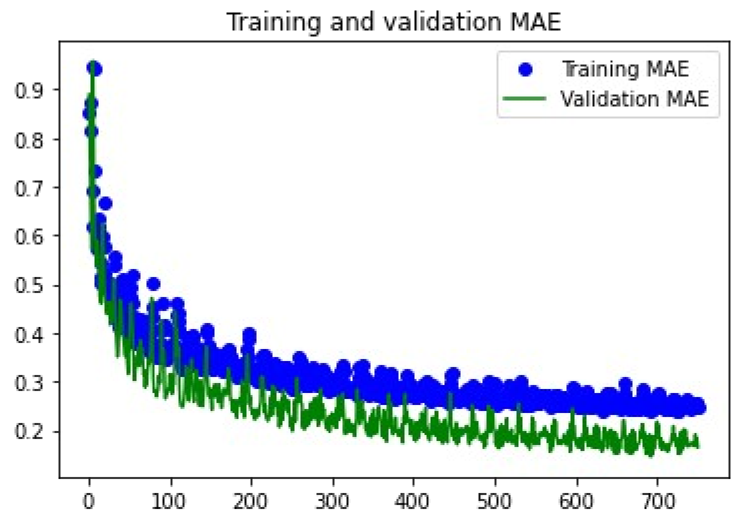
```
Model: "model"
```

```
 Layer (type)                Output Shape              Param #   
=================================================================
 input_1 (InputLayer)        [(None, 24, 2)]           0         
                                                                 
 gru (GRU)                   (None, 24, 50)            8100      
                                                                 
 gru_1 (GRU)                 (None, 24, 100)           45600     
                                                                 
 lstm (LSTM)                 (None, 24, 32)            17024     
                                                                 
 lstm_1 (LSTM)               (None, 16)                3136      
                                                                 
 dropout (Dropout)           (None, 16)                0         
                                                                 
 dense (Dense)               (None, 1)                 17        
                                                                 
=================================================================
Total params: 73,877
Trainable params: 73,877
Non-trainable params: 0
```

d) Training and Validation Loss



e) Training and Validation MAE



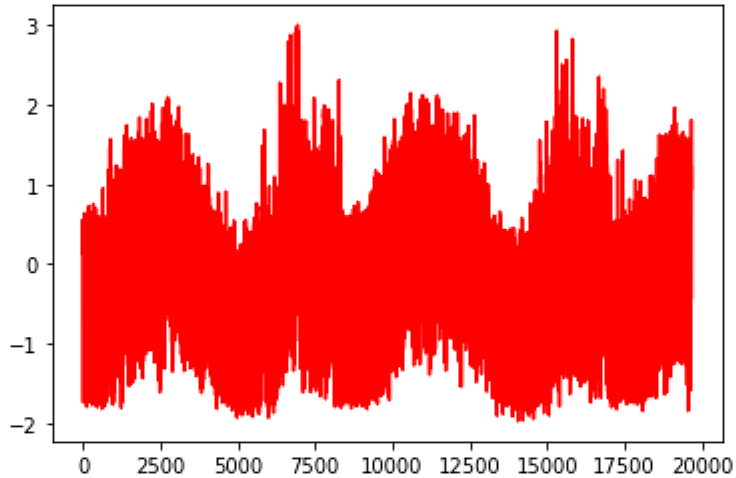
f) Final Training and Validation (loss & mae)

```
Final Training loss:  0.10740343481302261
Final Training MAE:  0.24863161146640778
Final Validation loss:  0.05227213725447655
Final Validation MAE:  0.16494640707969666
```

