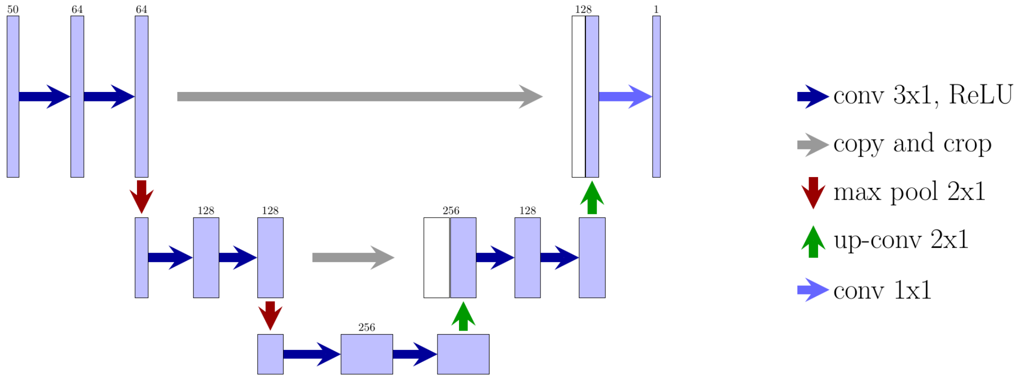
Sarah Brown - ANN - Homework 4

May 3, 2021

1 ANN Homework 4

1.1 Question 1

Build a U-net like predictor for the stock market data. - You may want to truncate the training and test input lengths to factors of 4 to avoid complication from downsampling and upsampling. - Use the following structure.



[8]:

```
#import and setup training set
import yfinance as yf
import pandas as pd
import numpy as np

def get_price(tick,start='2020-10-01',end=None):
    return yf.Ticker(tick).history(start=start,end=end)['Close']

def get_prices(tickers,start='2020-10-01',end=None):
    df=pd.DataFrame()
    for s in tickers:
        df[s]=get_price(s,start,end)
    return df

feature_stocks=['tsla','fb','twtr','amzn','nflx','gbtc','gdx','intc']␣
 ↪#excluding 'dal' and'c' to make a factor of 4
predict_stock='msft'

# training set
start_date_train='2020-10-01'
```

```
end_date_train='2020-12-31'

X_train=get_prices(feature_stocks,start=start_date_train,end=end_date_train)
y_train=get_prices([predict_stock],start=start_date_train,end=end_date_train)

# testing set
start_date_test='2021-01-01' # end date omit, default is today
X_test=get_prices(feature_stocks,start=start_date_test)
y_test=get_prices([predict_stock],start=start_date_test)

X_train=np.array(X_train)
Y_train=np.array(y_train)
X_test=np.array(X_test)
Y_test=np.array(y_test)
```

