25401119601-no2

July 7, 2023

Nama : Andrew

NIM : 2540119601

Kelas : LA09

Mata Kuliah : Deep Learning

Jurusan : Data Science

Link Video : https://www.youtube.com/watch?v=HulYcu6RbYs

Import Dataset

[ ]:
```python
# libary
import pandas as pd
```

Data diambil melalui drive

[ ]:
```python
# connect to drive to easily get data
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Parse colomn date sehingga sesuai dengan format date dan juga membuat kolom date menjadi index, karena time series data

[ ]:
```python
# specify to make date as a index
df1 = pd.read_csv('/content/drive/MyDrive/UAS_deepLearning_data/Dataset C/GOOGL.
 ↪csv',parse_dates=["Date"],index_col=["Date"])
df2 = pd.read_csv('/content/drive/MyDrive/UAS_deepLearning_data/Dataset C/INTC.
 ↪csv',parse_dates=["Date"],index_col=["Date"])
```

[ ]:
```python
print(df1)
print(df2)
```

```
                 Open       High        Low      Close  Adj Close  \
Date
2004-08-19  50.050049  52.082081  48.028027  50.220219  50.220219
2004-08-20  50.555557  54.594593  50.300301  54.209209  54.209209
2004-08-23  55.430431  56.796795  54.579578  54.754753  54.754753
2004-08-24  55.675674  55.855854  51.836838  52.487488  52.487488
```

```
2004-08-25    52.532532    54.054054    51.991993    53.053055    53.053055
…                   …           …           …           …           …
2020-03-26  1114.719971  1171.479980  1092.030029  1162.920044  1162.920044
2020-03-27  1127.469971  1151.050049  1104.000000  1110.260010  1110.260010
2020-03-30  1132.640015  1151.000000  1098.489990  1146.310059  1146.310059
2020-03-31  1148.729980  1173.400024  1136.719971  1161.949951  1161.949951
2020-04-01  1124.000000  1129.420044  1093.489990  1102.099976  1102.099976

              Volume
Date
2004-08-19  44659000
2004-08-20  22834300
2004-08-23  18256100
2004-08-24  15247300
2004-08-25   9188600
…                 …
2020-03-26   3828100
2020-03-27   3139700
2020-03-30   2936800
2020-03-31   3261400
2020-04-01   2597100

[3932 rows x 6 columns]
                 Open       High        Low      Close  Adj Close     Volume
Date
1980-03-17   0.325521   0.330729   0.325521   0.325521   0.204750   10924800
1980-03-18   0.325521   0.328125   0.322917   0.322917   0.203112   17068800
1980-03-19   0.330729   0.335938   0.330729   0.330729   0.208026   18508800
1980-03-20   0.330729   0.334635   0.329427   0.329427   0.207207   11174400
1980-03-21   0.322917   0.322917   0.317708   0.317708   0.199836   12172800
…                 …          …          …          …          …          …
2020-03-26  51.740002  55.950001  51.660000  55.540001  55.540001   41459800
2020-03-27  53.419998  54.639999  52.070000  52.369999  52.369999   31633500
2020-03-30  52.990002  56.099998  52.830002  55.490002  55.490002   31628600
2020-03-31  55.060001  55.799999  53.220001  54.119999  54.119999   48074700
2020-04-01  52.500000  54.689999  51.430000  51.880001  51.880001   29582100

[10098 rows x 6 columns]
```

Pada soal yang dipakai hanya kolom close untuk kedua data, sehingga dapat membuat dataframe baru yang berisikan colomn close.

[ ]:
```python
# takes only index and close on each day
google = pd.DataFrame(df1["Close"])
intc = pd.DataFrame(df2["Close"])
print(google.head())
print(intc.head())
```

```
                Close
Date
2004-08-19  50.220219
2004-08-20  54.209209
2004-08-23  54.754753
2004-08-24  52.487488
2004-08-25  53.053055
               Close
Date
1980-03-17  0.325521
1980-03-18  0.322917
1980-03-19  0.330729
1980-03-20  0.329427
1980-03-21  0.317708
```

Data b ase telah terbentuk dengan berisikan kolom close dan index date

With this we can proceeed to the next step, which is LSTM preprocessing

[LO 3, LO 4, 10 poin] Lakukan praproses data dengan memisahkan data time series tersebut menjadi dua bagian input dan output dengan window size = 5 [dari hari senin s.d jumat] dan horizon = 5 [dari hari senin s.d jumat]. Kemudian pisahkan dataset menjadi 80% training set, 10% validation set dan 10% test set.

[ ]:
```python
# Library
from matplotlib import pyplot as plt
import numpy as np
import math
from sklearn.preprocessing import MinMaxScaler
```

[ ]:
```python
print(google.head(20))
print()
print(intc.head(20))
```

```
                Close
Date
2004-08-19  50.220219
2004-08-20  54.209209
2004-08-23  54.754753
2004-08-24  52.487488
2004-08-25  53.053055
2004-08-26  54.009010
2004-08-27  53.128128
2004-08-30  51.056057
2004-08-31  51.236237
2004-09-01  50.175175
2004-09-02  50.805805
2004-09-03  50.055054
2004-09-07  50.840839
```

```
2004-09-08  51.201202
2004-09-09  51.206207
2004-09-10  52.717716
2004-09-13  53.803802
2004-09-14  55.800800
2004-09-15  56.056057
2004-09-16  57.042042

               Close
Date
1980-03-17  0.325521
1980-03-18  0.322917
1980-03-19  0.330729
1980-03-20  0.329427
1980-03-21  0.317708
1980-03-24  0.311198
1980-03-25  0.312500
1980-03-26  0.309896
1980-03-27  0.299479
1980-03-28  0.311198
1980-03-31  0.321615
1980-04-01  0.322917
1980-04-02  0.325521
1980-04-03  0.319010
1980-04-07  0.311198
1980-04-08  0.312500
1980-04-09  0.305990
1980-04-10  0.304688
1980-04-11  0.304688
1980-04-14  0.307292
```

Disini dapat dilihat terdapat beberapa hari libur dimana pada hari tersebut bursa saham tutup, selain itu juga setiap hari sabtu dan minggu bursa sama tutup.

[ ]:
```python
# Exploration data on each dataset
# Checking null value
google.info()
print()
intc.info()
```

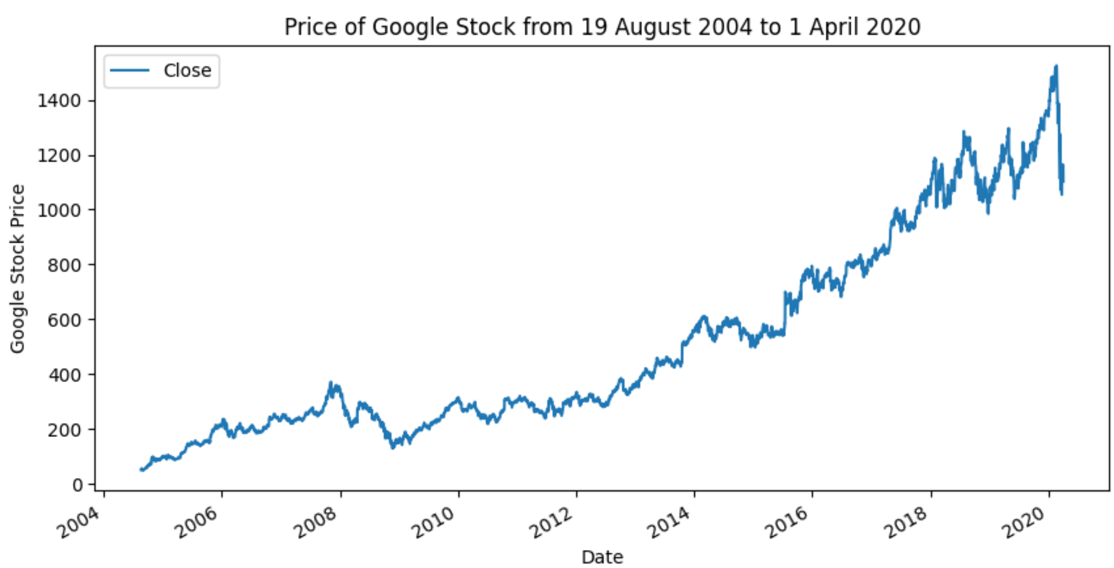
```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 3932 entries, 2004-08-19 to 2020-04-01
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Close   3932 non-null   float64
dtypes: float64(1)
memory usage: 61.4 KB
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 10098 entries, 1980-03-17 to 2020-04-01
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Close   10098 non-null  float64
dtypes: float64(1)
memory usage: 157.8 KB
```

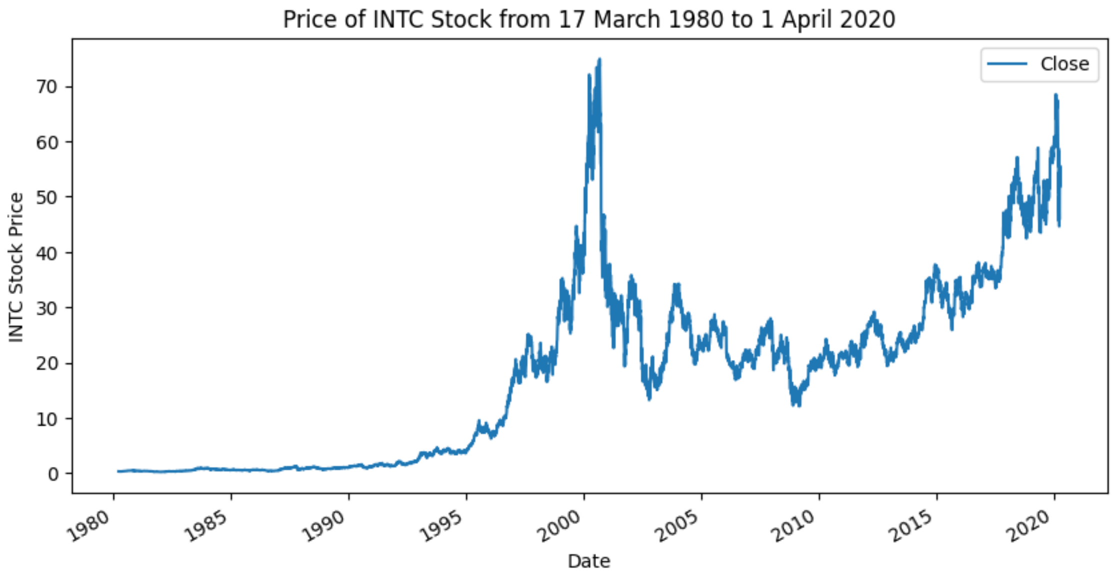
Dari sini diketahui tidak ada null value yang dihasilkan dan terdapat 3932 data pada saham google dan 10098 data pada saham intc

disini data duplicated tidak cek, karena ada kemungkinan saham memiliki value yang sama pada saat close di hari yang berbeda dan juga untuk outlier tidak cek, karena ada kemungkinan di dalam data saham memiliki data yang melonjak tinggi pada hari-hari tertentu atau suatu event.

[ ]:
```
# see the data in plot
google.plot(figsize=(10, 5))
plt.ylabel("Google Stock Price")
plt.title("Price of Google Stock from 19 August 2004 to 1 April 2020",␣
 ↪fontsize=12)
plt.legend(fontsize=10);
```



[ ]:
```
# see the data in plot
intc.plot(figsize=(10, 5))
plt.ylabel("INTC Stock Price")
plt.title("Price of INTC Stock from 17 March 1980 to 1 April 2020", fontsize=12)
plt.legend(fontsize=10);
```



Dari kedau plot yang telah terbentuk kita mengetahui intel memiliki kenaikan harga close lebih kecil dibandingkan dengan google.

Selain itu intc memiliki data dari tahun 1980, sedangkan google memiliki data tahun 2004.

Untuk trend dari data sendiri dapat dilihat google meningkat setiap tahun, tetapi untuk saham intel megalami kenaikan tinggi disekitar tahun 200.

Before windowing changing make new variable of array to hold a value of the clossing price

[ ]:
```
# to hold value of price
close_goo = google["Close"].to_numpy()
close_intc = intc["Close"].to_numpy()

print(close_goo)
print(close_intc)
```

```
[  50.22021866   54.20920944   54.75475311 … 1146.31005859 1161.94995117
 1102.09997559]
[ 0.32552084  0.32291666  0.33072916 … 55.49000168 54.11999893
 51.88000107]
```

Membuat function untuk pembuatan window dan horizon yang sekaligus melakukan splitting.

Window yang diminta adalah 5 dari senin hingga jumat dan horizon adalah 5 dari senin hingga jumat.

Define scaler untuk melakukan scalling

[ ]:
```
scaler = MinMaxScaler(feature_range=(0,1))
```

[ ]:
```
def window_data(df_close,scaling=False,train_size=0.8,check_value=False):

    training_data_len = math.ceil(len(df_close)* train_size)

    if scaling is True :
      data = scaler.fit_transform(df_close.values.reshape(-1, 1))
    else:
      data = df_close.values

    train_df = df_close.iloc[: training_data_len]
    train_data = data[:training_data_len]


    #Train set data
    # Define variable for train
    train_window = []
    train_horizon = []

    # using for loop with validate only accept closing when there is start from␣
↪monday to friday(window)
    # Also have next following data of monday to friday (horizon)
    for i in range(len(train_df)):
      if train_df.index[i].weekday() == 0 and i+9 < len(train_df) and train_df.
↪index[i + 5].weekday() == 0 and train_df.index[i + 9].weekday() == 4:
        train_window.append(train_data[i:i+5])
        train_horizon.append(train_data[i+5:i+10])

    #Determine where the start value of validation and test
    val_test_df = df_close.iloc[training_data_len: ]
    val_test_data = data[training_data_len: ]
    val_test_len = len(val_test_data)
    val_len = int(val_test_len * 0.5)
    test_len = val_test_len - val_len

    # Validation set data
    # Define variable for validation
    val_window = []
    val_horizon = []
    val_df =val_test_df.iloc[:val_len]
    val_data = val_test_data[:val_len]
    # using for loop with validate only accept closing when there is start from␣
↪monday to friday(window)
    # Also have next following data of monday to friday (horizon)
    for i in range(len(val_df)):
      if val_df.index[i].weekday() == 0  and i+9 < len(val_df) and val_df.
↪index[i + 5].weekday() == 0 and val_df.index[i +9].weekday() == 4:
        val_window.append(val_data[i:i+5])
```

```
        val_horizon.append(val_data[i+5:i+10])

    # Test set
    # Define variable for validation
    test_window = []
    test_horizon = []
    test_df = val_test_df.iloc[test_len:]
    test_data = val_test_data[test_len:]
    # using for loop with validate only accept closing when there is start from␣
↪monday to friday(window)
    # Also have next following data of monday to friday (horizon)
    for i in range(len(test_df)):
      if test_df.index[i].weekday() == 0  and i+9 < len(test_df) and test_df.
↪index[i + 5].weekday() == 0 and test_df.index[i + 9].weekday() == 4:
        test_window.append(test_data[i:i+5])
        test_horizon.append(test_data[i+5:i+10])

    # change the window data to array
    train_window = np.array(train_window)
    train_horizon = np.array(train_horizon)
    val_window = np.array(val_window)
    val_horizon = np.array(val_horizon)
    test_window = np.array(test_window)
    test_horizon = np.array(test_horizon)

    # Reshape the data so it can use in training data
    train_window = np.reshape(train_window, (train_window.shape[0],␣
↪train_window.shape[1], 1))
    val_window = np.reshape(val_window, (val_window.shape[0], val_window.
↪shape[1], 1))
    test_window = np.reshape(test_window, (test_window.shape[0], test_window.
↪shape[1], 1))
    if check_value is True :
      print("Sample Window with horizon in training:")
      for i in range(5):
        print("window :",train_window[i].flatten(),"-> Horizon :
↪",train_horizon[i].flatten())
    else:
      return train_window, train_horizon, val_window, val_horizon, test_window,␣
↪test_horizon
```

Contoh dari window dan horizon yang akan terbentuk pada train set

[ ]:
```
print("google")
window_data(google,check_value=True)
print()
print("intel")
```

