Introduction:

The objective of this task is to implement Recurrent Neural Networks (RNNs) on a weather time-series dataset and enhance its effectiveness. The goal is to improve the accuracy of weather pattern forecasting, specifically focusing on temperature variations, through the exploration of different deep learning methods. This task presents techniques for addressing temporal relationships in the data and enhancing model predictions by adjusting and integrating various neural network layers.

Background:

Forecasting future weather variables in meteorology through time-series analysis is done by using past data as a basis. Because of their capacity to maintain temporal information throughout sequences, Recurrent Neural Networks (RNNs) are well-suited for this task. Yet, difficulties are encountered in optimizing models and achieving accuracy because of the intricate nature of weather data. This task explores methods such as Long Short-Term Memory (LSTM) networks, 1D convolutional layers, and modifications in model structure to improve model effectiveness.

Problem Statement:

The task of forecasting weather variables, such as temperature, poses unique challenges due to the inherent temporal dependencies in the data. Traditional time-series methods may struggle to capture the complex and dynamic patterns found in meteorological data. This assignment aims to leverage Recurrent Neural Networks (RNNs) to improve weather time-series forecasting accuracy. The primary objectives are:

1. To develop and evaluate RNN models, including GRU and LSTM architectures, for weather data forecasting.
2. To explore the effectiveness of hybrid models that combine convolutional layers with RNNs to capture both spatial and temporal patterns.
3. To optimize model performance by adjusting the network structure, layer types, and hyperparameters, with the goal of minimizing validation mean absolute error (MAE).

By addressing these objectives, this assignment seeks to identify the optimal deep learning architecture for accurate temperature predictions in weather forecasting, contributing to advancements in time-series analysis and meteorological predictions.

Methodology:

1. **Data Preprocessing**:
    - The weather dataset is preprocessed by normalizing the temperature values to prepare them for modeling.
    - Data is divided into training, validation, and test sets, with time series windows created for model input.
2. **Model Architecture**:

- **Model Variants**: Multiple RNN architectures are explored:
    - **Baseline GRU Model**: An initial model using Gated Recurrent Units (GRU) to establish baseline performance.
    - **LSTM-based Model**: The GRU layers are replaced with LSTM layers to examine potential improvements.
    - **Hybrid CNN-RNN Model**: A model combining 1D convolutional layers with LSTM or GRU to capture spatial and temporal dependencies.

3. **Hyperparameter Tuning**:

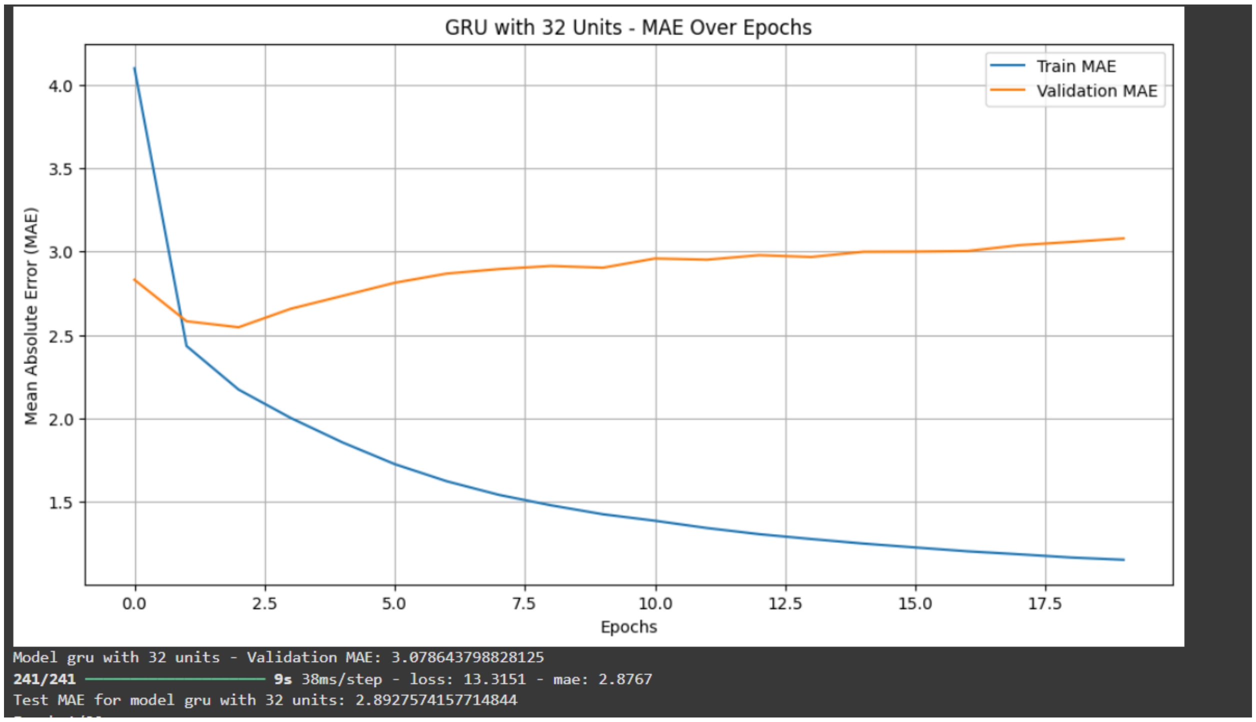
- The number of units in each RNN layer and convolutional filters are adjusted to optimize the validation mean absolute error (MAE).
- Early stopping and dropout regularization are used to prevent overfitting.