[48]:

```
import tensorflow as tf

input_length = len(X_train[0])
output_length = len(Y_train[0])

inputs = tf.keras.layers.Input((input_length, 1))
#print("input: ", inputs)

#contraction path
c1 = tf.keras.layers.Conv1D(50, (3), activation='relu', padding='same',␣
 ↪kernel_regularizer = tf.keras.regularizers.L2(l2=0.0001))(inputs)
c1 = tf.keras.layers.Conv1D(64, (3), activation='relu', padding='same',␣
 ↪kernel_regularizer = tf.keras.regularizers.L2(l2=0.0001))(c1)
c1 = tf.keras.layers.Conv1D(64, (3), activation='relu', padding='same',␣
 ↪kernel_regularizer = tf.keras.regularizers.L2(l2=0.0001))(c1)
#c1 = tf.keras.layers.Dropout((0.01))(c1)
p1 = tf.keras.layers.MaxPool1D((2))(c1)


c2 = tf.keras.layers.Conv1D(128, (3), activation='relu', padding='same',␣
 ↪kernel_regularizer = tf.keras.regularizers.L2(l2=0.0001))(p1)
c2 = tf.keras.layers.Conv1D(128, (3), activation='relu', padding='same',␣
 ↪kernel_regularizer = tf.keras.regularizers.L2(l2=0.0001))(c2)
#c2 = tf.keras.layers.Dropout((0.01))(c2)
p2 = tf.keras.layers.MaxPool1D((2))(c2)

c3 = tf.keras.layers.Conv1D(256, (3), activation='relu', padding='same',␣
 ↪kernel_regularizer = tf.keras.regularizers.L2(l2=0.0001))(p2)
#c3 = tf.keras.layers.Dropout((0.01))(c3)
u4 = tf.keras.layers.Conv1DTranspose(128, (2), strides=(2), padding='same')(c3)
```

```
u4 = tf.keras.layers.concatenate([u4, c2])
c4 = tf.keras.layers.Conv1D(128, (3), activation='relu', padding='same',␣
 ↪kernel_regularizer = tf.keras.regularizers.L2(l2=0.0001))(u4)
c4 = tf.keras.layers.Conv1D(128, (3), activation='relu', padding='same',␣
 ↪kernel_regularizer = tf.keras.regularizers.L2(l2=0.0001))(c4)
#c4 = tf.keras.layers.Dropout((0.01))(c4)

u5 = tf.keras.layers.Conv1DTranspose(64,(2), strides=(2), padding='same')(c4)
u5 = tf.keras.layers.concatenate([u5, c1])

outputs = tf.keras.layers.Conv1D(1, (1),  activation='relu')(u5)
outputs = tf.keras.layers.Dense(1, activation = 'relu')(outputs)
outputs = tf.keras.layers.Flatten()(outputs)
outputs = tf.keras.layers.Dense(1)(outputs)
print("out: ", outputs)

model = tf.keras.Model(inputs=[inputs], outputs=[outputs])
model.compile(optimizer='adam', loss='MSE')
model.summary()
```

```
out:  KerasTensor(type_spec=TensorSpec(shape=(None, 1), dtype=tf.float32,
name=None), name='dense_29/BiasAdd:0', description="created by layer
'dense_29'")
Model: "model_14"
________________________________________________________________________________
__________________
Layer (type)                    Output Shape         Param #     Connected to
================================================================================
==================
input_15 (InputLayer)           [(None, 8, 1)]       0

________________________________________________________________________________
__________________
conv1d_72 (Conv1D)              (None, 8, 50)        200         input_15[0][0]

________________________________________________________________________________
__________________
conv1d_73 (Conv1D)              (None, 8, 64)        9664        conv1d_72[0][0]

________________________________________________________________________________
__________________
conv1d_74 (Conv1D)              (None, 8, 64)        12352       conv1d_73[0][0]

________________________________________________________________________________
__________________
max_pooling1d_16 (MaxPooling1D) (None, 4, 64)        0           conv1d_74[0][0]

________________________________________________________________________________
__________________
conv1d_75 (Conv1D)              (None, 4, 128)       24704
max_pooling1d_16[0][0]
________________________________________________________________________________
```

```
------------------
conv1d_76 (Conv1D)              (None, 4, 128)       49280       conv1d_75[0][0]
--------------------------------------------------------------------------------
------------------
max_pooling1d_17 (MaxPooling1D) (None, 2, 128)       0           conv1d_76[0][0]
--------------------------------------------------------------------------------
------------------
conv1d_77 (Conv1D)              (None, 2, 256)       98560
max_pooling1d_17[0][0]
--------------------------------------------------------------------------------
------------------
conv1d_transpose_16 (Conv1DTran (None, 4, 128)       65664       conv1d_77[0][0]
--------------------------------------------------------------------------------
------------------
concatenate_16 (Concatenate)    (None, 4, 256)       0
conv1d_transpose_16[0][0]
                                                                 conv1d_76[0][0]
--------------------------------------------------------------------------------
------------------
conv1d_78 (Conv1D)              (None, 4, 128)       98432
concatenate_16[0][0]
--------------------------------------------------------------------------------
------------------
conv1d_79 (Conv1D)              (None, 4, 128)       49280       conv1d_78[0][0]
--------------------------------------------------------------------------------
------------------
conv1d_transpose_17 (Conv1DTran (None, 8, 64)        16448       conv1d_79[0][0]
--------------------------------------------------------------------------------
------------------
concatenate_17 (Concatenate)    (None, 8, 128)       0
conv1d_transpose_17[0][0]
                                                                 conv1d_74[0][0]
--------------------------------------------------------------------------------
------------------
conv1d_80 (Conv1D)              (None, 8, 1)         129
concatenate_17[0][0]
--------------------------------------------------------------------------------
------------------
dense_28 (Dense)                (None, 8, 1)         2           conv1d_80[0][0]
--------------------------------------------------------------------------------
------------------
flatten_14 (Flatten)            (None, 8)            0           dense_28[0][0]
--------------------------------------------------------------------------------
------------------
dense_29 (Dense)                (None, 1)            9
flatten_14[0][0]
================================================================================
==================
Total params: 424,724
Trainable params: 424,724
Non-trainable params: 0
________________________________________________________________________________
_________________
```