```
window_data(intc,check_value=True)
```

```
google
Sample Window with horizon in training:
window : [54.75475311 52.48748779 53.05305481 54.00901031 53.12812805] ->
Horizon : [51.05605698 51.23623657 50.17517471 50.80580521 50.05505371]
window : [53.80380249 55.80080032 56.05605698 57.04204178 58.80380249] ->
Horizon : [59.73973846 58.9789772  59.2492485  60.47047043 59.97497559]
window : [59.73973846 58.9789772  59.2492485  60.47047043 59.97497559] ->
Horizon : [59.18918991 63.49349213 65.60560608 64.86486816 66.35635376]
window : [59.18918991 63.49349213 65.60560608 64.86486816 66.35635376] ->
Horizon : [67.59759521 69.2542572  68.60861206 69.49449158 68.93393707]
window : [67.59759521 69.2542572  68.60861206 69.49449158 68.93393707] ->
Horizon : [67.6977005  68.76876831 70.52052307 71.07106781 72.1271286 ]

intel
Sample Window with horizon in training:
window : [0.32552084 0.32291666 0.33072916 0.32942709 0.31770834] -> Horizon :
[0.31119791 0.3125     0.30989584 0.29947916 0.31119791]
window : [0.31119791 0.3125     0.30598959 0.3046875  0.3046875 ] -> Horizon :
[0.30729166 0.30338541 0.29166666 0.28645834 0.29036459]
window : [0.30729166 0.30338541 0.29166666 0.28645834 0.29036459] -> Horizon :
[0.28776041 0.30078125 0.31901041 0.3203125  0.31510416]
window : [0.28776041 0.30078125 0.31901041 0.3203125  0.31510416] -> Horizon :
[0.3125     0.31510416 0.31901041 0.3203125  0.32552084]
window : [0.3125     0.31510416 0.31901041 0.3203125  0.32552084] -> Horizon :
[0.32942709 0.328125   0.328125   0.32421875 0.328125  ]
```

Splitting Data

```
[ ]: train_win_goo, train_lab_goo, val_win_goo, val_lab_goo, test_win_goo,␣
      ↪test_lab_goo = window_data(google,scaling=True)
     len(train_win_goo), len(train_lab_goo), len(val_win_goo), len(val_lab_goo),␣
      ↪len(test_win_goo), len(test_lab_goo)
```

```
[ ]: (437, 437, 55, 55, 52, 52)
```

```
[ ]: train_win_int, train_lab_int, val_win_int, val_lab_int, test_win_int,␣
      ↪test_lab_int = window_data(intc,scaling=True)
     len(train_win_int), len(train_lab_int), len(val_win_int), len(val_lab_int),␣
      ↪len(test_win_int), len(test_lab_int)
```

```
[ ]: (1156, 1156, 137, 137, 139, 139)
```

```
[ ]: test_win_int_check = scaler.inverse_transform(test_win_int[:,:,0])
     test_win_goo_check = scaler.inverse_transform(test_win_goo[:,:,0])
```

Degan begitu kedua dataset telah siap untuk dimodelkan

[LO 3, LO 4, 10 poin] Buatlah arsitektur baseline sesuai dengan gambar arsitektur Transformer for Stocks berikut ini: (Catatan: bagian FEED FORWARD menggunakan satu layer Conv1D saja dengan Activation function menggunakan ReLU dan bagian node Perceptron pada output disesuaikan dengan horizon datanya)

[ ]:
```
# library
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
```

[ ]:
```
days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"]
```

Fucntion untuk mengevaluasi model

[ ]:
```
# function for evaluate model
def evaluate_preds(y_true, y_pred):

  y_true = tf.cast(y_true, dtype=tf.float32)
  y_pred = tf.cast(y_pred, dtype=tf.float32)

  # Calculate various metrics
  mae = tf.keras.metrics.mean_absolute_error(y_true, y_pred)
  mse = tf.keras.metrics.mean_squared_error(y_true, y_pred)
  rmse = tf.sqrt(mse)
  mape = tf.keras.metrics.mean_absolute_percentage_error(y_true, y_pred)

  return mae.numpy(),rmse.numpy(),mape.numpy()
```

Model tranfomers, sebelum masuk ke input. Model akan dimulai transformer block. Di dalam transformer block terdapat beberapa bagian. Multi head attention dan juga feed forward. Jika mengikuti model didalam soal. Selain itu terdapat juga add dan normalization setelah layernya.

[ ]:
```
def transformer_encoder(inputs, head_size, num_heads, ff_dim, dropout=0):

    x = layers.MultiHeadAttention(
        key_dim=head_size, num_heads=num_heads, dropout=dropout)(inputs, inputs)
    res = x + inputs

    x = layers.LayerNormalization(epsilon=1e-6)(res)
    x = layers.Conv1D(filters=ff_dim, kernel_size=1, activation = "relu")(x)
    x = layers.LayerNormalization(epsilon=1e-6)(res)
    return x + res
```

Setelah bagian penting dibuat akan masuk ke bagian input ke ouput. Pada model akan terdapat feature embedding. Embedding digunakan untuk mengubah input mnejadi beberapa vektor yang mewakili si kategorinya. Selanjutnya akan ada positional embeding, dimana ini digunakan untuk menambahkan informasi lebih.

Lalu akan masuk ke tranformer block yang diloop sesuai keiinginan. Setelah itu baru ke output.

[ ]:
```
def build_model(
    input_shape,
    embedding_dim,
    head_size,
    num_heads,
    ff_dim,
    num_transformer_blocks,
    mlp_units,
    dropout=0,
):
    inputs = keras.Input(shape=input_shape)
    x = layers.Embedding(input_dim=10, output_dim=embedding_dim)(inputs)
    x = layers.Dropout(dropout)(x)
    position = tf.range(start=0, limit=5, delta=1)
    position_embedding = layers.Embedding(input_dim=5,␣
 ↪output_dim=embedding_dim)(position)
    x = x + position_embedding

    for _ in range(num_transformer_blocks):
        x = transformer_encoder(x, head_size, num_heads, ff_dim, dropout)

    x = layers.GlobalAveragePooling2D()(x)
    outputs = layers.Dense(5, activation="linear")(x)
    return keras.Model(inputs, outputs)
```

[ ]:
```
input_shape = train_win_goo.shape[1:]
```

Untuk menapung hasil MAE, RMSE, dan MAPE

[ ]:
```
intel_model = pd.DataFrame(columns=['Model', 'MAE', 'RMSE', 'MAPE'])
google_model = pd.DataFrame(columns=['Model', 'MAE', 'RMSE', 'MAPE'])
```

[ ]:
```
base_goo = build_model(
    input_shape,
    embedding_dim=32,
    head_size=46,
    num_heads=60,
    ff_dim=55,
    num_transformer_blocks=5,
    mlp_units=[256],
    dropout=0.15,
)
```

```python
base_goo.compile(
    loss="mse",
    metrics=["mean_squared_error"],
)

base_goo.fit(
    train_win_goo,
    train_lab_goo,
    validation_data=(val_win_goo, val_lab_goo),
    epochs=20,
    batch_size=32
)
```

```
Epoch 1/20
14/14 [==============================] - 14s 83ms/step - loss: 439.4121 -
mean_squared_error: 439.4121 - val_loss: 14.9374 - val_mean_squared_error:
14.9374
Epoch 2/20
14/14 [==============================] - 0s 34ms/step - loss: 10.6449 -
mean_squared_error: 10.6449 - val_loss: 24.6729 - val_mean_squared_error:
24.6729
Epoch 3/20
14/14 [==============================] - 0s 33ms/step - loss: 13.1108 -
mean_squared_error: 13.1108 - val_loss: 0.7548 - val_mean_squared_error: 0.7548
Epoch 4/20
14/14 [==============================] - 0s 34ms/step - loss: 3.9043 -
mean_squared_error: 3.9043 - val_loss: 4.1138 - val_mean_squared_error: 4.1138
Epoch 5/20
14/14 [==============================] - 0s 34ms/step - loss: 0.8278 -
mean_squared_error: 0.8278 - val_loss: 1.1921 - val_mean_squared_error: 1.1921
Epoch 6/20
14/14 [==============================] - 0s 35ms/step - loss: 96.5431 -
mean_squared_error: 96.5431 - val_loss: 244.6326 - val_mean_squared_error:
244.6326
Epoch 7/20
14/14 [==============================] - 0s 34ms/step - loss: 48.3955 -
mean_squared_error: 48.3955 - val_loss: 0.2296 - val_mean_squared_error: 0.2296
Epoch 8/20
14/14 [==============================] - 0s 32ms/step - loss: 4.4256 -
mean_squared_error: 4.4256 - val_loss: 3.0293 - val_mean_squared_error: 3.0293
Epoch 9/20
14/14 [==============================] - 0s 30ms/step - loss: 0.4395 -
mean_squared_error: 0.4395 - val_loss: 0.3051 - val_mean_squared_error: 0.3051
Epoch 10/20
14/14 [==============================] - 0s 30ms/step - loss: 6.3078 -
mean_squared_error: 6.3078 - val_loss: 435.1536 - val_mean_squared_error:
435.1536
```

```
Epoch 11/20
14/14 [==============================] - 0s 32ms/step - loss: 76.3267 -
mean_squared_error: 76.3267 - val_loss: 0.2082 - val_mean_squared_error: 0.2082
Epoch 12/20
14/14 [==============================] - 0s 32ms/step - loss: 0.6895 -
mean_squared_error: 0.6895 - val_loss: 0.6735 - val_mean_squared_error: 0.6735
Epoch 13/20
14/14 [==============================] - 0s 30ms/step - loss: 0.3609 -
mean_squared_error: 0.3609 - val_loss: 0.4560 - val_mean_squared_error: 0.4560
Epoch 14/20
14/14 [==============================] - 0s 30ms/step - loss: 1111.6035 -
mean_squared_error: 1111.6035 - val_loss: 3.5140 - val_mean_squared_error:
3.5140
Epoch 15/20
14/14 [==============================] - 0s 30ms/step - loss: 0.2827 -
mean_squared_error: 0.2827 - val_loss: 0.2101 - val_mean_squared_error: 0.2101
Epoch 16/20
14/14 [==============================] - 0s 30ms/step - loss: 0.0899 -
mean_squared_error: 0.0899 - val_loss: 5.3004 - val_mean_squared_error: 5.3004
Epoch 17/20
14/14 [==============================] - 0s 30ms/step - loss: 3.4346 -
mean_squared_error: 3.4346 - val_loss: 1.7101 - val_mean_squared_error: 1.7101
Epoch 18/20
14/14 [==============================] - 0s 31ms/step - loss: 1.5620 -
mean_squared_error: 1.5620 - val_loss: 1.4202 - val_mean_squared_error: 1.4202
Epoch 19/20
14/14 [==============================] - 0s 30ms/step - loss: 1.1510 -
mean_squared_error: 1.1510 - val_loss: 3.0578 - val_mean_squared_error: 3.0578
Epoch 20/20
14/14 [==============================] - 0s 31ms/step - loss: 101.3497 -
mean_squared_error: 101.3497 - val_loss: 36.1800 - val_mean_squared_error:
36.1800
```

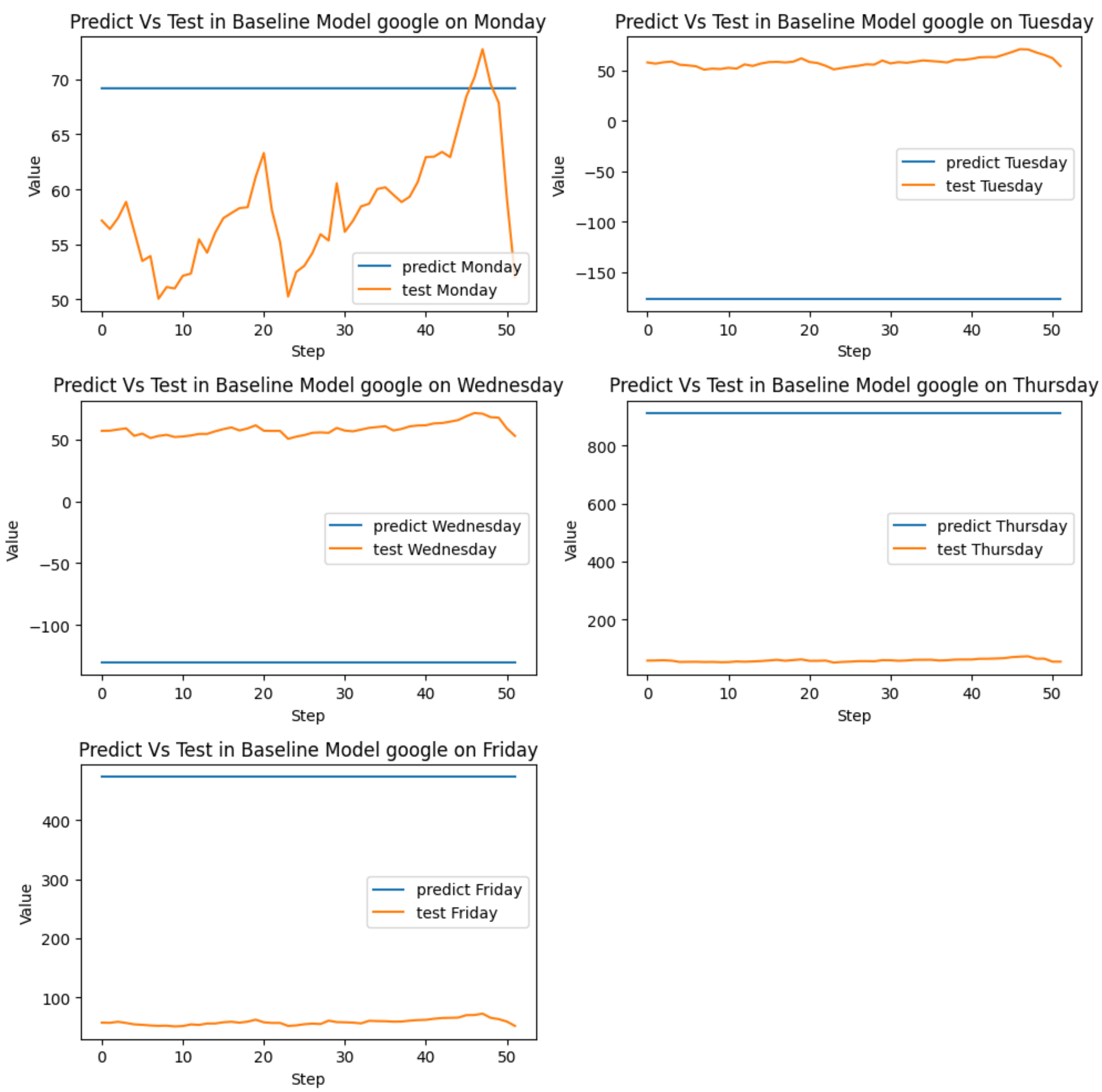
[ ]: <keras.callbacks.History at 0x7f83601300d0>

[ ]:
```python
base_goo_pred = base_goo.predict(test_win_goo)
base_goo_pred_check = scaler.inverse_transform(base_goo_pred)
```

