Applying RNN to Time-Series Data

Taking weather forecasting data

```
!pip install tensorflow==2.15
```

```
Requirement already satisfied: tensorflow==2.15 in /usr/local/lib/python3.10/
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/di
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10
Requirement already satisfied: flatbuffers>=23.5.26 in /usr/local/lib/python3
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/lo
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/
Requirement already satisfied: ml-dtypes~=0.2.0 in /usr/local/lib/python3.10/
Requirement already satisfied: numpy<2.0.0,>=1.23.5 in /usr/local/lib/python3
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/pyt
Requirement already satisfied: wrapt<1.15,>=1.11.0 in /usr/local/lib/python3.
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/l
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.
Requirement already satisfied: tensorboard<2.16,>=2.15 in /usr/local/lib/pyth
Requirement already satisfied: tensorflow-estimator<2.16,>=2.15.0 in /usr/loc
Requirement already satisfied: keras<2.16,>=2.15.0 in /usr/local/lib/python3.
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.1
Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python
Requirement already satisfied: google-auth-oauthlib<2,>=0.5 in /usr/local/lib
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/d
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10/d
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/pytho
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.10/dis
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/pyt
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/pyt
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.1
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.1
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10
Requirement already satisfied: pyasn1<0.7.0,>=0.4.6 in /usr/local/lib/python3
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.10/d
```

```
!wget https://s3.amazonaws.com/keras-datasets/jena_climate_2009_2016.csv.zip
!unzip jena_climate_2009_2016.csv.zip
```

```
--2024-07-19 20:56:59--  https://s3.amazonaws.com/keras-datasets/jena_climate_
Resolving s3.amazonaws.com (s3.amazonaws.com)... 16.182.34.88, 16.182.104.216,
Connecting to s3.amazonaws.com (s3.amazonaws.com)|16.182.34.88|:443... connec
HTTP request sent, awaiting response... 200 OK
Length: 13565642 (13M) [application/zip]
Saving to: 'jena_climate_2009_2016.csv.zip'

jena_climate_2009_2 100%[===================>]  12.94M  18.7MB/s    in 0.7s

2024-07-19 20:57:00 (18.7 MB/s) - 'jena_climate_2009_2016.csv.zip' saved [1356

Archive:  jena_climate_2009_2016.csv.zip
  inflating: jena_climate_2009_2016.csv
  inflating: __MACOSX/._jena_climate_2009_2016.csv
```

Importing the dataset

```
import os
fname = os.path.join("jena_climate_2009_2016.csv")  # This is the file

with open(fname) as f:
    data = f.read()

lines = data.split("\n")
header = lines[0].split(",")
lines = lines[1:]
print(header)     # Printing the initial values
print(len(lines))
```

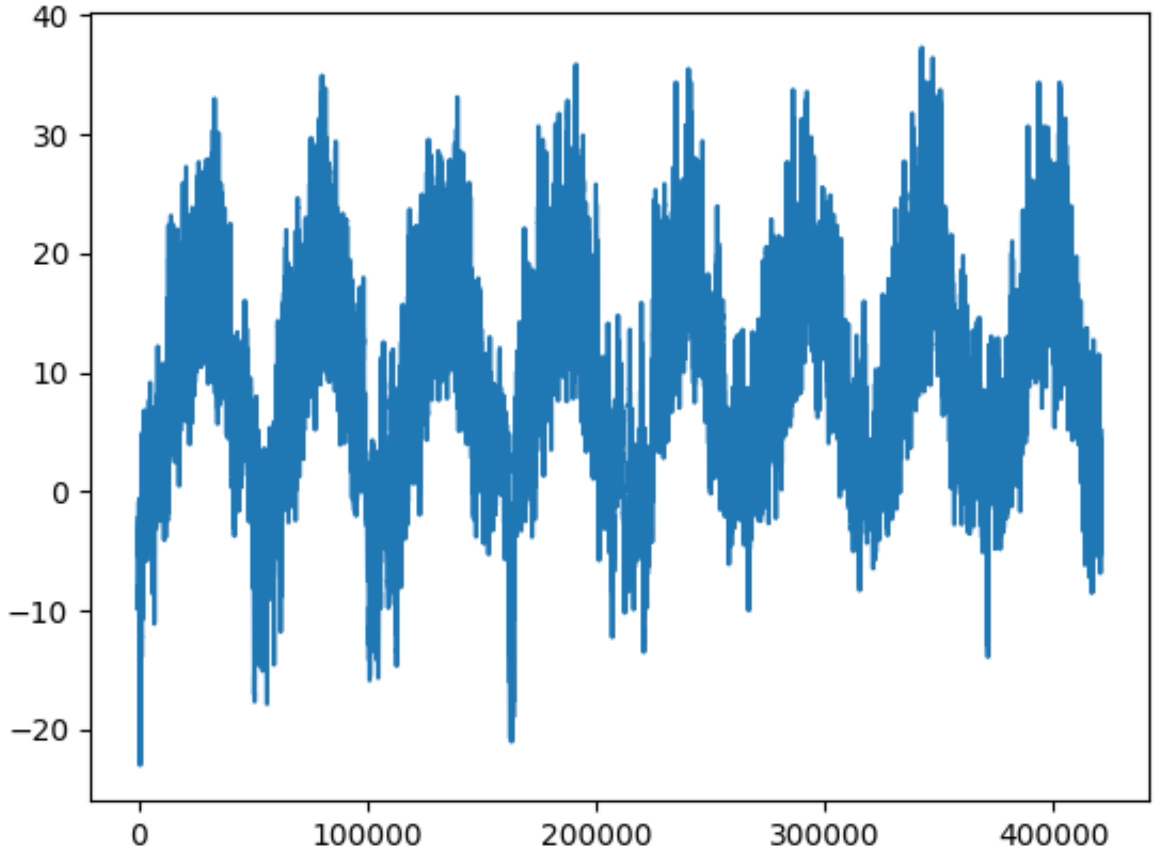
```
['"Date Time"', '"p (mbar)"', '"T (degC)"', '"Tpot (K)"', '"Tdew (degC)"', '"
420451
```

```
import numpy as np
temp = np.zeros((len(lines),))
pmry_data = np.zeros((len(lines), len(header) - 1))
for i, line in enumerate(lines):
    values = [float(x) for x in line.split(",")[1:]]
    temp[i] = values[1]
    pmry_data[i, :] = values[:]
```

