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

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

```
--2024-07-20 00:23:38--  https://s3.amazonaws.com/keras-datasets/jena_clima
Resolving s3.amazonaws.com (s3.amazonaws.com)... 52.216.95.29, 16.182.66.21
Connecting to s3.amazonaws.com (s3.amazonaws.com)|52.216.95.29|:443... conn
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  18.8MB/s    in 0.7s

2024-07-20 00:23:39 (18.8 MB/s) - ‘jena_climate_2009_2016.csv.zip’ saved [1

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))
```

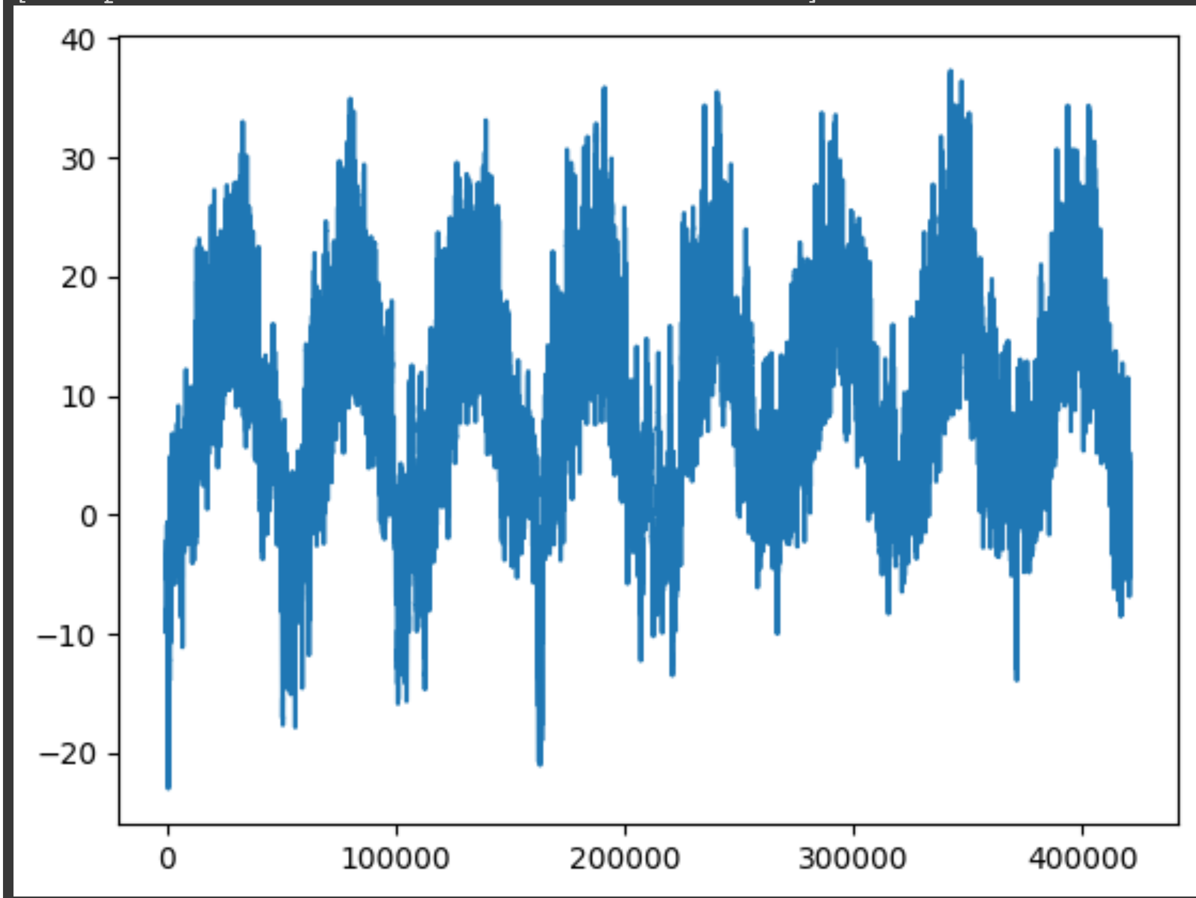
```
['"Date Time"', '"p (mbar)"', '"T (degC)"', '"Tpot (K)"', '"Tdew (degC)"',
420451
```

```
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)
```