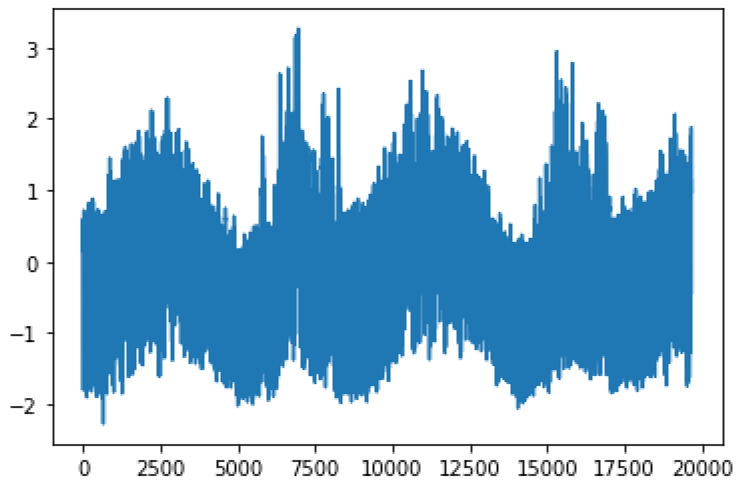
g) MAE of the Test Set

```
1/1 [==============================] - 2s 2s/step - loss:
0.0448 - mae: 0.1560
```

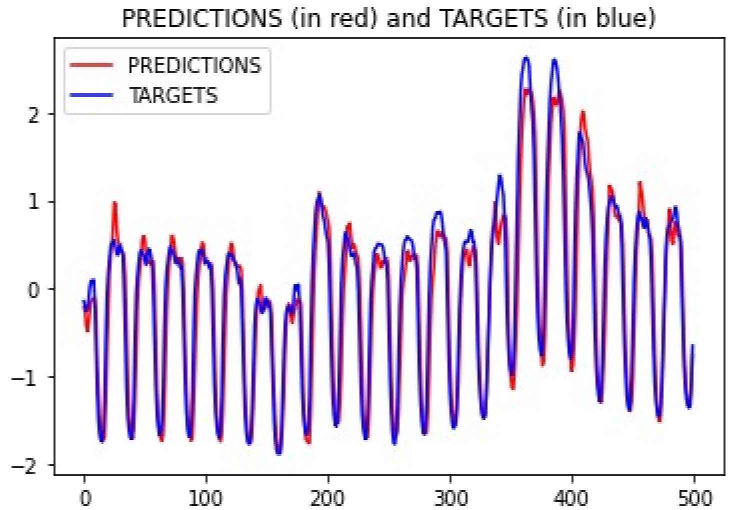
h) Time series plot of all the values predicted by the model on the test set



