```
!pip install tensorflow==2.12
```

```
Requirement already satisfied: tensorflow==2.12 in /usr/local/lib/python3.10/dist-packages (2.12.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.12) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.12) (1.6.3)
Requirement already satisfied: flatbuffers>=2.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.12) (23.5.26)
Requirement already satisfied: gast<=0.4.0,>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.12) (0.4.0)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.12) (0.2.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.12) (1.59.0)
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.12) (3.9.0)
Requirement already satisfied: jax>=0.3.15 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.12) (0.3.25)
Requirement already satisfied: keras<2.13,>=2.12.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.12) (2.12.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.12) (16.0.6)
Requirement already satisfied: numpy<1.24,>=1.22 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.12) (1.23.5)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.12) (3.3.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.12) (23.2)
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/python
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.12) (67.7.2)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.12) (1.16.0)
Requirement already satisfied: tensorboard<2.13,>=2.12 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.12) (2.12.0)
Requirement already satisfied: tensorflow-estimator<2.13,>=2.12.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.12) (2.
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.12) (2.3.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.12) (4.5.0)
Requirement already satisfied: wrapt<1.15,>=1.11.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.12) (1.14.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.12) (
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.10/dist-packages (from astunparse>=1.6.0->tensorflow==2.12)
Requirement already satisfied: scipy>=1.5 in /usr/local/lib/python3.10/dist-packages (from jax>=0.3.15->tensorflow==2.12) (1.11.3)
Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.13,>=2.12->tensorflo
Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.13,>=2.12
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.13,>=2.12->tensorflow==2.1
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.13,>=2.12->tensorflow=
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.13,>
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.13,>=2.12->t
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.13,>=2.12->tensorflow==2.1
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensorboar
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensorboard
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensorboard<2.13,>=
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from google-auth-oauthlib<0.5,>=0.4.
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboar
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.13,>=2.1
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.13
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.13
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-packages (from werkzeug>=1.0.1->tensorboard<2.13,>=2.
Requirement already satisfied: pyasn1<0.6.0,>=0.4.6 in /usr/local/lib/python3.10/dist-packages (from pyasn1-modules>=0.2.1->google-auth<
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.10/dist-packages (from requests-oauthlib>=0.7.0->google-auth-oa
```

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

```
--2023-11-07 02:10:59--  https://s3.amazonaws.com/keras-datasets/jena_climate_2009_2016.csv.zip
Resolving s3.amazonaws.com (s3.amazonaws.com)... 54.231.194.48, 54.231.228.72, 52.216.9.237, ...
Connecting to s3.amazonaws.com (s3.amazonaws.com)|54.231.194.48|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 13565642 (13M) [application/zip]
Saving to: ‘jena_climate_2009_2016.csv.zip.1’

jena_climate_2009_2 100%[===================>]  12.94M  47.7MB/s    in 0.3s

2023-11-07 02:11:00 (47.7 MB/s) - ‘jena_climate_2009_2016.csv.zip.1’ saved [13565642/13565642]

Archive:  jena_climate_2009_2016.csv.zip
replace jena_climate_2009_2016.csv? [y]es, [n]o, [A]ll, [N]one, [r]ename:
```

Lets Examine Jena meteorological data: 15, characteristics for analysis out of 420,451 rows.

```
import os
fname = os.path.join("jena_climate_2009_2016.csv")

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

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

```
feature_count = len(header)
print("Number of variables:", feature_count)
num_rows = len(lines)
print("Number of rows:", num_rows)
```

```
['"Date Time"', '"p (mbar)"', '"T (degC)"', '"Tpot (K)"', '"Tdew (degC)"', '"rh (%)"', '"VPmax (mbar)"', '"VPact (mbar)"', '"VPdef (mbar
420451
Number of variables: 15
Number of rows: 420451
```

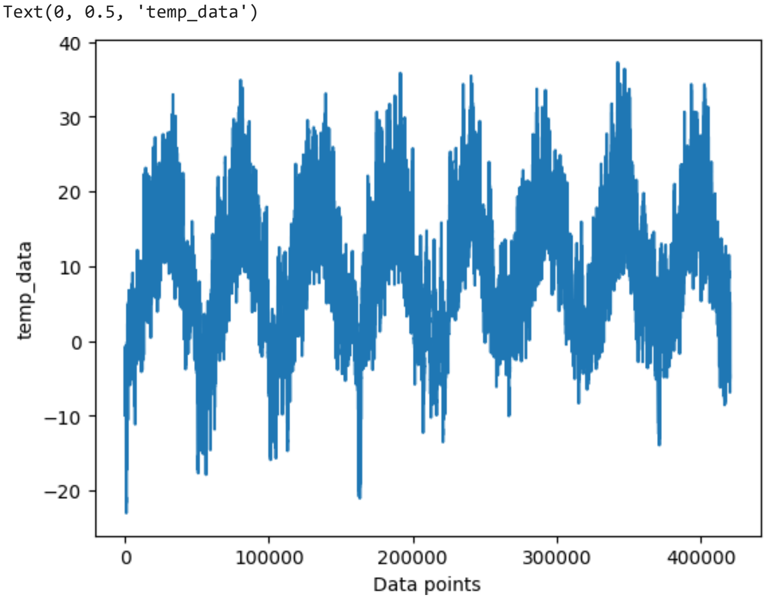
During parsing, values that are separated by commas are transformed into floating-point values. Important data is stored in the temp_datand raw_data arrays for further processing and analysis, which enables the development of insightful understandings.

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

Graphing the temp_data time series for examination.

```
from matplotlib import pyplot as plt
plt.plot(range(len(temp_data)), temp_data)
plt.xlabel('Data points')
plt.ylabel('temp_data')
```

```
Text(0, 0.5, 'temp_data')
```

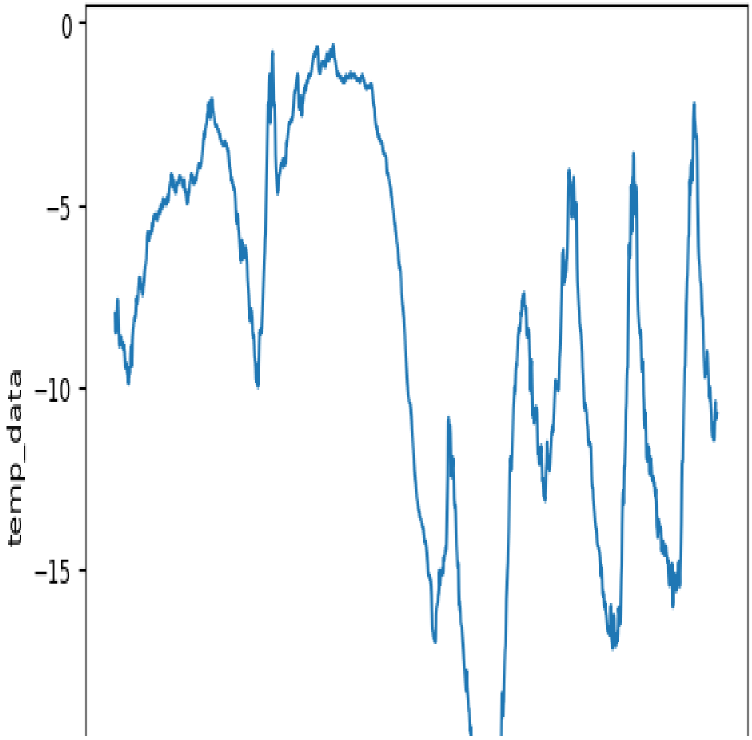


Lets plot the first ten days of the temp_data timeseries, which contains 1440 data points, in order to conduct a thorough analysis of short-term patterns and trends.

```
plt.plot(range(1440), temp_data[:1440])
plt.xlabel('Data points')
plt.ylabel('temp_data')
```

```
Text(0, 0.5, 'temp_data')
```

Getting the data ready



Let us Optimize dataset segmentation for model development by calculating sample distribution for data splits: assigning 25% for validation and 50% for training.

```
training_sample_count= int(0.5 * len(raw_data))
num_val_samples = int(0.25 * len(raw_data))
num_test_samples = len(raw_data) - training_sample_count - num_val_samples
print("training_sample_count:", training_sample_count)
print("num_val_samples:", num_val_samples)
print("num_test_samples:", num_test_samples)
```

```
training_sample_count: 210225
num_val_samples: 105112
num_test_samples: 105114
```

During the data normalization process, vectorization is not necessary because the data is already represented numerically. However, standardizing these variables is encouraged because some variables, such temp_data (-20 to +30) and pressure reported in millibars, have various scales.