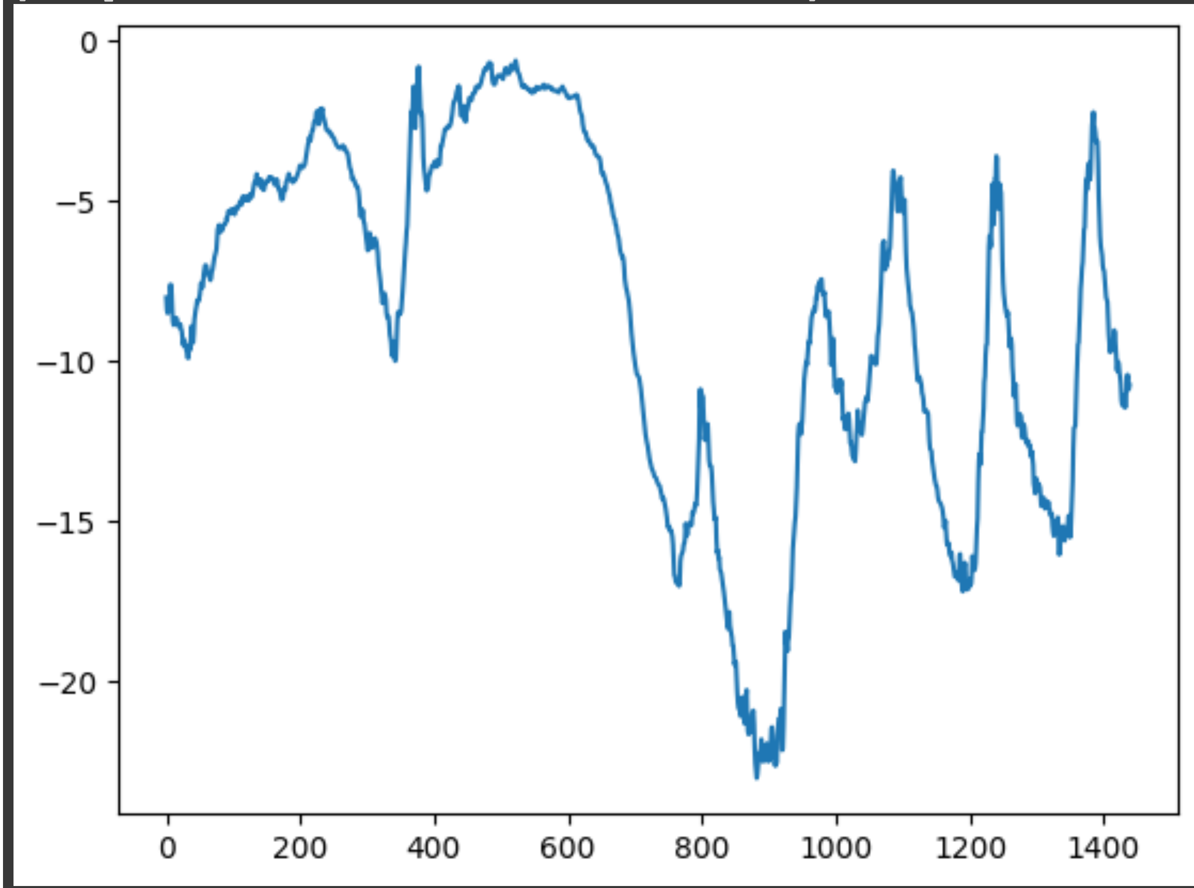
```
[<matplotlib.lines.Line2D at 0x79c84ddd79d0>]
```



Temperatues in °C

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

```
[<matplotlib.lines.Line2D at 0x79c849958430>]
```



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

```
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 t
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
```

```
Epoch 1/5
819/819 [==============================] - 55s 66ms/step - loss: 12.7710 -
Epoch 2/5
819/819 [==============================] - 54s 65ms/step - loss: 9.2414 - m
Epoch 3/5
819/819 [==============================] - 48s 58ms/step - loss: 8.4763 - m
Epoch 4/5
819/819 [==============================] - 56s 68ms/step - loss: 7.9824 - m
Epoch 5/5
819/819 [==============================] - 49s 60ms/step - loss: 7.6106 - m
405/405 [==============================] - 19s 44ms/step - loss: 11.4094 -
Test MAE: 2.68
```

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 t
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
```

