Applying RNN to Time-Series Data

Taking weather forecasting data

In [1]:
```
!pip install tensorflow==2.15
```

```
Requirement already satisfied: tensorflow==2.15 in /usr/local/lib/python3.10/dist-pac
kages (2.15.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packa
ges (from tensorflow==2.15) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-pa
ckages (from tensorflow==2.15) (1.6.3)
Requirement already satisfied: flatbuffers>=23.5.26 in /usr/local/lib/python3.10/dist
-packages (from tensorflow==2.15) (24.3.25)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/
python3.10/dist-packages (from tensorflow==2.15) (0.5.4)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-
packages (from tensorflow==2.15) (0.2.0)
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.10/dist-packages
(from tensorflow==2.15) (3.9.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dist-pac
kages (from tensorflow==2.15) (18.1.1)
Requirement already satisfied: ml-dtypes~=0.2.0 in /usr/local/lib/python3.10/dist-pac
kages (from tensorflow==2.15) (0.2.0)
Requirement already satisfied: numpy<2.0.0,>=1.23.5 in /usr/local/lib/python3.10/dist
-packages (from tensorflow==2.15) (1.25.2)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/dist-pa
ckages (from tensorflow==2.15) (3.3.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages
(from tensorflow==2.15) (24.0)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!
=4.21.5,<5.0.0dev,>=3.20.3 in /usr/local/lib/python3.10/dist-packages (from tensorflo
w==2.15) (3.20.3)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages
(from tensorflow==2.15) (67.7.2)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-packages
(from tensorflow==2.15) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-pac
kages (from tensorflow==2.15) (2.4.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.10/
dist-packages (from tensorflow==2.15) (4.10.0)
Requirement already satisfied: wrapt<1.15,>=1.11.0 in /usr/local/lib/python3.10/dist-
packages (from tensorflow==2.15) (1.14.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/li
b/python3.10/dist-packages (from tensorflow==2.15) (0.36.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-
packages (from tensorflow==2.15) (1.62.1)
Requirement already satisfied: tensorboard<2.16,>=2.15 in /usr/local/lib/python3.10/d
ist-packages (from tensorflow==2.15) (2.15.2)
Requirement already satisfied: tensorflow-estimator<2.16,>=2.15.0 in /usr/local/lib/p
ython3.10/dist-packages (from tensorflow==2.15) (2.15.0)
Requirement already satisfied: keras<2.16,>=2.15.0 in /usr/local/lib/python3.10/dist-
packages (from tensorflow==2.15) (2.15.0)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.10/dist-p
ackages (from astunparse>=1.6.0->tensorflow==2.15) (0.43.0)
Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3.10/dis
t-packages (from tensorboard<2.16,>=2.15->tensorflow==2.15) (2.27.0)
Requirement already satisfied: google-auth-oauthlib<2,>=0.5 in /usr/local/lib/python
3.10/dist-packages (from tensorboard<2.16,>=2.15->tensorflow==2.15) (1.2.0)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-pack
ages (from tensorboard<2.16,>=2.15->tensorflow==2.15) (3.6)
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.10/dist-
packages (from tensorboard<2.16,>=2.15->tensorflow==2.15) (2.31.0)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/li
b/python3.10/dist-packages (from tensorboard<2.16,>=2.15->tensorflow==2.15) (0.7.2)
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10/dist-pack
```

```
ages (from tensorboard<2.16,>=2.15->tensorflow==2.15) (3.0.2)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.10/di
st-packages (from google-auth<3,>=1.6.3->tensorboard<2.16,>=2.15->tensorflow==2.15)
(5.3.3)
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.10/dis
t-packages (from google-auth<3,>=1.6.3->tensorboard<2.16,>=2.15->tensorflow==2.15)
(0.4.0)
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.10/dist-packag
es (from google-auth<3,>=1.6.3->tensorboard<2.16,>=2.15->tensorflow==2.15) (4.9)
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.10/
dist-packages (from google-auth-oauthlib<2,>=0.5->tensorboard<2.16,>=2.15->tensorflow
==2.15) (1.3.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/
dist-packages (from requests<3,>=2.21.0->tensorboard<2.16,>=2.15->tensorflow==2.15)
(3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-package
s (from requests<3,>=2.21.0->tensorboard<2.16,>=2.15->tensorflow==2.15) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-p
ackages (from requests<3,>=2.21.0->tensorboard<2.16,>=2.15->tensorflow==2.15) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-p
ackages (from requests<3,>=2.21.0->tensorboard<2.16,>=2.15->tensorflow==2.15) (2024.
2.2)
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-pa
ckages (from werkzeug>=1.0.1->tensorboard<2.16,>=2.15->tensorflow==2.15) (2.1.5)
Requirement already satisfied: pyasn1<0.7.0,>=0.4.6 in /usr/local/lib/python3.10/dist
-packages (from pyasn1-modules>=0.2.1->google-auth<3,>=1.6.3->tensorboard<2.16,>=2.15
->tensorflow==2.15) (0.6.0)
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.10/dist-pack
ages (from requests-oauthlib>=0.7.0->google-auth-oauthlib<2,>=0.5->tensorboard<2.16,>
=2.15->tensorflow==2.15) (3.2.2)
```

In [2]:

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

```
--2024-04-07 23:32:20--  https://s3.amazonaws.com/keras-datasets/jena_climate_2009_20
16.csv.zip
Resolving s3.amazonaws.com (s3.amazonaws.com)... 52.217.134.8, 52.216.220.160, 52.21
6.57.48, ...
Connecting to s3.amazonaws.com (s3.amazonaws.com)|52.217.134.8|:443... connected.
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  19.2MB/s    in 0.7s

2024-04-07 23:32:21 (19.2 MB/s) - ‘jena_climate_2009_2016.csv.zip’ saved [13565642/13
565642]

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

Importing the dataset

In [3]:

```python
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))
```

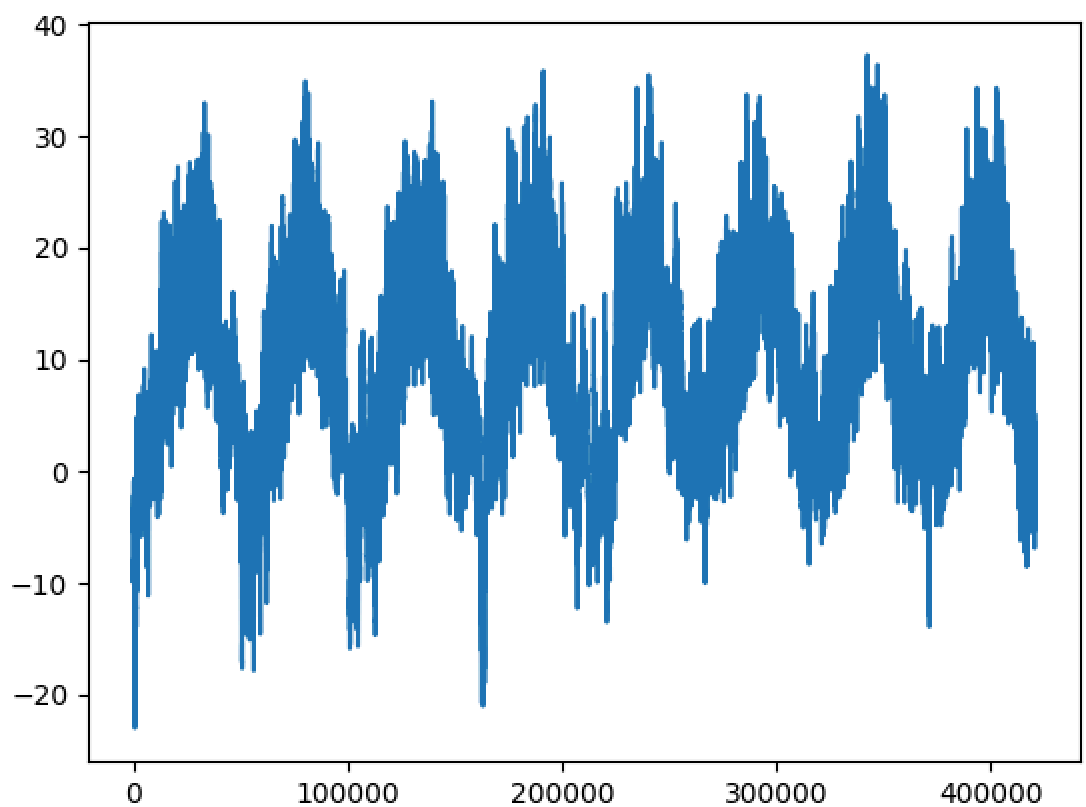
```
['"Date Time"', '"p (mbar)"', '"T (degC)"', '"Tpot (K)"', '"Tdew (degC)"', '"rh
(%)"', '"VPmax (mbar)"', '"VPact (mbar)"', '"VPdef (mbar)"', '"sh (g/kg)"', '"H2OC (m
mol/mol)"', '"rho (g/m**3)"', '"wv (m/s)"', '"max. wv (m/s)"', '"wd (deg)"']
420451
```

In [4]:
```
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

In [5]:
```
from matplotlib import pyplot as plt # Using matplotlib to plot the values
plt.plot(range(len(temp)), temp)
```

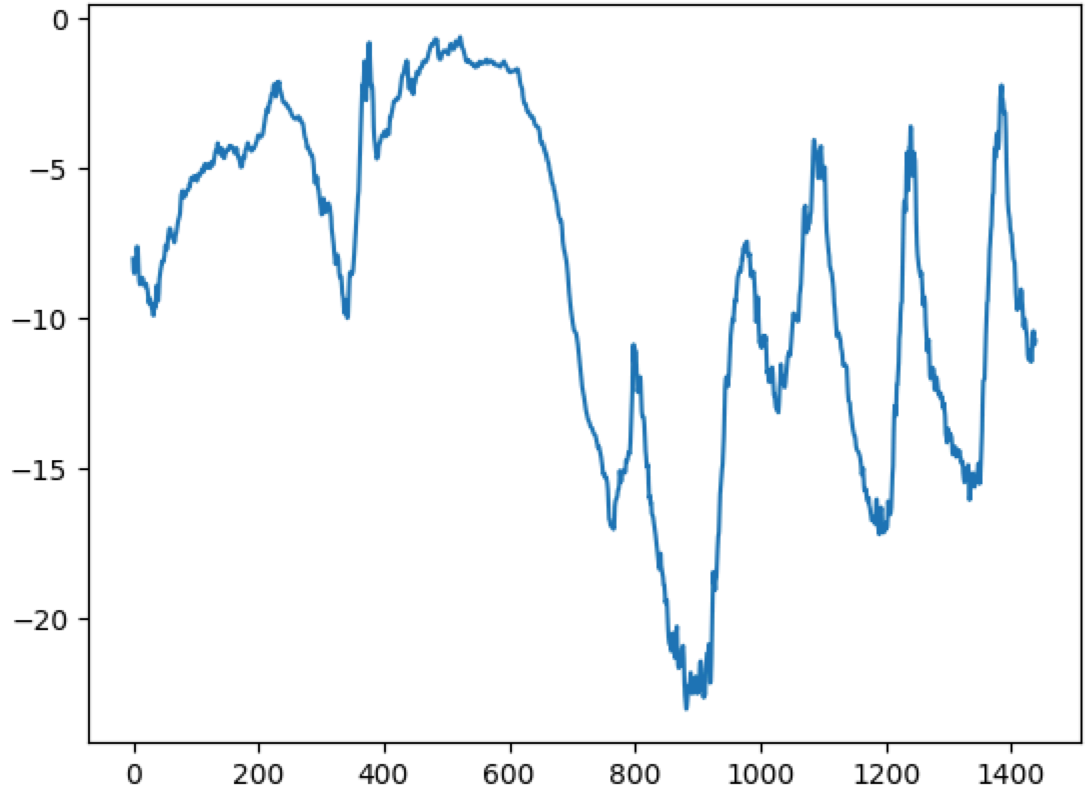
Out[5]:
```
[<matplotlib.lines.Line2D at 0x7a70ca3d6b60>]
```