```
mean = raw_data[:training_sample_count].mean(axis=0)
raw_data -= mean
std = raw_data[:training_sample_count].std(axis=0)
raw_data /= std
```

```
import numpy as np
from tensorflow import keras
int_sequence = np.arange(10)
dummy_dataset = keras.utils.timeseries_dataset_from_array(
    data=int_sequence[:-3],
    targets=int_sequence[3:],
    sequence_length=3,
    batch_size=2,
)

for inputs, targets in dummy_dataset:
    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
```

Datasets for training, validation, and testing must be created due to the substantial sample duplication. Creating samples dynamically instead of allocating RAM for each one allows for resource management.

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

train_samples = keras.utils.timeseries_dataset_from_array(
    raw_data[:-delay],
    targets=temp_data[delay:],
    sampling_rate=sampling_rate,
    sequence_length=sequence_length,
```

```
    shuffle=True,
    batch_size=batch_size,
    start_index=0,
    end_index=training_sample_count)

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

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

looking for trends and information in the output of a dataset.

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

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

a basis built on common sense rather than artificial intelligence

computing the Mean Absolute Error (MAE) of the common-sense baseline. The "evaluate_naive_method" function establishes a simple forecasting method by predicting a value based on the final input sequence.

```
def evaluate_naive_method(dataset):
    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}")
print(f"Test MAE: {evaluate_naive_method(test_dataset):.2f}")
```

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

The common-sense baseline is a simple approach that makes the assumption that the temperature will remain the same in 24 hours. The validation MAE of 2.44 degrees Celsius and the test MAE of 2.62 degrees Celsius represent the mean absolute error between the expected and actual temperatures. The MAE of 8.83 degrees Celsius for validation and 8.87 degrees Celsius for testing indicate that when estimating the temperature 24 hours in advance using the common-sense baseline–which presumes a steady temperature–an average variance of about 2.5 degrees Celsius is attained. This suggests that there is an average discrepancy of about 2.5 degrees Celsius between the common-sense baseline's projections and the actual temperature data.

A fundamental machine learning model is the Dense Layer.

Building and evaluating a densely connected model

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

```
inputs = keras.Input(shape=(sequence_length, raw_data.shape[-1]))
x = layers.Flatten()(inputs)
x = layers.Dense(16, activation="relu")(x)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)


callbacks = [
    keras.callbacks.ModelCheckpoint("jena_dense.keras",
                                    save_best_only=True)]


model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])


history = model.fit(train_samples, epochs=10,
                    validation_data = val_dataset, callbacks=callbacks)
```

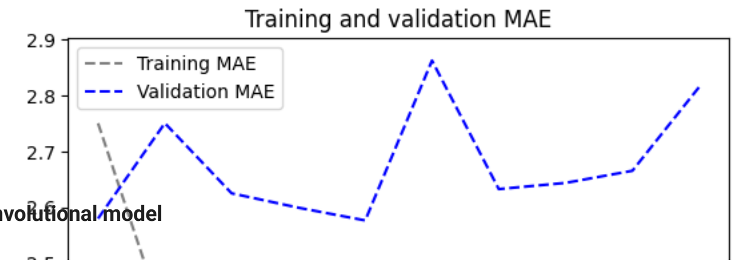
```
Epoch 1/10
819/819 [==============================] - 10s 11ms/step - loss: 12.6471 - mae: 2.7506 - val_loss: 10.7542 - val_mae: 2.5801
Epoch 2/10
819/819 [==============================] - 9s 11ms/step - loss: 9.2546 - mae: 2.3925 - val_loss: 12.1191 - val_mae: 2.7508
Epoch 3/10
819/819 [==============================] - 9s 11ms/step - loss: 8.5611 - mae: 2.3044 - val_loss: 11.0248 - val_mae: 2.6252
Epoch 4/10
819/819 [==============================] - 9s 10ms/step - loss: 8.0672 - mae: 2.2404 - val_loss: 10.6972 - val_mae: 2.5995
Epoch 5/10
819/819 [==============================] - 9s 11ms/step - loss: 7.7178 - mae: 2.1919 - val_loss: 10.5859 - val_mae: 2.5767
Epoch 6/10
819/819 [==============================] - 9s 10ms/step - loss: 7.4411 - mae: 2.1534 - val_loss: 12.8458 - val_mae: 2.8628
Epoch 7/10
819/819 [==============================] - 9s 10ms/step - loss: 7.2468 - mae: 2.1272 - val_loss: 10.9810 - val_mae: 2.6328
Epoch 8/10
819/819 [==============================] - 9s 10ms/step - loss: 7.0612 - mae: 2.0989 - val_loss: 11.0690 - val_mae: 2.6439
Epoch 9/10
819/819 [==============================] - 9s 11ms/step - loss: 6.9155 - mae: 2.0792 - val_loss: 11.2632 - val_mae: 2.6656
Epoch 10/10
819/819 [==============================] - 8s 10ms/step - loss: 6.7914 - mae: 2.0591 - val_loss: 12.5739 - val_mae: 2.8154
```

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

```
405/405 [==============================] - 3s 7ms/step - loss: 11.8603 - mae: 2.7181
Test MAE: 2.72
```

PLOTTING

```
import matplotlib.pyplot as plt
loss = history.history["mae"]
validation_error = history.history["val_mae"]

epochs = range(1, len(loss) + 1)
plt.figure()
plt.plot(epochs, loss, color="grey", linestyle="dashed", label="Training MAE")
plt.plot(epochs, validation_error, color="blue",linestyle="dashed", label="Validation MAE")
plt.title("Training and validation MAE")
plt.xlabel("Epochs")
plt.ylabel("MAE")
plt.legend()
plt.show()
```

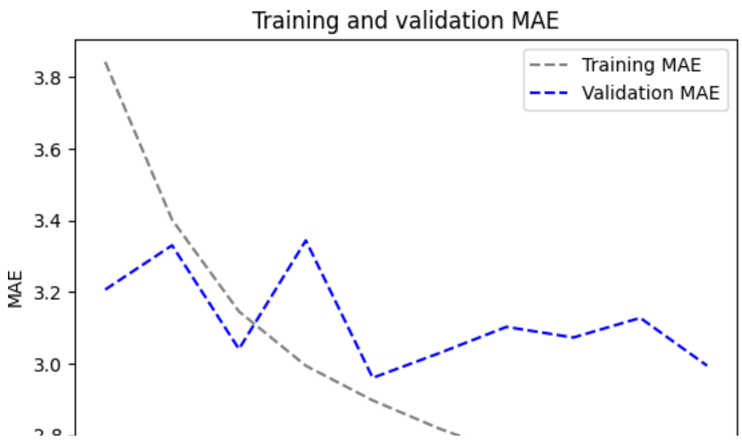


1D convolutional model

```
inputs = keras.Input(shape=(sequence_length, raw_data.shape[-1]))
x = layers.Conv1D(8, 24, activation="relu")(inputs)
x = layers.MaxPooling1D(2)(x)
x = layers.Conv1D(8, 12, activation="relu")(x)
x = layers.MaxPooling1D(2)(x)
x = layers.Conv1D(8, 6, activation="relu")(x)
x = layers.GlobalAveragePooling1D()(x)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)

callbacks = [
    keras.callbacks.ModelCheckpoint("jena_conv.keras",
                                    save_best_only=True)
]
model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(train_samples,
                    epochs=10,
                    validation_data=val_dataset,
                    callbacks=callbacks)