```
Epoch 1/5
819/819 [==============================] - 49s 59ms/step - loss: 21.3422 -
Epoch 2/5
819/819 [==============================] - 51s 62ms/step - loss: 12.0865 -
Epoch 3/5
819/819 [==============================] - 52s 63ms/step - loss: 11.2515 -
Epoch 4/5
819/819 [==============================] - 57s 69ms/step - loss: 10.7244 -
Epoch 5/5
819/819 [==============================] - 54s 66ms/step - loss: 10.3188 -
405/405 [==============================] - 18s 42ms/step - loss: 11.2767 -
Test MAE: 2.61
```

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

inputs = keras.Input(shape=(sequence_length, pmry_data.shape[-1])) # Defining t
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
```

```
Epoch 1/5
819/819 [==============================] - 54s 65ms/step - loss: 12.4065 -
Epoch 2/5
819/819 [==============================] - 54s 66ms/step - loss: 8.7979 - m
Epoch 3/5
819/819 [==============================] - 50s 60ms/step - loss: 7.8028 - m
Epoch 4/5
819/819 [==============================] - 53s 64ms/step - loss: 7.2062 - m
Epoch 5/5
819/819 [==============================] - 56s 68ms/step - loss: 6.7568 - m
405/405 [==============================] - 16s 40ms/step - loss: 11.4883 -
Test MAE: 2.69
```

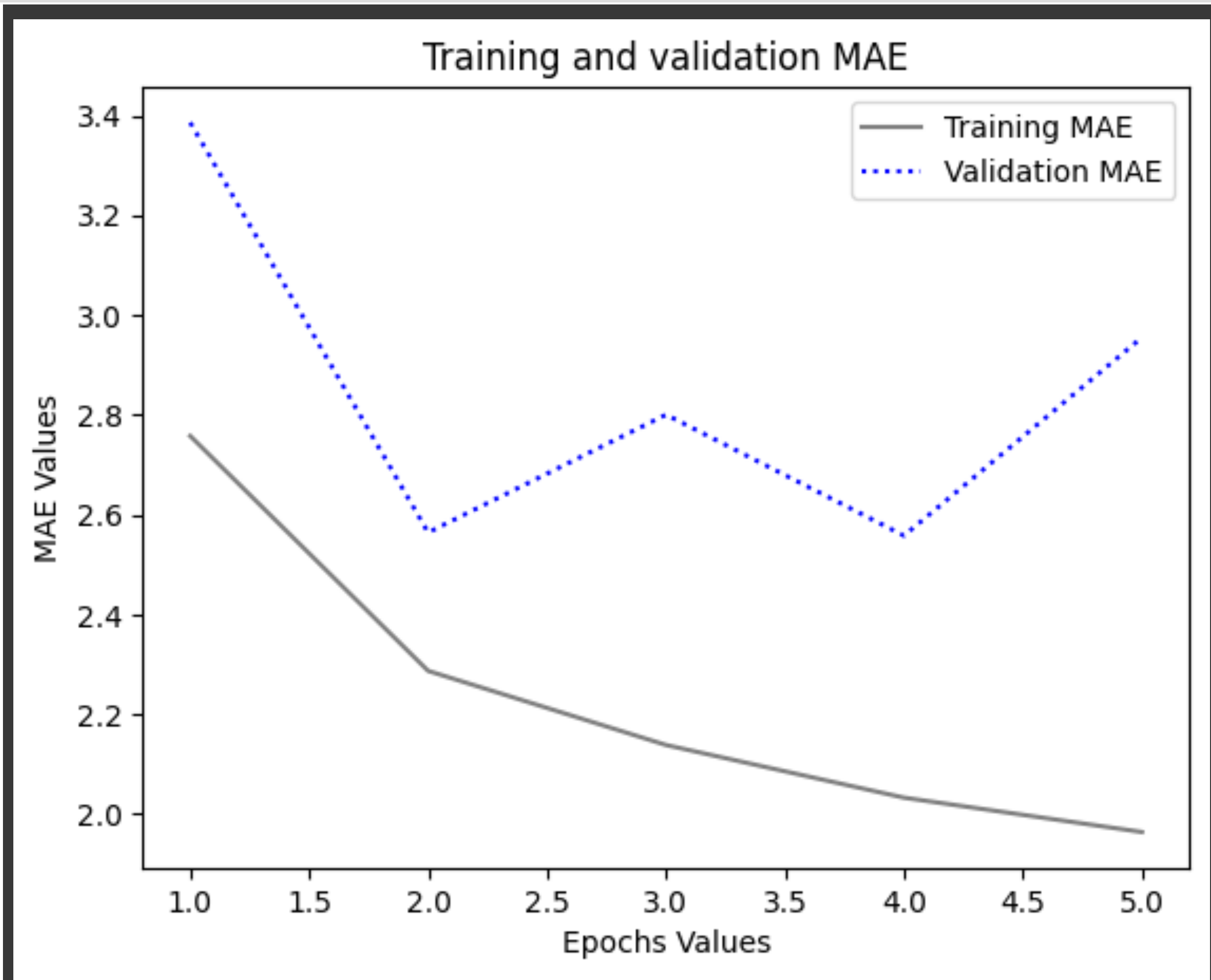
```
from tensorflow import keras
from tensorflow.keras import layers

inputs = keras.Input(shape=(sequence_length, pmry_data.shape[-1])) # Defining t
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
```

```
Epoch 1/5
819/819 [==============================] - 57s 68ms/step - loss: 12.7053 -
Epoch 2/5
819/819 [==============================] - 53s 64ms/step - loss: 8.4730 - m
Epoch 3/5
819/819 [==============================] - 55s 67ms/step - loss: 7.3854 - m
Epoch 4/5
819/819 [==============================] - 56s 68ms/step - loss: 6.6785 - m
Epoch 5/5
819/819 [==============================] - 56s 68ms/step - loss: 6.2089 - m
405/405 [==============================] - 17s 40ms/step - loss: 11.5738 -
Test MAE: 2.70
```