Temperatues in °C

In [6]:
```
plt.plot(range(1440), temp[:1440])
```

Out[6]:
```
[<matplotlib.lines.Line2D at 0x7a70ca2db9a0>]
```



Calculating the quantity of samples that each data split will require

In [7]:
```
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

In [8]:
```
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.

In [9]:
```
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 data
    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

In [10]:
```
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

In [11]:
```
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

In [12]:
```
def evaluate_naive_method(dataset): # using evaluate_naive_method to calculate MAE
    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 the va
print(f"Test MAE: {evaluate_naive_method(test_dataset):.2f}") #  # Displaying the vali
```

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

2nd Model:

Basic machine-learning model

Simple neural network model for forecasting using Keras.

In [13]:
```
from tensorflow import keras
from tensorflow.keras import layers

inputs = keras.Input(shape=(sequence_length, pmry_data.shape[-1])) # Defining the inpu
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 test d
```

```
Epoch 1/5
819/819 [==============================] - 40s 47ms/step - loss: 13.8354 - mae: 2.869
3 - val_loss: 10.6964 - val_mae: 2.5819
Epoch 2/5
819/819 [==============================] - 39s 48ms/step - loss: 9.4806 - mae: 2.4187
- val_loss: 10.5222 - val_mae: 2.5631
Epoch 3/5
819/819 [==============================] - 39s 47ms/step - loss: 8.7072 - mae: 2.3219
- val_loss: 11.6787 - val_mae: 2.7143
Epoch 4/5
819/819 [==============================] - 48s 59ms/step - loss: 8.1823 - mae: 2.2513
- val_loss: 12.1607 - val_mae: 2.7588
Epoch 5/5
819/819 [==============================] - 39s 47ms/step - loss: 7.8062 - mae: 2.2012
- val_loss: 13.1117 - val_mae: 2.8712
405/405 [==============================] - 13s 32ms/step - loss: 11.3766 - mae: 2.661
7
Test MAE: 2.66
```

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

In [14]:

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

inputs = keras.Input(shape=(sequence_length, pmry_data.shape[-1])) # Defining the inpu
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 test d
```

```
Epoch 1/5
819/819 [==============================] - 38s 46ms/step - loss: 13.7336 - mae: 2.861
4 - val_loss: 11.3828 - val_mae: 2.6792
Epoch 2/5
819/819 [==============================] - 48s 58ms/step - loss: 9.8448 - mae: 2.4805
- val_loss: 10.5232 - val_mae: 2.5619
Epoch 3/5
819/819 [==============================] - 38s 47ms/step - loss: 9.2856 - mae: 2.4071
- val_loss: 11.7458 - val_mae: 2.7147
Epoch 4/5
819/819 [==============================] - 48s 58ms/step - loss: 8.8895 - mae: 2.3537
- val_loss: 10.8861 - val_mae: 2.6056
Epoch 5/5
819/819 [==============================] - 37s 45ms/step - loss: 8.6078 - mae: 2.3172
- val_loss: 10.6583 - val_mae: 2.5765
405/405 [==============================] - 13s 31ms/step - loss: 11.5658 - mae: 2.668
4
Test MAE: 2.67
```

In [15]:

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

inputs = keras.Input(shape=(sequence_length, pmry_data.shape[-1])) # Defining the inpu
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 test d
```