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

```
Epoch 1/10
819/819 [==============================] - 15s 17ms/step - loss: 24.0907 - mae: 3.8420 - val_loss: 16.6355 - val_mae: 3.2067
Epoch 2/10
819/819 [==============================] - 13s 16ms/step - loss: 19.0095 - mae: 3.4016 - val_loss: 17.7197 - val_mae: 3.3302
Epoch 3/10
819/819 [==============================] - 13s 16ms/step - loss: 16.4579 - mae: 3.1457 - val_loss: 15.0676 - val_mae: 3.0408
Epoch 4/10
819/819 [==============================] - 14s 16ms/step - loss: 14.9690 - mae: 2.9943 - val_loss: 17.9949 - val_mae: 3.3439
Epoch 5/10
819/819 [==============================] - 13s 16ms/step - loss: 14.0384 - mae: 2.8975 - val_loss: 14.3401 - val_mae: 2.9605
Epoch 6/10
819/819 [==============================] - 14s 16ms/step - loss: 13.2569 - mae: 2.8185 - val_loss: 14.9526 - val_mae: 3.0297
Epoch 7/10
819/819 [==============================] - 14s 16ms/step - loss: 12.4977 - mae: 2.7509 - val_loss: 15.6282 - val_mae: 3.1027
Epoch 8/10
819/819 [==============================] - 13s 16ms/step - loss: 11.5332 - mae: 2.6741 - val_loss: 15.3343 - val_mae: 3.0730
Epoch 9/10
819/819 [==============================] - 14s 16ms/step - loss: 11.0916 - mae: 2.6231 - val_loss: 15.7738 - val_mae: 3.1279
Epoch 10/10
819/819 [==============================] - 13s 16ms/step - loss: 10.7964 - mae: 2.5908 - val_loss: 14.5987 - val_mae: 2.9944
405/405 [==============================] - 3s 8ms/step - loss: 15.4298 - mae: 3.0796
Test MAE: 3.08
```

```
import matplotlib.pyplot as plt
loss = history.history["mae"]
validation_error = history.history["val_mae"]

epochs = range(1, len(loss) + 1)
plt.figure()
plt.plot(epochs, loss, color="grey", linestyle="dashed", label="Training MAE")
plt.plot(epochs, validation_error, color="blue",linestyle="dashed", label="Validation MAE")
plt.title("Training and validation MAE")
plt.xlabel("Epochs")
plt.ylabel("MAE")
plt.legend()
plt.show()
```



A Simple RNN

A layer of an RNN capable of handling any length of sequence

```
num_features = 14
inputs = keras.Input(shape=(None, num_features))
outputs = layers.SimpleRNN(16)(inputs)

model = keras.Model(inputs, outputs)

callbacks = [
    keras.callbacks.ModelCheckpoint("jena_SimRNN.keras",
                                    save_best_only=True)
]
model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(train_samples,
                    epochs=10,
                    validation_data=val_dataset,
                    callbacks=callbacks)

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

```
Epoch 1/10
819/819 [==============================] - 20s 24ms/step - loss: 138.8040 - mae: 9.7072 - val_loss: 143.8892 - val_mae: 9.8878
Epoch 2/10
819/819 [==============================] - 19s 24ms/step - loss: 136.3773 - mae: 9.5569 - val_loss: 143.7802 - val_mae: 9.8783
Epoch 3/10
819/819 [==============================] - 19s 23ms/step - loss: 136.3101 - mae: 9.5512 - val_loss: 143.7887 - val_mae: 9.8772
Epoch 4/10
819/819 [==============================] - 19s 23ms/step - loss: 136.2598 - mae: 9.5464 - val_loss: 143.6304 - val_mae: 9.8580
Epoch 5/10
819/819 [==============================] - 20s 24ms/step - loss: 136.2858 - mae: 9.5494 - val_loss: 143.6036 - val_mae: 9.8568
Epoch 6/10
819/819 [==============================] - 19s 23ms/step - loss: 136.1999 - mae: 9.5404 - val_loss: 143.5964 - val_mae: 9.8557
Epoch 7/10
819/819 [==============================] - 19s 24ms/step - loss: 136.1950 - mae: 9.5404 - val_loss: 143.6058 - val_mae: 9.8550
Epoch 8/10
819/819 [==============================] - 19s 23ms/step - loss: 136.1778 - mae: 9.5387 - val_loss: 143.6402 - val_mae: 9.8608
Epoch 9/10
819/819 [==============================] - 19s 23ms/step - loss: 136.1872 - mae: 9.5408 - val_loss: 143.5659 - val_mae: 9.8511
Epoch 10/10
819/819 [==============================] - 19s 23ms/step - loss: 136.1420 - mae: 9.5348 - val_loss: 143.5593 - val_mae: 9.8516
405/405 [==============================] - 4s 10ms/step - loss: 151.2890 - mae: 9.9178
Test MAE: 9.92
```

2.Simple RNN - Stacking RNN layers

```
num_features = 14
steps = 120
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)
model = keras.Model(inputs, outputs)

callbacks = [
    keras.callbacks.ModelCheckpoint("jena_SRNN2.keras",
```

```
                                    save_best_only=True)
]
model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(train_samples,
                    epochs=10,
                    validation_data=val_dataset,
                    callbacks=callbacks)

model = keras.models.load_model("jena_SRNN2.keras")
print(f"Test MAE: {model.evaluate(test_dataset)[1]:.2f}")
```

```
Epoch 1/10
819/819 [==============================] - 59s 69ms/step - loss: 136.9978 - mae: 9.5776 - val_loss: 143.4734 - val_mae: 9.8423
Epoch 2/10
819/819 [==============================] - 56s 69ms/step - loss: 135.9836 - mae: 9.5187 - val_loss: 143.4255 - val_mae: 9.8368
Epoch 3/10
819/819 [==============================] - 57s 69ms/step - loss: 135.9145 - mae: 9.5090 - val_loss: 143.4162 - val_mae: 9.8353
Epoch 4/10
819/819 [==============================] - 56s 68ms/step - loss: 135.8775 - mae: 9.5028 - val_loss: 143.4394 - val_mae: 9.8405
Epoch 5/10
819/819 [==============================] - 56s 68ms/step - loss: 135.8553 - mae: 9.4995 - val_loss: 143.4258 - val_mae: 9.8349
Epoch 6/10
819/819 [==============================] - 56s 68ms/step - loss: 135.8438 - mae: 9.4981 - val_loss: 143.4247 - val_mae: 9.8340
Epoch 7/10
819/819 [==============================] - 56s 69ms/step - loss: 135.8262 - mae: 9.4950 - val_loss: 143.4252 - val_mae: 9.8365
Epoch 8/10
819/819 [==============================] - 57s 69ms/step - loss: 135.8211 - mae: 9.4946 - val_loss: 143.4378 - val_mae: 9.8365
Epoch 9/10
819/819 [==============================] - 56s 68ms/step - loss: 135.8087 - mae: 9.4920 - val_loss: 143.4214 - val_mae: 9.8318
Epoch 10/10
819/819 [==============================] - 56s 68ms/step - loss: 135.7930 - mae: 9.4882 - val_loss: 143.4281 - val_mae: 9.8378
405/405 [==============================] - 9s 22ms/step - loss: 151.1356 - mae: 9.9059
Test MAE: 9.91
```

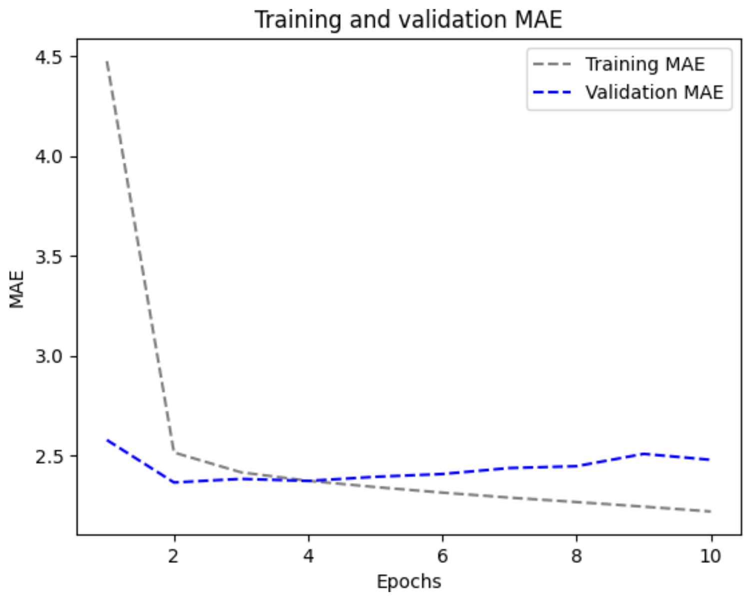
A Simple GRU (Gated Recurrent Unit)

```
inputs = keras.Input(shape=(sequence_length, raw_data.shape[-1]))
x = layers.GRU(16)(inputs)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)

callbacks = [
    keras.callbacks.ModelCheckpoint("jena_gru.keras",
                                    save_best_only=True)
]
model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(train_samples,
                    epochs=10,
                    validation_data=val_dataset,
                    callbacks=callbacks)