Graph which shows the timeseries of temperatues as we took the weather forecasting dataset

```
from matplotlib import pyplot as plt # Using matplotlib to plot the values
plt.plot(range(len(temp)), temp)
```

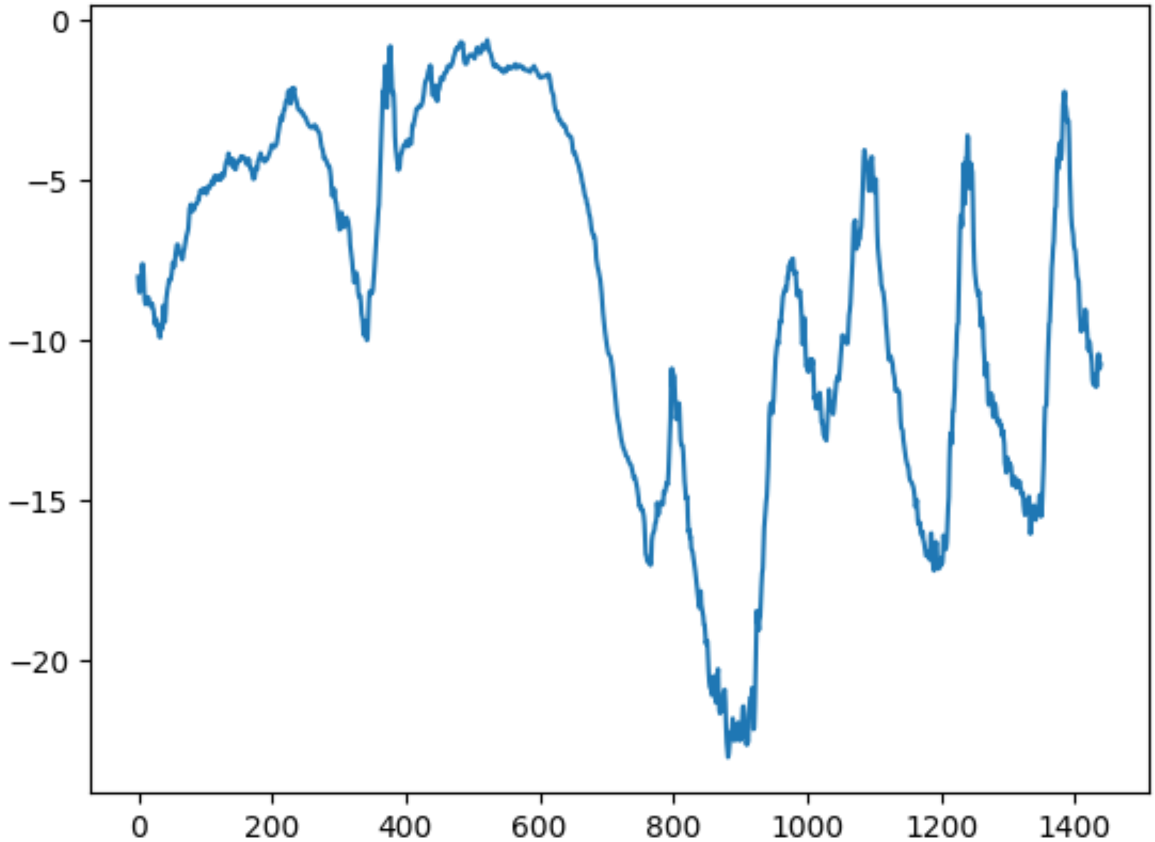
```
[<matplotlib.lines.Line2D at 0x7bb08741d030>]
```



Temperatues in °C

```
plt.plot(range(1440), temp[:1440])
```

```
[<matplotlib.lines.Line2D at 0x7bb0833dd780>]
```



Calculating the quantity of samples that each data split will require

```
num_train_samples = int(0.5 * len(pmry_data))
num_val_samples = int(0.25 * len(pmry_data))
num_test_samples = len(pmry_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)
```

```
num_train_samples: 210225
num_val_samples: 105112
num_test_samples: 105114
```

Data Standardization

Computing the mean and standard deviation on train data

```
mean = pmry_data[:num_train_samples].mean(axis=0)
pmry_data-= mean
std = pmry_data[:num_train_samples].std(axis=0)
pmry_data/= std
```

Here we use Numpy array to produce data sets in bulk for time series model training.

```
import numpy as np
from tensorflow import keras
int_sequence = np.arange(10)
dataset_1 = keras.utils.timeseries_dataset_from_array(
    data=int_sequence[:-3],           # Taking input sequence of length 3
    targets=int_sequence[3:],
    sequence_length=3,
    batch_size=2,
)

for inputs, targets in dataset_1:     # Using for loop to iterate over batches of
    for i in range(inputs.shape[0]):
        print([int(x) for x in inputs[i]], int(targets[i]))
```

```
[0, 1, 2] 3
[1, 2, 3] 4
[2, 3, 4] 5
[3, 4, 5] 6
[4, 5, 6] 7
```

Creating training, testing, and validation of datasets

```
sampling_rate = 6
sequence_length = 120
delay = sampling_rate * (sequence_length + 24 - 1)
batch_size = 256

train_dataset = keras.utils.timeseries_dataset_from_array(
    pmry_data[:-delay],
    targets=temp[delay:],
    sampling_rate=sampling_rate,
    sequence_length=sequence_length,
    shuffle=True,
    batch_size=batch_size,
    start_index=0,
    end_index=num_train_samples)

val_dataset = keras.utils.timeseries_dataset_from_array(
    pmry_data[:-delay],
    targets=temp[delay:],
    sampling_rate=sampling_rate,
    sequence_length=sequence_length,
    shuffle=True,
    batch_size=batch_size,
    start_index=num_train_samples,
    end_index=num_train_samples + num_val_samples)

test_dataset = keras.utils.timeseries_dataset_from_array(
    pmry_data[:-delay],
    targets=temp[delay:],
    sampling_rate=sampling_rate,
    sequence_length=sequence_length,
    shuffle=True,
    batch_size=batch_size,
    start_index=num_train_samples + num_val_samples)
```