```
Epoch 1/5
819/819 [==============================] - 48s 58ms/step - loss: 12.0396 - mae: 2.689
5 - val_loss: 11.5963 - val_mae: 2.6892
Epoch 2/5
819/819 [==============================] - 48s 59ms/step - loss: 8.5856 - mae: 2.3024
- val_loss: 10.2363 - val_mae: 2.5226
Epoch 3/5
819/819 [==============================] - 47s 57ms/step - loss: 7.6594 - mae: 2.1751
- val_loss: 10.2962 - val_mae: 2.5360
Epoch 4/5
819/819 [==============================] - 38s 46ms/step - loss: 7.0765 - mae: 2.0904
- val_loss: 11.0559 - val_mae: 2.6372
Epoch 5/5
819/819 [==============================] - 38s 45ms/step - loss: 6.6199 - mae: 2.0229
- val_loss: 11.4999 - val_mae: 2.6905
405/405 [==============================] - 13s 31ms/step - loss: 11.3889 - mae: 2.665
3
Test MAE: 2.67
```

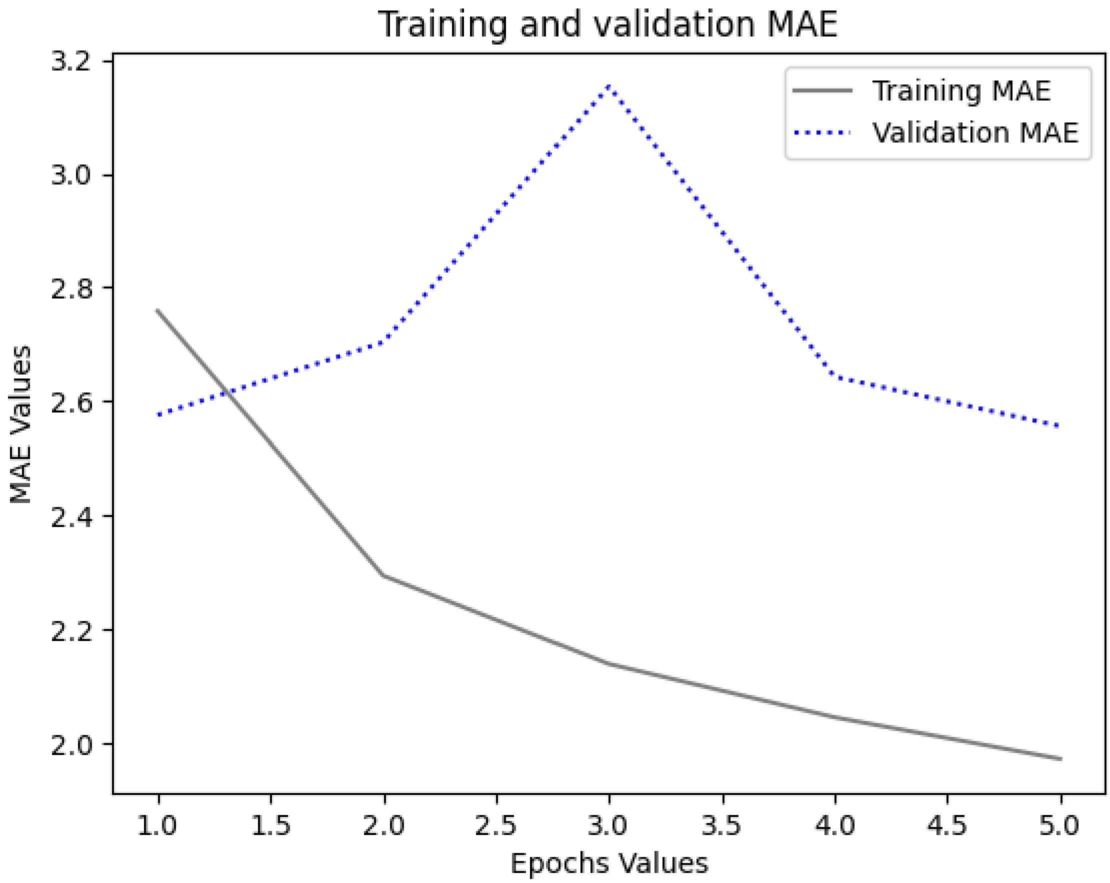
In [ ]:
```
from tensorflow import keras
from tensorflow.keras import layers

inputs = keras.Input(shape=(sequence_length, pmry_data.shape[-1])) # Defining the inpu
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 test d
```

```
Epoch 1/5
819/819 [==============================] - 37s 44ms/step - loss: 12.6638 - mae: 2.758
4 - val_loss: 10.6554 - val_mae: 2.5757
Epoch 2/5
819/819 [==============================] - 46s 56ms/step - loss: 8.5352 - mae: 2.2939
- val_loss: 11.7338 - val_mae: 2.7034
Epoch 3/5
819/819 [==============================] - 37s 45ms/step - loss: 7.4101 - mae: 2.1386
- val_loss: 15.6021 - val_mae: 3.1535
Epoch 4/5
819/819 [==============================] - 46s 56ms/step - loss: 6.7594 - mae: 2.0449
- val_loss: 11.1811 - val_mae: 2.6429
Epoch 5/5
819/819 [==============================] - 38s 46ms/step - loss: 6.2839 - mae: 1.9717
- val_loss: 10.5104 - val_mae: 2.5561
405/405 [==============================] - 13s 31ms/step - loss: 11.3549 - mae: 2.669
7
Test MAE: 2.67
```

Tried various dense units of 8, 32 and 64

Graph of Training and Validation MAE Values

In [ ]:
```
# 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 MAE")
plt.title("Training and validation MAE")
plt.xlabel("Epochs Values")
plt.ylabel("MAE Values")
plt.legend()
plt.show()
```



3rd Model:

1D convolutional model

In [ ]:
```
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 layer
convol_x = layers.MaxPooling1D(2)(convol_x)                    # Max pooling Layer
convol_x = layers.Conv1D(8, 12, activation="relu")(convol_x)   # 1D conventional layer
convol_x = layers.MaxPooling1D(2)(convol_x)                    # Max pooling Layer
convol_x = layers.Conv1D(8, 6, activation="relu")(convol_x)    # 1D conventional layer
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 test d
```