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

```
Epoch 1/10
819/819 [==============================] - 49s 57ms/step - loss: 38.4780 - mae: 4.4740 - val_loss: 11.5883 - val_mae: 2.5778
Epoch 2/10
819/819 [==============================] - 48s 58ms/step - loss: 10.4187 - mae: 2.5151 - val_loss: 9.4144 - val_mae: 2.3651
Epoch 3/10
819/819 [==============================] - 47s 58ms/step - loss: 9.5569 - mae: 2.4157 - val_loss: 9.6382 - val_mae: 2.3831
Epoch 4/10
819/819 [==============================] - 47s 58ms/step - loss: 9.2296 - mae: 2.3734 - val_loss: 9.5597 - val_mae: 2.3728
Epoch 5/10
819/819 [==============================] - 47s 58ms/step - loss: 8.9776 - mae: 2.3421 - val_loss: 9.8327 - val_mae: 2.3935
Epoch 6/10
819/819 [==============================] - 47s 58ms/step - loss: 8.7347 - mae: 2.3143 - val_loss: 9.9845 - val_mae: 2.4074
Epoch 7/10
819/819 [==============================] - 48s 58ms/step - loss: 8.5281 - mae: 2.2893 - val_loss: 10.2104 - val_mae: 2.4374
Epoch 8/10
819/819 [==============================] - 47s 58ms/step - loss: 8.3632 - mae: 2.2670 - val_loss: 10.2642 - val_mae: 2.4466
Epoch 9/10
819/819 [==============================] - 47s 57ms/step - loss: 8.1948 - mae: 2.2440 - val_loss: 10.9398 - val_mae: 2.5082
Epoch 10/10
819/819 [==============================] - 47s 58ms/step - loss: 8.0273 - mae: 2.2198 - val_loss: 10.5012 - val_mae: 2.4784
405/405 [==============================] - 8s 18ms/step - loss: 10.4031 - mae: 2.5228
Test MAE: 2.52
```

```
import matplotlib.pyplot as plt
loss = history.history["mae"]
```

```
validation_error = history.history["val_mae"]

epochs = range(1, len(loss) + 1)
plt.figure()
plt.plot(epochs, loss, color="grey", linestyle="dashed", label="Training MAE")
plt.plot(epochs, validation_error, color="blue",linestyle="dashed", label="Validation MAE")
plt.title("Training and validation MAE")
plt.xlabel("Epochs")
plt.ylabel("MAE")
plt.legend()
plt.show()
```



LSTM(Long Short-Term Memory)

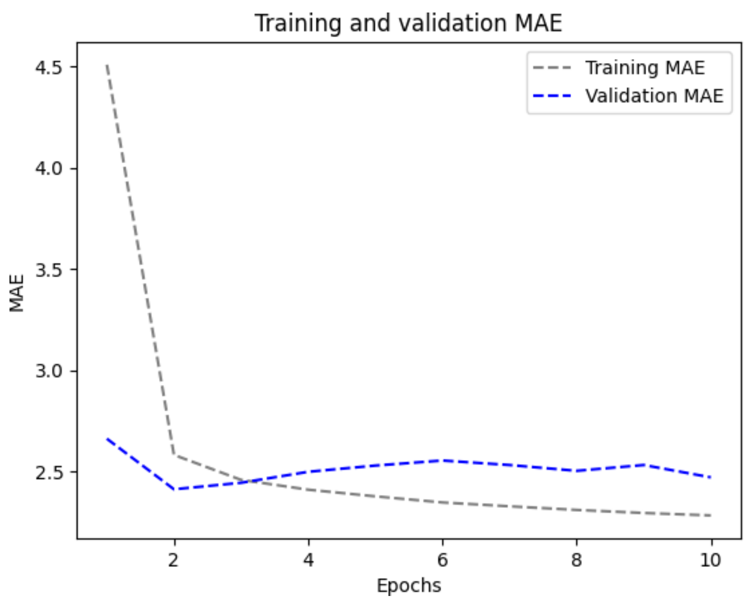
1.LSTM-Simple

```
inputs = keras.Input(shape=(sequence_length, raw_data.shape[-1]))
x = layers.LSTM(16)(inputs)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)

callbacks = [
    keras.callbacks.ModelCheckpoint("jena_lstm.keras",
                                    save_best_only=True)
]
model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(train_samples,
                    epochs=10,
                    validation_data=val_dataset,
                    callbacks=callbacks)

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

```
Epoch 1/10
819/819 [==============================] - 51s 60ms/step - loss: 38.7214 - mae: 4.5075 - val_loss: 12.0747 - val_mae: 2.6621
Epoch 2/10
819/819 [==============================] - 49s 59ms/step - loss: 10.9602 - mae: 2.5821 - val_loss: 9.7205 - val_mae: 2.4125
Epoch 3/10
819/819 [==============================] - 49s 60ms/step - loss: 9.8482 - mae: 2.4589 - val_loss: 10.0488 - val_mae: 2.4446
Epoch 4/10
819/819 [==============================] - 49s 59ms/step - loss: 9.5094 - mae: 2.4106 - val_loss: 10.6286 - val_mae: 2.4988
Epoch 5/10
819/819 [==============================] - 48s 59ms/step - loss: 9.2738 - mae: 2.3776 - val_loss: 10.8954 - val_mae: 2.5295
Epoch 6/10
819/819 [==============================] - 48s 59ms/step - loss: 9.0606 - mae: 2.3475 - val_loss: 10.7639 - val_mae: 2.5544
Epoch 7/10
819/819 [==============================] - 48s 59ms/step - loss: 8.9118 - mae: 2.3285 - val_loss: 10.6657 - val_mae: 2.5322
Epoch 8/10
819/819 [==============================] - 48s 59ms/step - loss: 8.7924 - mae: 2.3107 - val_loss: 10.3997 - val_mae: 2.5036
Epoch 9/10
819/819 [==============================] - 49s 60ms/step - loss: 8.6908 - mae: 2.2955 - val_loss: 10.6887 - val_mae: 2.5323
```

```
Epoch 10/10
819/819 [==============================] - 49s 59ms/step - loss: 8.5973 - mae: 2.2836 - val_loss: 10.1320 - val_mae: 2.4714
405/405 [==============================] - 10s 22ms/step - loss: 10.8881 - mae: 2.5730
Test MAE: 2.57
```

```
import matplotlib.pyplot as plt
loss = history.history["mae"]
validation_error = history.history["val_mae"]

epochs = range(1, len(loss) + 1)
plt.figure()
plt.plot(epochs, loss, color="grey", linestyle="dashed", label="Training MAE")
plt.plot(epochs, validation_error, color="blue",linestyle="dashed", label="Validation MAE")
plt.title("Training and validation MAE")
plt.xlabel("Epochs")
plt.ylabel("MAE")
plt.legend()
plt.show()
```



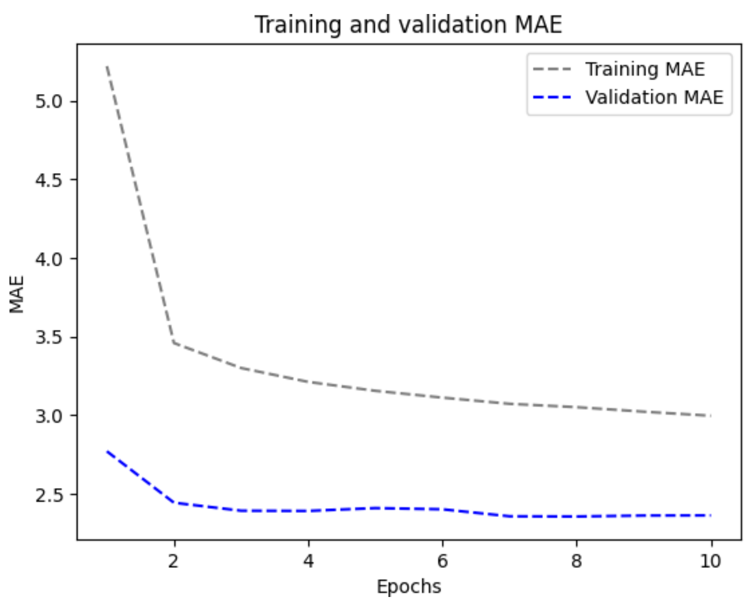
2. LSTM - dropout Regularization