```
2/2 [==============================] - 0s 9ms/step
```

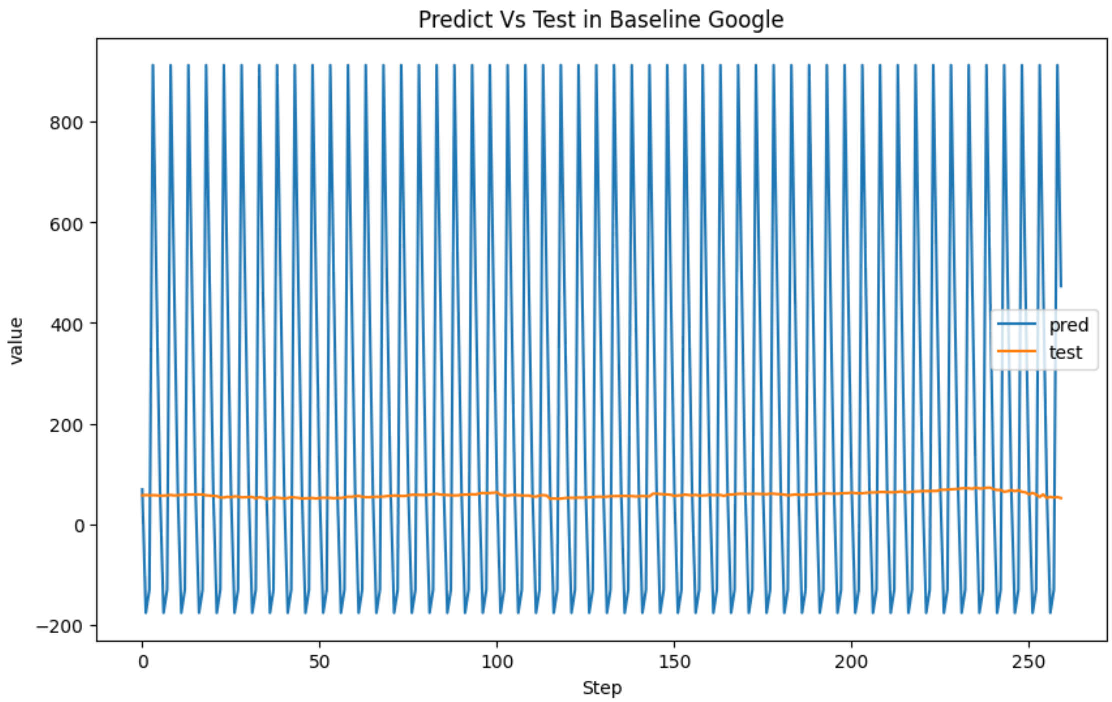
[ ]:
```python
fig, axs = plt.subplots(3, 2, figsize=(10, 10))



for i in range(5):
    ax = axs.flat[i]
    ax.plot(base_goo_pred_check[:,i], label=f'predict {days[i]}')
    ax.plot(test_win_goo_check[:, i], label=f'test {days[i]}')
    ax.set_title(f"Predict Vs Test in Baseline Model google on {days[i]}")
```

```
    ax.set_xlabel("Step")
    ax.set_ylabel("Value")
    ax.legend()
for j in range(5, 6):
    axs.flat[j].set_visible(False)

plt.tight_layout()
plt.show()
```



[ ]:
```
plt.figure(figsize=(10,6))
plt.plot(base_goo_pred_check.flatten(),label="pred")
plt.plot(test_win_goo_check.flatten(),label="test")
plt.title("Predict Vs Test in Baseline Google")
```

```
plt.xlabel("Step")
plt.ylabel("value")
plt.legend()
plt.show()
```



Dari sini dapat dilihat dengan menggunakan embedding di didalam univariate data memberikan hasil output yang konstan dan berulang. Sehingga dapat dikatakan modelnya tidak baik dan perlu diubah model.

```
[ ]: base_goo_results = evaluate_preds(y_true=test_lab_goo.flatten(),␣
      ↪y_pred=base_goo_pred.flatten())
     google_model.loc[0] =␣
      ↪["Base",base_goo_results[0],base_goo_results[1],base_goo_results[2]]
     print("MAE :",base_goo_results[0])
     print("RMSE :",base_goo_results[1])
     print("MAPE :",base_goo_results[2])
```

```
MAE : 4.5667076
RMSE : 5.9719186
MAPE : 591.5103
```

untuk buktinya dari model yang tidak baik adalah pada plot dan hasil dari MAPE, MAE, RMSE. Dimana angka yang diciptakan sangatlah tinggi untuk baseline ini.

[ ]:
```
base_int = build_model(
    input_shape,
    head_size=46,
    num_heads=60,
    embedding_dim=32,
    ff_dim=55,
    num_transformer_blocks=5,
    mlp_units=[256],
    dropout=0.15,
)

base_int.compile(
    loss="mean_squared_error",
    metrics=["mean_squared_error"],
)
#model.summary()


base_int.fit(
    train_win_int,
    train_lab_int,
    validation_data=(val_win_int, val_lab_int),
    epochs=20,
    batch_size=32
)
```

```
Epoch 1/20
37/37 [==============================] - 12s 43ms/step - loss: 157.3802 -
mean_squared_error: 157.3802 - val_loss: 1.0638 - val_mean_squared_error: 1.0638
Epoch 2/20
37/37 [==============================] - 1s 30ms/step - loss: 3.0675 -
mean_squared_error: 3.0675 - val_loss: 0.3894 - val_mean_squared_error: 0.3894
Epoch 3/20
37/37 [==============================] - 1s 31ms/step - loss: 49.4950 -
mean_squared_error: 49.4950 - val_loss: 1.8308 - val_mean_squared_error: 1.8308
Epoch 4/20
37/37 [==============================] - 1s 31ms/step - loss: 317.1993 -
mean_squared_error: 317.1993 - val_loss: 0.0633 - val_mean_squared_error: 0.0633
Epoch 5/20
37/37 [==============================] - 1s 33ms/step - loss: 2.4607 -
mean_squared_error: 2.4607 - val_loss: 0.2920 - val_mean_squared_error: 0.2920
Epoch 6/20
37/37 [==============================] - 1s 34ms/step - loss: 2.4575 -
mean_squared_error: 2.4575 - val_loss: 8.3145 - val_mean_squared_error: 8.3145
Epoch 7/20
37/37 [==============================] - 1s 34ms/step - loss: 28.1611 -
mean_squared_error: 28.1611 - val_loss: 0.2219 - val_mean_squared_error: 0.2219
```

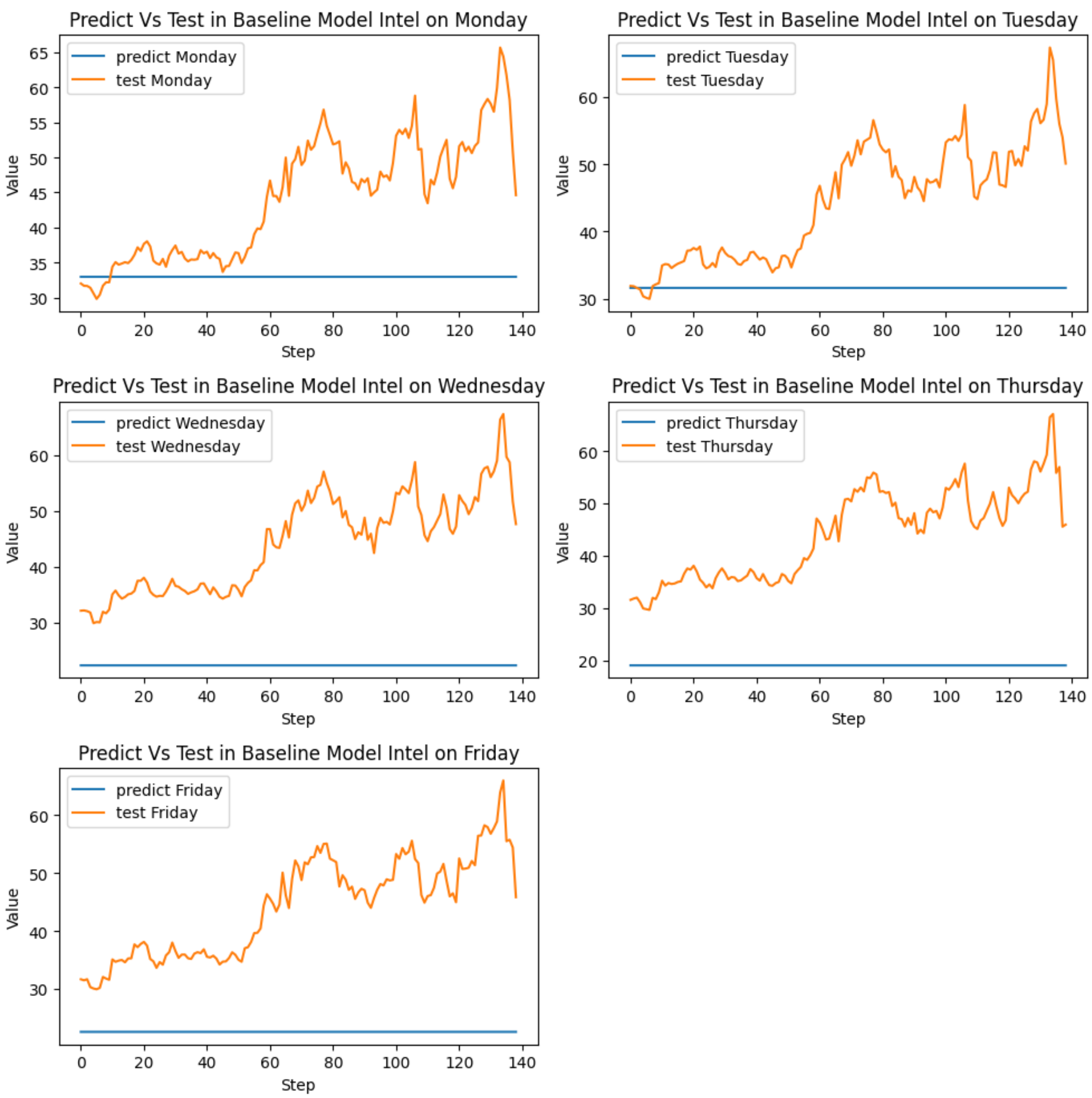
```
Epoch 8/20
37/37 [==============================] - 1s 32ms/step - loss: 23.7475 -
mean_squared_error: 23.7475 - val_loss: 0.3399 - val_mean_squared_error: 0.3399
Epoch 9/20
37/37 [==============================] - 1s 31ms/step - loss: 0.2787 -
mean_squared_error: 0.2787 - val_loss: 0.1334 - val_mean_squared_error: 0.1334
Epoch 10/20
37/37 [==============================] - 1s 32ms/step - loss: 61.4556 -
mean_squared_error: 61.4556 - val_loss: 0.2898 - val_mean_squared_error: 0.2898
Epoch 11/20
37/37 [==============================] - 1s 30ms/step - loss: 0.8806 -
mean_squared_error: 0.8806 - val_loss: 0.0928 - val_mean_squared_error: 0.0928
Epoch 12/20
37/37 [==============================] - 1s 30ms/step - loss: 173.2373 -
mean_squared_error: 173.2373 - val_loss: 0.0985 - val_mean_squared_error: 0.0985
Epoch 13/20
37/37 [==============================] - 1s 29ms/step - loss: 1.6142 -
mean_squared_error: 1.6142 - val_loss: 3.8689 - val_mean_squared_error: 3.8689
Epoch 14/20
37/37 [==============================] - 1s 29ms/step - loss: 47.1105 -
mean_squared_error: 47.1105 - val_loss: 3.9635 - val_mean_squared_error: 3.9635
Epoch 15/20
37/37 [==============================] - 1s 30ms/step - loss: 0.8107 -
mean_squared_error: 0.8107 - val_loss: 0.3607 - val_mean_squared_error: 0.3607
Epoch 16/20
37/37 [==============================] - 1s 29ms/step - loss: 35.4130 -
mean_squared_error: 35.4130 - val_loss: 1.5318 - val_mean_squared_error: 1.5318
Epoch 17/20
37/37 [==============================] - 1s 31ms/step - loss: 14.8994 -
mean_squared_error: 14.8994 - val_loss: 0.0392 - val_mean_squared_error: 0.0392
Epoch 18/20
37/37 [==============================] - 1s 32ms/step - loss: 17.9783 -
mean_squared_error: 17.9783 - val_loss: 0.0808 - val_mean_squared_error: 0.0808
Epoch 19/20
37/37 [==============================] - 1s 32ms/step - loss: 0.4178 -
mean_squared_error: 0.4178 - val_loss: 0.0317 - val_mean_squared_error: 0.0317
Epoch 20/20
37/37 [==============================] - 1s 30ms/step - loss: 17.9241 -
mean_squared_error: 17.9241 - val_loss: 0.0102 - val_mean_squared_error: 0.0102
```

[ ]: <keras.callbacks.History at 0x7f82cf1127d0>

[ ]:
```
base_int_pred = base_int.predict(test_win_int)
base_int_pred_check = scaler.inverse_transform(base_int_pred)
```

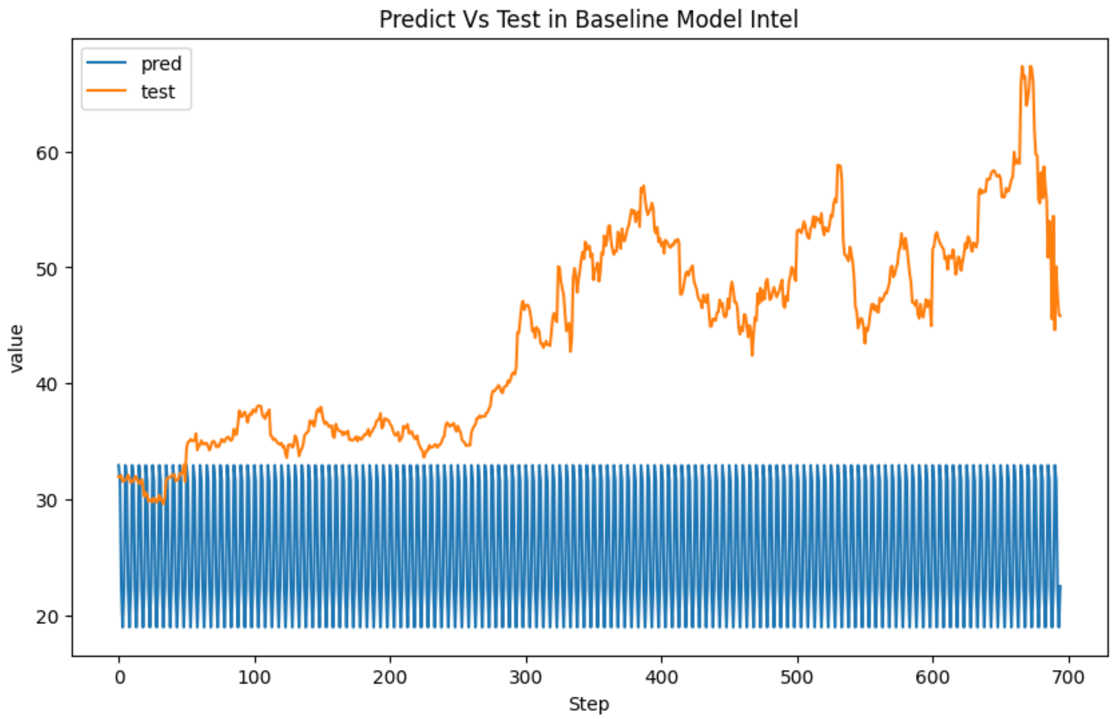
```
5/5 [==============================] - 0s 9ms/step
```

[ ]:
```python
fig, axs = plt.subplots(3, 2, figsize=(10, 10))


for i in range(5):
    ax = axs.flat[i]
    ax.plot(base_int_pred_check[:,i], label=f'predict {days[i]}')
    ax.plot(test_win_int_check[:, i], label=f'test {days[i]}')
    ax.set_title(f"Predict Vs Test in Baseline Model Intel on {days[i]}")
    ax.set_xlabel("Step")
    ax.set_ylabel("Value")
    ax.legend()
for j in range(5, 6):
    axs.flat[j].set_visible(False)

plt.tight_layout()
plt.show()
```



[ ]:

```
plt.figure(figsize=(10,6))
plt.plot(base_int_pred_check.flatten(), label = "pred")
plt.plot(test_win_int_check.flatten(), label = "test")
plt.title("Predict Vs Test in Baseline Model Intel")
plt.xlabel("Step")
plt.ylabel("value")
plt.legend()
plt.show()
```



[ ]:
```
base_int_results = evaluate_preds(y_true=test_lab_int.flatten(),␣
↪y_pred=base_int_pred.flatten())
intel_model.loc[0] =␣
↪["Base",base_int_results[0],base_int_results[1],base_int_results[2]]
print("MAE :",base_int_results[0])
print("RMSE :",base_int_results[1])
print("MAPE :",base_int_results[2])
```

```
MAE : 0.24888042
RMSE : 0.28294024
MAPE : 40.060062
```

Pada model intel base yang dihasilkan dengan embedding menghasilkan hasil yang tidak baik pada model. Sehingga pada modified perlu diperbaiki masalah ini.

[LO 1, LO 2, LO 3, LO 4, 20 poin] Modifikasi arsitektur Transformer for Stocks di atas agar mendapatkan hasil klasifikasi yang optimal. Kalian dapat menambahkan atau mengurangi arsitektur tersebut dan melakukan mengubah arsitektur pada nomor 2c dengan menggunakan dropout, batch normalization dan lain-lainnya. Dan selanjutnya lakukan proses tuning hyperparameter agar unjuk kerjanya meningkat. Berikan alasan mengapa modifikasi arsitektur dan metode tuning hyperparameter kalian lebih baik.

Pada base model terdapat beberapa masalah dalam model, khususnya masalah embedding yang tidak cocok pada data univariate. Sehingga perlu di hilangkan.

Dengan begitu dapat mengulang step building model seperti di baseline, tetapi menambahkan optimizer adam untuk mempercepat model.

Selain dari embedding terdapat juga penggantian parameter yang menjadi pengaruh penting. Seperti head size, jumlah head, hidden layer, dropoout setelah tranformer block.

[ ]:
```
def transformer_encoder1(inputs, head_size, num_heads, ff_dim, dropout=0):

    x = layers.LayerNormalization(epsilon=1e-6)(inputs)

    x = layers.MultiHeadAttention(
        key_dim=head_size, num_heads=num_heads, dropout=dropout
    )(x, x)
    x = layers.Dropout(dropout)(x)
    res = x + inputs

    x = layers.LayerNormalization(epsilon=1e-6)(res)
    x = layers.Conv1D(filters=ff_dim, kernel_size=1, activation = "relu")(x)
    x = layers.Dropout(dropout)(x)
    return x + res
```

[ ]:
```
def build_model1(
    input_shape,
    head_size,
    num_heads,
    ff_dim,
    num_transformer_blocks,
    mlp_units,
    dropout=0,
    mlp_dropout=0,
):
    inputs = keras.Input(shape=input_shape)
    x = inputs

    for _ in range(num_transformer_blocks):
        x = transformer_encoder1(x, head_size, num_heads, ff_dim, dropout)

    x = layers.GlobalAveragePooling1D(data_format="channels_first")(x)
    for dim in mlp_units:
        x = layers.Dense(dim, activation="elu")(x)
        x = layers.Dropout(mlp_dropout)(x)
    outputs = layers.Dense(5, activation="linear")(x)
    return keras.Model(inputs, outputs)
```

[ ]:
```
model1_goo = build_model1(
    input_shape,
    head_size=35,
    num_heads=75,
    ff_dim=256,
```

```
    num_transformer_blocks=5,
    mlp_units=[256],
    mlp_dropout=0.3,
    dropout=0.15,
)

model1_goo.compile(
    loss="mean_squared_error",
    optimizer=keras.optimizers.Adam(learning_rate=1e-4),
    metrics=["mean_squared_error"],
)

model1_goo.fit(
    train_win_goo,
    train_lab_goo,
    validation_data=(val_win_goo, val_lab_goo),
    epochs=40,
    batch_size=32
)
```

```
Epoch 1/40
14/14 [==============================] - 20s 113ms/step - loss: 0.0992 -
mean_squared_error: 0.0992 - val_loss: 0.4304 - val_mean_squared_error: 0.4304
Epoch 2/40
14/14 [==============================] - 0s 32ms/step - loss: 0.0511 -
mean_squared_error: 0.0511 - val_loss: 0.3690 - val_mean_squared_error: 0.3690
Epoch 3/40
14/14 [==============================] - 0s 30ms/step - loss: 0.0397 -
mean_squared_error: 0.0397 - val_loss: 0.2794 - val_mean_squared_error: 0.2794
Epoch 4/40
14/14 [==============================] - 0s 30ms/step - loss: 0.0314 -
mean_squared_error: 0.0314 - val_loss: 0.2284 - val_mean_squared_error: 0.2284
Epoch 5/40
14/14 [==============================] - 0s 31ms/step - loss: 0.0248 -
mean_squared_error: 0.0248 - val_loss: 0.2028 - val_mean_squared_error: 0.2028
Epoch 6/40
14/14 [==============================] - 0s 31ms/step - loss: 0.0200 -
mean_squared_error: 0.0200 - val_loss: 0.1806 - val_mean_squared_error: 0.1806
Epoch 7/40
14/14 [==============================] - 0s 29ms/step - loss: 0.0171 -
mean_squared_error: 0.0171 - val_loss: 0.1564 - val_mean_squared_error: 0.1564
Epoch 8/40
14/14 [==============================] - 0s 27ms/step - loss: 0.0158 -
mean_squared_error: 0.0158 - val_loss: 0.1485 - val_mean_squared_error: 0.1485
Epoch 9/40
14/14 [==============================] - 0s 23ms/step - loss: 0.0136 -
mean_squared_error: 0.0136 - val_loss: 0.1301 - val_mean_squared_error: 0.1301
```

```
Epoch 10/40
14/14 [==============================] - 0s 21ms/step - loss: 0.0124 -
mean_squared_error: 0.0124 - val_loss: 0.1352 - val_mean_squared_error: 0.1352
Epoch 11/40
14/14 [==============================] - 0s 21ms/step - loss: 0.0109 -
mean_squared_error: 0.0109 - val_loss: 0.1216 - val_mean_squared_error: 0.1216
Epoch 12/40
14/14 [==============================] - 0s 23ms/step - loss: 0.0096 -
mean_squared_error: 0.0096 - val_loss: 0.1097 - val_mean_squared_error: 0.1097
Epoch 13/40
14/14 [==============================] - 0s 21ms/step - loss: 0.0087 -
mean_squared_error: 0.0087 - val_loss: 0.0967 - val_mean_squared_error: 0.0967
Epoch 14/40
14/14 [==============================] - 0s 22ms/step - loss: 0.0088 -
mean_squared_error: 0.0088 - val_loss: 0.1122 - val_mean_squared_error: 0.1122
Epoch 15/40
14/14 [==============================] - 0s 22ms/step - loss: 0.0080 -
mean_squared_error: 0.0080 - val_loss: 0.0980 - val_mean_squared_error: 0.0980
Epoch 16/40
14/14 [==============================] - 0s 22ms/step - loss: 0.0071 -
mean_squared_error: 0.0071 - val_loss: 0.0985 - val_mean_squared_error: 0.0985
Epoch 17/40
14/14 [==============================] - 0s 21ms/step - loss: 0.0068 -
mean_squared_error: 0.0068 - val_loss: 0.1002 - val_mean_squared_error: 0.1002
Epoch 18/40
14/14 [==============================] - 0s 23ms/step - loss: 0.0072 -
mean_squared_error: 0.0072 - val_loss: 0.0824 - val_mean_squared_error: 0.0824
Epoch 19/40
14/14 [==============================] - 0s 22ms/step - loss: 0.0059 -
mean_squared_error: 0.0059 - val_loss: 0.0819 - val_mean_squared_error: 0.0819
Epoch 20/40
14/14 [==============================] - 0s 22ms/step - loss: 0.0058 -
mean_squared_error: 0.0058 - val_loss: 0.0788 - val_mean_squared_error: 0.0788
Epoch 21/40
14/14 [==============================] - 0s 21ms/step - loss: 0.0062 -
mean_squared_error: 0.0062 - val_loss: 0.0760 - val_mean_squared_error: 0.0760
Epoch 22/40
14/14 [==============================] - 0s 21ms/step - loss: 0.0051 -
mean_squared_error: 0.0051 - val_loss: 0.0591 - val_mean_squared_error: 0.0591
Epoch 23/40
14/14 [==============================] - 0s 21ms/step - loss: 0.0048 -
mean_squared_error: 0.0048 - val_loss: 0.0618 - val_mean_squared_error: 0.0618
Epoch 24/40
14/14 [==============================] - 0s 21ms/step - loss: 0.0045 -
mean_squared_error: 0.0045 - val_loss: 0.0572 - val_mean_squared_error: 0.0572
Epoch 25/40
14/14 [==============================] - 0s 22ms/step - loss: 0.0041 -
mean_squared_error: 0.0041 - val_loss: 0.0448 - val_mean_squared_error: 0.0448
```

```
Epoch 26/40
14/14 [==============================] - 0s 21ms/step - loss: 0.0044 -
mean_squared_error: 0.0044 - val_loss: 0.0483 - val_mean_squared_error: 0.0483
Epoch 27/40
14/14 [==============================] - 0s 21ms/step - loss: 0.0039 -
mean_squared_error: 0.0039 - val_loss: 0.0451 - val_mean_squared_error: 0.0451
Epoch 28/40
14/14 [==============================] - 0s 23ms/step - loss: 0.0035 -
mean_squared_error: 0.0035 - val_loss: 0.0457 - val_mean_squared_error: 0.0457
Epoch 29/40
14/14 [==============================] - 0s 22ms/step - loss: 0.0034 -
mean_squared_error: 0.0034 - val_loss: 0.0497 - val_mean_squared_error: 0.0497
Epoch 30/40
14/14 [==============================] - 0s 21ms/step - loss: 0.0033 -
mean_squared_error: 0.0033 - val_loss: 0.0455 - val_mean_squared_error: 0.0455
Epoch 31/40
14/14 [==============================] - 0s 22ms/step - loss: 0.0028 -
mean_squared_error: 0.0028 - val_loss: 0.0361 - val_mean_squared_error: 0.0361
Epoch 32/40
14/14 [==============================] - 0s 22ms/step - loss: 0.0026 -
mean_squared_error: 0.0026 - val_loss: 0.0322 - val_mean_squared_error: 0.0322
Epoch 33/40
14/14 [==============================] - 0s 21ms/step - loss: 0.0025 -
mean_squared_error: 0.0025 - val_loss: 0.0395 - val_mean_squared_error: 0.0395
Epoch 34/40
14/14 [==============================] - 0s 21ms/step - loss: 0.0025 -
mean_squared_error: 0.0025 - val_loss: 0.0367 - val_mean_squared_error: 0.0367
Epoch 35/40
14/14 [==============================] - 0s 21ms/step - loss: 0.0023 -
mean_squared_error: 0.0023 - val_loss: 0.0290 - val_mean_squared_error: 0.0290
Epoch 36/40
14/14 [==============================] - 0s 21ms/step - loss: 0.0024 -
mean_squared_error: 0.0024 - val_loss: 0.0234 - val_mean_squared_error: 0.0234
Epoch 37/40
14/14 [==============================] - 0s 22ms/step - loss: 0.0020 -
mean_squared_error: 0.0020 - val_loss: 0.0245 - val_mean_squared_error: 0.0245
Epoch 38/40
14/14 [==============================] - 0s 22ms/step - loss: 0.0020 -
mean_squared_error: 0.0020 - val_loss: 0.0259 - val_mean_squared_error: 0.0259
Epoch 39/40
14/14 [==============================] - 0s 22ms/step - loss: 0.0021 -
mean_squared_error: 0.0021 - val_loss: 0.0154 - val_mean_squared_error: 0.0154
Epoch 40/40
14/14 [==============================] - 0s 21ms/step - loss: 0.0020 -
mean_squared_error: 0.0020 - val_loss: 0.0262 - val_mean_squared_error: 0.0262
```

```
[ ]: <keras.callbacks.History at 0x7f82beb06620>
```

[ ]:
```
model1_goo_pred = model1_goo.predict(test_win_goo)
model1_goo_pred_check = scaler.inverse_transform(model1_goo_pred)
```

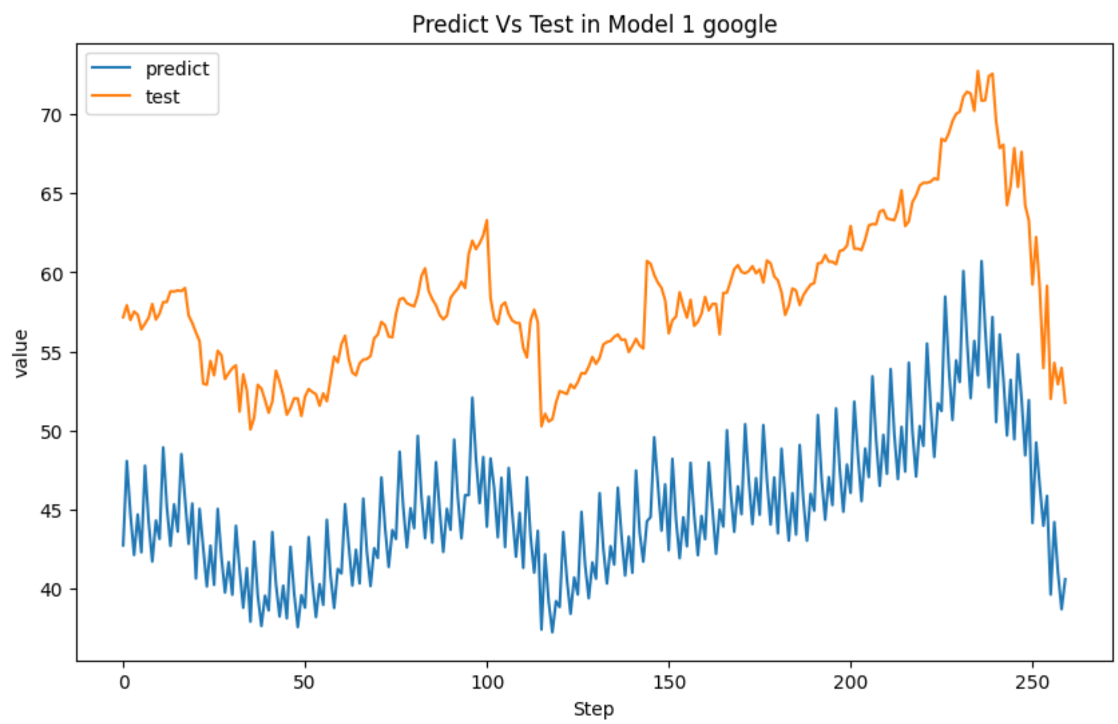
```
2/2 [==============================] - 1s 59ms/step
```

[ ]:
```
model1_goo_results = evaluate_preds(y_true=test_lab_goo.flatten(),␣
 ↪y_pred=model1_goo_pred.flatten())
google_model.loc[1] =␣
 ↪["Parameter",model1_goo_results[0],model1_goo_results[1],model1_goo_results[2]]
print("MAE :",model1_goo_results[0])
print("RMSE :",model1_goo_results[1])
print("MAPE :",model1_goo_results[2])
```

