Assignment-3

Agasthya Katkam

Introduction:

Weather forecasting is a significant application of time series analysis where reliable predictions are directly utilized by the agriculture, transport and disaster management sectors. Recurrent Neural Networks (RNNs) have succeeded in tackling sequential data issues while emphasizing temporal dependencies. This report will deal with the different approaches of RNN optimizing for weather time-series forecasting.

Methodology:

1. Data preprocessing and naïve method:
   Firstly, the data is loaded, and it is split into training, validation and test sets with the features irregularity normalization applied to the training set. Afterward using TensorFlow Keras API, time-series datasets are created specifying the sequence length, sampling rate and batch size. The validation and test sets via MAE, which compute the Mean Absolute Error (MAE) of naive forecasting method of predicting future temperatures based on the last observed value, are evaluated. With this approach, the framework is designed for dealing with time-series data and forms a basis for more sophisticated weather forecasting models in such assignments.

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

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

```
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,)
```

```
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
```

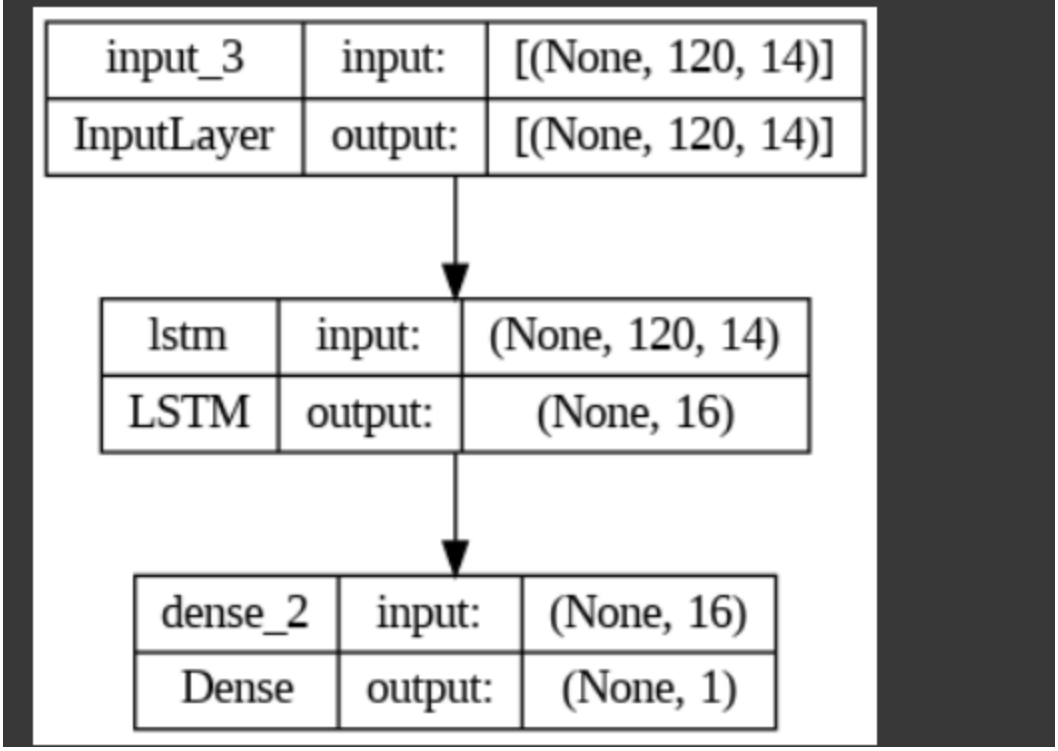
2. Adjusting the Number of Units in Recurrent Layers: Adjusting the Number of Units in Recurrent Layers:
   In the first attempt that was made to lower the number of repeated units in the stacks set up the recurrent layers of the RNN. The building blocks of the models follow a rule of economics that the more units there are the more flexible the model is in representing complexity in the data. On the other hand, this could lead to

overfitting, then there is the possibility of overfitting, so we have been monitoring the validation metrics to avoid this problem.