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
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 conventiona
convol_x = layers.MaxPooling1D(2)(convol_x)                    # Max pooling La
convol_x = layers.Conv1D(8, 12, activation="relu")(convol_x)   # 1D conventiona
convol_x = layers.MaxPooling1D(2)(convol_x)                    # Max pooling La
convol_x = layers.Conv1D(8, 6, activation="relu")(convol_x)    # 1D conventiona
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
```

```
Epoch 1/5
819/819 [==============================] - 85s 103ms/step - loss: 23.3811 -
Epoch 2/5
819/819 [==============================] - 88s 107ms/step - loss: 16.3592 -
Epoch 3/5
819/819 [==============================] - 81s 98ms/step - loss: 14.8700 -
Epoch 4/5
819/819 [==============================] - 85s 104ms/step - loss: 14.0818 -
Epoch 5/5
819/819 [==============================] - 81s 98ms/step - loss: 13.3889 -
405/405 [==============================] - 22s 53ms/step - loss: 16.2532 -
Test MAE: 3.22
```

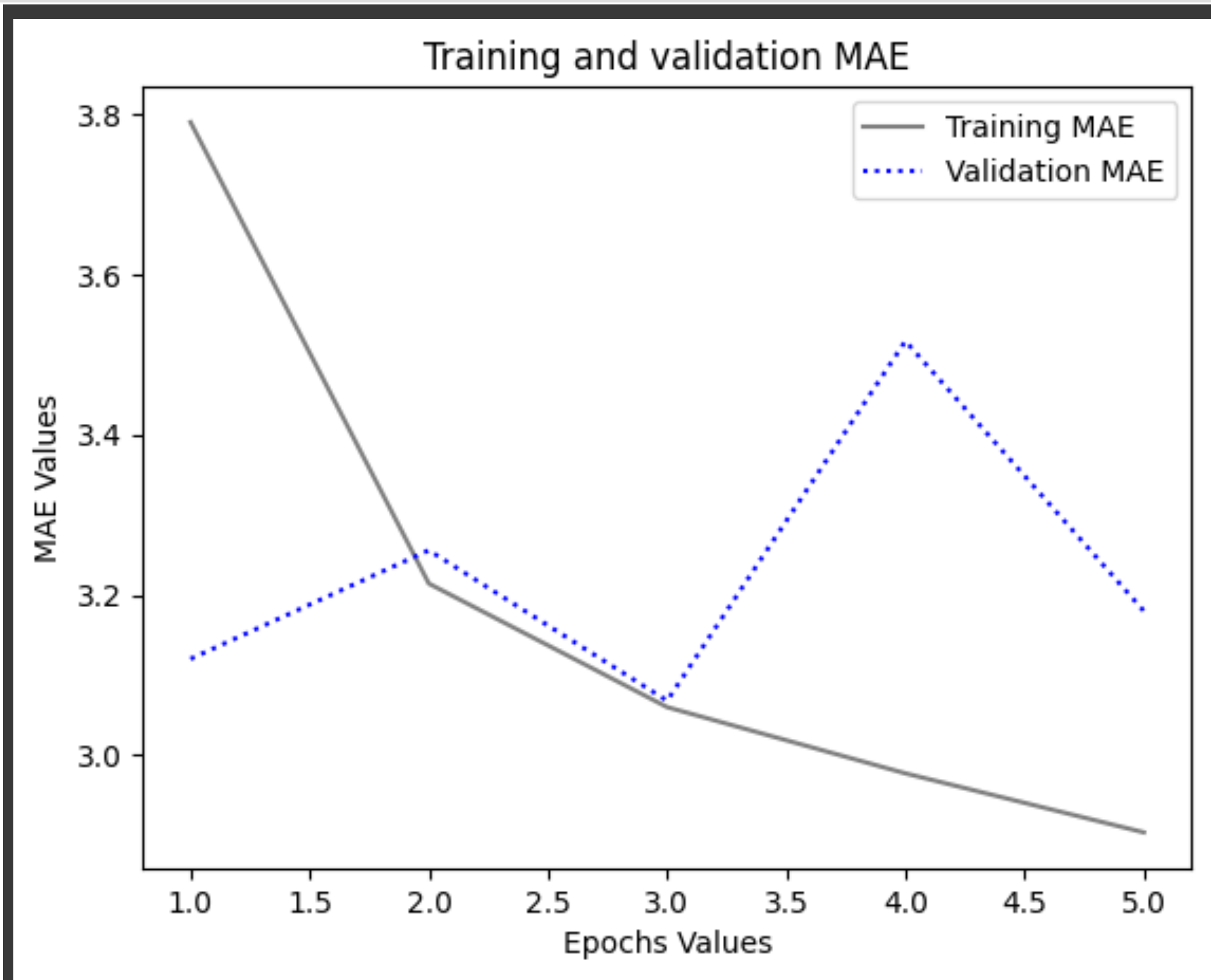
We received

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
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 t
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}")
```