i) Time series plot of the corresponding targets

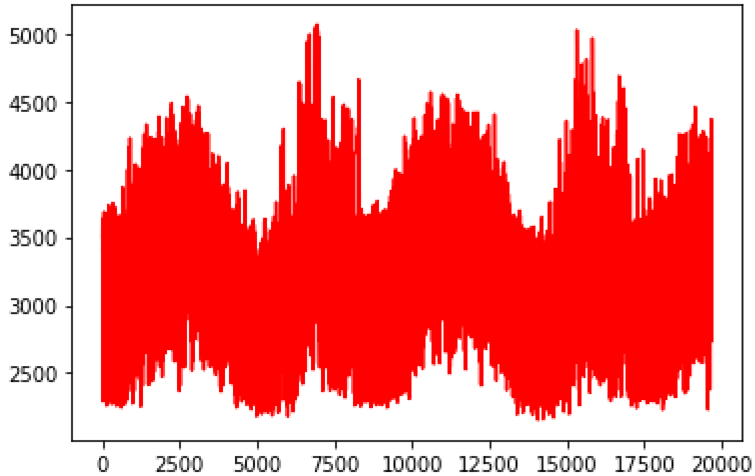


j) Overlay plot predictions and targets

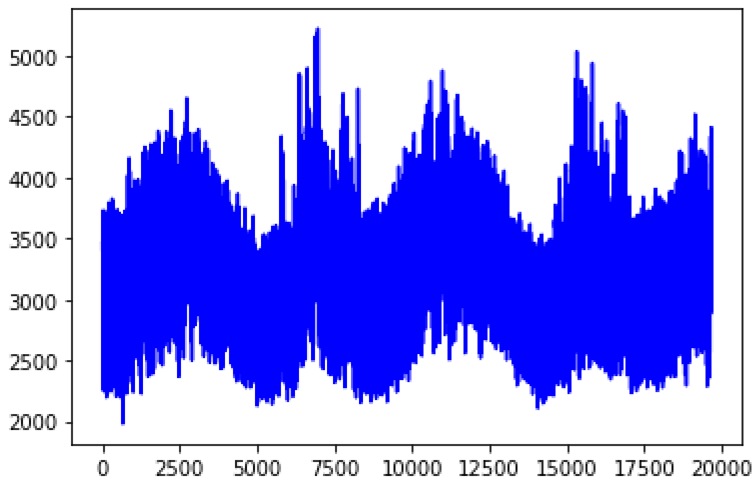


g2) EFFECTIVE real-scale MAE: 168.34

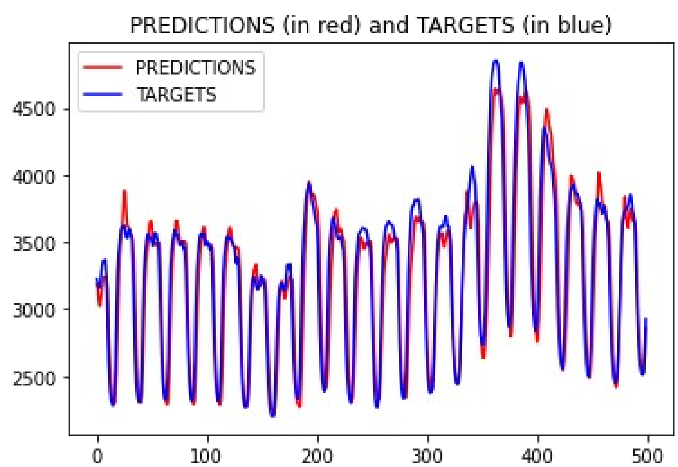
h2) Unnormalized time series plot of all the values predicted by the model on the test set



i2) Unnormalized Time series plot of the corresponding targets



j2) Unnormalized Overlay plot predictions and targets



k) PMAE = 5.187757013295373

Discussion/Observations of 2-Input Predictor (3 hours and 6 hours) :

The results of a 2-input predictor for a horizon of 3 hours and a horizon of 6 hours were very interesting. For starters, it was noticed that both of these models did not overfit even when it was running at 750 - 1000 epoch range. This potentially suggests that this model can potentially be always improved by just training for a longer period of time. Naturally, training for a longer period of time is not always the best solution when it comes to improving a model but the fact that the model can be trained for long periods of time without overfitting is definitely an important aspect. Advancing towards the difference found when the horizon was changed from 3 hours to 6 hours, it was noticed that the 3 hours model performed slightly better. This can be observed not only in the overlay plots, but also in the fact that the PMAE for the 3 hours predictor was *4.461068025316224*, while for the 6 hours predictor is *5.187757013295373*. Similar to the 1-input predictor this suggests that it is better to predict on a 3 hours horizon.

Conclusion:

In conclusion, what was learned in this project is the power of Recurrent Neural Networks and how versatile they can be in time series data. This project had many interesting outcomes. For example, one of the things that was expected was that RNN's would do a good job of predicting this time series. In addition, the network actually performed better when the 2-input predictor was used as opposed to the 1-input predictor. Although more testing would have to be accomplished it can be surmised that sometimes inputting unnecessary data can actually benefit the performance of the neural network. There are many adjustments that can be made to the models, however one that can have an immediate impact is introducing bidirectional LSTM. In addition, when talking about the model's performance, the 2-input predictor outperformed the 1-input predictor on both the 3 hours and 6 hours horizon. In addition, the expected better performance for the 3 hours horizon was met by both of the predictors. I believe that the reason that the 2-input predictor performed better is because in neural networks it has been established that most of the time it is better to provide additional data, even if the data provided is not correlated with the prediction that is being made.