Shape of the data chunks

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

```
samples shape: (256, 120, 14)
targets shape: (256,)
```

1st Model:

A common-sense, non-machine-learning baseline

Baseline MAE caluculation

```
def evaluate_naive_method(dataset): # using evaluate_naive_method to calculate MA
    total_abs_err = 0.
    samples_seen = 0
    for samples, targets in dataset:
        preds = samples[:, -1, 1] * std[1] + mean[1]
        total_abs_err += np.sum(np.abs(preds - targets))
        samples_seen += samples.shape[0]
    return total_abs_err / samples_seen

print(f"Validation MAE: {evaluate_naive_method(val_dataset):.2f}") # Displaying t
print(f"Test MAE: {evaluate_naive_method(test_dataset):.2f}") #  # Displaying the
```

```
Validation MAE: 2.44
Test MAE: 2.62
```

2nd Model:

Basic machine-learning model

Simple neural network model for forecasting using Keras.

```
from tensorflow import keras
from tensorflow.keras import layers

inputs = keras.Input(shape=(sequence_length, pmry_data.shape[-1])) # Defining the
x = layers.Flatten()(inputs)
x = layers.Dense(16, activation="relu")(x)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)
# Specifying a callback list to be utilized in training.
callbacks = [
    keras.callbacks.ModelCheckpoint("jena_dense.x",
                                    save_best_only=True)
]
model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(train_dataset,
                    epochs=5,
                    validation_data=val_dataset,
                    callbacks=callbacks)

model = keras.models.load_model("jena_dense.x")
print(f"Test MAE: {model.evaluate(test_dataset)[1]:.2f}") # Printing the MAE of t
```

```
Epoch 1/5
819/819 [==============================] - 49s 57ms/step - loss: 12.6822 - mae
Epoch 2/5
819/819 [==============================] - 39s 48ms/step - loss: 9.2481 - mae
Epoch 3/5
819/819 [==============================] - 36s 44ms/step - loss: 8.4552 - mae
Epoch 4/5
819/819 [==============================] - 35s 42ms/step - loss: 7.9575 - mae
Epoch 5/5
819/819 [==============================] - 36s 44ms/step - loss: 7.6156 - mae
405/405 [==============================] - 12s 27ms/step - loss: 11.6896 - mae
Test MAE: 2.69
```

The above model takes as input a sequence of data points and outputs a single value.

```
from tensorflow import keras
from tensorflow.keras import layers

inputs = keras.Input(shape=(sequence_length, pmry_data.shape[-1])) # Defining the
x = layers.Flatten()(inputs)
x = layers.Dense(8, activation="relu")(x)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)
# Specifying a callback list to be utilized in training.
callbacks = [
    keras.callbacks.ModelCheckpoint("jena_dense.x",
                                    save_best_only=True)
]
model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(train_dataset,
                    epochs=5,
                    validation_data=val_dataset,
                    callbacks=callbacks)

model = keras.models.load_model("jena_dense.x")
print(f"Test MAE: {model.evaluate(test_dataset)[1]:.2f}") # Printing the MAE of t
```

```
Epoch 1/5
819/819 [==============================] - 47s 56ms/step - loss: 13.1631 - mae
Epoch 2/5
819/819 [==============================] - 40s 49ms/step - loss: 9.6131 - mae:
Epoch 3/5
819/819 [==============================] - 37s 45ms/step - loss: 9.1471 - mae:
Epoch 4/5
819/819 [==============================] - 35s 43ms/step - loss: 8.8006 - mae:
Epoch 5/5
819/819 [==============================] - 46s 56ms/step - loss: 8.5644 - mae:
405/405 [==============================] - 14s 33ms/step - loss: 11.3995 - mae
Test MAE: 2.66
```

```
from tensorflow import keras
from tensorflow.keras import layers

inputs = keras.Input(shape=(sequence_length, pmry_data.shape[-1])) # Defining the
x = layers.Flatten()(inputs)
x = layers.Dense(32, activation="relu")(x)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)
# Specifying a callback list to be utilized in training.
callbacks = [
    keras.callbacks.ModelCheckpoint("jena_dense.x",
                                    save_best_only=True)
]
model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(train_dataset,
                    epochs=5,
                    validation_data=val_dataset,
                    callbacks=callbacks)

model = keras.models.load_model("jena_dense.x")
print(f"Test MAE: {model.evaluate(test_dataset)[1]:.2f}") # Printing the MAE of t
```

```
Epoch 1/5
819/819 [==============================] - 37s 44ms/step - loss: 11.7960 - mae
Epoch 2/5
819/819 [==============================] - 45s 54ms/step - loss: 8.6042 - mae:
Epoch 3/5
819/819 [==============================] - 35s 43ms/step - loss: 7.7225 - mae:
Epoch 4/5
819/819 [==============================] - 36s 44ms/step - loss: 7.1612 - mae:
Epoch 5/5
819/819 [==============================] - 45s 55ms/step - loss: 6.7127 - mae:
405/405 [==============================] - 13s 31ms/step - loss: 11.4643 - mae
Test MAE: 2.68
```