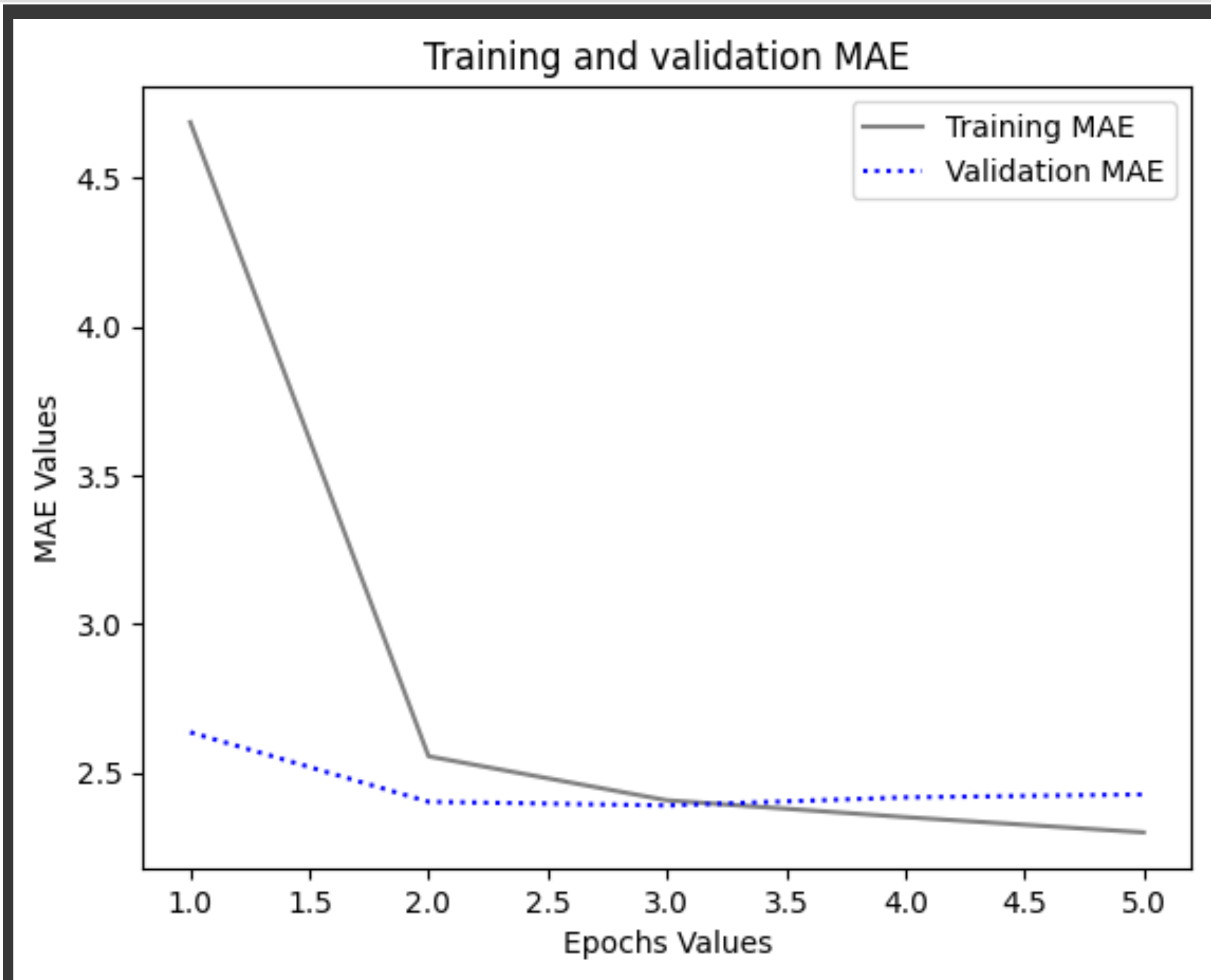
```
Epoch 1/5
819/819 [==============================] - 123s 141ms/step - loss: 41.6451
Epoch 2/5
819/819 [==============================] - 109s 132ms/step - loss: 10.8549
Epoch 3/5
819/819 [==============================] - 108s 131ms/step - loss: 9.5623 -
Epoch 4/5
819/819 [==============================] - 104s 126ms/step - loss: 9.1359 -
Epoch 5/5
819/819 [==============================] - 122s 148ms/step - loss: 8.7567 -
405/405 [==============================] - 26s 62ms/step - loss: 10.4601 -
Test MAE: 2.53
```