```
inputs = keras.Input(shape=(sequence_length, raw_data.shape[-1]))
x = layers.LSTM(16, recurrent_dropout=0.25)(inputs)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)

callbacks = [
    keras.callbacks.ModelCheckpoint("jena_lstm_dropout.keras",
                                    save_best_only=True)
]
model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(train_samples,
                    epochs=10,
                    validation_data=val_dataset,
                    callbacks=callbacks)

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

```
Epoch 1/10
819/819 [==============================] - 67s 78ms/step - loss: 48.9475 - mae: 5.2180 - val_loss: 13.5574 - val_mae: 2.7706
Epoch 2/10
819/819 [==============================] - 64s 79ms/step - loss: 20.2573 - mae: 3.4586 - val_loss: 9.9276 - val_mae: 2.4437
Epoch 3/10
819/819 [==============================] - 64s 79ms/step - loss: 18.4003 - mae: 3.2998 - val_loss: 9.4476 - val_mae: 2.3930
Epoch 4/10
819/819 [==============================] - 64s 78ms/step - loss: 17.4288 - mae: 3.2116 - val_loss: 9.4263 - val_mae: 2.3916
Epoch 5/10
819/819 [==============================] - 64s 78ms/step - loss: 16.7961 - mae: 3.1550 - val_loss: 9.5583 - val_mae: 2.4096
Epoch 6/10
```

```
819/819 [==============================] - 64s 78ms/step - loss: 16.3914 - mae: 3.1117 - val_loss: 9.5176 - val_mae: 2.4020
Epoch 7/10
819/819 [==============================] - 64s 78ms/step - loss: 15.9436 - mae: 3.0723 - val_loss: 9.1500 - val_mae: 2.3576
Epoch 8/10
819/819 [==============================] - 64s 78ms/step - loss: 15.6992 - mae: 3.0509 - val_loss: 9.1769 - val_mae: 2.3563
Epoch 9/10
819/819 [==============================] - 64s 78ms/step - loss: 15.3724 - mae: 3.0223 - val_loss: 9.1631 - val_mae: 2.3623
Epoch 10/10
819/819 [==============================] - 64s 78ms/step - loss: 15.0742 - mae: 2.9964 - val_loss: 9.1676 - val_mae: 2.3631
405/405 [==============================] - 8s 19ms/step - loss: 10.8827 - mae: 2.6074
Test MAE: 2.61
```

```
import matplotlib.pyplot as plt
loss = history.history["mae"]
validation_error = history.history["val_mae"]

epochs = range(1, len(loss) + 1)
plt.figure()
plt.plot(epochs, loss, color="grey", linestyle="dashed", label="Training MAE")
plt.plot(epochs, validation_error, color="blue",linestyle="dashed", label="Validation MAE")
plt.title("Training and validation MAE")
plt.xlabel("Epochs")
plt.ylabel("MAE")
plt.legend()
plt.show()
```



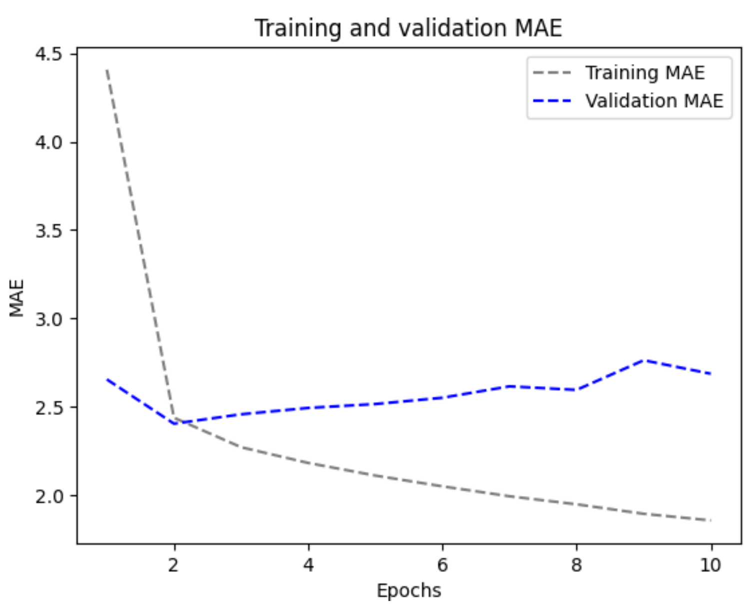
3. LSTM - Stacked setup with 16 units

```
inputs = keras.Input(shape=(sequence_length, raw_data.shape[-1]))
x = layers.LSTM(16, return_sequences=True)(inputs)
x = layers.LSTM(16)(x)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)

callbacks = [
    keras.callbacks.ModelCheckpoint("jena_LSTM_stacked1.keras",
                                    save_best_only=True)
]
model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(train_samples,
                    epochs=10,
                    validation_data=val_dataset,
                    callbacks=callbacks)
model = keras.models.load_model("jena_LSTM_stacked1.keras")
print(f"Test MAE: {model.evaluate(test_dataset)[1]:.2f}")
```

```
Epoch 1/10
819/819 [==============================] - 100s 117ms/step - loss: 37.2840 - mae: 4.4072 - val_loss: 12.1664 - val_mae: 2.6536
Epoch 2/10
819/819 [==============================] - 96s 117ms/step - loss: 9.8683 - mae: 2.4363 - val_loss: 9.5078 - val_mae: 2.4028
Epoch 3/10
```

```
819/819 [==============================] - 96s 117ms/step - loss: 8.4699 - mae: 2.2695 - val_loss: 9.8601 - val_mae: 2.4559
Epoch 4/10
819/819 [==============================] - 96s 117ms/step - loss: 7.8093 - mae: 2.1804 - val_loss: 10.2179 - val_mae: 2.4920
Epoch 5/10
819/819 [==============================] - 96s 117ms/step - loss: 7.3202 - mae: 2.1091 - val_loss: 10.4011 - val_mae: 2.5141
Epoch 6/10
819/819 [==============================] - 96s 117ms/step - loss: 6.9107 - mae: 2.0480 - val_loss: 10.7807 - val_mae: 2.5496
Epoch 7/10
819/819 [==============================] - 96s 117ms/step - loss: 6.5250 - mae: 1.9917 - val_loss: 11.2180 - val_mae: 2.6142
Epoch 8/10
819/819 [==============================] - 96s 117ms/step - loss: 6.2207 - mae: 1.9462 - val_loss: 11.1105 - val_mae: 2.5941
Epoch 9/10
819/819 [==============================] - 96s 117ms/step - loss: 5.8793 - mae: 1.8924 - val_loss: 12.2374 - val_mae: 2.7619
Epoch 10/10
819/819 [==============================] - 96s 117ms/step - loss: 5.6643 - mae: 1.8550 - val_loss: 11.7993 - val_mae: 2.6856
405/405 [==============================] - 18s 42ms/step - loss: 11.4130 - mae: 2.6134
Test MAE: 2.61
```

```
import matplotlib.pyplot as plt
loss = history.history["mae"]
validation_error = history.history["val_mae"]

epochs = range(1, len(loss) + 1)
plt.figure()
plt.plot(epochs, loss, color="grey", linestyle="dashed", label="Training MAE")
plt.plot(epochs, validation_error, color="blue",linestyle="dashed", label="Validation MAE")
plt.title("Training and validation MAE")
plt.xlabel("Epochs")
plt.ylabel("MAE")
plt.legend()
plt.show()
```