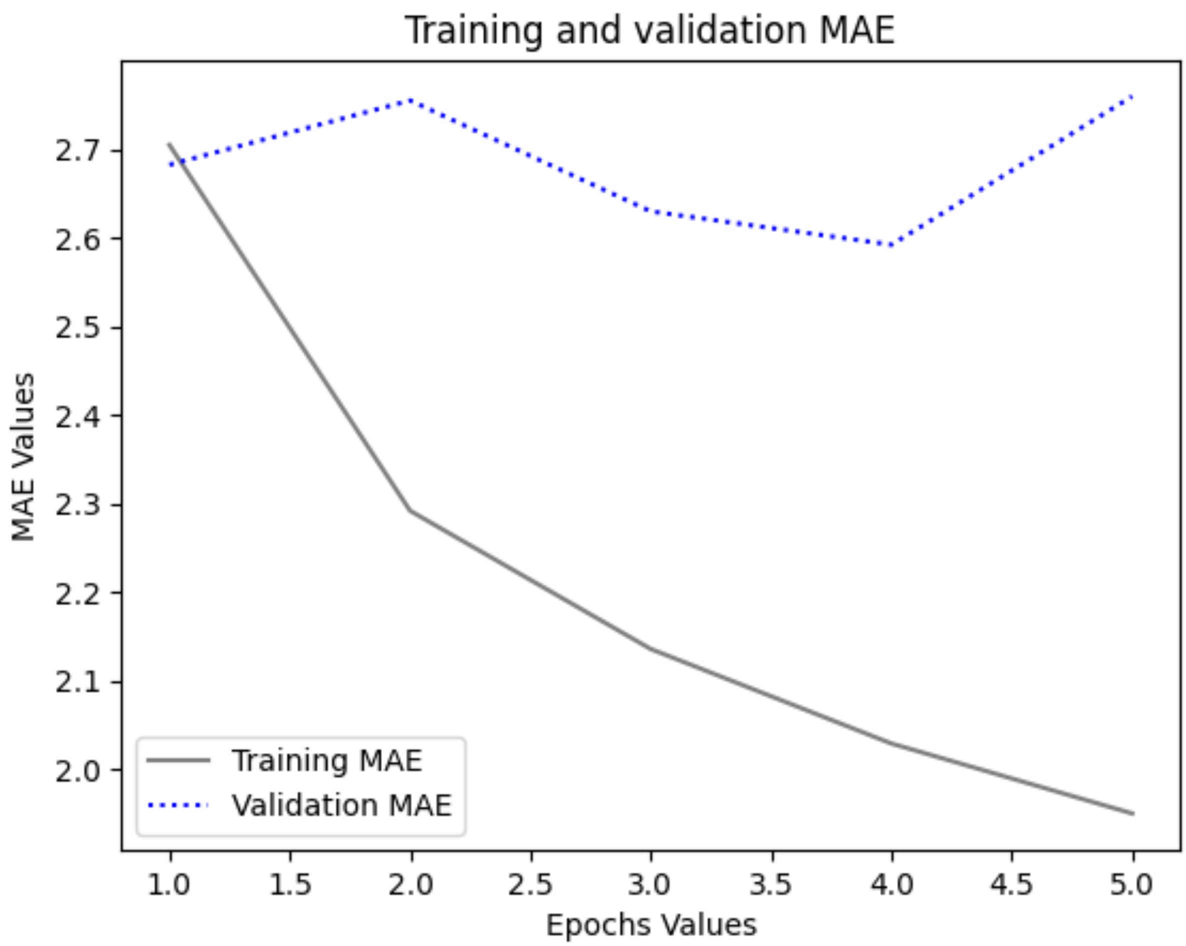
```
from tensorflow import keras
from tensorflow.keras import layers

inputs = keras.Input(shape=(sequence_length, pmry_data.shape[-1])) # Defining the
x = layers.Flatten()(inputs)
x = layers.Dense(64, activation="relu")(x)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)
# Specifying a callback list to be utilized in training.
callbacks = [
    keras.callbacks.ModelCheckpoint("jena_dense.x",
                                    save_best_only=True)
]
model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(train_dataset,
                    epochs=5,
                    validation_data=val_dataset,
                    callbacks=callbacks)

model = keras.models.load_model("jena_dense.x")
print(f"Test MAE: {model.evaluate(test_dataset)[1]:.2f}") # Printing the MAE of t
```

```
Epoch 1/5
819/819 [==============================] - 34s 41ms/step - loss: 12.1653 - mae
Epoch 2/5
819/819 [==============================] - 34s 42ms/step - loss: 8.4944 - mae
Epoch 3/5
819/819 [==============================] - 36s 44ms/step - loss: 7.3859 - mae
Epoch 4/5
819/819 [==============================] - 47s 57ms/step - loss: 6.6551 - mae
Epoch 5/5
819/819 [==============================] - 45s 55ms/step - loss: 6.1415 - mae
405/405 [==============================] - 12s 29ms/step - loss: 11.8821 - mae
Test MAE: 2.72
```

Tried various dense units of 8, 32 and 64

Graph of Training and Validation MAE Values

```
# matplotlib.pyplot for creating plots
import matplotlib.pyplot as plt
loss = history.history["mae"]
val_loss = history.history["val_mae"]
epochs = range(1, len(loss) + 1)
plt.figure()
plt.plot(epochs, loss, color="grey", linestyle="solid", label="Training MAE")
plt.plot(epochs, val_loss, color="blue", linestyle="dotted", label="Validation MA
plt.title("Training and validation MAE")
plt.xlabel("Epochs Values")
plt.ylabel("MAE Values")
plt.legend()
plt.show()
```



3rd Model:

1D convolutional model

```
from tensorflow import keras
from tensorflow.keras import layers
inputs = keras.Input(shape=(sequence_length, pmry_data.shape[-1]))
convol_x = layers.Conv1D(8, 24, activation="relu")(inputs)     # 1D conventional
convol_x = layers.MaxPooling1D(2)(convol_x)                    # Max pooling Laye
convol_x = layers.Conv1D(8, 12, activation="relu")(convol_x)   # 1D conventional
convol_x = layers.MaxPooling1D(2)(convol_x)                    # Max pooling Laye
convol_x = layers.Conv1D(8, 6, activation="relu")(convol_x)    # 1D conventional
convol_x = layers.GlobalAveragePooling1D()(convol_x)
outputs = layers.Dense(1)(convol_x)
model = keras.Model(inputs, outputs)
# Specifying a callback list to be utilized in training.
callbacks = [
    keras.callbacks.ModelCheckpoint("jena_conv.convol_x",
                                    save_best_only=True)
]
model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(train_dataset,
                    epochs=5,
                    validation_data=val_dataset,
                    callbacks=callbacks)

model = keras.models.load_model("jena_conv.convol_x")
print(f"Test MAE: {model.evaluate(test_dataset)[1]:.2f}") # Printing the MAE of t
```

```
Epoch 1/5
819/819 [==============================] - 42s 48ms/step - loss: 25.6978 - mae
Epoch 2/5
819/819 [==============================] - 50s 60ms/step - loss: 15.9845 - mae
Epoch 3/5
819/819 [==============================] - 47s 58ms/step - loss: 14.4231 - mae
Epoch 4/5
819/819 [==============================] - 37s 45ms/step - loss: 13.6848 - mae
Epoch 5/5
819/819 [==============================] - 48s 58ms/step - loss: 13.1804 - mae
405/405 [==============================] - 12s 29ms/step - loss: 16.0210 - mae
Test MAE: 3.18
```

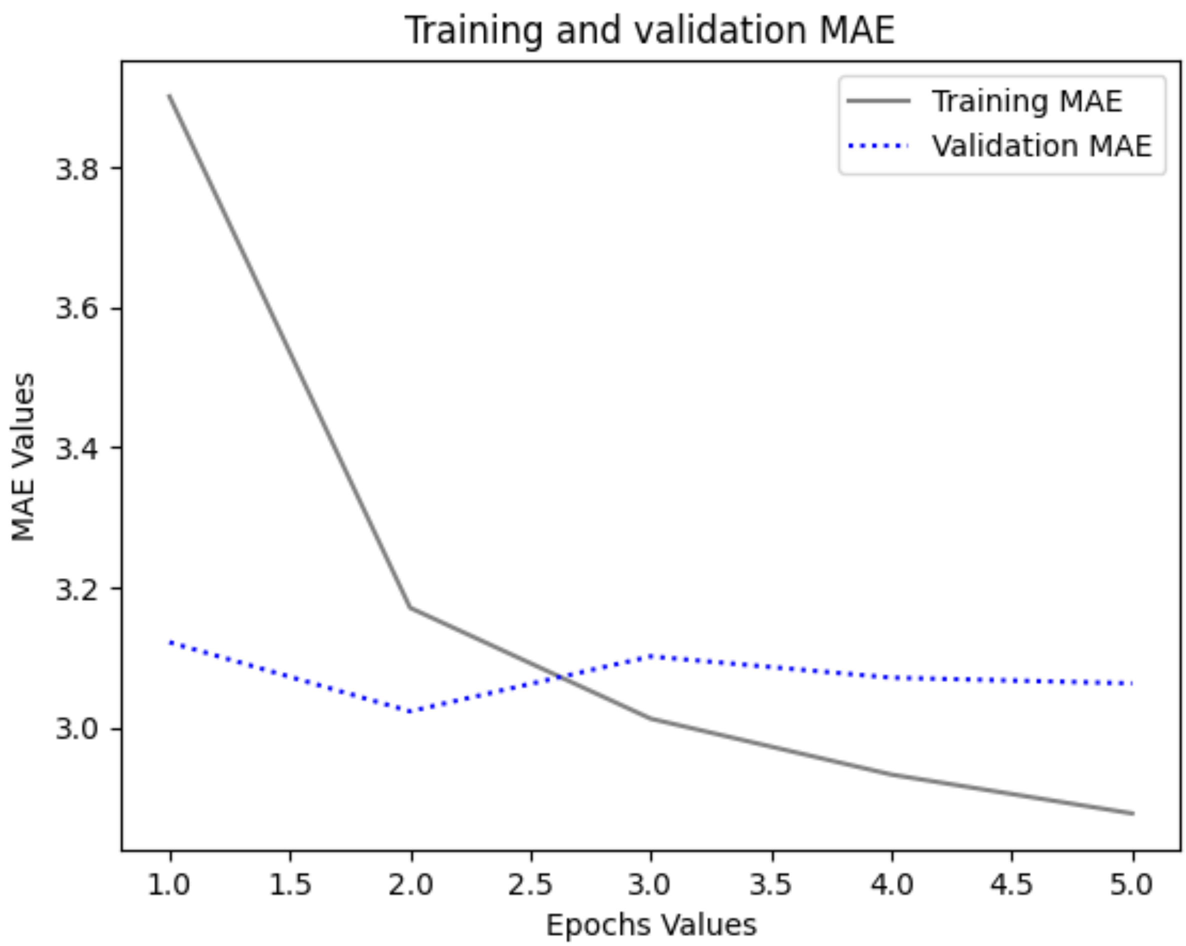
We received

Validation MAE: 3.2278

Test MAE : 3.20

Graph of Training and Validation MAE Values

```
import matplotlib.pyplot as plt
loss = history.history["mae"]
val_loss = history.history["val_mae"]
epochs = range(1, len(loss) + 1)
plt.figure()
plt.plot(epochs, loss, color="grey", linestyle="solid", label="Training MAE")
plt.plot(epochs, val_loss, color="blue", linestyle="dotted", label="Validation MA
plt.title("Training and validation MAE")
plt.xlabel("Epochs Values")
plt.ylabel("MAE Values")
plt.legend()
plt.show()
```



The first recurrent baseline

4th Model:

Simple LSTM-based model

```
inputs = keras.Input(shape=(sequence_length, pmry_data.shape[-1])) # Defining the
x = layers.LSTM(16)(inputs)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)
# Specifying a callback list to be utilized in training.
callbacks = [
    keras.callbacks.ModelCheckpoint("jena_lstm.x",
                                    save_best_only=True)
]
model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(train_dataset,
                    epochs=5,
                    validation_data=val_dataset,
                    callbacks=callbacks)

model = keras.models.load_model("jena_lstm.x")
print(f"Test MAE: {model.evaluate(test_dataset)[1]:.2f}")
```