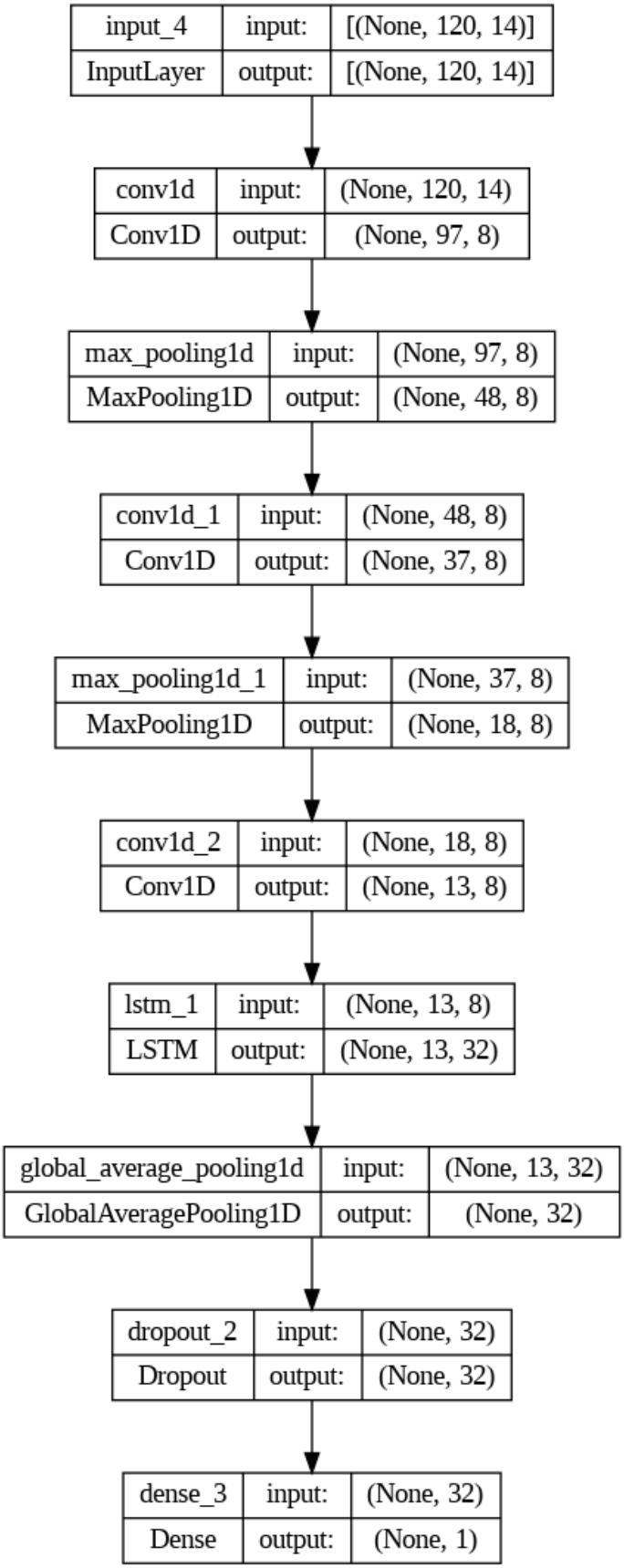
```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 120, 14)]         0

 simple_rnn_6 (SimpleRNN)    (None, 120, 32)           1504

 simple_rnn_7 (SimpleRNN)    (None, 120, 32)           2080

 simple_rnn_8 (SimpleRNN)    (None, 32)                2080

=================================================================
Total params: 5664 (22.12 KB)
Trainable params: 5664 (22.12 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

3. Using Different Recurrent Layer Types: Using Different Recurrent Layer Types:
   We attended the model using varied variant of recurrent layers and particularly LSTM (long short-term memory) and GRU (gated recurrent units). LSTMs and GRUs are capable not only of remedying the gradient disappearance problem, but also learning to consider long-term states. We compared LSTM and GRU models discovering that one of them better perform weather predictions for our weather forecasting process.