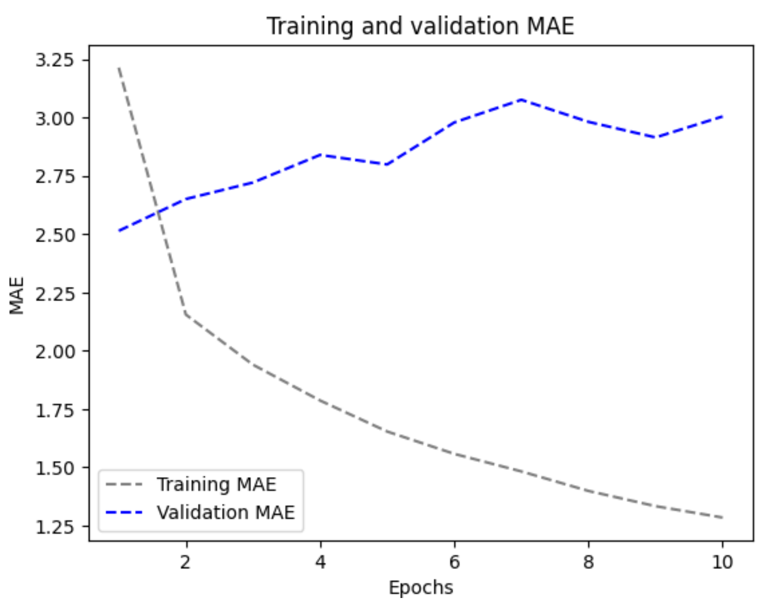
4. LSTM - Stacked setup with 32 units

```
inputs = keras.Input(shape=(sequence_length, raw_data.shape[-1]))
x = layers.LSTM(32, return_sequences=True)(inputs)
x = layers.LSTM(32)(x)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)

callbacks = [
    keras.callbacks.ModelCheckpoint("jena_LSTM_stacked2.keras",
                                    save_best_only=True)
]
model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(train_samples,
                    epochs=10,
                    validation_data=val_dataset,
                    callbacks=callbacks)
```

```
model = keras.models.load_model("jena_LSTM_stacked2.keras")
print(f"Test MAE: {model.evaluate(test_dataset)[1]:.2f}")
```

```
Epoch 1/10
819/819 [==============================] - 147s 175ms/step - loss: 19.9646 - mae: 3.2134 - val_loss: 10.4564 - val_mae: 2.5137
Epoch 2/10
819/819 [==============================] - 143s 174ms/step - loss: 7.7233 - mae: 2.1550 - val_loss: 11.5179 - val_mae: 2.6501
Epoch 3/10
819/819 [==============================] - 143s 175ms/step - loss: 6.3384 - mae: 1.9403 - val_loss: 12.0244 - val_mae: 2.7209
Epoch 4/10
819/819 [==============================] - 143s 174ms/step - loss: 5.4036 - mae: 1.7868 - val_loss: 13.2104 - val_mae: 2.8391
Epoch 5/10
819/819 [==============================] - 142s 173ms/step - loss: 4.6420 - mae: 1.6540 - val_loss: 12.9779 - val_mae: 2.7983
Epoch 6/10
819/819 [==============================] - 143s 175ms/step - loss: 4.1172 - mae: 1.5581 - val_loss: 14.4337 - val_mae: 2.9782
Epoch 7/10
819/819 [==============================] - 143s 174ms/step - loss: 3.7138 - mae: 1.4829 - val_loss: 15.5428 - val_mae: 3.0754
Epoch 8/10
819/819 [==============================] - 143s 174ms/step - loss: 3.3060 - mae: 1.3992 - val_loss: 14.6169 - val_mae: 2.9809
Epoch 9/10
819/819 [==============================] - 143s 174ms/step - loss: 3.0121 - mae: 1.3340 - val_loss: 14.0916 - val_mae: 2.9141
Epoch 10/10
819/819 [==============================] - 142s 174ms/step - loss: 2.8138 - mae: 1.2852 - val_loss: 14.8427 - val_mae: 3.0041
405/405 [==============================] - 28s 66ms/step - loss: 11.8370 - mae: 2.6720
Test MAE: 2.67
```

```
import matplotlib.pyplot as plt
loss = history.history["mae"]
validation_error = history.history["val_mae"]

epochs = range(1, len(loss) + 1)
plt.figure()
plt.plot(epochs, loss, color="grey", linestyle="dashed", label="Training MAE")
plt.plot(epochs, validation_error, color="blue",linestyle="dashed", label="Validation MAE")
plt.title("Training and validation MAE")
plt.xlabel("Epochs")
plt.ylabel("MAE")
plt.legend()
plt.show()
```



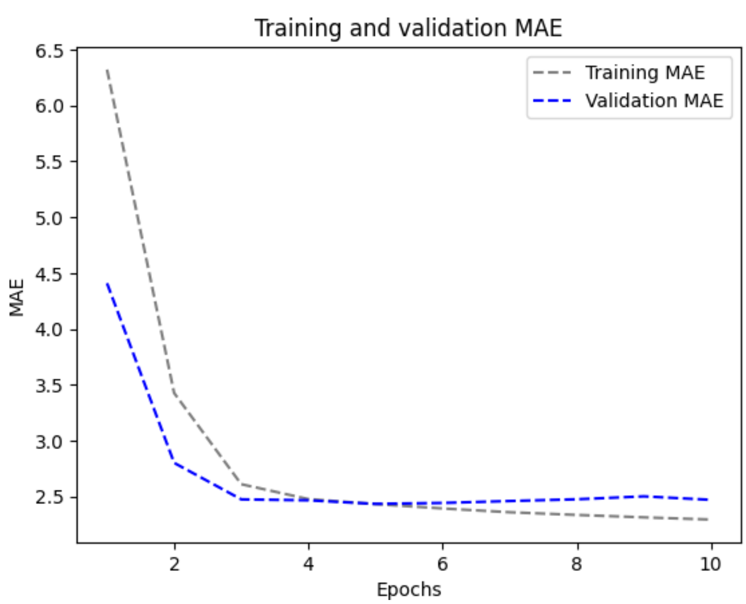
4. LSTM - Stacked setup with 8 units

```
inputs = keras.Input(shape=(sequence_length, raw_data.shape[-1]))
x = layers.LSTM(8, return_sequences=True)(inputs)
x = layers.LSTM(8)(x)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)

callbacks = [
    keras.callbacks.ModelCheckpoint("jena_LSTM_stacked3.keras",
                                    save_best_only=True)
]
```

```
model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(train_samples,
                    epochs=10,
                    validation_data=val_dataset,
                    callbacks=callbacks)
model = keras.models.load_model("jena_LSTM_stacked3.keras")
print(f"Test MAE: {model.evaluate(test_dataset)[1]:.2f}")
```

```
Epoch 1/10
819/819 [==============================] - 83s 97ms/step - loss: 68.0861 - mae: 6.3214 - val_loss: 35.1029 - val_mae: 4.4090
Epoch 2/10
819/819 [==============================] - 78s 96ms/step - loss: 21.3275 - mae: 3.4267 - val_loss: 15.1067 - val_mae: 2.8014
Epoch 3/10
819/819 [==============================] - 78s 96ms/step - loss: 11.3140 - mae: 2.6103 - val_loss: 10.3403 - val_mae: 2.4745
Epoch 4/10
819/819 [==============================] - 78s 95ms/step - loss: 10.0888 - mae: 2.4770 - val_loss: 10.2443 - val_mae: 2.4660
Epoch 5/10
819/819 [==============================] - 78s 95ms/step - loss: 9.7604 - mae: 2.4321 - val_loss: 9.8560 - val_mae: 2.4341
Epoch 6/10
819/819 [==============================] - 79s 96ms/step - loss: 9.4531 - mae: 2.3933 - val_loss: 9.8933 - val_mae: 2.4425
Epoch 7/10
819/819 [==============================] - 79s 97ms/step - loss: 9.1625 - mae: 2.3592 - val_loss: 10.2762 - val_mae: 2.4597
Epoch 8/10
819/819 [==============================] - 79s 96ms/step - loss: 8.9663 - mae: 2.3353 - val_loss: 10.1649 - val_mae: 2.4754
Epoch 9/10
819/819 [==============================] - 78s 96ms/step - loss: 8.7931 - mae: 2.3134 - val_loss: 10.4764 - val_mae: 2.5015
Epoch 10/10
819/819 [==============================] - 78s 95ms/step - loss: 8.6351 - mae: 2.2939 - val_loss: 10.1826 - val_mae: 2.4704
405/405 [==============================] - 15s 33ms/step - loss: 10.6855 - mae: 2.5328
Test MAE: 2.53
```

```
import matplotlib.pyplot as plt
loss = history.history["mae"]
validation_error = history.history["val_mae"]

epochs = range(1, len(loss) + 1)
plt.figure()
plt.plot(epochs, loss, color="grey", linestyle="dashed", label="Training MAE")
plt.plot(epochs, validation_error, color="blue",linestyle="dashed", label="Validation MAE")
plt.title("Training and validation MAE")
plt.xlabel("Epochs")
plt.ylabel("MAE")
plt.legend()
plt.show()
```