```
Epoch 1/5
819/819 [==============================] - 42s 48ms/step - loss: 22.4171 - mae: 3.687
8 - val_loss: 15.6087 - val_mae: 3.1092
Epoch 2/5
819/819 [==============================] - 38s 46ms/step - loss: 15.7093 - mae: 3.138
8 - val_loss: 15.3861 - val_mae: 3.1285
Epoch 3/5
819/819 [==============================] - 49s 59ms/step - loss: 14.6198 - mae: 3.032
3 - val_loss: 14.7381 - val_mae: 3.0495
Epoch 4/5
819/819 [==============================] - 48s 58ms/step - loss: 13.8313 - mae: 2.947
6 - val_loss: 15.6607 - val_mae: 3.1263
Epoch 5/5
819/819 [==============================] - 38s 47ms/step - loss: 13.1960 - mae: 2.877
7 - val_loss: 16.6953 - val_mae: 3.2278
405/405 [==============================] - 13s 32ms/step - loss: 16.3099 - mae: 3.204
1
Test MAE: 3.20
```

We received

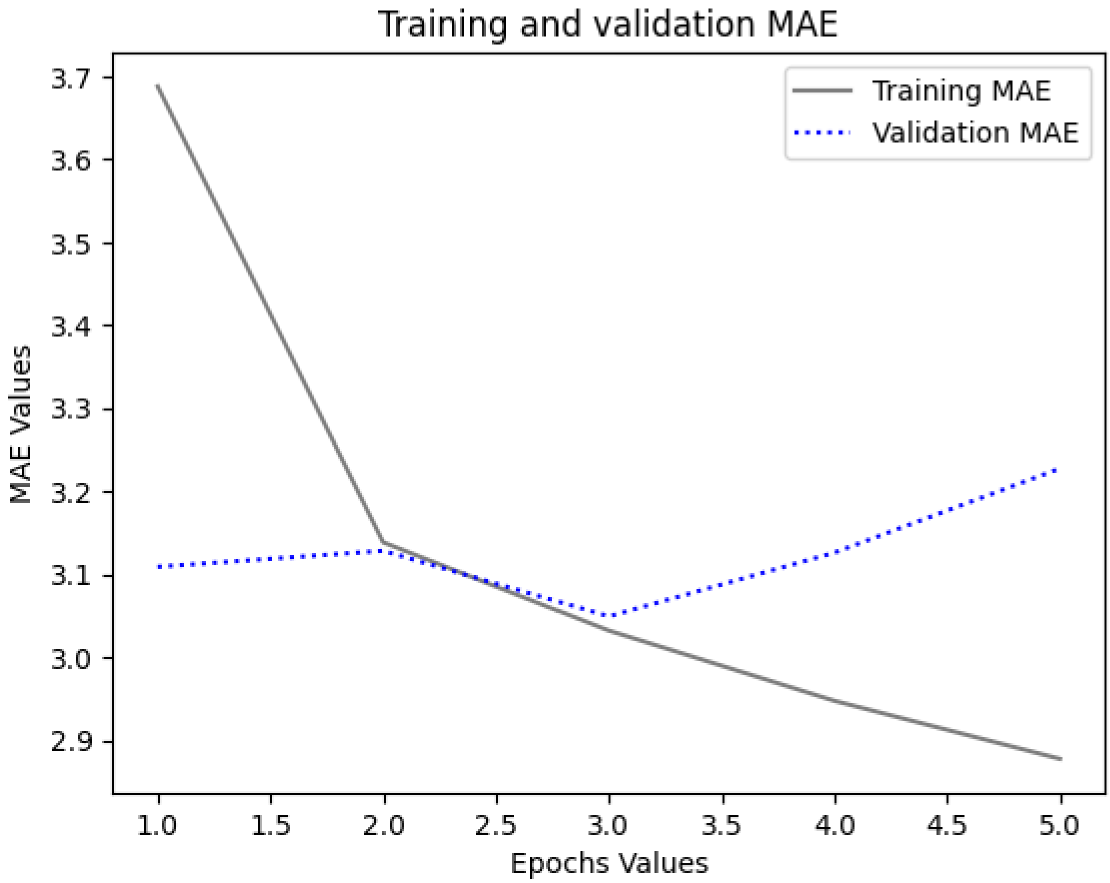
Validation MAE: 3.2278

Test MAE : 3.20

Graph of Training and Validation MAE Values

In [ ]:

```
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 MAE")
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

In [ ]:
```
inputs = keras.Input(shape=(sequence_length, pmry_data.shape[-1])) # Defining the inpu
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 [==============================] - 46s 53ms/step - loss: 48.2989 - mae: 5.081
2 - val_loss: 13.8085 - val_mae: 2.7983
Epoch 2/5
819/819 [==============================] - 44s 53ms/step - loss: 11.4708 - mae: 2.632
7 - val_loss: 9.5353 - val_mae: 2.3913
Epoch 3/5
819/819 [==============================] - 43s 52ms/step - loss: 10.1155 - mae: 2.482
3 - val_loss: 9.4213 - val_mae: 2.3761
Epoch 4/5
819/819 [==============================] - 53s 64ms/step - loss: 9.5540 - mae: 2.4103
- val_loss: 9.3574 - val_mae: 2.3653
Epoch 5/5
819/819 [==============================] - 39s 48ms/step - loss: 9.1957 - mae: 2.3645
- val_loss: 9.4934 - val_mae: 2.3790
405/405 [==============================] - 13s 31ms/step - loss: 10.6071 - mae: 2.547
5
Test MAE: 2.55
```