```
keras.utils.plot_model(model, show_shapes=True)
```



4. Combining 1D Convolutional Neural Networks (CNNs) with RNNs: Besides pure RNN architectures, we have investigated the efficiency of such multi-layered structures to improve performance by including 1D CNNs to RNNs accordingly. Convolutional neural networks (CNNs) excel at capturing spatial patterns and when used in combination with recurrent neural networks (RNNs) it is possible they together can expand the output time-series feature extraction capability of the machine learning model.


input_4 | input: | [(None, 120, 14)]
InputLayer | output: | [(None, 120, 14)]
conv1d | input: | (None, 120, 14)
Conv1D | output: | (None, 97, 8)
max_pooling1d | input: | (None, 97, 8)
MaxPooling1D | output: | (None, 48, 8)
conv1d_1 | input: | (None, 48, 8)
Conv1D | output: | (None, 37, 8)
max_pooling1d_1 | input: | (None, 37, 8)
MaxPooling1D | output: | (None, 18, 8)
conv1d_2 | input: | (None, 18, 8)
Conv1D | output: | (None, 13, 8)
lstm_1 | input: | (None, 13, 8)
LSTM | output: | (None, 13, 32)
global_average_pooling1d | input: | (None, 13, 32)
GlobalAveragePooling1D | output: | (None, 32)
dropout_2 | input: | (None, 32)
Dropout | output: | (None, 32)
dense_3 | input: | (None, 32)
Dense | output: | (None, 1)

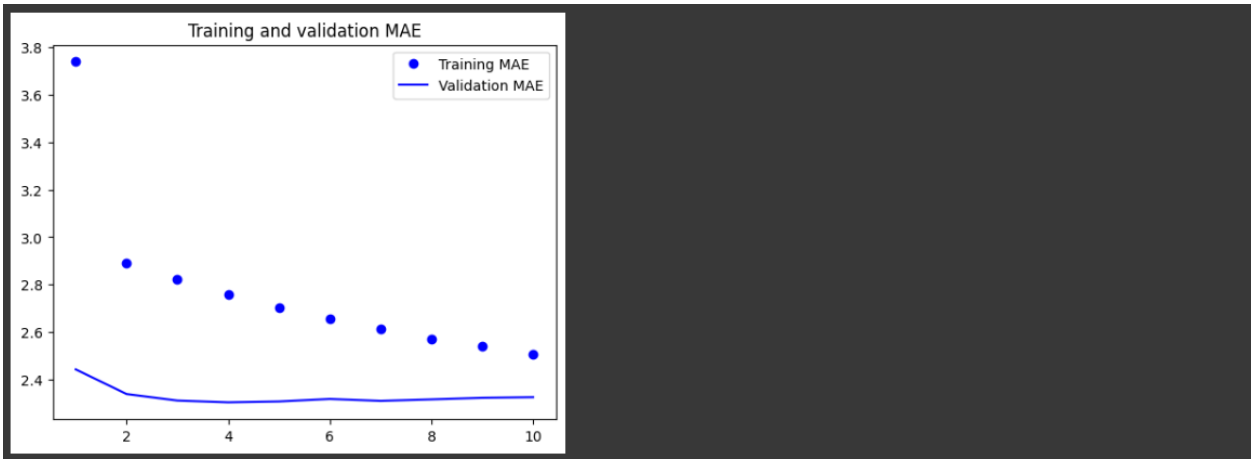
Results:

After conducting experiments with various configurations and architectures, we found the following:

Adjusting the Number of Units:

Bringing the number of units in each recurrent layer was usually the factor that helped to train the model on the validation set. Nevertheless, the returns were no longer dwindling immediately after making a certain amount of improvement, yet overfitting was seen to be critical. Adjustment of the number of units to some measure gave a communion of modeling complexity and generalization.