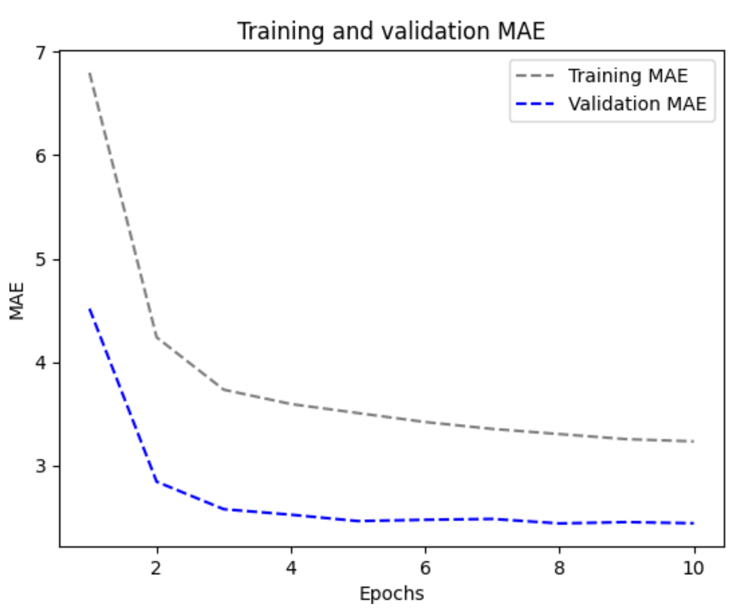
5. LSTM - dropout-regularized, stacked model

```
inputs = keras.Input(shape=(sequence_length, raw_data.shape[-1]))
x = layers.LSTM(8, recurrent_dropout=0.5, return_sequences=True)(inputs)
x = layers.LSTM(8, recurrent_dropout=0.5)(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1)(x)
```

```
model = keras.Model(inputs, outputs)

callbacks = [
    keras.callbacks.ModelCheckpoint("jena_stacked_LSTM_dropout.keras",
                                    save_best_only=True)
]
model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(train_samples,
                    epochs=10,
                    validation_data=val_dataset,
                    callbacks=callbacks)
model = keras.models.load_model("jena_stacked_LSTM_dropout.keras")
print(f"Test MAE: {model.evaluate(test_dataset)[1]:.2f}")
```

```
Epoch 1/10
819/819 [==============================] - 116s 136ms/step - loss: 77.8257 - mae: 6.7982 - val_loss: 36.7763 - val_mae: 4.5160
Epoch 2/10
819/819 [==============================] - 111s 136ms/step - loss: 32.2162 - mae: 4.2387 - val_loss: 14.3888 - val_mae: 2.8423
Epoch 3/10
819/819 [==============================] - 112s 136ms/step - loss: 24.3356 - mae: 3.7321 - val_loss: 11.2612 - val_mae: 2.5764
Epoch 4/10
819/819 [==============================] - 112s 136ms/step - loss: 22.4220 - mae: 3.5936 - val_loss: 10.6822 - val_mae: 2.5232
Epoch 5/10
819/819 [==============================] - 112s 137ms/step - loss: 21.3244 - mae: 3.5059 - val_loss: 10.0708 - val_mae: 2.4614
Epoch 6/10
819/819 [==============================] - 113s 137ms/step - loss: 20.2979 - mae: 3.4194 - val_loss: 10.1543 - val_mae: 2.4741
Epoch 7/10
819/819 [==============================] - 112s 137ms/step - loss: 19.4256 - mae: 3.3529 - val_loss: 10.1505 - val_mae: 2.4824
Epoch 8/10
819/819 [==============================] - 112s 137ms/step - loss: 18.7569 - mae: 3.3030 - val_loss: 9.7592 - val_mae: 2.4390
Epoch 9/10
819/819 [==============================] - 113s 138ms/step - loss: 18.2290 - mae: 3.2538 - val_loss: 9.8178 - val_mae: 2.4518
Epoch 10/10
819/819 [==============================] - 113s 138ms/step - loss: 17.9255 - mae: 3.2323 - val_loss: 9.7635 - val_mae: 2.4406
405/405 [==============================] - 12s 28ms/step - loss: 10.9811 - mae: 2.5801
Test MAE: 2.58
```

```
import matplotlib.pyplot as plt
loss = history.history["mae"]
validation_error = history.history["val_mae"]

epochs = range(1, len(loss) + 1)
plt.figure()
plt.plot(epochs, loss, color="grey", linestyle="dashed", label="Training MAE")
plt.plot(epochs, validation_error, color="blue",linestyle="dashed", label="Validation MAE")
plt.title("Training and validation MAE")
plt.xlabel("Epochs")
plt.ylabel("MAE")
plt.legend()
plt.show()
```

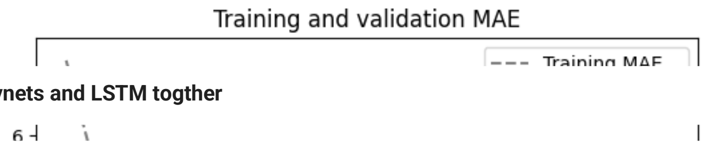


Bidirectional LSTM

```
inputs = keras.Input(shape=(sequence_length, raw_data.shape[-1]))
x = layers.LSTM(8, recurrent_dropout=0.5, return_sequences=True)(inputs)
x = layers.LSTM(8, recurrent_dropout=0.5)(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)

callbacks = [
    keras.callbacks.ModelCheckpoint("jena_stacked_LSTM_dropout.keras",
                                    save_best_only=True)
]
model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(training_data,
                    epochs=10,
                    validation_data=val_dataset,
                    callbacks=callbacks)
model = keras.models.load_model("jena_stacked_LSTM_dropout.keras")
print(f"Test MAE: {model.evaluate(test_dataset)[1]:.2f}")
```

```
Epoch 1/10
819/819 [==============================] - 118s 139ms/step - loss: 76.4131 - mae: 6.7332 - val_loss: 37.2751 - val_mae: 4.5627
Epoch 2/10
819/819 [==============================] - 113s 137ms/step - loss: 32.9669 - mae: 4.2902 - val_loss: 14.3714 - val_mae: 2.8133
Epoch 3/10
819/819 [==============================] - 113s 138ms/step - loss: 24.8019 - mae: 3.7681 - val_loss: 11.2731 - val_mae: 2.5588
Epoch 4/10
819/819 [==============================] - 113s 137ms/step - loss: 22.5858 - mae: 3.6063 - val_loss: 10.5699 - val_mae: 2.5025
Epoch 5/10
819/819 [==============================] - 112s 137ms/step - loss: 21.3166 - mae: 3.5045 - val_loss: 9.7225 - val_mae: 2.3978
Epoch 6/10
819/819 [==============================] - 113s 138ms/step - loss: 20.3250 - mae: 3.4295 - val_loss: 9.4756 - val_mae: 2.3669
Epoch 7/10
819/819 [==============================] - 113s 138ms/step - loss: 19.4387 - mae: 3.3616 - val_loss: 9.5240 - val_mae: 2.3797
Epoch 8/10
819/819 [==============================] - 113s 138ms/step - loss: 18.7158 - mae: 3.3000 - val_loss: 9.3367 - val_mae: 2.3607
Epoch 9/10
819/819 [==============================] - 113s 138ms/step - loss: 18.1702 - mae: 3.2488 - val_loss: 9.3664 - val_mae: 2.3601
Epoch 10/10
819/819 [==============================] - 113s 138ms/step - loss: 17.8043 - mae: 3.2170 - val_loss: 9.5666 - val_mae: 2.3820
405/405 [==============================] - 12s 28ms/step - loss: 10.8555 - mae: 2.5616
Test MAE: 2.56
```

```
import matplotlib.pyplot as plt
loss = history.history["mae"]
validation_error = history.history["val_mae"]

epochs = range(1, len(loss) + 1)
plt.figure()
plt.plot(epochs, loss, color="grey", linestyle="dashed", label="Training MAE")
plt.plot(epochs, validation_error, color="blue",linestyle="dashed", label="Validation MAE")
plt.title("Training and validation MAE")
plt.xlabel("Epochs")
plt.ylabel("MAE")
plt.legend()
plt.show()
```