[49]:

```
#print(Y_test)
epochs=100
results = model.fit(X_train, Y_train,batch_size=64, epochs=epochs,␣
 ↪verbose=False)
print("average loss: ", np.average(results.history['loss']))
print("final loss: ", results.history['loss'][epochs-1])
```

```
average loss:  557.1130572509766
final loss:  15.257489204406738
```

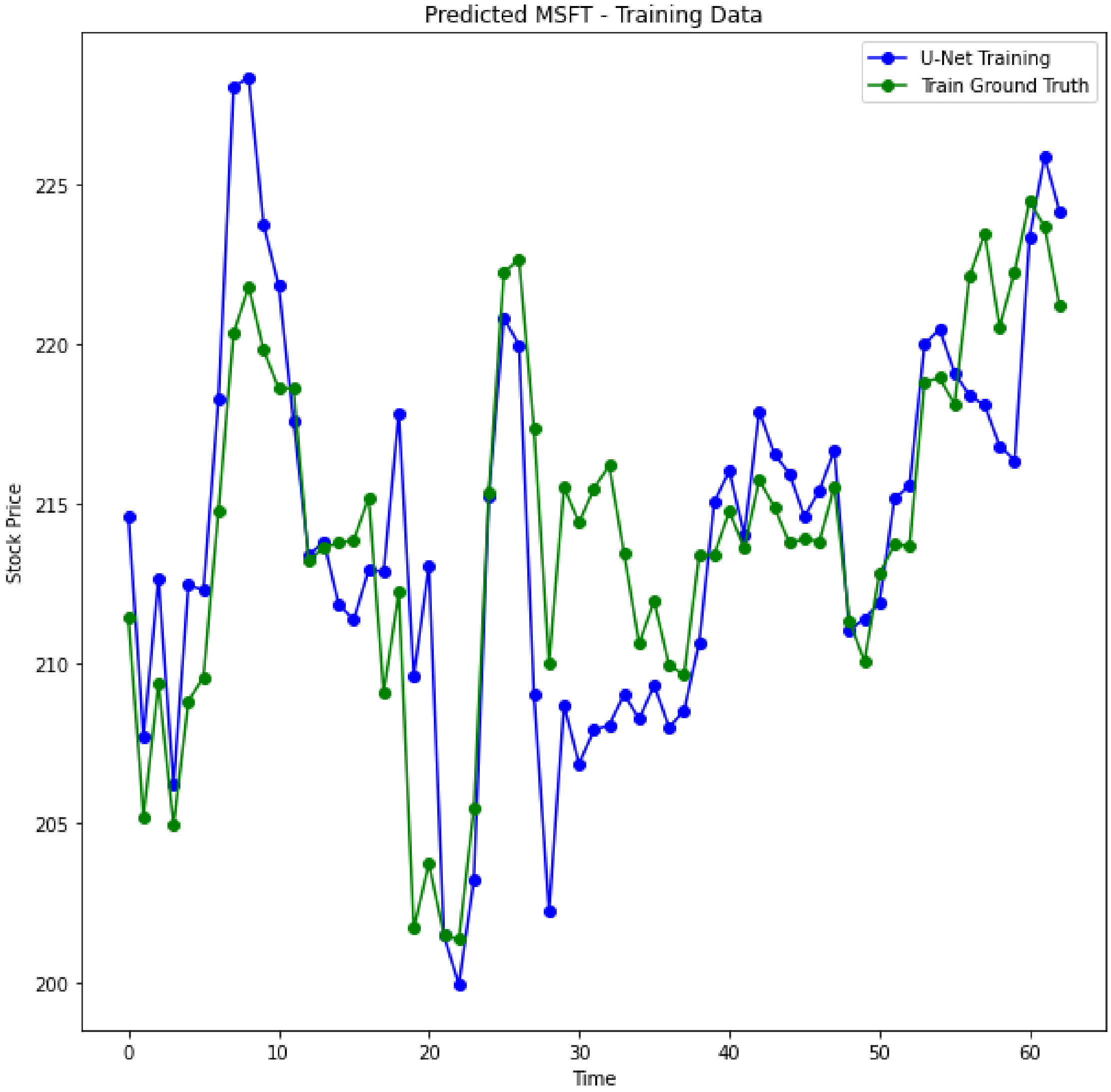
[50]:

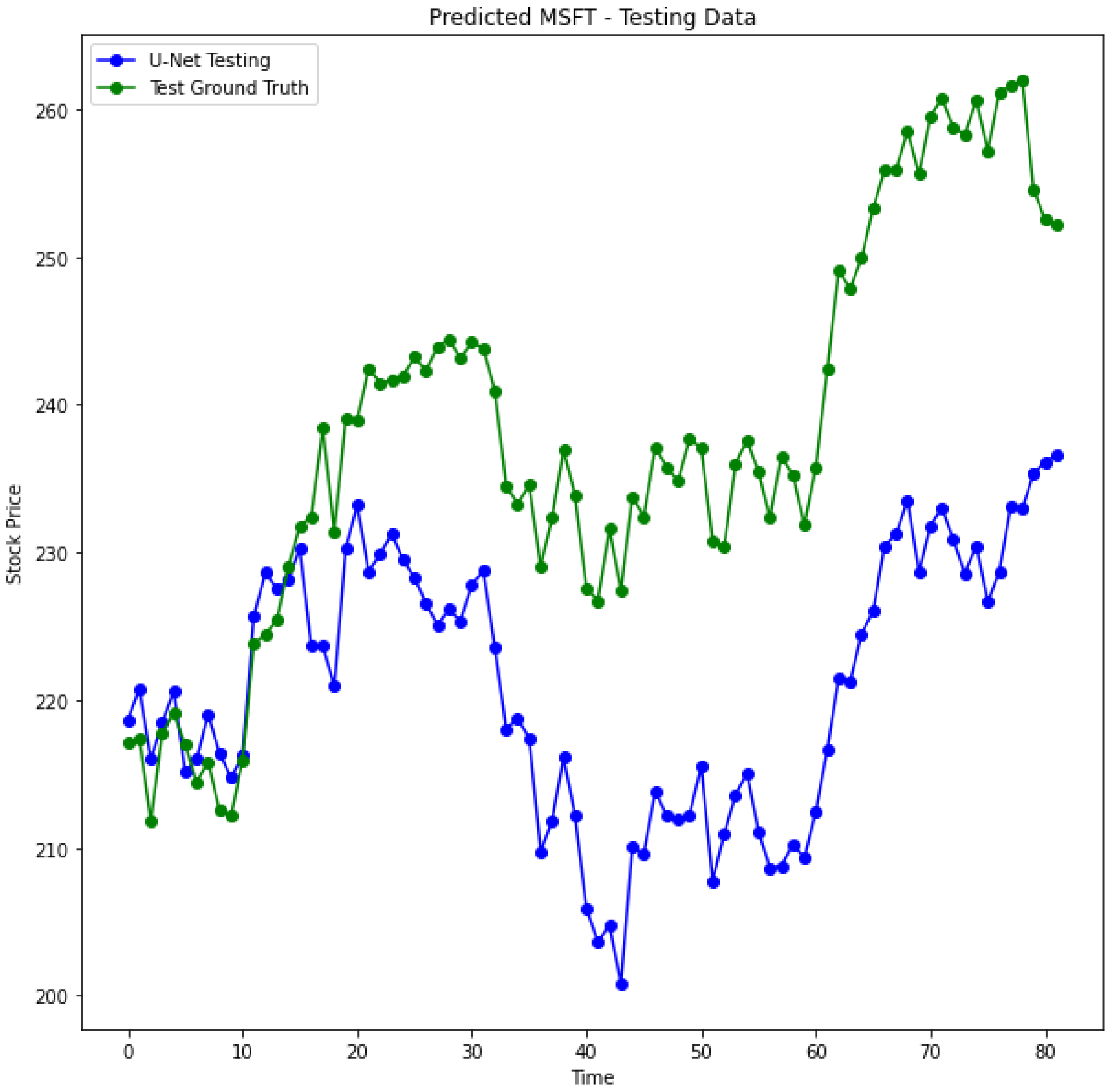
```
model_train = model.predict(X_train)
model_test = model.predict(X_test)
```