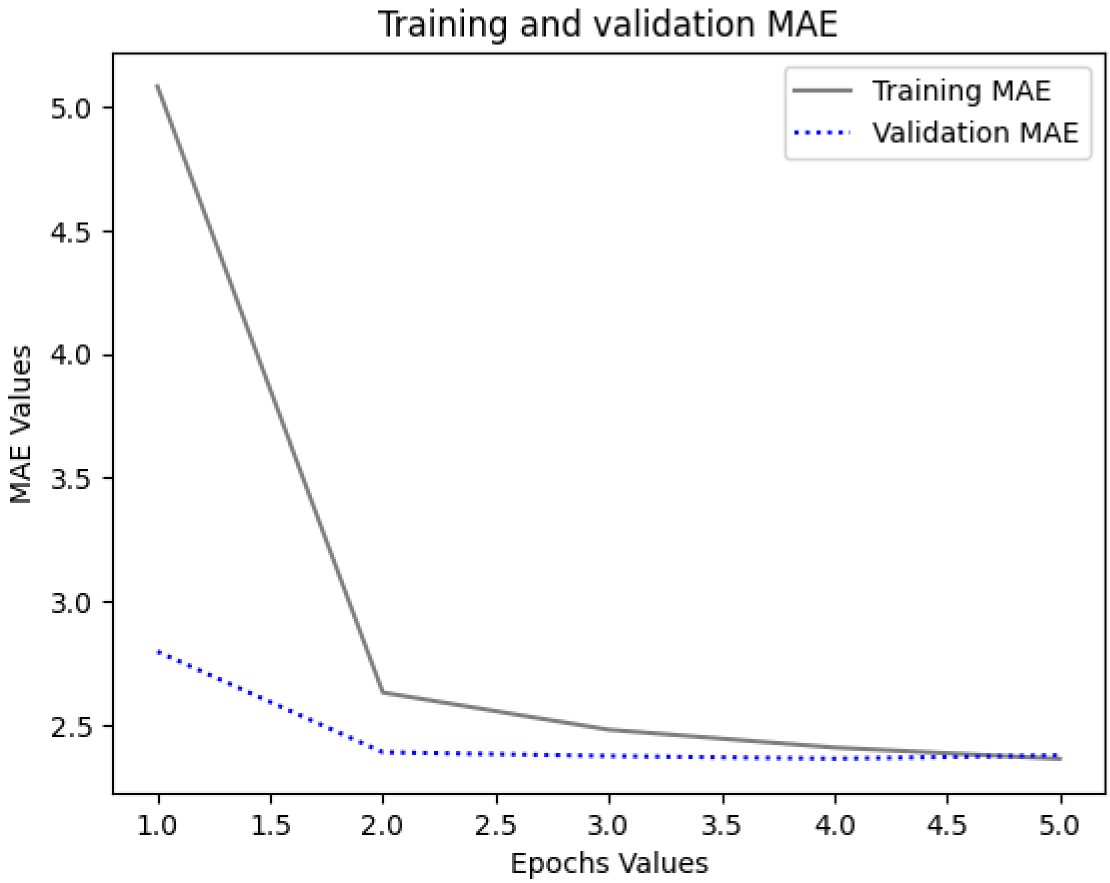
We received

Validation MAE: 2.3790

Test MAE : 2.55

In [ ]:

```python
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 MAE")
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

In [ ]:
```
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)
```

In [ ]:
```
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

In [ ]:
```
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)
```

In [ ]:
```
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

In [ ]:
```
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

In [ ]:
```
inputs = keras.Input(shape=(sequence_length, pmry_data.shape[-1]))           # Defi
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 sample M
```

```
WARNING:tensorflow:Layer lstm will not use cuDNN kernels since it doesn't meet the cr
iteria. It will use a generic GPU kernel as fallback when running on GPU.
```

```
Epoch 1/5
819/819 [==============================] - 287s 346ms/step - loss: 46.1028 - mae: 5.0
708 - val_loss: 13.3208 - val_mae: 2.7662
Epoch 2/5
819/819 [==============================] - 287s 351ms/step - loss: 19.9860 - mae: 3.4
336 - val_loss: 9.9788 - val_mae: 2.4582
Epoch 3/5
819/819 [==============================] - 284s 347ms/step - loss: 18.3525 - mae: 3.2
934 - val_loss: 9.7477 - val_mae: 2.4391
Epoch 4/5
819/819 [==============================] - 287s 349ms/step - loss: 17.5025 - mae: 3.2
177 - val_loss: 9.6928 - val_mae: 2.4370
Epoch 5/5
819/819 [==============================] - 298s 363ms/step - loss: 16.8856 - mae: 3.1
660 - val_loss: 9.5326 - val_mae: 2.4221
```

```
WARNING:tensorflow:Layer lstm will not use cuDNN kernels since it doesn't meet the cr
iteria. It will use a generic GPU kernel as fallback when running on GPU.
```

```
405/405 [==============================] - 30s 73ms/step - loss: 11.0949 - mae: 2.617
8
Test MAE: 2.62
```