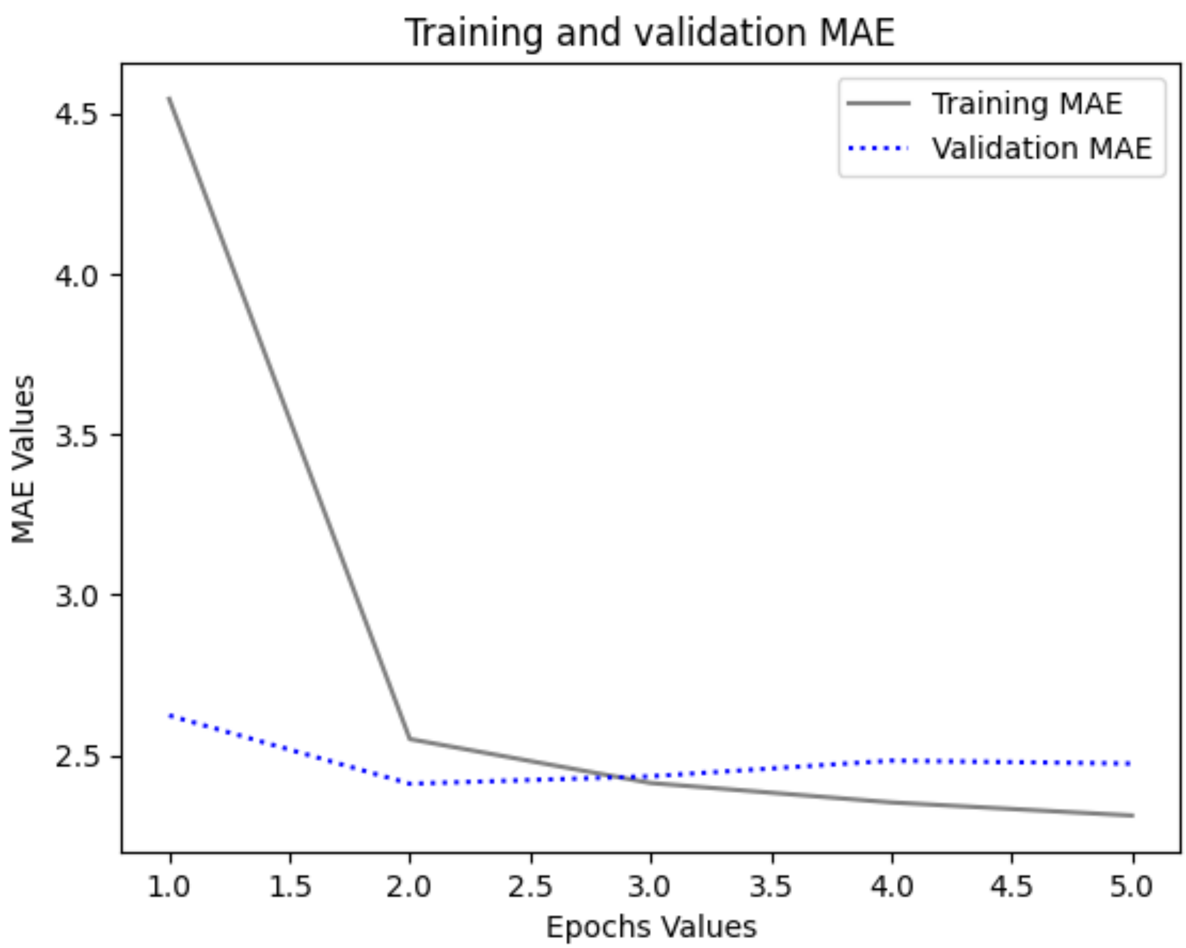
```
Epoch 1/5
819/819 [==============================] - 44s 51ms/step - loss: 39.6707 - mae
Epoch 2/5
819/819 [==============================] - 51s 62ms/step - loss: 10.7152 - mae
Epoch 3/5
819/819 [==============================] - 42s 51ms/step - loss: 9.5902 - mae:
Epoch 4/5
819/819 [==============================] - 38s 46ms/step - loss: 9.1430 - mae:
Epoch 5/5
819/819 [==============================] - 39s 48ms/step - loss: 8.8514 - mae:
405/405 [==============================] - 13s 31ms/step - loss: 11.0274 - mae
Test MAE: 2.58
```

We received

Validation MAE: 2.3790

Test MAE : 2.55

```
import matplotlib.pyplot as plt
loss = history.history["mae"]
val_loss = history.history["val_mae"]
epochs = range(1, len(loss) + 1)
plt.figure()
plt.plot(epochs, loss, color="grey", linestyle="solid", label="Training MAE")
plt.plot(epochs, val_loss, color="blue", linestyle="dotted", label="Validation MA
plt.title("Training and validation MAE")
plt.xlabel("Epochs Values")
plt.ylabel("MAE Values")
plt.legend()
plt.show()
```



5th Model:

Recurrent neural networks

Apllying Numpy to a simple RNN

```
import numpy as np
timesteps = 100
input_features = 32
output_features = 64
inputs = np.random.random((timesteps, input_features))
state_t = np.zeros((output_features,))
W = np.random.random((output_features, input_features))
U = np.random.random((output_features, output_features))
b = np.random.random((output_features,))
successive_outputs = []
for input_t in inputs:
    output_t = np.tanh(np.dot(W, input_t) + np.dot(U, state_t) + b)
    successive_outputs.append(output_t)
    state_t = output_t
final_output_sequence = np.stack(successive_outputs, axis=0)
```

```
num_features = 14  # Recurring network processing sequences of length
inputs = keras.Input(shape=(None, num_features))
outputs = layers.SimpleRNN(16)(inputs)
```

RNN layer returning output shape

```
num_features = 14
steps = 120
inputs = keras.Input(shape=(steps, num_features))
outputs = layers.SimpleRNN(16, return_sequences=False)(inputs)
print(outputs.shape)
```

```
(None, 16)
```

```
num_features = 14  # Full output sequence retrieval from an RNN layer
steps = 120
inputs = keras.Input(shape=(steps, num_features))
outputs = layers.SimpleRNN(16, return_sequences=True)(inputs)
print(outputs.shape)
```

```
(None, 120, 16)
```