```
MAE : 0.17486672
RMSE : 0.17982869
MAPE : 22.415958
```

[ ]:
```
fig, axs = plt.subplots(3, 2, figsize=(10, 10))

for i in range(5):
    ax = axs.flat[i]
    ax.plot(model1_goo_pred_check[:, i], label=f'predict {days[i]}')
    ax.plot(test_win_goo_check[:, i], label=f'test {days[i]}')
    ax.set_title(f"Predict Vs Test in Model 1 Google on {days[i]}")
    ax.set_xlabel("Step")
    ax.set_ylabel("Value")
    ax.legend()
for j in range(5, 6):
    axs.flat[j].set_visible(False)

plt.tight_layout()
plt.show()
```

[ ]:
```
plt.figure(figsize=(10,6))
plt.plot(model1_goo_pred_check.flatten(), label='predict')
plt.plot(test_win_goo_check.flatten(),label='test')
plt.title("Predict Vs Test in Model 1 google")
plt.xlabel("Step")
plt.ylabel("value")
plt.legend()
plt.show()
```



[ ]:
```
model1_int = build_model1(
    input_shape,
    head_size=46,
    num_heads=60,
    ff_dim=55,
    num_transformer_blocks=5,
    mlp_units=[256],
)

model1_int.compile(
    loss="mean_squared_error",
    optimizer=keras.optimizers.Adam(learning_rate=1e-4),
    metrics=["mean_squared_error"],
)

model1_int.fit(
    train_win_int,
    train_lab_int,
    validation_data=(val_win_int, val_lab_int),
    epochs=40,
    batch_size=32
)
```

```
Epoch 1/40
```

```
37/37 [==============================] - 18s 35ms/step - loss: 0.0619 -
mean_squared_error: 0.0619 - val_loss: 0.0661 - val_mean_squared_error: 0.0661
Epoch 2/40
37/37 [==============================] - 1s 19ms/step - loss: 0.0269 -
mean_squared_error: 0.0269 - val_loss: 0.0305 - val_mean_squared_error: 0.0305
Epoch 3/40
37/37 [==============================] - 1s 20ms/step - loss: 0.0204 -
mean_squared_error: 0.0204 - val_loss: 0.0279 - val_mean_squared_error: 0.0279
Epoch 4/40
37/37 [==============================] - 1s 19ms/step - loss: 0.0167 -
mean_squared_error: 0.0167 - val_loss: 0.0210 - val_mean_squared_error: 0.0210
Epoch 5/40
37/37 [==============================] - 1s 20ms/step - loss: 0.0144 -
mean_squared_error: 0.0144 - val_loss: 0.0165 - val_mean_squared_error: 0.0165
Epoch 6/40
37/37 [==============================] - 1s 19ms/step - loss: 0.0104 -
mean_squared_error: 0.0104 - val_loss: 0.0179 - val_mean_squared_error: 0.0179
Epoch 7/40
37/37 [==============================] - 1s 19ms/step - loss: 0.0089 -
mean_squared_error: 0.0089 - val_loss: 0.0048 - val_mean_squared_error: 0.0048
Epoch 8/40
37/37 [==============================] - 1s 24ms/step - loss: 0.0065 -
mean_squared_error: 0.0065 - val_loss: 0.0035 - val_mean_squared_error: 0.0035
Epoch 9/40
37/37 [==============================] - 1s 26ms/step - loss: 0.0049 -
mean_squared_error: 0.0049 - val_loss: 0.0035 - val_mean_squared_error: 0.0035
Epoch 10/40
37/37 [==============================] - 1s 25ms/step - loss: 0.0035 -
mean_squared_error: 0.0035 - val_loss: 0.0040 - val_mean_squared_error: 0.0040
Epoch 11/40
37/37 [==============================] - 1s 27ms/step - loss: 0.0026 -
mean_squared_error: 0.0026 - val_loss: 0.0043 - val_mean_squared_error: 0.0043
Epoch 12/40
37/37 [==============================] - 1s 19ms/step - loss: 0.0019 -
mean_squared_error: 0.0019 - val_loss: 0.0017 - val_mean_squared_error: 0.0017
Epoch 13/40
37/37 [==============================] - 1s 21ms/step - loss: 0.0014 -
mean_squared_error: 0.0014 - val_loss: 0.0013 - val_mean_squared_error: 0.0013
Epoch 14/40
37/37 [==============================] - 1s 19ms/step - loss: 9.9217e-04 -
mean_squared_error: 9.9217e-04 - val_loss: 4.9510e-04 - val_mean_squared_error:
4.9510e-04
Epoch 15/40
37/37 [==============================] - 1s 19ms/step - loss: 8.3260e-04 -
mean_squared_error: 8.3260e-04 - val_loss: 5.7446e-04 - val_mean_squared_error:
5.7446e-04
Epoch 16/40
37/37 [==============================] - 1s 19ms/step - loss: 6.4264e-04 -
```

```
mean_squared_error: 6.4264e-04 - val_loss: 8.3766e-04 - val_mean_squared_error:
8.3766e-04
Epoch 17/40
37/37 [==============================] - 1s 19ms/step - loss: 8.4089e-04 -
mean_squared_error: 8.4089e-04 - val_loss: 5.1482e-04 - val_mean_squared_error:
5.1482e-04
Epoch 18/40
37/37 [==============================] - 1s 20ms/step - loss: 4.4759e-04 -
mean_squared_error: 4.4759e-04 - val_loss: 6.0827e-04 - val_mean_squared_error:
6.0827e-04
Epoch 19/40
37/37 [==============================] - 1s 19ms/step - loss: 3.4257e-04 -
mean_squared_error: 3.4257e-04 - val_loss: 3.9817e-04 - val_mean_squared_error:
3.9817e-04
Epoch 20/40
37/37 [==============================] - 1s 20ms/step - loss: 3.3358e-04 -
mean_squared_error: 3.3358e-04 - val_loss: 3.8639e-04 - val_mean_squared_error:
3.8639e-04
Epoch 21/40
37/37 [==============================] - 1s 19ms/step - loss: 2.8269e-04 -
mean_squared_error: 2.8269e-04 - val_loss: 2.4173e-04 - val_mean_squared_error:
2.4173e-04
Epoch 22/40
37/37 [==============================] - 1s 19ms/step - loss: 2.7127e-04 -
mean_squared_error: 2.7127e-04 - val_loss: 1.3594e-04 - val_mean_squared_error:
1.3594e-04
Epoch 23/40
37/37 [==============================] - 1s 19ms/step - loss: 2.5793e-04 -
mean_squared_error: 2.5793e-04 - val_loss: 1.3871e-04 - val_mean_squared_error:
1.3871e-04
Epoch 24/40
37/37 [==============================] - 1s 21ms/step - loss: 2.5858e-04 -
mean_squared_error: 2.5858e-04 - val_loss: 1.7603e-04 - val_mean_squared_error:
1.7603e-04
Epoch 25/40
37/37 [==============================] - 1s 22ms/step - loss: 2.4738e-04 -
mean_squared_error: 2.4738e-04 - val_loss: 1.9218e-04 - val_mean_squared_error:
1.9218e-04
Epoch 26/40
37/37 [==============================] - 1s 27ms/step - loss: 2.9156e-04 -
mean_squared_error: 2.9156e-04 - val_loss: 1.6748e-04 - val_mean_squared_error:
1.6748e-04
Epoch 27/40
37/37 [==============================] - 1s 26ms/step - loss: 3.0129e-04 -
mean_squared_error: 3.0129e-04 - val_loss: 1.7551e-04 - val_mean_squared_error:
1.7551e-04
Epoch 28/40
37/37 [==============================] - 1s 26ms/step - loss: 2.7100e-04 -
```

```
mean_squared_error: 2.7100e-04 - val_loss: 1.3943e-04 - val_mean_squared_error:
1.3943e-04
Epoch 29/40
37/37 [==============================] - 1s 21ms/step - loss: 2.7748e-04 -
mean_squared_error: 2.7748e-04 - val_loss: 1.2312e-04 - val_mean_squared_error:
1.2312e-04
Epoch 30/40
37/37 [==============================] - 1s 19ms/step - loss: 3.5581e-04 -
mean_squared_error: 3.5581e-04 - val_loss: 1.8898e-04 - val_mean_squared_error:
1.8898e-04
Epoch 31/40
37/37 [==============================] - 1s 20ms/step - loss: 2.6160e-04 -
mean_squared_error: 2.6160e-04 - val_loss: 1.7808e-04 - val_mean_squared_error:
1.7808e-04
Epoch 32/40
37/37 [==============================] - 1s 18ms/step - loss: 2.7319e-04 -
mean_squared_error: 2.7319e-04 - val_loss: 3.2373e-04 - val_mean_squared_error:
3.2373e-04
Epoch 33/40
37/37 [==============================] - 1s 19ms/step - loss: 2.5200e-04 -
mean_squared_error: 2.5200e-04 - val_loss: 1.2945e-04 - val_mean_squared_error:
1.2945e-04
Epoch 34/40
37/37 [==============================] - 1s 19ms/step - loss: 2.3935e-04 -
mean_squared_error: 2.3935e-04 - val_loss: 4.7537e-04 - val_mean_squared_error:
4.7537e-04
Epoch 35/40
37/37 [==============================] - 1s 19ms/step - loss: 3.0066e-04 -
mean_squared_error: 3.0066e-04 - val_loss: 0.0024 - val_mean_squared_error:
0.0024
Epoch 36/40
37/37 [==============================] - 1s 19ms/step - loss: 6.0399e-04 -
mean_squared_error: 6.0399e-04 - val_loss: 1.4332e-04 - val_mean_squared_error:
1.4332e-04
Epoch 37/40
37/37 [==============================] - 1s 21ms/step - loss: 2.3077e-04 -
mean_squared_error: 2.3077e-04 - val_loss: 2.0913e-04 - val_mean_squared_error:
2.0913e-04
Epoch 38/40
37/37 [==============================] - 1s 22ms/step - loss: 2.4345e-04 -
mean_squared_error: 2.4345e-04 - val_loss: 1.2213e-04 - val_mean_squared_error:
1.2213e-04
Epoch 39/40
37/37 [==============================] - 1s 22ms/step - loss: 2.3581e-04 -
mean_squared_error: 2.3581e-04 - val_loss: 1.2179e-04 - val_mean_squared_error:
1.2179e-04
Epoch 40/40
37/37 [==============================] - 1s 25ms/step - loss: 2.3513e-04 -
```

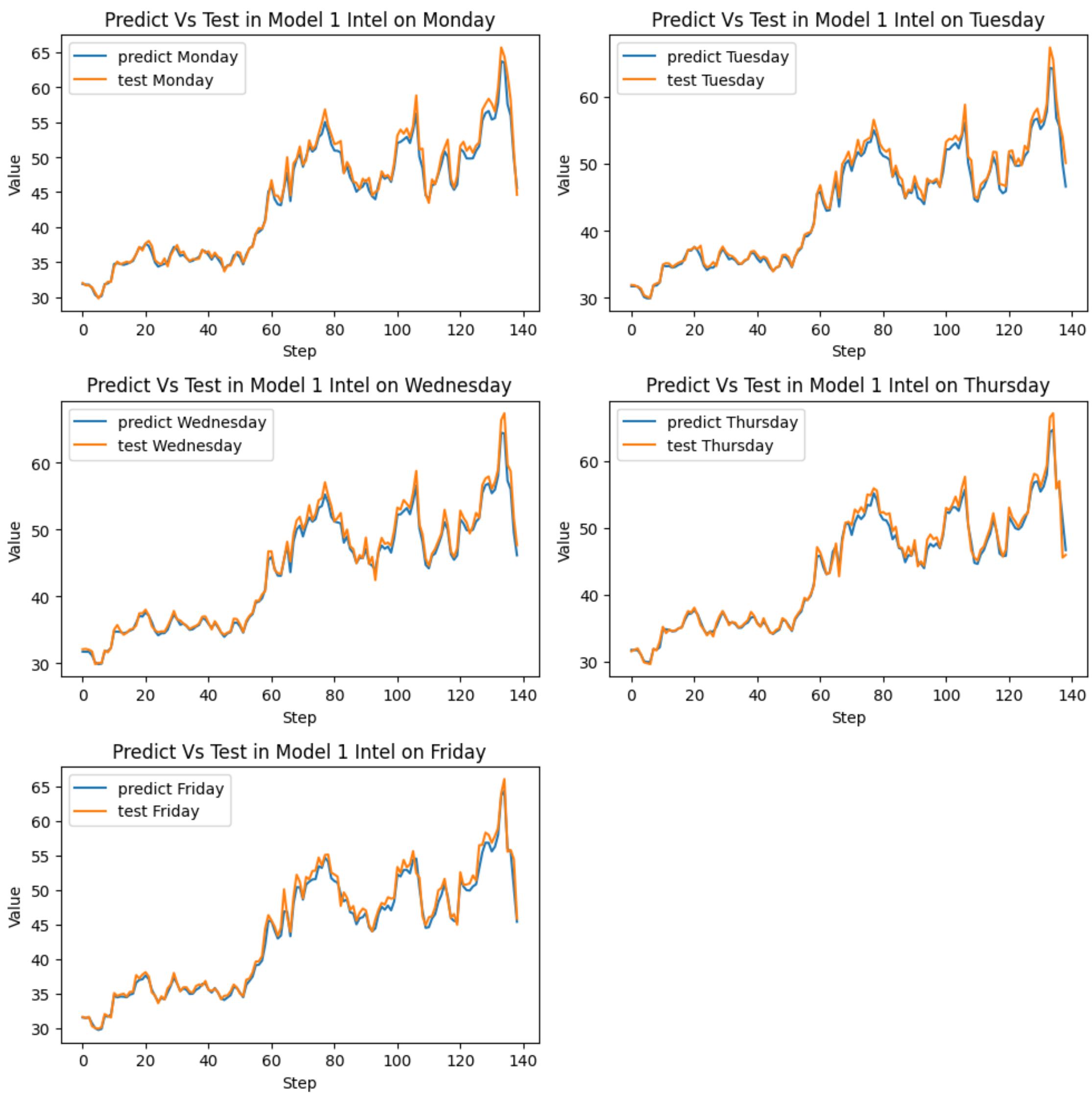
```
mean_squared_error: 2.3513e-04 - val_loss: 1.2308e-04 - val_mean_squared_error:
1.2308e-04
```

[ ]: <keras.callbacks.History at 0x7f82c0e94280>

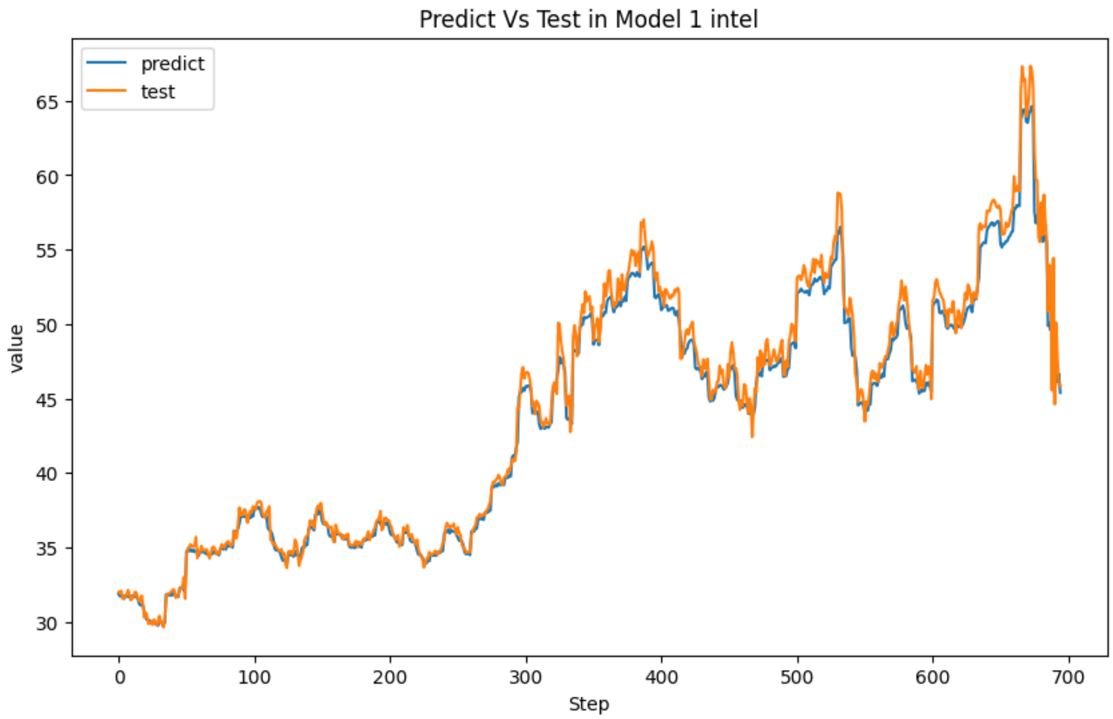
[ ]:
```python
model1_int_pred = model1_int.predict(test_win_int)
model1_int_pred_check = scaler.inverse_transform(model1_int_pred)
```