[51]:

```
#Plot the the estimated price along with the ground truth for both training and␣
 ↪test data in two separate plots

time_train = range(len(X_train))
time_test = range(len(X_test))

import matplotlib.pyplot as plt
%matplotlib inline

#training data
plt.figure(figsize=(10,10))
ax = plt.gca()
plt.xlabel("Time")
plt.ylabel("Stock Price")
plt.title("Predicted MSFT - Training Data")
ax.plot(time_train, model_train, color = "b", label="U-Net Training",␣
 ↪marker='o')
ax.plot(time_train, Y_train, color = "g", label = "Train Ground Truth",␣
 ↪marker='o')
ax.legend()

#testing data
plt.figure(figsize=(10,10))
ax = plt.gca()
plt.xlabel("Time")
plt.ylabel("Stock Price")
plt.title("Predicted MSFT - Testing Data")
ax.plot(time_test, model_test, color = "b", label="U-Net Testing", marker='o')
ax.plot(time_test, Y_test, color = "g", label = "Test Ground Truth", marker='o')
ax.legend()
```

```
[51]: <matplotlib.legend.Legend at 0x7f4eebf14cd0>
```





1.2 Question 2 (Extra Credit)

Create an LSTM predictor for the stock market data. Please use two layers of LSTM (each with 50 hidden dimensions) for your predictor. Plot your result as in question 1. For simplicity, you can take the current prices of other stocks as input and the current MSFT price as output. This blog post should be helpful.
- https://towardsdatascience.com/from-a-lstm-cell-to-a-multilayer-lstm-network-with-pytorch-2899eb5696f3

1.2.1 Tensorflow Resources

- https://www.tensorflow.org/api_docs/python/tf/keras/
- https://machinelearningmastery.com/stacked-long-short-term-memory-networks/

- https://www.machinecurve.com/index.php/2021/01/07/build-an-lstm-model-with-tensorflow-and-keras/

[52]:

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
X_train_2 = X_train.copy()
X_train_2 = scaler.fit_transform(X_train_2)

scaler2 = MinMaxScaler()
X_test_2 = X_test.copy()
X_test_2 = scaler2.fit_transform(X_test_2)
```

[81]:

```
inputs = tf.keras.layers.Input((8, 1))
lstm1 = tf.keras.layers.LSTM(50, return_sequences=True)(inputs)
lstm1 = tf.keras.layers.Dropout((0.02))(lstm1)
lstm2 = tf.keras.layers.LSTM(100, return_sequences=True)(lstm1)
lstm2 = tf.keras.layers.Dropout((0.03))(lstm2)
avg = tf.keras.layers.AveragePooling1D(pool_size=8)(lstm2)

outputs = tf.keras.layers.Dense(50, activation = 'relu')(avg)
outputs = tf.keras.layers.Dense(1, activation = 'relu')(outputs)
outputs = tf.keras.layers.Flatten()(outputs)


model_lstm = tf.keras.Model(inputs=[inputs], outputs=[outputs])
model_lstm.compile(optimizer='adam', loss='MSE')
model_lstm.summary()
```

```
Model: "model_24"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_25 (InputLayer)        [(None, 8, 1)]            0
_________________________________________________________________
lstm_30 (LSTM)               (None, 8, 50)             10400
_________________________________________________________________
dropout_40 (Dropout)         (None, 8, 50)             0
_________________________________________________________________
lstm_31 (LSTM)               (None, 8, 100)            60400
_________________________________________________________________
dropout_41 (Dropout)         (None, 8, 100)            0
_________________________________________________________________
average_pooling1d_15 (Averag (None, 1, 100)            0
_________________________________________________________________
dense_48 (Dense)             (None, 1, 50)             5050
_________________________________________________________________
dense_49 (Dense)             (None, 1, 1)              51
```

```
_________________________________________________________________
flatten_24 (Flatten)         (None, 1)                 0
=================================================================
Total params: 75,901
Trainable params: 75,901
Non-trainable params: 0
_________________________________________________________________
```

[82]:

```python
epochs2=170
results_lstm = model_lstm.fit(X_train_2, Y_train, batch_size=32,␣
 ↪epochs=epochs2, verbose=False, shuffle=True)
```

[83]:

```python
print("average loss: ", np.average(results.history['loss']))
print("final loss: ", results_lstm.history['loss'][epochs2-1])
```