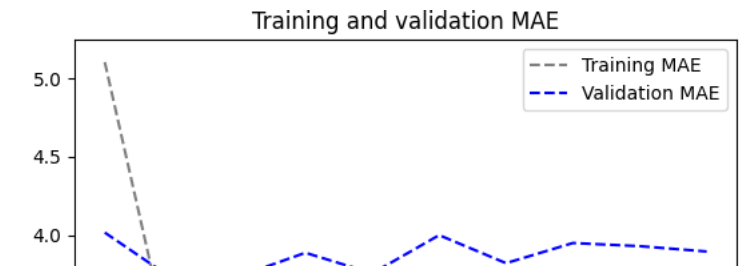


1D Convnets and LSTM togther

```
inputs = keras.Input(shape=(sequence_length, raw_data.shape[-1]))
x = layers.Conv1D(64, 3, activation='relu')(inputs)
x = layers.MaxPooling1D(3)(x)
x = layers.Conv1D(128, 3, activation='relu')(x)
x = layers.GlobalMaxPooling1D()(x)
x = layers.Reshape((-1, 128))(x)  # Reshape the data to be 3D
x = layers.LSTM(16)(x)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)

model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])

callbacks = [
    keras.callbacks.ModelCheckpoint("jena_Conv_LSTM.keras", save_best_only=True)
]

history = model.fit(training_data, epochs=10, validation_data=val_dataset, callbacks=callbacks)

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

```
Epoch 1/10
819/819 [==============================] - 19s 20ms/step - loss: 46.7288 - mae: 5.1012 - val_loss: 26.7469 - val_mae: 4.0160
Epoch 2/10
819/819 [==============================] - 16s 19ms/step - loss: 17.4231 - mae: 3.2207 - val_loss: 21.7069 - val_mae: 3.7261
Epoch 3/10
819/819 [==============================] - 16s 19ms/step - loss: 14.3686 - mae: 2.9403 - val_loss: 21.9876 - val_mae: 3.7359
Epoch 4/10
819/819 [==============================] - 15s 19ms/step - loss: 12.7082 - mae: 2.7624 - val_loss: 24.1101 - val_mae: 3.8873
Epoch 5/10
819/819 [==============================] - 16s 19ms/step - loss: 11.5917 - mae: 2.6308 - val_loss: 22.2855 - val_mae: 3.7561
Epoch 6/10
819/819 [==============================] - 15s 19ms/step - loss: 10.7314 - mae: 2.5237 - val_loss: 24.4376 - val_mae: 3.9995
Epoch 7/10
819/819 [==============================] - 16s 19ms/step - loss: 10.0606 - mae: 2.4404 - val_loss: 22.9792 - val_mae: 3.8209
Epoch 8/10
819/819 [==============================] - 16s 19ms/step - loss: 9.5435 - mae: 2.3689 - val_loss: 24.0852 - val_mae: 3.9494
Epoch 9/10
819/819 [==============================] - 15s 19ms/step - loss: 9.0646 - mae: 2.3039 - val_loss: 25.1463 - val_mae: 3.9293
Epoch 10/10
819/819 [==============================] - 16s 19ms/step - loss: 8.6630 - mae: 2.2482 - val_loss: 23.6115 - val_mae: 3.8953
405/405 [==============================] - 4s 8ms/step - loss: 24.8048 - mae: 3.9673
Test MAE: 3.97
```

```
import matplotlib.pyplot as plt
loss = history.history["mae"]
validation_error = history.history["val_mae"]

epochs = range(1, len(loss) + 1)
plt.figure()
plt.plot(epochs, loss, color="grey", linestyle="dashed", label="Training MAE")
plt.plot(epochs, validation_error, color="blue",linestyle="dashed", label="Validation MAE")
plt.title("Training and validation MAE")
plt.xlabel("Epochs")
plt.ylabel("MAE")
plt.legend()
plt.show()
```

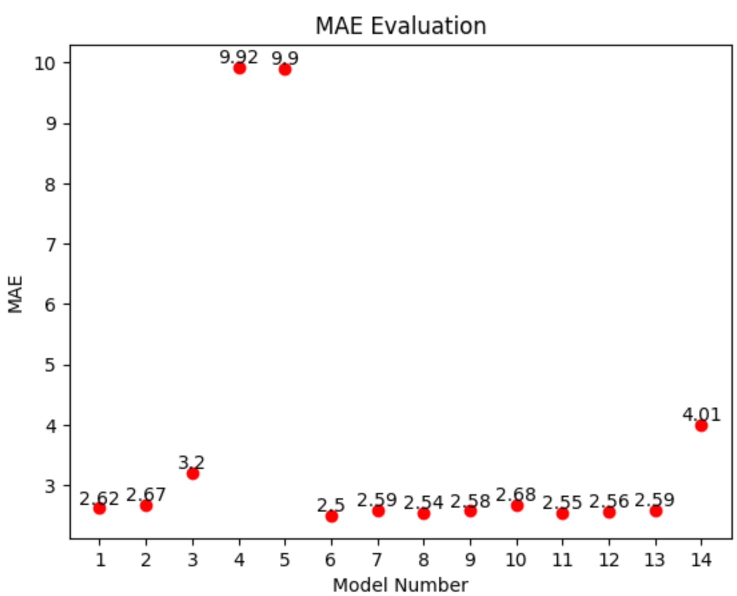


We built thirteen models: The details are as follows:

Model 1: not machine learning, but a baseline of common sense Model 2: A basic machine learning model Model 3: First-order convolutional model Model 4: A simple RNN layer that can process sequences of arbitrary length Model 5: Simple RNN - stacked RNN layers A Simple Gated Recurrent Unit (GRU) is Model 6. Model 7: Dropout Regularization Model 8 - Simple Long Short-Term Memory Model 9: stacked 16-unit LSTM layout

Model 10: stacked 32-unit LSTM arrangement Model 11: Stacking arrangement of eight LSTM units Model 12: Regularized dropout, LSTM, eight units stacked Model 13 Bidirectional LSTM Model 14: Integrating 1D Convolutions with LSTM

```
Models = ("1","2","3","4","5","6","7","8","9","10","11","12","13","14")
Mae = (2.62,2.67,3.2,9.92,9.9,2.5,2.59,2.54,2.58,2.68,2.55,2.56,2.59,4.01)

# MAE Evaluation
plt.scatter(Models, Mae, color="red")
plt.title("MAE Evaluation")
plt.xlabel("Model Number")
plt.ylabel("MAE")

for (xi, yi) in zip(Models,Mae):
    plt.text(xi, yi, yi, va='bottom', ha='center')

plt.show()
```



In summary, fourteen models were developed throughout the course of our study. An 2.62.Next, a straightforward machine learning model with a dense layer was created; this produced an MAE of 2.72, which is slightly higher. The first reasonable baseline, which expected temp_data to be precisely equal to the present value, was used to calculate MAE. Later on. The flattened timeseries removed time-related data, which resulted in the poor performance of the thick layer. Because the convolution model rearranged the order of the information and treated all data segments equally, it produced poor results.

When we realized that certain designs were needed for time series data, we turned to Recurrent Neural Networks (RNNs). The vanishing gradient problem limited SimpleRNN's applicability even though it was theoretically sound. LSTM and GRU RNNs were used to address this; basic GRU fared better than other models due to its greater ability to capture long-range relationships. Six different models with different units were tested on Long Short-Term Memory (LSTMs), a popular model for time series data. Dependable 8-unit LSTM fared better than the others. An examination of six models with varying units was conducted on LSTMs, which are widely recognized for time series data. An 8-unit LSTM regularly outperformed other models, with MAE values that were lower than those of the common sense model.

Techniques like recurrent dropout and bidirectional data helped to further improve accuracy. Convolution's limitations in terms of upsetting information order are highlighted by the subpar 3.97 MAE that results from combining 1D convolution with RNN. To summarise, the study highlights the importance of selecting architectures that align with data attributes to attain optimal model performance.

Recommendation: From what I've seen, simple RNNs struggle to capture long-term relationships because of the vanishing gradient problem. To address these issues, the adoption of advanced RNN designs such as LSTM and GRU is recommended. Tests indicate that while GRU would be a better option, LSTM is still a common option for time series data. I suggest that the number of units in stacked recurrent layers, bidirectional data, and recurrent dropout rate be used as hyperparameters to maximize GRU.

My analysis indicates that the combined performance of 1D convolution and RNN was not as good as it may have been. Considering the convolutional method's limitations in terms of disrupting the information order in time series data, focus should be placed on architectures meant for sequential data, like pure RNNs.