Appendix

```
import numpy as np
import pandas as pd
import os
import shutil
import matplotlib.pyplot as plt
#from common.utils import load_data, extract_data, download_file
%matplotlib inline
import pandas as pd
import os
import numpy as np
from matplotlib import pyplot as plt
from tensorflow import keras
from tensorflow import keras
from tensorflow.keras import layers
import copy

!wget https://mlftsfwp.blob.core.windows.net/mlftsfwp/GEFCom2014.zip

!unzip GEFCom2014.zip

!mv 'GEFCom2014 Data'/GEFCom2014-E_V2.zip ./
!unzip GEFCom2014-E_V2.zip

GEFDF = pd.read_excel('GEFCom2014-E.xlsx', skiprows=range(1, 17545),
dtype = {'A':np.int32,})

GEFDF.to_csv('GEF14.csv', encoding='utf-8', index=False, header=True,
columns=['Hour','load','T'])
with open('GEF14.csv') as f:
  lines = f.readlines()
  last = len(lines) - 1
  lines[last] = lines[last].replace('\r','').replace('\n','')
with open('GEF14.csv', 'w') as wr:
  wr.writelines(lines)

fname = os.path.join("GEF14.csv")
with open(fname) as f:
  data = f.read()
lines = data.split("\n")
header = lines[0].split(",")
lines = lines[1:]
print(header)
print(len(lines))
```

```
eload = np.zeros((len(lines),))
tempf = np.zeros((len(lines),))
raw_data = np.zeros((len(lines), len(header)-1)) #chgd )-1 to )-2 to
also
# remove the HOUR column, in addition to the DATE column
print(len(lines))
for m in range(78888):
  thisline = lines[m]
  values = [float(x) for x in thisline.split(",")[1:]]
  eload[m] = values[0] #Captures JUST E LOAD
  tempf[m] = values[1] #Captures JUST TEMPF
  #raw_data[m] = values[0] #Like this, raw_data Captures JUST E LOAD
  raw_data[m, :] = values[:] # Like this, raw_data CAPTURES BOTH

plt.plot(range(len(eload)), eload)

plt.plot(range(len(tempf)), tempf)

num_train_samples = int(0.5 * len(raw_data))
num_val_samples = int(0.25 * len(raw_data))
num_test_samples = len(raw_data) - num_train_samples - num_val_samples
print("num_train_samples:", num_train_samples)
print("num_val_samples:", num_val_samples)
print("num_test_samples:", num_test_samples)

plt.plot(range(240),eload[:240])

plt.plot(range(240),tempf[:240])

Copy_raw_data = copy.copy(raw_data)
mean = Copy_raw_data[:num_train_samples].mean(axis=0)
Copy_raw_data -= mean
std = Copy_raw_data[:num_train_samples].std(axis=0)
Copy_raw_data /= std

N_eload = Copy_raw_data[:,0]
print("Highest value of test set: ",raw_data[num_train_samples +
num_val_samples:,0].max(axis=0))
print("Smallest value of test set: ",raw_data[num_train_samples +
num_val_samples:,0].min(axis=0))
print("Full range of test set: ",raw_data[num_train_samples +
num_val_samples:,0].max(axis=0) - raw_data[num_train_samples +
num_val_samples:,0].min(axis=0))

horizon = 6 # Num. of hours ahead for forecast (3 or 6)
```

```
sampling_rate = 1
sequence_length = 24 # For horizon = 3 (sequence length < 15) & horizon
= 6 (sequence length < 36)
delay = sampling_rate * (sequence_length + horizon - 1)
batch_size = 128
train_dataset = keras.utils.timeseries_dataset_from_array(
Copy_raw_data[:-delay],
targets=Copy_raw_data[delay:,0], # This would used "Normalized Targets"
# targets=eload[delay:], # This would used "Not-normalized eload
targets"
sampling_rate=sampling_rate,
sequence_length=sequence_length,
shuffle=True, #changed to false JUST FOR VERIF
batch_size= num_train_samples,
start_index=0,
end_index=num_train_samples)
val_dataset = keras.utils.timeseries_dataset_from_array(
Copy_raw_data[:-delay], # changed from raw_data to just eload not really
targets=Copy_raw_data[delay:,0], # This would used "Normalized Targets"