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
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 s
```

```
Epoch 1/5
819/819 [==============================] - 178s 209ms/step - loss: 46.8937
Epoch 2/5
819/819 [==============================] - 158s 193ms/step - loss: 20.0621
Epoch 3/5
819/819 [==============================] - 161s 196ms/step - loss: 18.3131
Epoch 4/5
819/819 [==============================] - 163s 199ms/step - loss: 17.4915
Epoch 5/5
819/819 [==============================] - 163s 199ms/step - loss: 16.8222
405/405 [==============================] - 28s 66ms/step - loss: 10.9959 -
Test MAE: 2.62
```

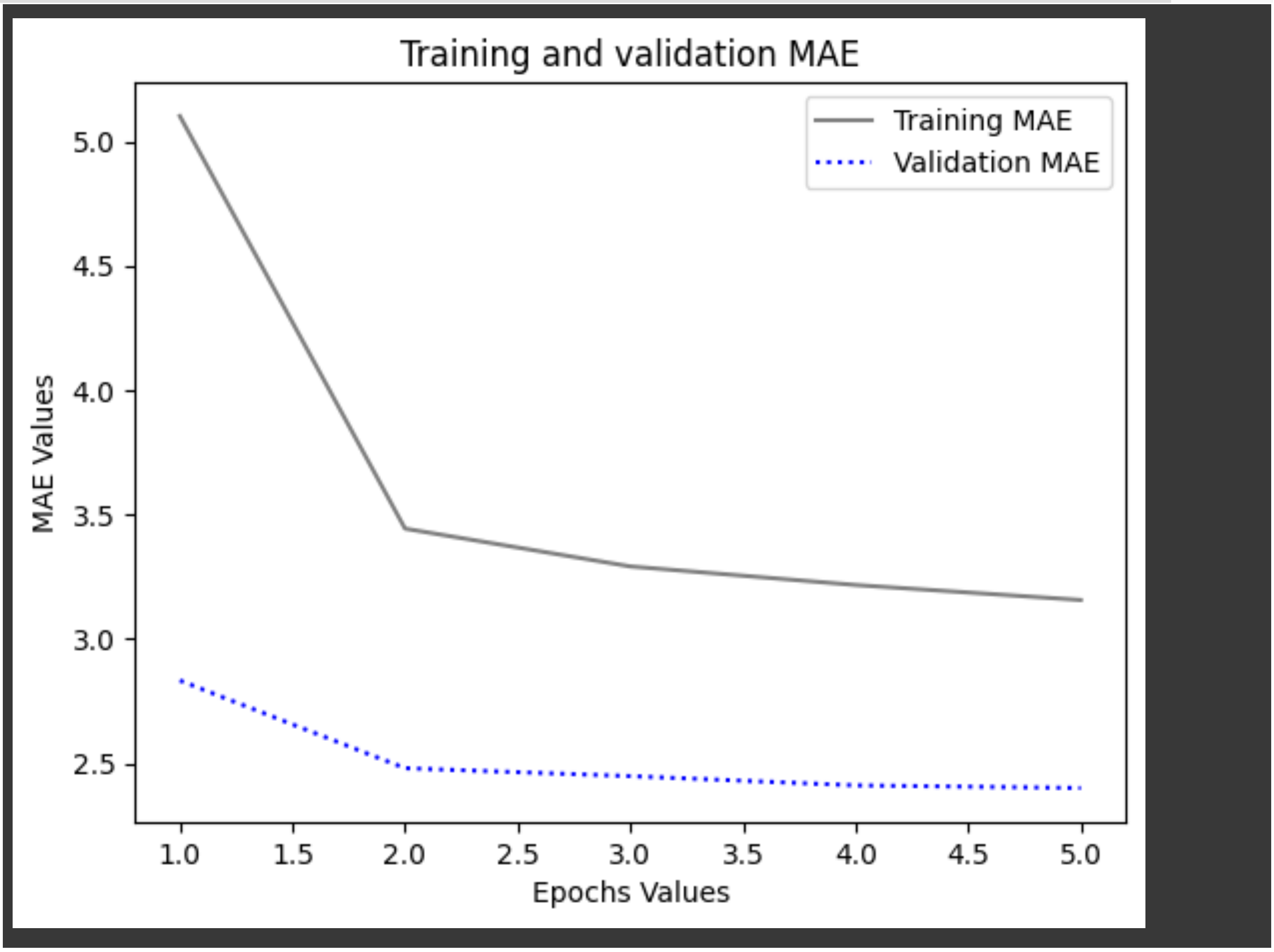
We received

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
plt.title("Training and validation MAE")
plt.xlabel("Epochs Values")
plt.ylabel("MAE Values")
plt.legend()
plt.show()
```



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

7th Model:

Stacked setup of recurrent layers

Computing dropout-regularized, stacked GRU model

```
inputs = keras.Input(shape=(sequence_length, pmry_data.shape[-1]))      # Defir
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 f
```

```
Epoch 1/5
819/819 [==============================] - 342s 409ms/step - loss: 25.0365
Epoch 2/5
819/819 [==============================] - 335s 409ms/step - loss: 13.9876
Epoch 3/5
819/819 [==============================] - 328s 400ms/step - loss: 13.1691
Epoch 4/5
819/819 [==============================] - 336s 410ms/step - loss: 12.5251
Epoch 5/5
819/819 [==============================] - 337s 411ms/step - loss: 12.1156
405/405 [==============================] - 42s 100ms/step - loss: 9.6172 -
Test MAE: 2.43
```