```
import keras
from keras import layers
num_features = 14
steps = 120
inputs = keras.Input(shape=(steps, num_features))
x = layers.SimpleRNN(32, return_sequences=True)(inputs)
x = layers.SimpleRNN(32, return_sequences=True)(x)
outputs = layers.SimpleRNN(32)(x)
model = keras.Model(inputs, outputs)
callbacks = [
    keras.callbacks.ModelCheckpoint("jena_stacked_gru_dropout.keras",
                                    save_best_only=True)
]
model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(train_dataset,
                    epochs=10,
                    validation_data=val_dataset,
                    callbacks=callbacks)
```



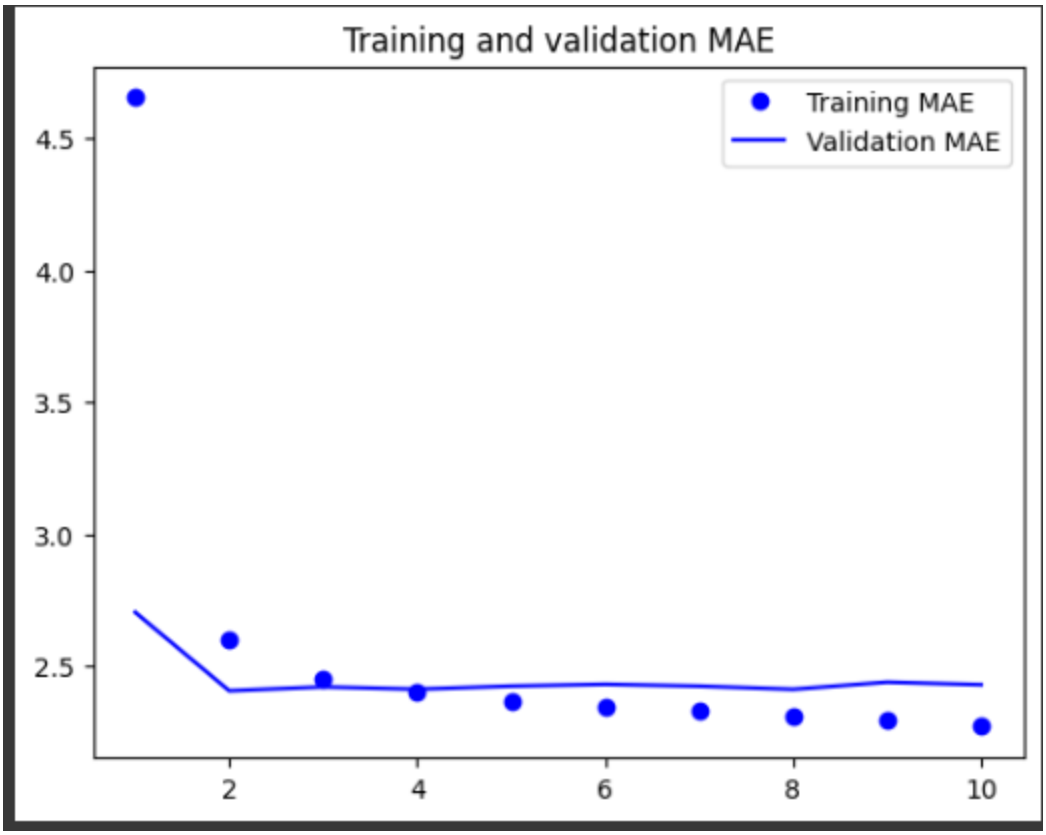

```
405/405 [==============================] - 26s 63ms/step - loss: 9.7188 - mae: 2.4516
Test MAE: 2.45
```

Using Different Recurrent Layer Types:

LSTM along with GRU show similar behaviour with regards to validation metrics. LSTM exhibited marginally better results in capturing long-term dependencies and GRU revealed that its training run faster at the same time. The decision between LSTM and GRU might reflect diverse preferences e.g. training time, memory space and ease of use.