# targets=eload[delay:], # This would used "Not-normalized eload
targets"
sampling_rate=sampling_rate,
sequence_length=sequence_length,
shuffle=True,
batch_size=num_val_samples,
start_index=num_train_samples,
end_index=num_train_samples + num_val_samples)
test_dataset = keras.utils.timeseries_dataset_from_array(
Copy_raw_data[:-delay], # changed from raw_data to just eload
targets=Copy_raw_data[delay:,0], # This would used "Normalized Targets"
# targets=eload[delay:], # This would used "Not-normalized eload
targets"
sampling_rate=sampling_rate,
sequence_length=sequence_length,
shuffle=False,
batch_size=num_test_samples,
start_index=num_train_samples + num_val_samples)

for samples, targets in train_dataset:
  print("samples shape:", samples.shape)
  print("targets shape:", targets.shape)
  break

inputs = keras.Input(shape=(sequence_length, Copy_raw_data.shape[-1]))
x = layers.GRU(50, recurrent_dropout=0.25, return_sequences=True)
```

```
(inputs)
x = layers.GRU(100, recurrent_dropout=0.5, return_sequences=True)(x)
x = layers.LSTM(32, recurrent_dropout=0.15, return_sequences=True)(x)
x = layers.LSTM(16, recurrent_dropout=0.25) (x)
x = layers.Dropout (.5) (x)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)
callbacks = [
keras.callbacks.ModelCheckpoint("predictor.keras",
save_best_only=True)
]

model.compile(optimizer=keras.optimizers.RMSprop(learning_rate=5e-3),
loss="mse", metrics=["mae"])
history = model.fit(train_dataset,
epochs=750,
validation_data=val_dataset,
callbacks=callbacks)
model = keras.models.load_model("predictor.keras")
print(f"Test MAE: {model.evaluate(test_dataset)[1]:.2f}")

model.summary()

acc = history.history["mae"]
val_acc = history.history["val_mae"]
loss = history.history["loss"]
val_loss = history.history["val_loss"]
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, "bo", label="Training MAE")
plt.plot(epochs, val_acc, "g", label="Validation MAE")
plt.title("Training and validation MAE")
plt.legend()
plt.figure()
plt.plot(epochs, loss, "bo", label="Training loss")
plt.plot(epochs, val_loss, "g", label="Validation loss")
plt.title("Training and validation loss")
plt.legend()
plt.show()
print("Final Training loss: ",history.history['loss'][-1],"\nFinal
Training MAE: ", history.history['mae'][-1])
print("Final Validation loss: ",history.history['val_loss'][-1],"\nFinal
Validation MAE: ", history.history['val_mae'][-1])

model = keras.models.load_model("predictor.keras") # bringing the "best
model"
```

```
predictions = model.predict(test_dataset)
model.evaluate(test_dataset)
lenpred = len(predictions)
plt.plot(range(lenpred), predictions, 'r')

# De normalizing the data
EFFpredictions1 = np.asarray(predictions * std[0])
EFFpredictions2 = EFFpredictions1.flatten()

MEANV = (np.ones(lenpred,)) * mean[0]

EFFpredictions = EFFpredictions2 + MEANV
print(EFFpredictions.shape)

EFFmidtargets1 = np.asarray(midtargets * std[0])
EFFmidtargets2 = EFFmidtargets1.flatten()

MEANV = (np.ones(lenpred,)) * mean[0]

EFFmidtargets = EFFmidtargets2 + MEANV
print(EFFpredictions.shape)

plt.plot(range(lenpred), EFFpredictions, 'r')
plt.show

plt.plot(range(lenpred), EFFmidtargets, 'b')
plt.show

plt.plot(EFFpredictions[6000:6500], "r", label="PREDICTIONS")
plt.plot(EFFmidtargets[6000:6500], "b", label="TARGETS")
plt.title("PREDICTIONS (in red) and TARGETS (in blue)")
plt.legend()
plt.show()

EFFECTIVE_MAE = np.mean(np.abs(EFFpredictions - EFFmidtargets))
print(f'EFFECTIVE real-scale MAE: {EFFECTIVE_MAE:.2f}')
PMAE = (EFFECTIVE_MAE /  (EFFmidtargets.max() - EFFmidtargets.min())) *
100
print(PMAE)
```