```
average loss:  557.1130572509766
final loss:  22.83536148071289
```

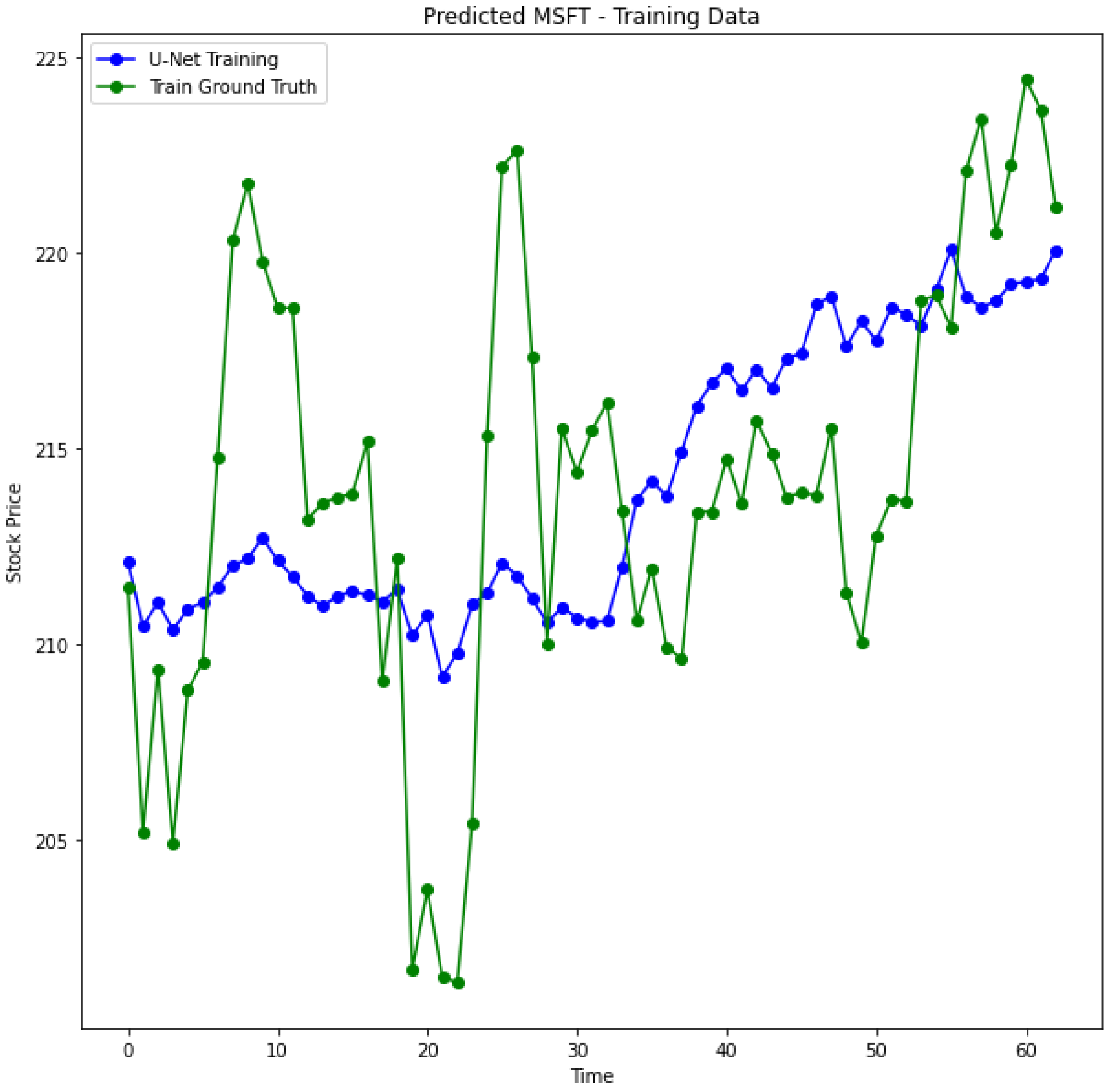
[84]:

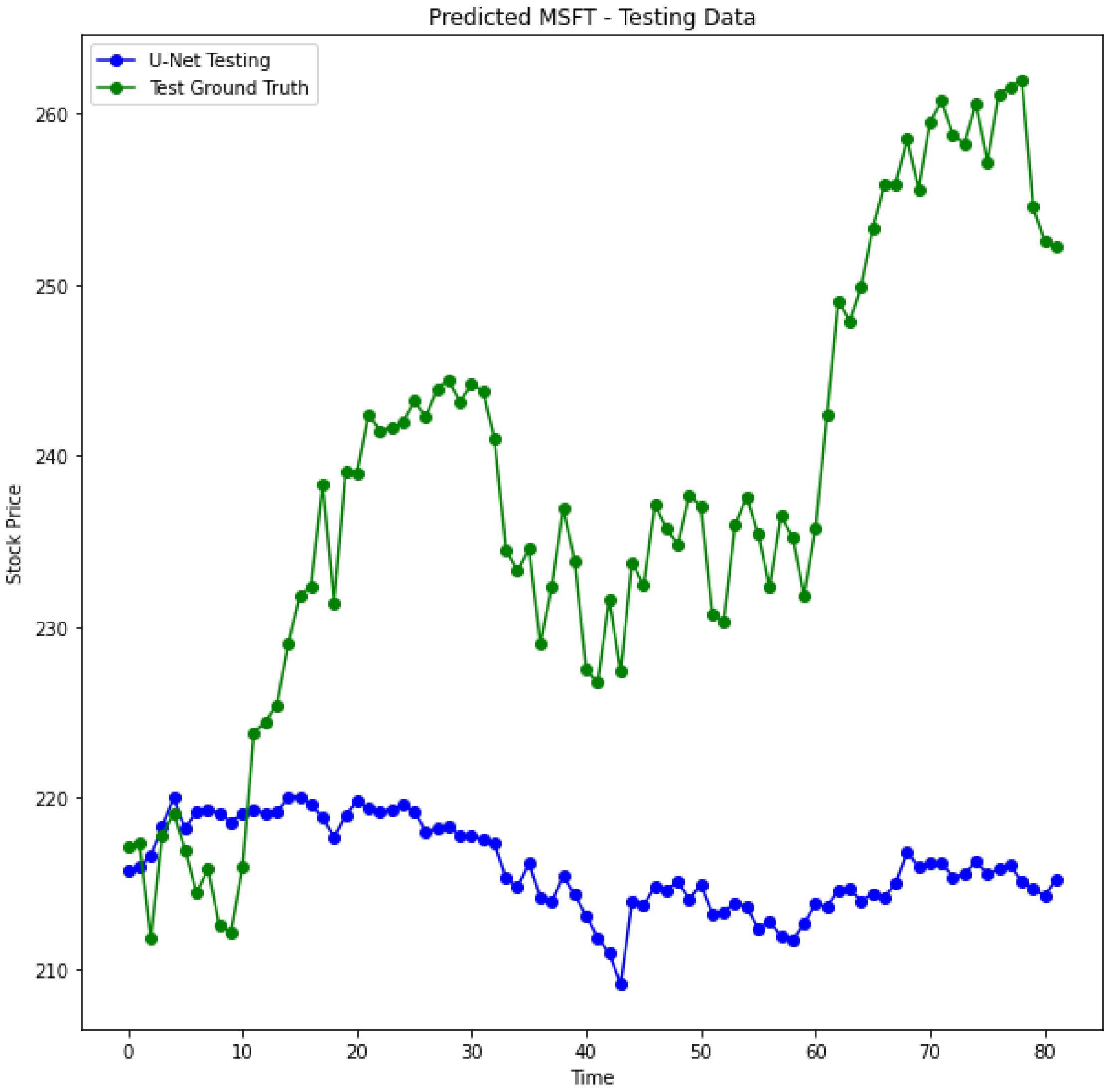
```python
model_train_lstm = model_lstm.predict(X_train_2)
model_test_lstm = model_lstm.predict(X_test_2)
```

[85]:

```python
#Plot the the estimated price along with the ground truth for both training and␣
 ↪test data in two separate plots

time_train = range(len(X_train_2))
time_test = range(len(X_test_2))

import matplotlib.pyplot as plt
%matplotlib inline

#training data
plt.figure(figsize=(10,10))
ax = plt.gca()
plt.xlabel("Time")
plt.ylabel("Stock Price")
plt.title("Predicted MSFT - Training Data")
ax.plot(time_train, model_train_lstm, color = "b", label="U-Net Training",␣
 ↪marker='o')
ax.plot(time_train, Y_train, color = "g", label = "Train Ground Truth",␣
 ↪marker='o')
ax.legend()

#testing data
plt.figure(figsize=(10,10))
ax = plt.gca()
plt.xlabel("Time")
plt.ylabel("Stock Price")
plt.title("Predicted MSFT - Testing Data")
ax.plot(time_test, model_test_lstm, color = "b", label="U-Net Testing",␣
 ↪marker='o')
ax.plot(time_test, Y_test, color = "g", label = "Test Ground Truth", marker='o')
ax.legend()
```

```
[85]: <matplotlib.legend.Legend at 0x7f4ef2916a30>
```




Predicted MSFT - Testing Data
U-Net Testing
Test Ground Truth
Stock Price
260
250
240
230
220
210
Time
0
10
20
30
40
50
60
70
80

```
[ ]:
```