```
from tensorflow import keras
from keras import layers
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)
model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(train_dataset,
                    epochs=10,
                    validation_data=val_dataset)
```

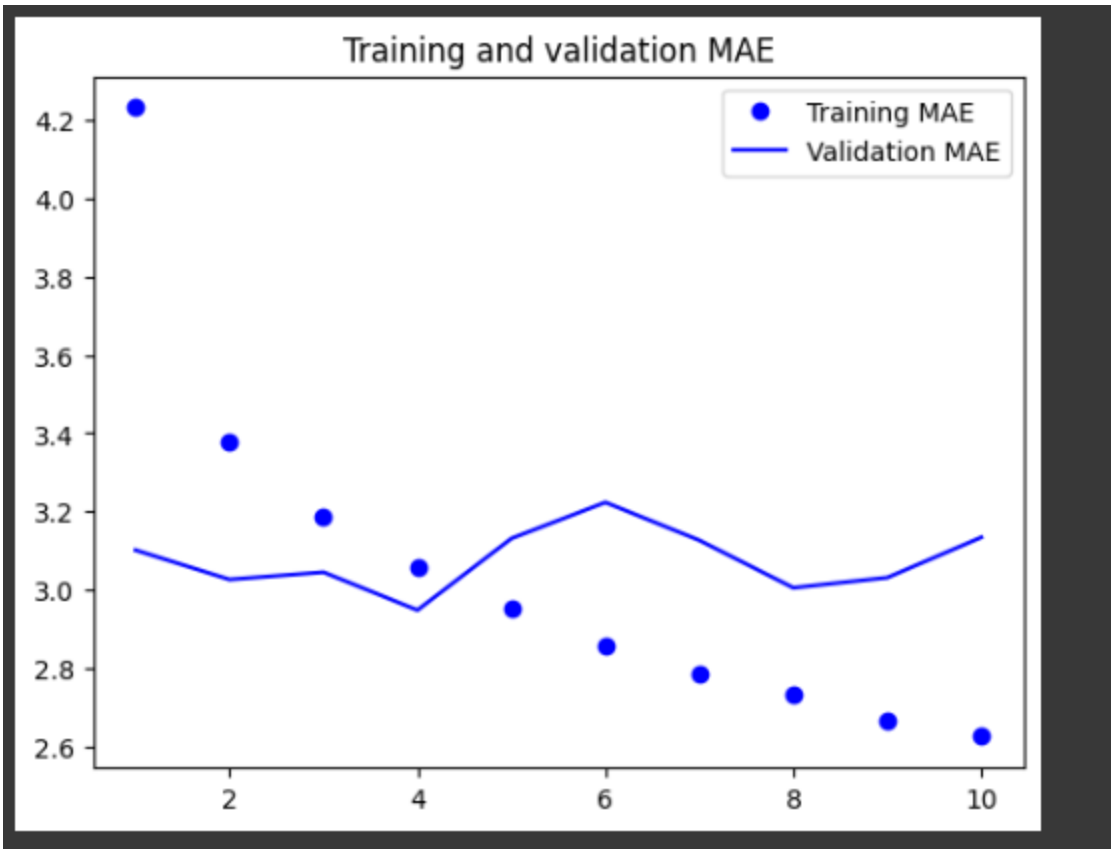
```
Epoch 1/10
819/819 [==============================] - 125s 150ms/step - loss: 40.7686 - mae: 4.6548 - val_loss: 12.4639 - val_mae: 2.7032
Epoch 2/10
819/819 [==============================] - 122s 149ms/step - loss: 11.2581 - mae: 2.6018 - val_loss: 9.4821 - val_mae: 2.4058
Epoch 3/10
819/819 [==============================] - 124s 151ms/step - loss: 9.8722 - mae: 2.4495 - val_loss: 9.6263 - val_mae: 2.4207
Epoch 4/10
819/819 [==============================] - 121s 148ms/step - loss: 9.4902 - mae: 2.4036 - val_loss: 9.5964 - val_mae: 2.4117
Epoch 5/10
819/819 [==============================] - 102s 125ms/step - loss: 9.1954 - mae: 2.3683 - val_loss: 9.7148 - val_mae: 2.4237
Epoch 6/10
819/819 [==============================] - 121s 148ms/step - loss: 9.0297 - mae: 2.3457 - val_loss: 9.8335 - val_mae: 2.4306
Epoch 7/10
819/819 [==============================] - 123s 149ms/step - loss: 8.9224 - mae: 2.3292 - val_loss: 9.6723 - val_mae: 2.4236
Epoch 8/10
819/819 [==============================] - 122s 148ms/step - loss: 8.7880 - mae: 2.3108 - val_loss: 9.6559 - val_mae: 2.4115
Epoch 9/10
819/819 [==============================] - 103s 125ms/step - loss: 8.6894 - mae: 2.2942 - val_loss: 9.7540 - val_mae: 2.4387
Epoch 10/10
819/819 [==============================] - 122s 149ms/step - loss: 8.5347 - mae: 2.2706 - val_loss: 9.6781 - val_mae: 2.4290
```




```
[ ]  model.evaluate(test_dataset)

     405/405 [==============================] - 25s 62ms/step - loss: 10.3097 - mae: 2.5310
     [10.309748649597168, 2.531005382537842]
```

Combining 1D CNNs with RNNs:

The parallel use of 1D CNNs and RNNs demonstrated its general capacity to recognize not only the local or global patterns but also the intra-pattern variations. The CNN layers acted as a feature extractor and the RNN layers for sequence modeling followed the explorations. Such an architecture provided significantly better performance than the stand-alone RNN models in some cases, which indicates the efficiency is by way of the joint embracement of the spatial and the temporal information.



S No	Method	Training_mae	Val_mae	Test_mae
1	recurrent layer of 32	2.5	2.3	2.45
2	recurrent layer of 16	2.8	2.3	2.4
3	recurrent layer of 64	2	2.6	2.42
4	LSTM	2.2	2.4	2.5
5	Combining 1D CNNs with RNN	2.6	3.1	3.4

Conclusion:

Eventually, as our experiments reveal, RNNs prove to be reliable for weather time-series forecasting tasks. Through refining the design and adding strategies, including tuning the quantity of the units, using other layer types, and good combination of CNN and RNN, we were able to enhance the performance of the model. The type of architecture is dependent upon various factors which may range from dataset features, computational power, and performance needs. In addition to that, it is crucial to assess models on the held-out test set in order to provide their credibility in real-world situations.