We received

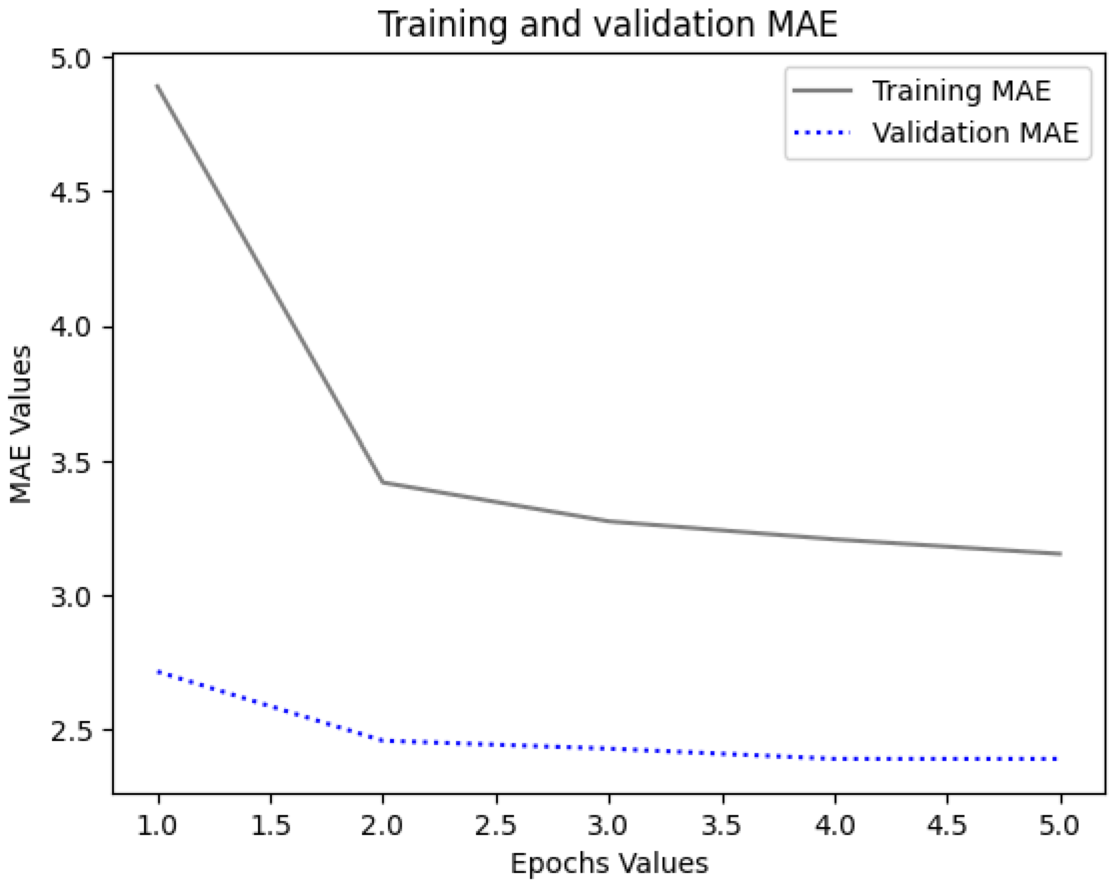
Validation MAE: 2.4221

Test MAE : 2.62

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

In [ ]:

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



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

```
WARNING:tensorflow:Layer lstm_2 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
```

7th Model:

Stacked setup of recurrent layers

Computing dropout-regularized, stacked GRU model

In [ ]:
```
inputs = keras.Input(shape=(sequence_length, pmry_data.shape[-1]))      # Defining the
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 test
```

```
WARNING:tensorflow:Layer gru will not use cuDNN kernels since it doesn't meet the cri
teria. It will use a generic GPU kernel as fallback when running on GPU.
WARNING:tensorflow:Layer gru_1 will not use cuDNN kernels since it doesn't meet the c
riteria. It will use a generic GPU kernel as fallback when running on GPU.
Epoch 1/5
819/819 [==============================] - 546s 658ms/step - loss: 26.5803 - mae: 3.7
781 - val_loss: 9.2917 - val_mae: 2.3584
Epoch 2/5
819/819 [==============================] - 538s 656ms/step - loss: 13.8852 - mae: 2.8
865 - val_loss: 8.9438 - val_mae: 2.3208
Epoch 3/5
819/819 [==============================] - 534s 652ms/step - loss: 13.1488 - mae: 2.8
081 - val_loss: 9.0930 - val_mae: 2.3425
Epoch 4/5
819/819 [==============================] - 538s 657ms/step - loss: 12.5502 - mae: 2.7
433 - val_loss: 8.8388 - val_mae: 2.3081
Epoch 5/5
819/819 [==============================] - 538s 656ms/step - loss: 12.1748 - mae: 2.7
013 - val_loss: 9.0895 - val_mae: 2.3444
WARNING:tensorflow:Layer gru will not use cuDNN kernels since it doesn't meet the cri
teria. It will use a generic GPU kernel as fallback when running on GPU.
WARNING:tensorflow:Layer gru_1 will not use cuDNN kernels since it doesn't meet the c
riteria. It will use a generic GPU kernel as fallback when running on GPU.
405/405 [==============================] - 41s 100ms/step - loss: 9.8169 - mae: 2.461
7
Test MAE: 2.46
```

We received

Validation MAE: 2.3444

Test MAE : 2.46

8th Model:

Bidirectional RNN

Computing the Bidirectional LSTM

In [ ]:

```
inputs = keras.Input(shape=(sequence_length, pmry_data.shape[-1]))
x = layers.Bidirectional(layers.LSTM(16))(inputs)                # Using the Bidir
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 [==============================] - 57s 66ms/step - loss: 25.4428 - mae: 3.631
6 - val_loss: 10.7021 - val_mae: 2.5358
Epoch 2/5
819/819 [==============================] - 52s 63ms/step - loss: 9.5631 - mae: 2.4127
- val_loss: 10.0460 - val_mae: 2.4504
Epoch 3/5
819/819 [==============================] - 43s 52ms/step - loss: 8.7561 - mae: 2.3080
- val_loss: 10.0814 - val_mae: 2.4698
Epoch 4/5
819/819 [==============================] - 45s 54ms/step - loss: 8.2579 - mae: 2.2456
- val_loss: 9.9260 - val_mae: 2.4377
Epoch 5/5
819/819 [==============================] - 43s 52ms/step - loss: 7.8541 - mae: 2.1957
- val_loss: 10.5242 - val_mae: 2.5226
405/405 [==============================] - 13s 32ms/step - loss: 10.8788 - mae: 2.596
0
Test MAE: 2.60
```

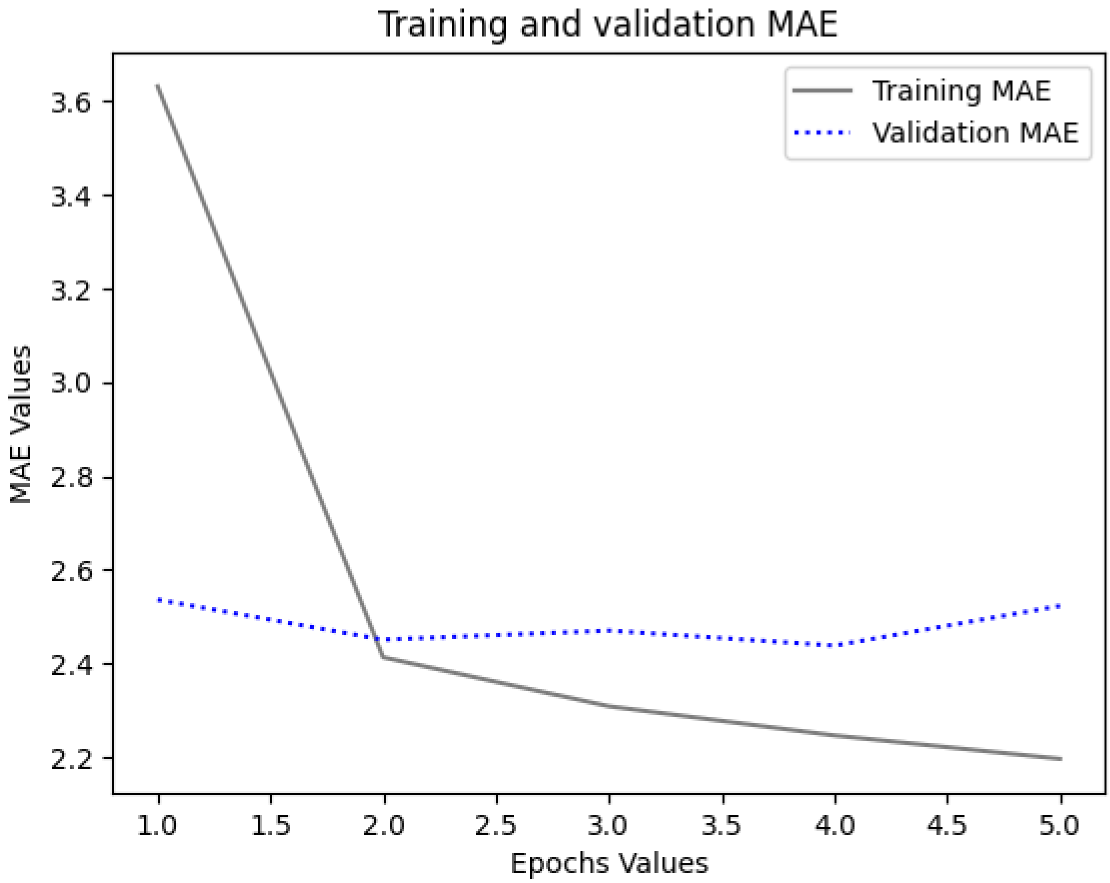
We received

Validation MAE: 2.5226

Test MAE : 2.60

In [ ]:

```
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 MAE")
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

In [ ]:
```
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 - mae: 3.1
180 - val_loss: 9.4154 - val_mae: 2.4084
Epoch 2/5
819/819 [==============================] - 284s 347ms/step - loss: 16.1007 - mae: 3.0
890 - val_loss: 9.5428 - val_mae: 2.4194
Epoch 3/5
819/819 [==============================] - 284s 346ms/step - loss: 15.7931 - mae: 3.0
641 - val_loss: 9.4338 - val_mae: 2.3982
Epoch 4/5
819/819 [==============================] - 291s 355ms/step - loss: 15.5608 - mae: 3.0
405 - val_loss: 9.5251 - val_mae: 2.4140
Epoch 5/5
819/819 [==============================] - 296s 361ms/step - loss: 15.3729 - mae: 3.0
256 - val_loss: 9.6930 - val_mae: 2.4410
405/405 [==============================] - 29s 72ms/step - loss: 10.9585 - mae: 2.595
3
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

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