Stacking Of Recurring Neural Network

```
inputs = keras.Input(shape=(steps, num_features))
x = layers.SimpleRNN(16, return_sequences=True)(inputs)
x = layers.SimpleRNN(16, return_sequences=True)(x)
outputs = layers.SimpleRNN(16)(x)
```

6th Model:

Recurring Neural Network(LSTM Layers)

Using recurrent dropout

Computing the dropout-regularized LSTM

```
inputs = keras.Input(shape=(sequence_length, pmry_data.shape[-1]))          #
lstm_x = layers.LSTM(16, recurrent_dropout=0.25)(inputs)
lstm_x = layers.Dropout(0.5)(lstm_x)  # Using droput function
outputs = layers.Dense(1)(lstm_x)
model = keras.Model(inputs, outputs)
# Specifying a callback list to be utilized in training.
callbacks = [
    keras.callbacks.ModelCheckpoint("jena_lstm_dropout.lstm_x",
                                    save_best_only=True)
]
model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(train_dataset,
                    epochs=5,
                    validation_data=val_dataset,
                    callbacks=callbacks)
model = keras.models.load_model("jena_lstm_dropout.lstm_x")
print(f"Test MAE: {model.evaluate(test_dataset)[1]:.2f}") # Printing the Test sam
```

```
WARNING:tensorflow:Layer lstm_2 will not use cuDNN kernels since it doesn't me
Epoch 1/5
819/819 [==============================] - 310s 371ms/step - loss: 49.2305 - r
Epoch 2/5
819/819 [==============================] - 285s 348ms/step - loss: 20.2056 - r
Epoch 3/5
819/819 [==============================] - 287s 350ms/step - loss: 18.1546 - r
Epoch 4/5
819/819 [==============================] - 288s 351ms/step - loss: 17.1655 - r
Epoch 5/5
819/819 [==============================] - 287s 350ms/step - loss: 16.4676 - r
WARNING:tensorflow:Layer lstm_2 will not use cuDNN kernels since it doesn't me
405/405 [==============================] - 27s 65ms/step - loss: 11.0051 - mae
Test MAE: 2.61
```

We received

Validation MAE: 2.4221

Test MAE : 2.62

Graph of dropout-regularized LSTM displaying the validation and training MAE

```
import matplotlib.pyplot as plt
loss = history.history["mae"]
val_loss = history.history["val_mae"]
epochs = range(1, len(loss) + 1)
plt.figure()
plt.plot(epochs, loss, color="grey", linestyle="solid", label="Training MAE")
plt.plot(epochs, val_loss, color="blue", linestyle="dotted", label="Validation MA
plt.title("Training and validation MAE")
plt.xlabel("Epochs Values")
plt.ylabel("MAE Values")
plt.legend()
plt.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-4-7e710dbb7431> in <cell line: 2>()
      1 import matplotlib.pyplot as plt
----> 2 loss = history.history["mae"]
      3 val_loss = history.history["val_mae"]
      4 epochs = range(1, len(loss) + 1)
      5 plt.figure()

NameError: name 'history' is not defined
```

```
inputs = keras.Input(shape=(sequence_length, num_features))
x = layers.LSTM(16, recurrent_dropout=0.2, unroll=True)(inputs) # Using the LSTM
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-2-3d1be80d7f1c> in <cell line: 1>()
----> 1 inputs = keras.Input(shape=(sequence_length, num_features))
      2 x = layers.LSTM(16, recurrent_dropout=0.2, unroll=True)(inputs) #
Using the LSTM

NameError: name 'keras' is not defined
```

7th Model:

Stacked setup of recurrent layers

Computing dropout-regularized, stacked GRU model

```
inputs = keras.Input(shape=(sequence_length, pmry_data.shape[-1]))      # Definin
x = layers.GRU(32, recurrent_dropout=0.5, return_sequences=True)(inputs)
x = layers.GRU(32, recurrent_dropout=0.5)(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)
# Specifying a callback list to be utilized in training.
callbacks = [
    keras.callbacks.ModelCheckpoint("jena_stacked_gru_dropout.x",
                                    save_best_only=True)
]
model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(train_dataset,
                    epochs=5,
                    validation_data=val_dataset,
                    callbacks=callbacks)
model = keras.models.load_model("jena_stacked_gru_dropout.x")
print(f"Test MAE: {model.evaluate(test_dataset)[1]:.2f}")  # Printing the MAE for
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-ac92c8a45d71> in <cell line: 1>()
----> 1 inputs = keras.Input(shape=(sequence_length, pmry_data.shape[-1]))
# Defining the input layer of the model
      2 x = layers.GRU(32, recurrent_dropout=0.5, return_sequences=True)
(inputs)
      3 x = layers.GRU(32, recurrent_dropout=0.5)(x)
      4 x = layers.Dropout(0.5)(x)
      5 outputs = layers.Dense(1)(x)

NameError: name 'keras' is not defined
```

We received

Validation MAE: 2.3444

Test MAE : 2.46

8th Model:

Bidirectional RNN

Computing the Bidirectional LSTM

```
inputs = keras.Input(shape=(sequence_length, pmry_data.shape[-1]))
x = layers.Bidirectional(layers.LSTM(16))(inputs)                    # Using the
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)

model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(train_dataset,
                    epochs=5,
                    validation_data=val_dataset)
test_mae = model.evaluate(test_dataset)[1]
print(f"Test MAE: {test_mae:.2f}")    # Printing The Testing dataset MAE
```