```
5/5 [==============================] - 0s 19ms/step
```

[ ]:
```python
fig, axs = plt.subplots(3, 2, figsize=(10, 10))

for i in range(5):
    ax = axs.flat[i]
    ax.plot(model1_int_pred_check[:, i], label=f'predict {days[i]}')
    ax.plot(test_win_int_check[:, i], label=f'test {days[i]}')
    ax.set_title(f"Predict Vs Test in Model 1 Intel on {days[i]}")
    ax.set_xlabel("Step")
    ax.set_ylabel("Value")
    ax.legend()
for j in range(5, 6):
    axs.flat[j].set_visible(False)

plt.tight_layout()
plt.show()
```



[ ]:
```
plt.figure(figsize=(10,6))
plt.plot(model1_int_pred_check.flatten(), label='predict')
plt.plot(test_win_int_check.flatten(),label='test')
plt.title("Predict Vs Test in Model 1 intel")
plt.xlabel("Step")
plt.ylabel("value")
plt.legend()
plt.show()
```




```
[ ]: model1_int_results = evaluate_preds(y_true=test_lab_int.flatten(),␣
      ↪y_pred=model1_int_pred.flatten())
     intel_model.loc[1] =␣
      ↪["Paramater",model1_int_results[0],model1_int_results[1],model1_int_results[2]]
     print("MAE :",model1_int_results[0])
     print("RMSE :",model1_int_results[1])
     print("MAPE :",model1_int_results[2])
```

```
MAE : 0.017313467
RMSE : 0.024075076
MAPE : 2.7895958
```

Dari hasil yang diberikan terdapat beberapa hal yang dapat disimpulkan pada model google khususnya pada predict dari model mengalami offset sangat jauh dari test yang diberikan. Maka dari itu salah satu cara untuk mengatasi hal ini adalah menambahkan convulsion layer di dalam tranformers block.

Sedangkan untuk model intel sendiri offset yang diberikan antara predicted dengan test tidaklah cukup jauh. Tetapi untuk menambahkan akurasi dari model (mengurangi MAPE) akan diperlakukan cara yang sama seperti google

Pada tahapan ini parameter akan sama seperti model 1 tidak ada yang diubah hanya penambahan layer convulsion

[ ]:
```
def transformer_encoder_2(inputs, head_size, num_heads, ff_dim, dropout=0):

    x = layers.LayerNormalization(epsilon=1e-6)(inputs)

    x = layers.MultiHeadAttention(
        key_dim=head_size, num_heads=num_heads, dropout=dropout
    )(x, x)
    x = layers.Dropout(dropout)(x)
    res = x + inputs

    x = layers.LayerNormalization(epsilon=1e-6)(res)
    x = layers.Conv1D(filters=ff_dim, kernel_size=1, activation = "relu")(x)
    x = layers.Dropout(dropout)(x)
    x = layers.Conv1D(filters=inputs.shape[-1], kernel_size=1)(x)
    return x + res
```

[ ]:
```
def model2(
    input_shape,
    head_size,
    num_heads,
    ff_dim,
    num_transformer_blocks,
    mlp_units,
    dropout=0,
    mlp_dropout=0,
):
    inputs = keras.Input(shape=input_shape)
    x = inputs

    for _ in range(num_transformer_blocks):
        x = transformer_encoder_2(x, head_size, num_heads, ff_dim, dropout)

    x = layers.GlobalAveragePooling1D(data_format="channels_first")(x)
    for dim in mlp_units:
        x = layers.Dense(dim, activation="elu")(x)
        x = layers.Dropout(mlp_dropout)(x)
    outputs = layers.Dense(5, activation="linear")(x)
    return keras.Model(inputs, outputs)
```

[ ]:
```
model2_goo = model2(
    input_shape,
    head_size=35,
    num_heads=75,
    ff_dim=256,
    num_transformer_blocks=5,
    mlp_units=[256],
    mlp_dropout=0.3,
```

```
    dropout=0.15,
)

model2_goo.compile(
    loss="mean_squared_error",
    optimizer=keras.optimizers.Adam(learning_rate=1e-4),
    metrics=["mean_squared_error"],
)

model2_goo.fit(
    train_win_goo,
    train_lab_goo,
    validation_data=(val_win_goo, val_lab_goo),
    epochs=40,
    batch_size=32
)
```

```
Epoch 1/40
14/14 [==============================] - 18s 77ms/step - loss: 0.0573 -
mean_squared_error: 0.0573 - val_loss: 0.3723 - val_mean_squared_error: 0.3723
Epoch 2/40
14/14 [==============================] - 0s 27ms/step - loss: 0.0423 -
mean_squared_error: 0.0423 - val_loss: 0.2624 - val_mean_squared_error: 0.2624
Epoch 3/40
14/14 [==============================] - 0s 27ms/step - loss: 0.0276 -
mean_squared_error: 0.0276 - val_loss: 0.1742 - val_mean_squared_error: 0.1742
Epoch 4/40
14/14 [==============================] - 0s 28ms/step - loss: 0.0179 -
mean_squared_error: 0.0179 - val_loss: 0.1189 - val_mean_squared_error: 0.1189
Epoch 5/40
14/14 [==============================] - 0s 29ms/step - loss: 0.0116 -
mean_squared_error: 0.0116 - val_loss: 0.0847 - val_mean_squared_error: 0.0847
Epoch 6/40
14/14 [==============================] - 0s 27ms/step - loss: 0.0089 -
mean_squared_error: 0.0089 - val_loss: 0.0571 - val_mean_squared_error: 0.0571
Epoch 7/40
14/14 [==============================] - 0s 28ms/step - loss: 0.0072 -
mean_squared_error: 0.0072 - val_loss: 0.0504 - val_mean_squared_error: 0.0504
Epoch 8/40
14/14 [==============================] - 0s 29ms/step - loss: 0.0064 -
mean_squared_error: 0.0064 - val_loss: 0.0438 - val_mean_squared_error: 0.0438
Epoch 9/40
14/14 [==============================] - 1s 40ms/step - loss: 0.0053 -
mean_squared_error: 0.0053 - val_loss: 0.0343 - val_mean_squared_error: 0.0343
Epoch 10/40
14/14 [==============================] - 1s 39ms/step - loss: 0.0046 -
mean_squared_error: 0.0046 - val_loss: 0.0275 - val_mean_squared_error: 0.0275
```

```
Epoch 11/40
14/14 [==============================] - 1s 38ms/step - loss: 0.0040 -
mean_squared_error: 0.0040 - val_loss: 0.0227 - val_mean_squared_error: 0.0227
Epoch 12/40
14/14 [==============================] - 1s 39ms/step - loss: 0.0037 -
mean_squared_error: 0.0037 - val_loss: 0.0182 - val_mean_squared_error: 0.0182
Epoch 13/40
14/14 [==============================] - 1s 38ms/step - loss: 0.0034 -
mean_squared_error: 0.0034 - val_loss: 0.0167 - val_mean_squared_error: 0.0167
Epoch 14/40
14/14 [==============================] - 1s 42ms/step - loss: 0.0030 -
mean_squared_error: 0.0030 - val_loss: 0.0171 - val_mean_squared_error: 0.0171
Epoch 15/40
14/14 [==============================] - 0s 31ms/step - loss: 0.0028 -
mean_squared_error: 0.0028 - val_loss: 0.0121 - val_mean_squared_error: 0.0121
Epoch 16/40
14/14 [==============================] - 0s 27ms/step - loss: 0.0028 -
mean_squared_error: 0.0028 - val_loss: 0.0090 - val_mean_squared_error: 0.0090
Epoch 17/40
14/14 [==============================] - 0s 26ms/step - loss: 0.0023 -
mean_squared_error: 0.0023 - val_loss: 0.0089 - val_mean_squared_error: 0.0089
Epoch 18/40
14/14 [==============================] - 0s 28ms/step - loss: 0.0022 -
mean_squared_error: 0.0022 - val_loss: 0.0074 - val_mean_squared_error: 0.0074
Epoch 19/40
14/14 [==============================] - 0s 30ms/step - loss: 0.0020 -
mean_squared_error: 0.0020 - val_loss: 0.0059 - val_mean_squared_error: 0.0059
Epoch 20/40
14/14 [==============================] - 0s 29ms/step - loss: 0.0021 -
mean_squared_error: 0.0021 - val_loss: 0.0047 - val_mean_squared_error: 0.0047
Epoch 21/40
14/14 [==============================] - 0s 27ms/step - loss: 0.0020 -
mean_squared_error: 0.0020 - val_loss: 0.0042 - val_mean_squared_error: 0.0042
Epoch 22/40
14/14 [==============================] - 0s 26ms/step - loss: 0.0018 -
mean_squared_error: 0.0018 - val_loss: 0.0030 - val_mean_squared_error: 0.0030
Epoch 23/40
14/14 [==============================] - 0s 24ms/step - loss: 0.0017 -
mean_squared_error: 0.0017 - val_loss: 0.0028 - val_mean_squared_error: 0.0028
Epoch 24/40
14/14 [==============================] - 0s 23ms/step - loss: 0.0018 -
mean_squared_error: 0.0018 - val_loss: 0.0031 - val_mean_squared_error: 0.0031
Epoch 25/40
14/14 [==============================] - 0s 22ms/step - loss: 0.0018 -
mean_squared_error: 0.0018 - val_loss: 0.0022 - val_mean_squared_error: 0.0022
Epoch 26/40
14/14 [==============================] - 0s 25ms/step - loss: 0.0018 -
mean_squared_error: 0.0018 - val_loss: 0.0015 - val_mean_squared_error: 0.0015
```

```
Epoch 27/40
14/14 [==============================] - 0s 24ms/step - loss: 0.0016 -
mean_squared_error: 0.0016 - val_loss: 0.0018 - val_mean_squared_error: 0.0018
Epoch 28/40
14/14 [==============================] - 0s 24ms/step - loss: 0.0016 -
mean_squared_error: 0.0016 - val_loss: 0.0011 - val_mean_squared_error: 0.0011
Epoch 29/40
14/14 [==============================] - 0s 24ms/step - loss: 0.0016 -
mean_squared_error: 0.0016 - val_loss: 0.0012 - val_mean_squared_error: 0.0012
Epoch 30/40
14/14 [==============================] - 0s 23ms/step - loss: 0.0015 -
mean_squared_error: 0.0015 - val_loss: 0.0011 - val_mean_squared_error: 0.0011
Epoch 31/40
14/14 [==============================] - 0s 22ms/step - loss: 0.0015 -
mean_squared_error: 0.0015 - val_loss: 0.0010 - val_mean_squared_error: 0.0010
Epoch 32/40
14/14 [==============================] - 0s 23ms/step - loss: 0.0014 -
mean_squared_error: 0.0014 - val_loss: 0.0010 - val_mean_squared_error: 0.0010
Epoch 33/40
14/14 [==============================] - 0s 24ms/step - loss: 0.0015 -
mean_squared_error: 0.0015 - val_loss: 8.1397e-04 - val_mean_squared_error:
8.1397e-04
Epoch 34/40
14/14 [==============================] - 0s 22ms/step - loss: 0.0015 -
mean_squared_error: 0.0015 - val_loss: 8.3077e-04 - val_mean_squared_error:
8.3077e-04
Epoch 35/40
14/14 [==============================] - 0s 23ms/step - loss: 0.0014 -
mean_squared_error: 0.0014 - val_loss: 9.9896e-04 - val_mean_squared_error:
9.9896e-04
Epoch 36/40
14/14 [==============================] - 0s 22ms/step - loss: 0.0013 -
mean_squared_error: 0.0013 - val_loss: 5.7254e-04 - val_mean_squared_error:
5.7254e-04
Epoch 37/40
14/14 [==============================] - 0s 24ms/step - loss: 0.0014 -
mean_squared_error: 0.0014 - val_loss: 7.3727e-04 - val_mean_squared_error:
7.3727e-04
Epoch 38/40
14/14 [==============================] - 0s 23ms/step - loss: 0.0014 -
mean_squared_error: 0.0014 - val_loss: 6.8149e-04 - val_mean_squared_error:
6.8149e-04
Epoch 39/40
14/14 [==============================] - 0s 23ms/step - loss: 0.0014 -
mean_squared_error: 0.0014 - val_loss: 8.0497e-04 - val_mean_squared_error:
8.0497e-04
Epoch 40/40
14/14 [==============================] - 0s 23ms/step - loss: 0.0014 -
```

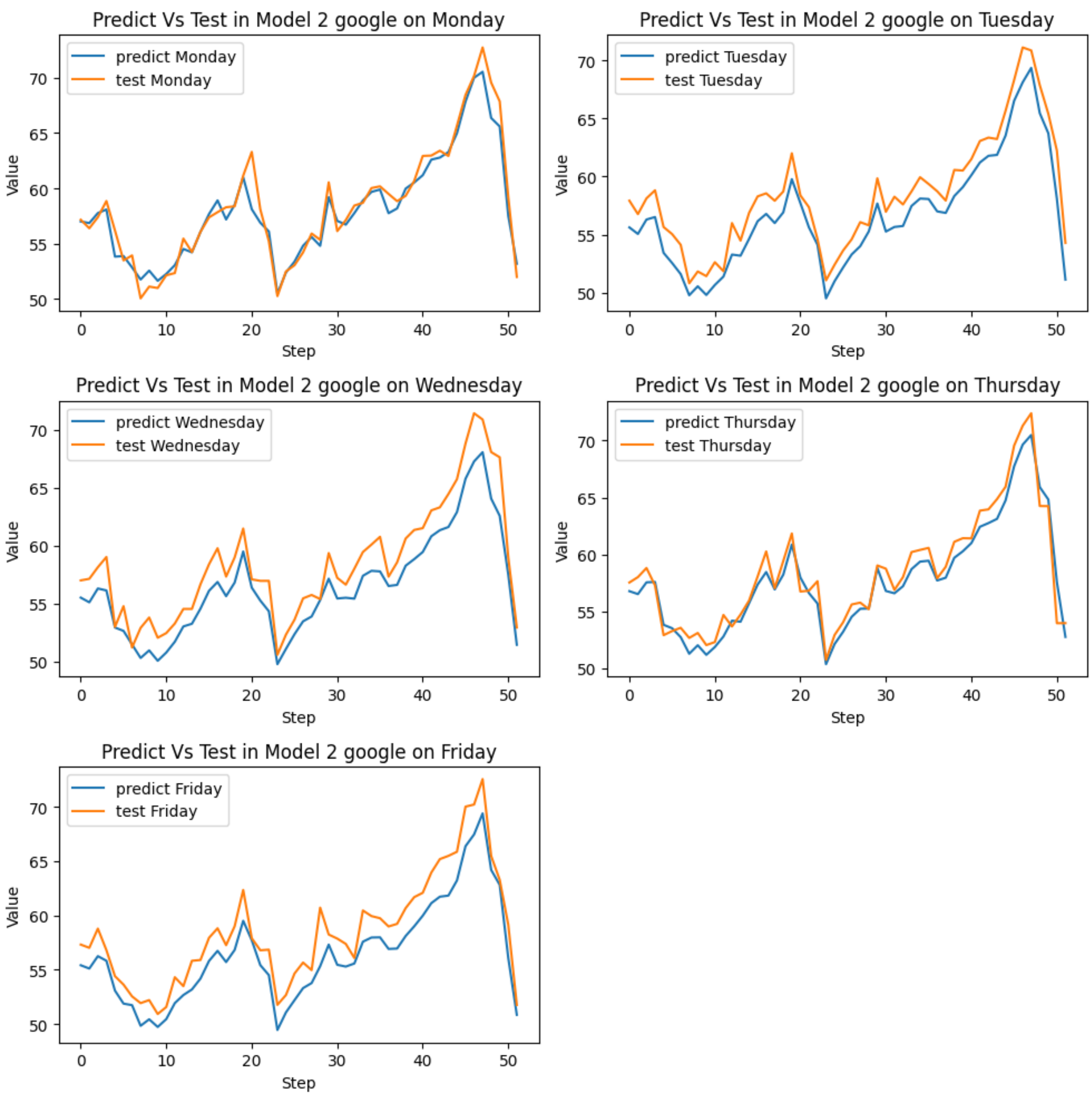
```
mean_squared_error: 0.0014 - val_loss: 7.4521e-04 - val_mean_squared_error:
7.4521e-04
```

[ ]: <keras.callbacks.History at 0x7f82c04bc310>

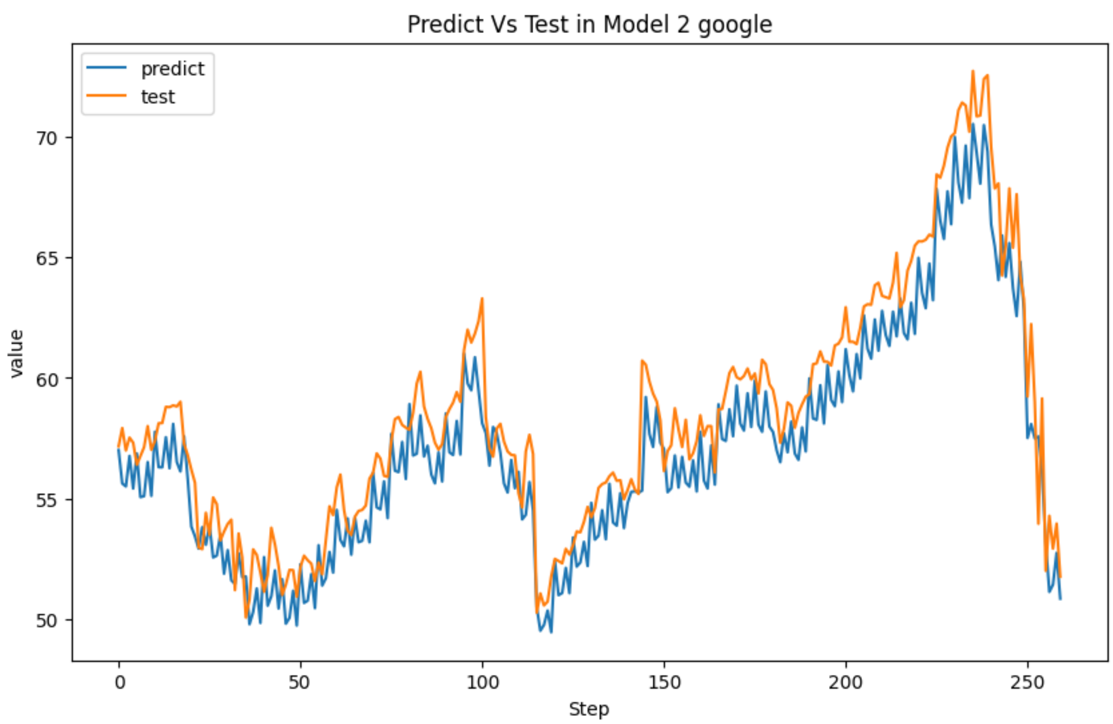
[ ]:
```python
model2_goo_pred = model2_goo.predict(test_win_goo)
model2_goo_pred_check = scaler.inverse_transform(model2_goo_pred)
```