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 th
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
```

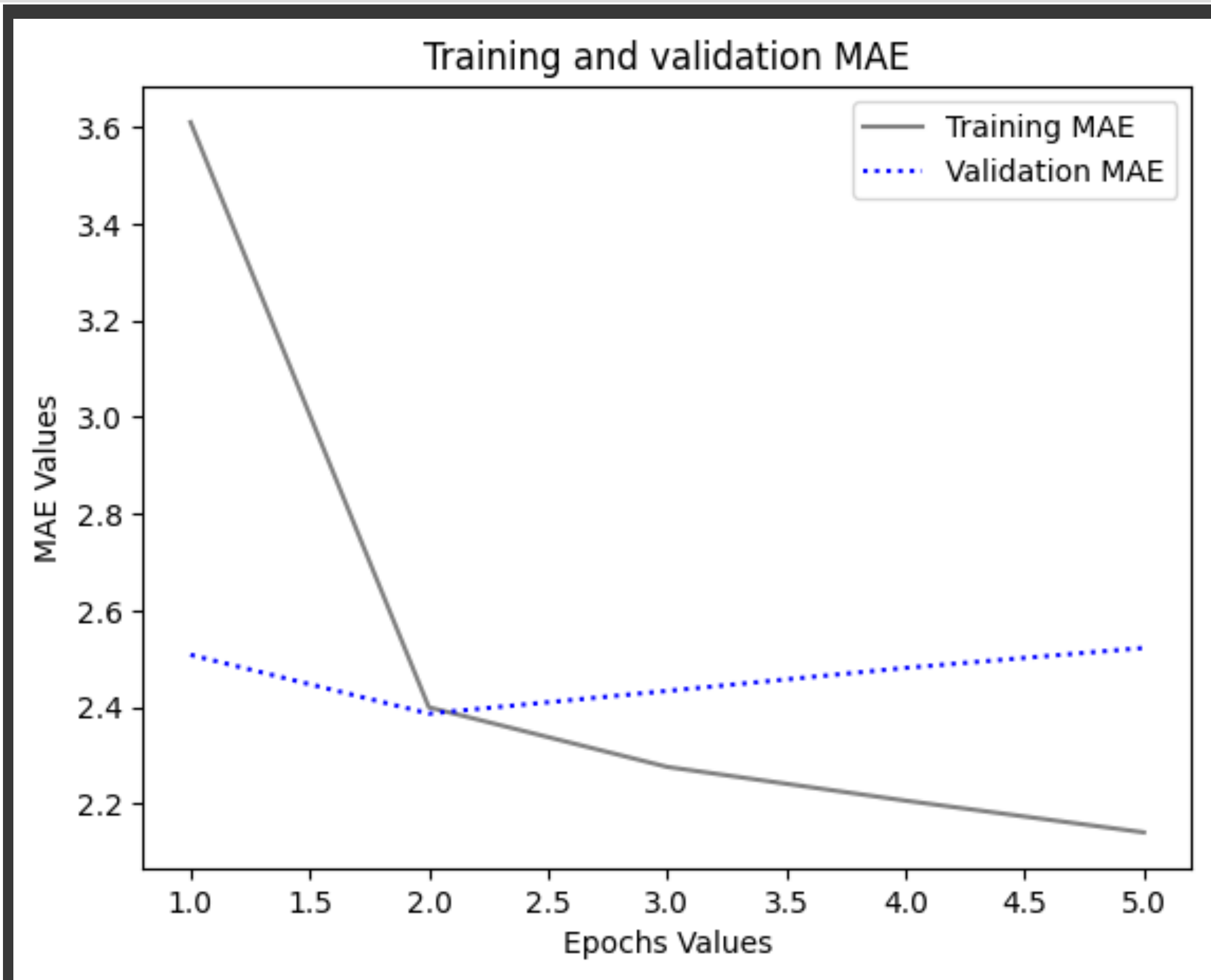
```
Epoch 1/5
819/819 [==============================] - 155s 176ms/step - loss: 24.8898
Epoch 2/5
819/819 [==============================] - 148s 180ms/step - loss: 9.4318 -
Epoch 3/5
819/819 [==============================] - 156s 190ms/step - loss: 8.5433 -
Epoch 4/5
819/819 [==============================] - 146s 178ms/step - loss: 8.0306 -
Epoch 5/5
819/819 [==============================] - 147s 179ms/step - loss: 7.5397 -
405/405 [==============================] - 36s 88ms/step - loss: 10.8439 -
Test MAE: 2.60
```

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
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 [==============================] - 152s 179ms/step - loss: 7.1739 -
Epoch 2/5
819/819 [==============================] - 156s 190ms/step - loss: 6.9490 -
Epoch 3/5
819/819 [==============================] - 147s 180ms/step - loss: 6.6829 -
Epoch 4/5
819/819 [==============================] - 150s 182ms/step - loss: 6.4780 -
Epoch 5/5
819/819 [==============================] - 145s 177ms/step - loss: 6.3117 -
405/405 [==============================] - 32s 77ms/step - loss: 12.4554 -
Test MAE: 2.78
```

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
plt.title("Training and validation MAE")
plt.xlabel("Epochs Values")
plt.ylabel("MAE Values")
plt.legend()
plt.show()
```