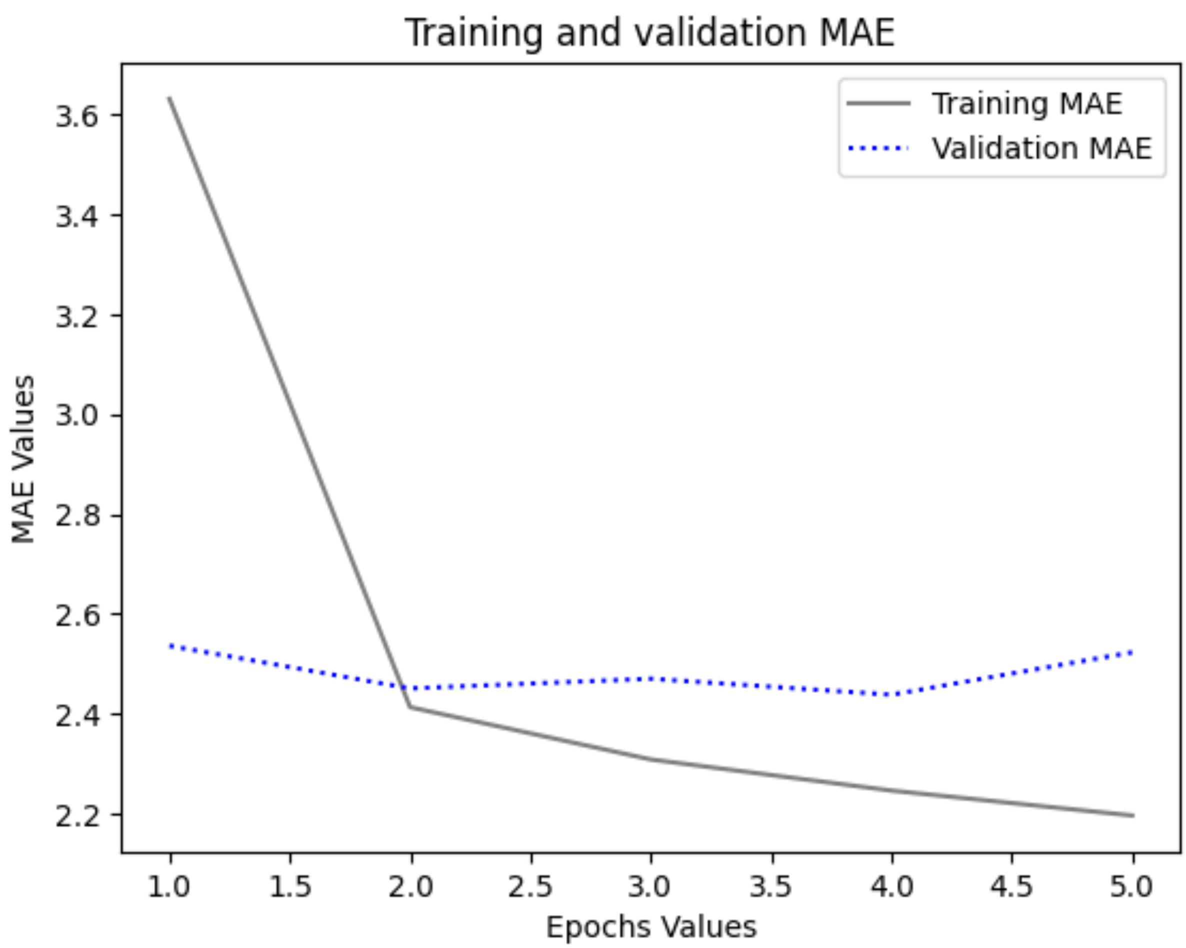
```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-3-2f092585a653> in <cell line: 1>()
----> 1 inputs = keras.Input(shape=(sequence_length, pmry_data.shape[-1]))
      2 x = layers.Bidirectional(layers.LSTM(16))(inputs)
# Using the Bidirectional function for the model
      3 outputs = layers.Dense(1)(x)
      4 model = keras.Model(inputs, outputs)
      5

NameError: name 'keras' is not defined
```

We received

Validation MAE: 2.5226

Test MAE : 2.60

```
import matplotlib.pyplot as plt
loss = history.history["mae"]
val_loss = history.history["val_mae"]
epochs = range(1, len(loss) + 1)
plt.figure()
plt.plot(epochs, loss, color="grey", linestyle="solid", label="Training MAE")
plt.plot(epochs, val_loss, color="blue", linestyle="dotted", label="Validation MA
plt.title("Training and validation MAE")
plt.xlabel("Epochs Values")
plt.ylabel("MAE Values")
plt.legend()
plt.show()
```



9th Model:

Combination Of 1D convent and dropout-regularized LSTM

```
mix_1d_RNN = layers.concatenate([convol_x, lstm_x]) # Using 1D convent and RNN
outputs = layers.Dense(1)(mix_1d_RNN)
model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(train_dataset, epochs=5, validation_data=val_dataset)
test_mae = model.evaluate(test_dataset)[1]
print(f"Test MAE: {test_mae:.2f}") # Printing the Testing MAE
```

```
Epoch 1/5
819/819 [==============================] - 289s 349ms/step - loss: 16.3800 - m
Epoch 2/5
819/819 [==============================] - 284s 347ms/step - loss: 16.1007 - m
Epoch 3/5
819/819 [==============================] - 284s 346ms/step - loss: 15.7931 - m
Epoch 4/5
819/819 [==============================] - 291s 355ms/step - loss: 15.5608 - m
Epoch 5/5
819/819 [==============================] - 296s 361ms/step - loss: 15.3729 - m
405/405 [==============================] - 29s 72ms/step - loss: 10.9585 - mae
Test MAE: 2.60
```

We received

Validation MAE: 2.4410

Test MAE : 2.60

Graph of Training and Validation MAE of the combination of 1D Convent and RNN

```
import matplotlib.pyplot as plt
loss = history.history["mae"]
val_loss = history.history["val_mae"]
epochs = range(1, len(loss) + 1)
plt.figure()
plt.plot(epochs, loss, color="grey", linestyle="solid", label="Training MAE")
plt.plot(epochs, val_loss, color="blue", linestyle="dotted", label="Validation MA
plt.title("Training and validation MAE")
plt.xlabel("Epochs Values")
plt.ylabel("MAE Values")
plt.legend()
plt.show()
```