```
2/2 [==============================] - 1s 36ms/step
```

[ ]:
```python
fig, axs = plt.subplots(3, 2, figsize=(10, 10))

for i in range(5):
    ax = axs.flat[i]
    ax.plot(model2_goo_pred_check[:, i], label=f'predict {days[i]}')
    ax.plot(test_win_goo_check[:, i], label=f'test {days[i]}')
    ax.set_title(f"Predict Vs Test in Model 2 google on {days[i]}")
    ax.set_xlabel("Step")
    ax.set_ylabel("Value")
    ax.legend()
for j in range(5, 6):
    axs.flat[j].set_visible(False)

plt.tight_layout()
plt.show()
```



[ ]:
```
plt.figure(figsize=(10,6))
plt.plot(model2_goo_pred_check.flatten(), label='predict')
plt.plot(test_win_goo_check.flatten(),label='test')
plt.title("Predict Vs Test in Model 2 google")
plt.xlabel("Step")
plt.ylabel("value")
plt.legend()
plt.show()
```



[ ]:
```
model2_goo_results = evaluate_preds(y_true=test_lab_goo.flatten(),␣
↪y_pred=model2_goo_pred.flatten())
google_model.loc[2] =␣
↪["Conv",model2_goo_results[0],model2_goo_results[1],model2_goo_results[2]]
print("MAE :",model2_goo_results[0])
print("RMSE :",model2_goo_results[1])
print("MAPE :",model2_goo_results[2])
```

```
MAE : 0.028406017
RMSE : 0.034612
MAPE : 3.6062129
```

[ ]:
```
model2_int = model2(
    input_shape,
    head_size=35,
    num_heads=75,
    ff_dim=256,
    num_transformer_blocks=5,
    mlp_units=[256],
    mlp_dropout=0.3,
    dropout=0.15,
)

model2_int.compile(
```

```
    loss="mean_squared_error",
    optimizer=keras.optimizers.Adam(learning_rate=1e-4),
    metrics=["mean_squared_error"],
)

model2_int.fit(
    train_win_int,
    train_lab_int,
    validation_data=(val_win_int, val_lab_int),
    epochs=40,
    batch_size=32
)
```

```
Epoch 1/40
37/37 [==============================] - 19s 38ms/step - loss: 0.0322 -
mean_squared_error: 0.0322 - val_loss: 0.0274 - val_mean_squared_error: 0.0274
Epoch 2/40
37/37 [==============================] - 1s 23ms/step - loss: 0.0148 -
mean_squared_error: 0.0148 - val_loss: 0.0140 - val_mean_squared_error: 0.0140
Epoch 3/40
37/37 [==============================] - 1s 22ms/step - loss: 0.0092 -
mean_squared_error: 0.0092 - val_loss: 0.0073 - val_mean_squared_error: 0.0073
Epoch 4/40
37/37 [==============================] - 1s 21ms/step - loss: 0.0055 -
mean_squared_error: 0.0055 - val_loss: 0.0030 - val_mean_squared_error: 0.0030
Epoch 5/40
37/37 [==============================] - 1s 21ms/step - loss: 0.0038 -
mean_squared_error: 0.0038 - val_loss: 0.0014 - val_mean_squared_error: 0.0014
Epoch 6/40
37/37 [==============================] - 1s 32ms/step - loss: 0.0026 -
mean_squared_error: 0.0026 - val_loss: 0.0012 - val_mean_squared_error: 0.0012
Epoch 7/40
37/37 [==============================] - 1s 33ms/step - loss: 0.0022 -
mean_squared_error: 0.0022 - val_loss: 3.9977e-04 - val_mean_squared_error:
3.9977e-04
Epoch 8/40
37/37 [==============================] - 1s 34ms/step - loss: 0.0019 -
mean_squared_error: 0.0019 - val_loss: 3.4321e-04 - val_mean_squared_error:
3.4321e-04
Epoch 9/40
37/37 [==============================] - 1s 40ms/step - loss: 0.0019 -
mean_squared_error: 0.0019 - val_loss: 2.6841e-04 - val_mean_squared_error:
2.6841e-04
Epoch 10/40
37/37 [==============================] - 1s 25ms/step - loss: 0.0018 -
mean_squared_error: 0.0018 - val_loss: 3.4575e-04 - val_mean_squared_error:
3.4575e-04
```

```
Epoch 11/40
37/37 [==============================] - 1s 25ms/step - loss: 0.0018 -
mean_squared_error: 0.0018 - val_loss: 2.9974e-04 - val_mean_squared_error:
2.9974e-04
Epoch 12/40
37/37 [==============================] - 1s 27ms/step - loss: 0.0017 -
mean_squared_error: 0.0017 - val_loss: 1.1896e-04 - val_mean_squared_error:
1.1896e-04
Epoch 13/40
37/37 [==============================] - 1s 26ms/step - loss: 0.0018 -
mean_squared_error: 0.0018 - val_loss: 2.7164e-04 - val_mean_squared_error:
2.7164e-04
Epoch 14/40
37/37 [==============================] - 1s 28ms/step - loss: 0.0018 -
mean_squared_error: 0.0018 - val_loss: 1.3027e-04 - val_mean_squared_error:
1.3027e-04
Epoch 15/40
37/37 [==============================] - 1s 35ms/step - loss: 0.0018 -
mean_squared_error: 0.0018 - val_loss: 1.9441e-04 - val_mean_squared_error:
1.9441e-04
Epoch 16/40
37/37 [==============================] - 1s 36ms/step - loss: 0.0017 -
mean_squared_error: 0.0017 - val_loss: 1.4261e-04 - val_mean_squared_error:
1.4261e-04
Epoch 17/40
37/37 [==============================] - 3s 82ms/step - loss: 0.0017 -
mean_squared_error: 0.0017 - val_loss: 2.1224e-04 - val_mean_squared_error:
2.1224e-04
Epoch 18/40
37/37 [==============================] - 4s 95ms/step - loss: 0.0017 -
mean_squared_error: 0.0017 - val_loss: 1.4290e-04 - val_mean_squared_error:
1.4290e-04
Epoch 19/40
37/37 [==============================] - 2s 61ms/step - loss: 0.0016 -
mean_squared_error: 0.0016 - val_loss: 2.4571e-04 - val_mean_squared_error:
2.4571e-04
Epoch 20/40
37/37 [==============================] - 2s 47ms/step - loss: 0.0016 -
mean_squared_error: 0.0016 - val_loss: 1.2167e-04 - val_mean_squared_error:
1.2167e-04
Epoch 21/40
37/37 [==============================] - 2s 44ms/step - loss: 0.0016 -
mean_squared_error: 0.0016 - val_loss: 1.1809e-04 - val_mean_squared_error:
1.1809e-04
Epoch 22/40
37/37 [==============================] - 2s 46ms/step - loss: 0.0014 -
mean_squared_error: 0.0014 - val_loss: 1.2875e-04 - val_mean_squared_error:
1.2875e-04
```

```
Epoch 23/40
37/37 [==============================] - 2s 48ms/step - loss: 0.0016 -
mean_squared_error: 0.0016 - val_loss: 1.4345e-04 - val_mean_squared_error:
1.4345e-04
Epoch 24/40
37/37 [==============================] - 3s 67ms/step - loss: 0.0014 -
mean_squared_error: 0.0014 - val_loss: 1.1734e-04 - val_mean_squared_error:
1.1734e-04
Epoch 25/40
37/37 [==============================] - 2s 55ms/step - loss: 0.0015 -
mean_squared_error: 0.0015 - val_loss: 1.9036e-04 - val_mean_squared_error:
1.9036e-04
Epoch 26/40
37/37 [==============================] - 2s 44ms/step - loss: 0.0014 -
mean_squared_error: 0.0014 - val_loss: 1.6296e-04 - val_mean_squared_error:
1.6296e-04
Epoch 27/40
37/37 [==============================] - 2s 45ms/step - loss: 0.0014 -
mean_squared_error: 0.0014 - val_loss: 1.2734e-04 - val_mean_squared_error:
1.2734e-04
Epoch 28/40
37/37 [==============================] - 2s 41ms/step - loss: 0.0014 -
mean_squared_error: 0.0014 - val_loss: 1.3421e-04 - val_mean_squared_error:
1.3421e-04
Epoch 29/40
37/37 [==============================] - 2s 51ms/step - loss: 0.0015 -
mean_squared_error: 0.0015 - val_loss: 1.5006e-04 - val_mean_squared_error:
1.5006e-04
Epoch 30/40
37/37 [==============================] - 1s 33ms/step - loss: 0.0014 -
mean_squared_error: 0.0014 - val_loss: 1.1604e-04 - val_mean_squared_error:
1.1604e-04
Epoch 31/40
37/37 [==============================] - 2s 61ms/step - loss: 0.0014 -
mean_squared_error: 0.0014 - val_loss: 1.2758e-04 - val_mean_squared_error:
1.2758e-04
Epoch 32/40
37/37 [==============================] - 1s 30ms/step - loss: 0.0013 -
mean_squared_error: 0.0013 - val_loss: 2.8310e-04 - val_mean_squared_error:
2.8310e-04
Epoch 33/40
37/37 [==============================] - 1s 32ms/step - loss: 0.0015 -
mean_squared_error: 0.0015 - val_loss: 3.2297e-04 - val_mean_squared_error:
3.2297e-04
Epoch 34/40
37/37 [==============================] - 1s 23ms/step - loss: 0.0014 -
mean_squared_error: 0.0014 - val_loss: 1.4336e-04 - val_mean_squared_error:
1.4336e-04
```

```
Epoch 35/40
37/37 [==============================] - 1s 22ms/step - loss: 0.0012 -
mean_squared_error: 0.0012 - val_loss: 1.6022e-04 - val_mean_squared_error:
1.6022e-04
Epoch 36/40
37/37 [==============================] - 1s 22ms/step - loss: 0.0012 -
mean_squared_error: 0.0012 - val_loss: 2.1992e-04 - val_mean_squared_error:
2.1992e-04
Epoch 37/40
37/37 [==============================] - 1s 22ms/step - loss: 0.0014 -
mean_squared_error: 0.0014 - val_loss: 1.4859e-04 - val_mean_squared_error:
1.4859e-04
Epoch 38/40
37/37 [==============================] - 1s 21ms/step - loss: 0.0013 -
mean_squared_error: 0.0013 - val_loss: 1.3259e-04 - val_mean_squared_error:
1.3259e-04
Epoch 39/40
37/37 [==============================] - 1s 22ms/step - loss: 0.0013 -
mean_squared_error: 0.0013 - val_loss: 1.1579e-04 - val_mean_squared_error:
1.1579e-04
Epoch 40/40
37/37 [==============================] - 1s 25ms/step - loss: 0.0012 -
mean_squared_error: 0.0012 - val_loss: 1.1288e-04 - val_mean_squared_error:
1.1288e-04
```

[ ]:
```
<keras.callbacks.History at 0x7f82881f32b0>
```

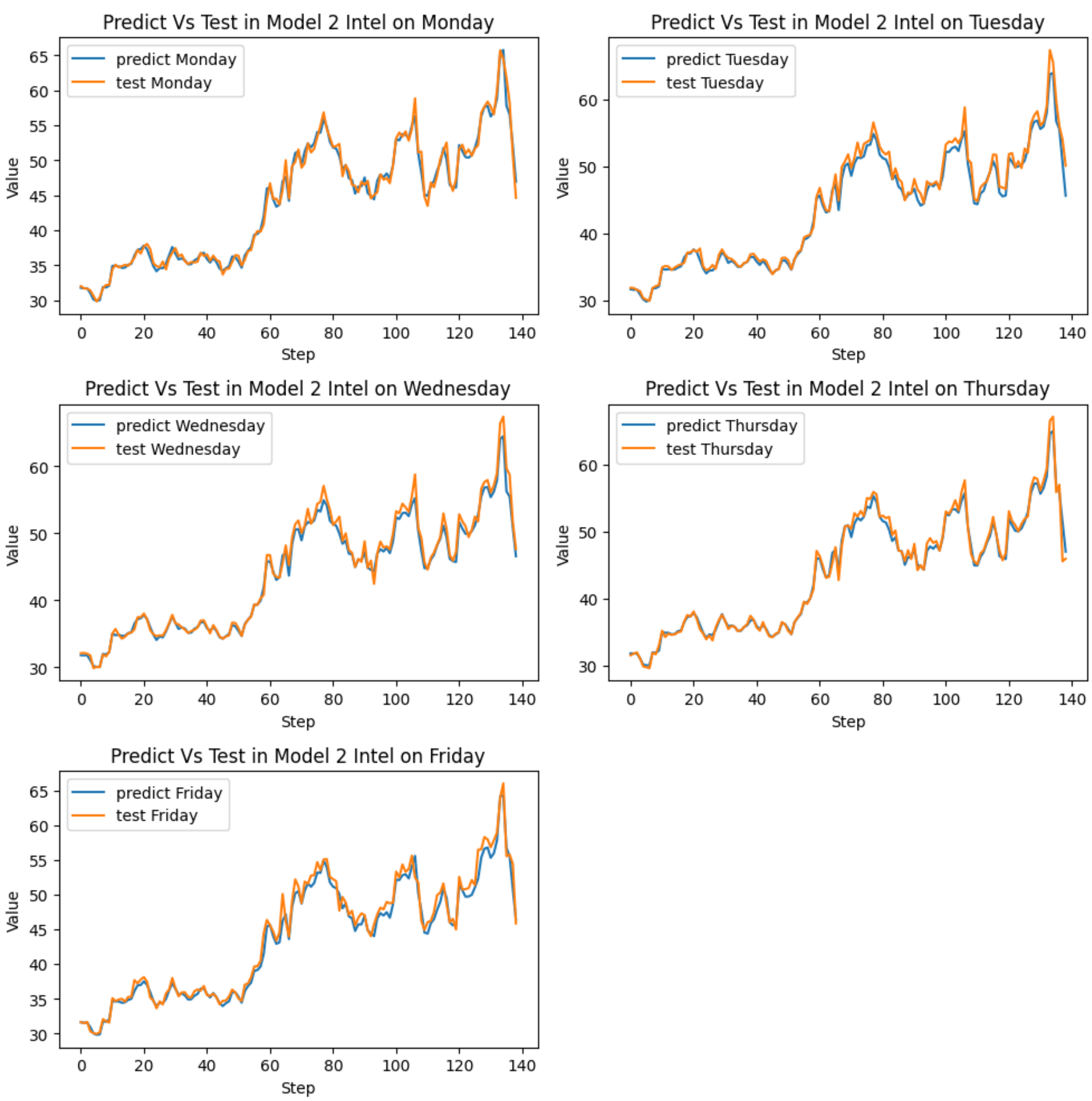
[ ]:
```
model2_int_pred = model2_int.predict(test_win_int)
model2_int_pred_check = scaler.inverse_transform(model2_int_pred)
```

```
WARNING:tensorflow:5 out of the last 15 calls to <function
Model.make_predict_function.<locals>.predict_function at 0x7f82902c7d90>
triggered tf.function retracing. Tracing is expensive and the excessive number
of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2)
passing tensors with different shapes, (3) passing Python objects instead of
tensors. For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has reduce_retracing=True option that can avoid unnecessary
retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
```

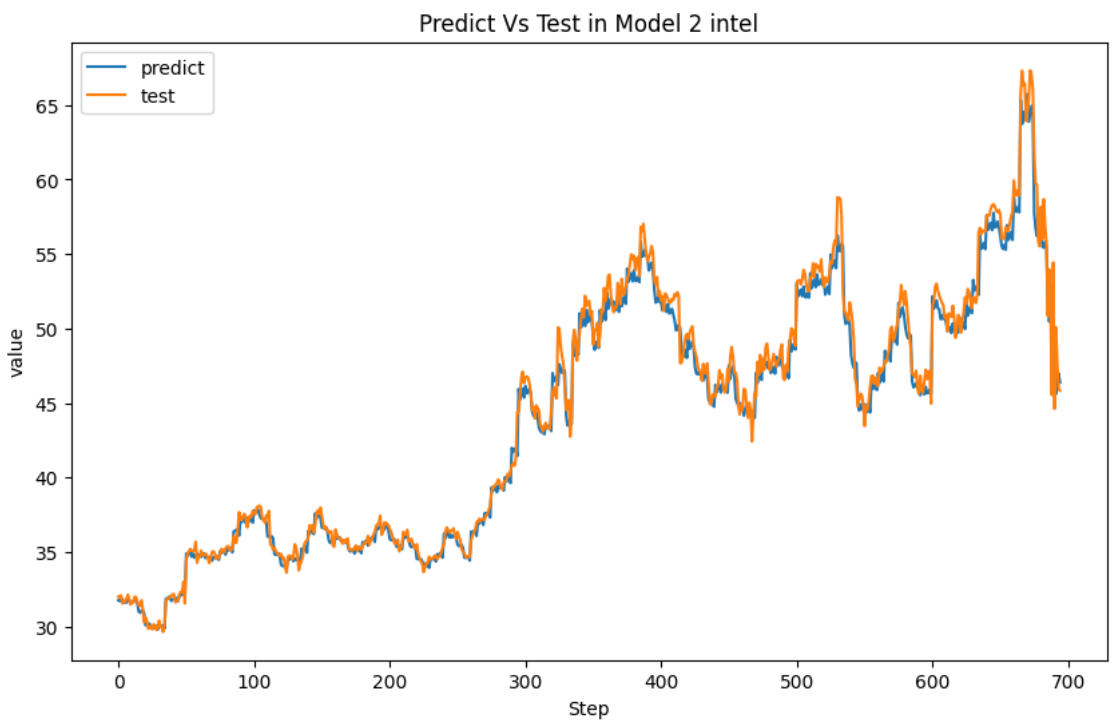
```
5/5 [==============================] - 1s 20ms/step
```

[ ]:
```
fig, axs = plt.subplots(3, 2, figsize=(10, 10))

for i in range(5):
    ax = axs.flat[i]
    ax.plot(model2_int_pred_check[:, i], label=f'predict {days[i]}')
```

```
    ax.plot(test_win_int_check[:, i], label=f'test {days[i]}')
    ax.set_title(f"Predict Vs Test in Model 2 Intel on {days[i]}")
    ax.set_xlabel("Step")
    ax.set_ylabel("Value")
    ax.legend()
for j in range(5, 6):
    axs.flat[j].set_visible(False)

plt.tight_layout()
plt.show()
```



[ ]:
```
plt.figure(figsize=(10,6))
plt.plot(model2_int_pred_check.flatten(), label='predict')
plt.plot(test_win_int_check.flatten(),label='test')
plt.title("Predict Vs Test in Model 2 intel")
plt.xlabel("Step")
plt.ylabel("value")
plt.legend()
plt.show()
```



[ ]:
```
model2_int_results = evaluate_preds(y_true=test_lab_int.flatten(),␣
    ↪y_pred=model2_int_pred.flatten())
intel_model.loc[2] =␣
    ↪["Conv",model2_int_results[0],model2_int_results[1],model2_int_results[2]]
print("MAE :",model2_int_results[0])
print("RMSE :",model2_int_results[1])
print("MAPE :",model2_int_results[2])
```

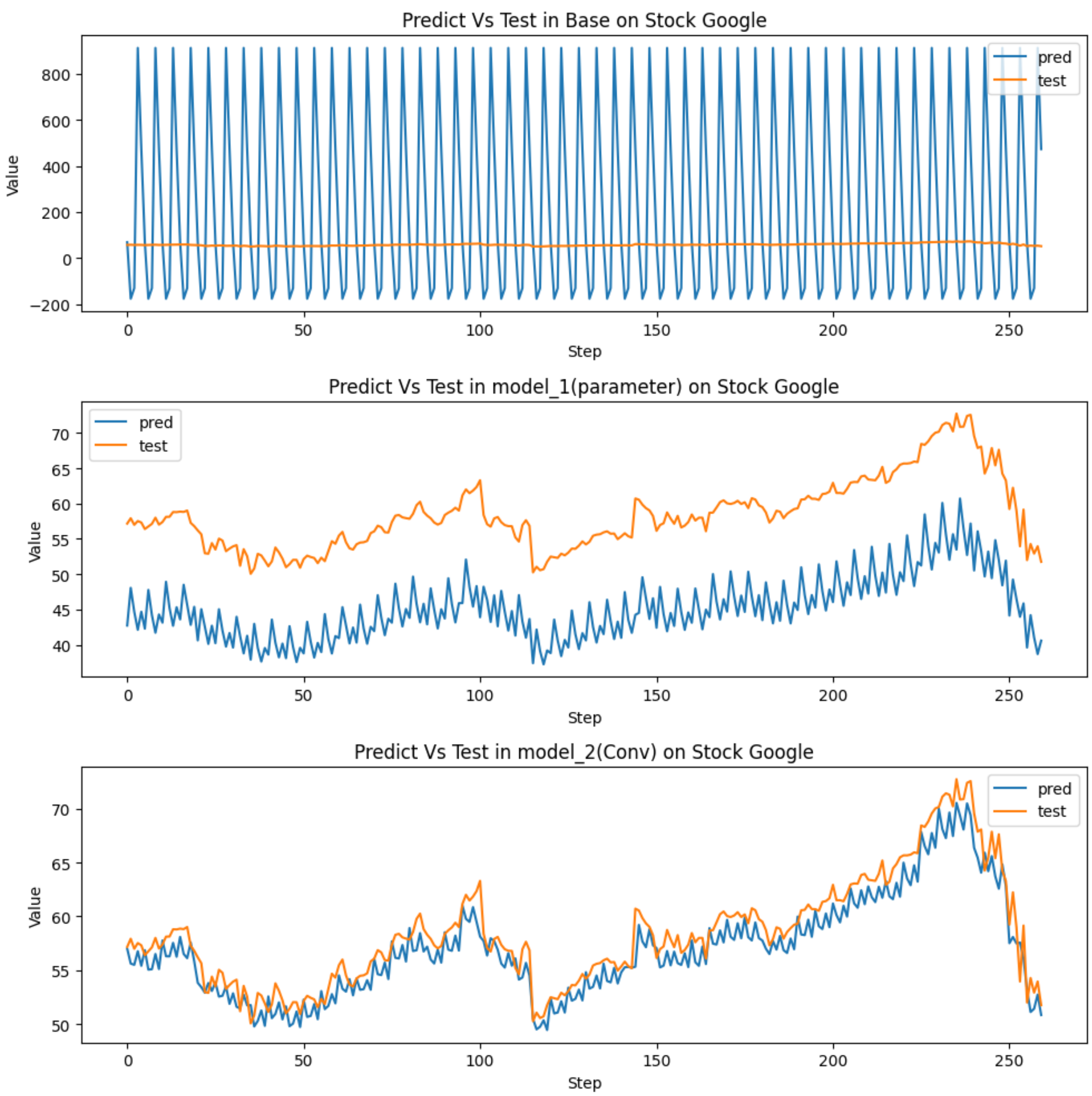
```
MAE : 0.01600866
RMSE : 0.022798454
MAPE : 2.589915
```

Dari percobaan kedua didapatkan hasil yang cukup memuaskan dibandingkan model. Pada model googe error yang diberikan 3 persen yang mana cukup kecil dibandingkan dari model kedua. Sedangkan untuk model pada intel untuk errornya mengurang dan berada di 2 persen. Walaupun tidak sebanyak seperti google

[LO 3, LO 4, 5 poin] Evaluasi performa dari arsitektur nomor 2d secara rinci dan jelaskan hasil yang kalian dapatkan. Gunakan testing set yang diberikan untuk memprediksi nilai ground truth dengan predicted result.

[ ]:
```
label = ["Base","model_1(parameter)","model_2(Conv)"]
```

[ ]:
```
google_array = [base_goo_pred_check.flatten(),model1_goo_pred_check.
 ↪flatten(),model2_goo_pred_check.flatten()]
fig, axs = plt.subplots(3, 1, figsize=(10, 10))


for i in range(3):
    ax = axs.flat[i]
    ax.plot(google_array[i],label="pred")
    ax.plot(test_win_goo_check.flatten(),label="test")
    ax.set_title(f"Predict Vs Test in {label[i]} on Stock Google")
    ax.set_xlabel("Step")
    ax.set_ylabel("Value")
    ax.legend()

plt.tight_layout()

plt.show()
```



[ ]:
```
google_model
```

[ ]:

	Model	MAE	RMSE	MAPE
0	Base	4.566708	5.971919	591.510315
1	Parameter	0.174867	0.179829	22.415958
2	Conv	0.028406	0.034612	3.606213

Dari model yang didapatkan : - Pada base model, arsitektur yang digunakan kurang cocok untuk time series data yang bertipe univariate, hal ini dikarenakan terdapat fitur embedding yang akan melakukan representasi dari fitur-fitur data, tetapi didalam univariate data hanya terdapat 1 fitur. Selain itu arsitektur dari baseline terlalu kompleks yang membuat model tidak baik.

- Pada model 1 parameter dari base diganti beberapa untuk menaikan performa model. parameter yang diganti adalah head_size, number head, feed forward dimesion, dan dropout setelah transformer block.

- Pada model 2 penabahakan convulsion layer yang mana unit akan berisi sesuai input. Dengan cara ini menaikan performa model, Hal ini dikarenakan MAPE dari model yang menurun. Dimana hal ini menjadi best model pada data google, dengan error 3 persen.

Kesimpulan yang didapatkan dengan merubah arsitektur menjadi lebih sederhana dan dengan menggunakan 2 convulsion layer pada feed forward menghasilkan error model yang cukup kecil dibandingkan model 1 dan 2, yaitu 3 persen.

[ ]:
```
intel_array = [base_int_pred_check.flatten(),model1_int_pred_check.
 ↪flatten(),model2_int_pred_check.flatten()]
fig, axs = plt.subplots(3, 1, figsize=(10, 10))


for i in range(3):
    ax = axs.flat[i]
    ax.plot(intel_array[i],label="pred")
    ax.plot(test_win_int_check.flatten(),label="test")
    ax.set_title(f"Predict Vs Test in {label[i]} on Stock Intel")
    ax.set_xlabel("Step")
    ax.set_ylabel("Value")
    ax.legend()

plt.tight_layout()

plt.show()
```

[ ]:
```
intel_model
```

[ ]:

	Model	MAE	RMSE	MAPE
0	Base	0.248880	0.282940	40.060062
1	Paramater	0.017313	0.024075	2.789596
2	Conv	0.016009	0.022798	2.589915

Dari model yang didapatkan : - Pada base model, arsitektur yang digunakan kurang cocok untuk time series data yang bertipe univariate, hal ini dikarenakan terdapat fitur embedding yang akan melakukan representasi dari fitur-fitur data, tetapi didalam univariate data hanya terdapat 1 fitur. Selain itu arsitektur dari baseline terlalu kompleks yang membuat model tidak baik.

- Pada model 1 parameter dari base diganti beberapa untuk menaikan performa model. parameter yang diganti adalah head_size, number head, feed forward dimesion, dan dropout setelah transformer block. Pada model ini error dari model sudah menurun sangat drastis menjadi 3 persen.

- Pada model 2 penabahakan convulsion layer yang mana unit akan berisi sesuai input. Dengan cara ini menaikan performa model, Hal ini dikarenakan MAPE dari model yang menurun. Dimana hal ini menjadi best model pada data intel, dengan error 2.7 persen.

Kesimpulan yang didapatkan dengan merubah arsitektur menjadi lebih sederhana dan dengan menggunakan 2 convulsion layer pada feed forward menghasilkan error model yang cukup kecil dibandingkan model 1 dan 2, yaitu 2.5 persen.