4. **Evaluation**:

- Validation MAE is used to assess model performance during training.
- The best-performing models based on validation MAE are further evaluated on the test set for final performance metrics.

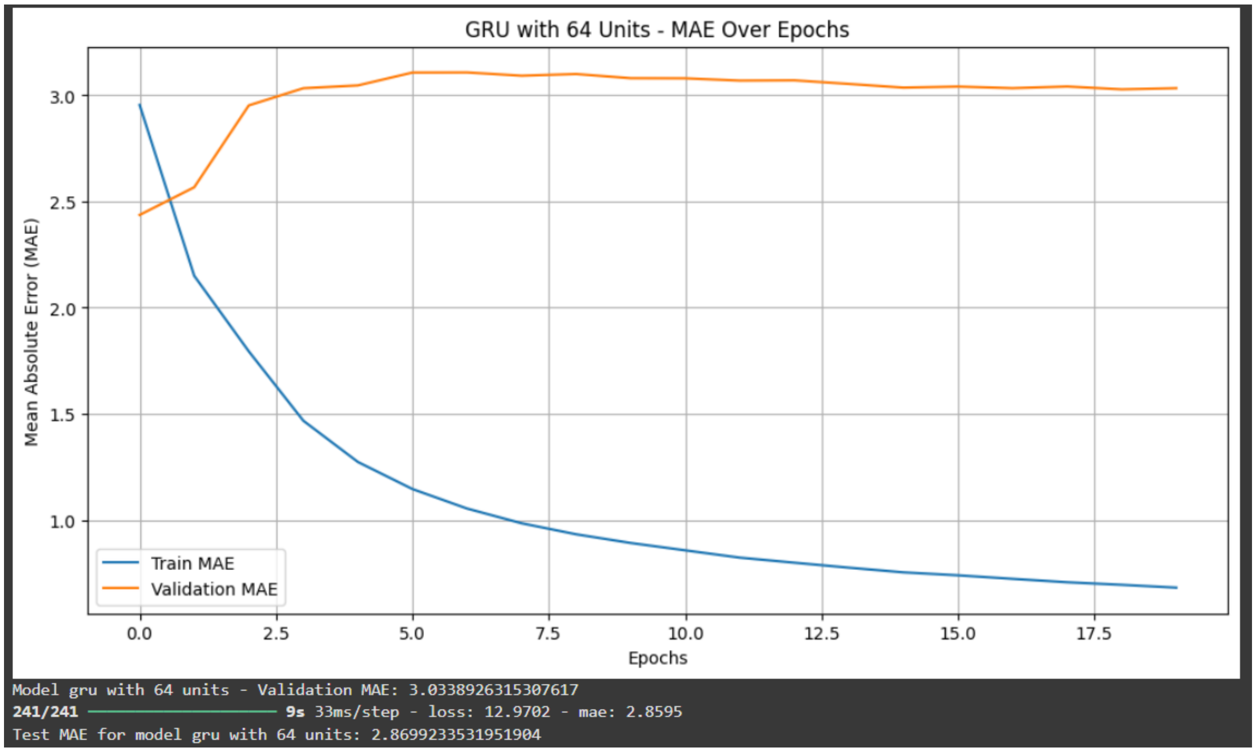
Results:

GRU model with 32 hidden units:



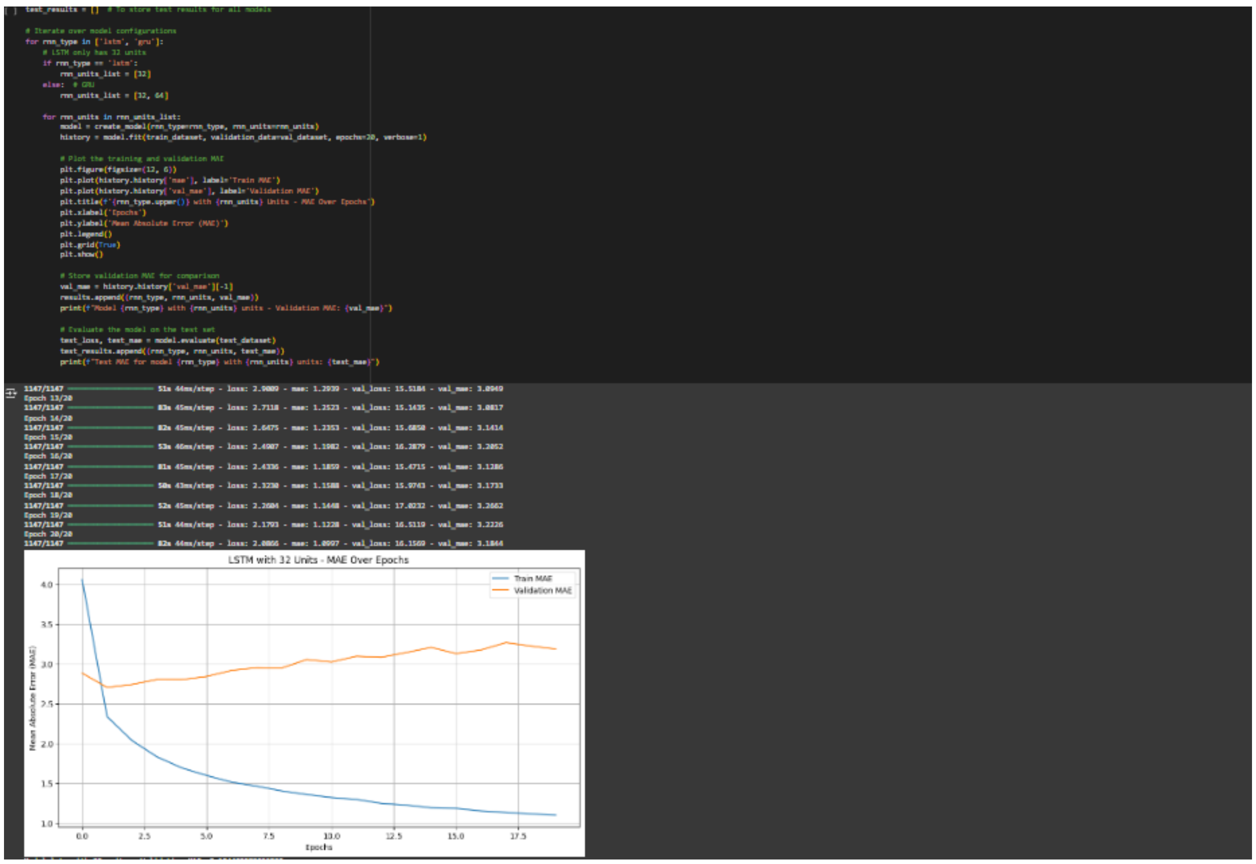
```
Model gru with 32 units - Validation MAE: 3.078643798828125
241/241 ━━━━━━━━━━━━━━━━━━━━ 9s 38ms/step - loss: 13.3151 - mae: 2.8767
Test MAE for model gru with 32 units: 2.8927574157714844
```

Gru model with 64 units:



```
Model gru with 64 units - Validation MAE: 3.0338926315307617
241/241 ━━━━━━━━━━━━━━━━━━━━ 9s 33ms/step - loss: 12.9702 - mae: 2.8595
Test MAE for model gru with 64 units: 2.8699233531951904
```

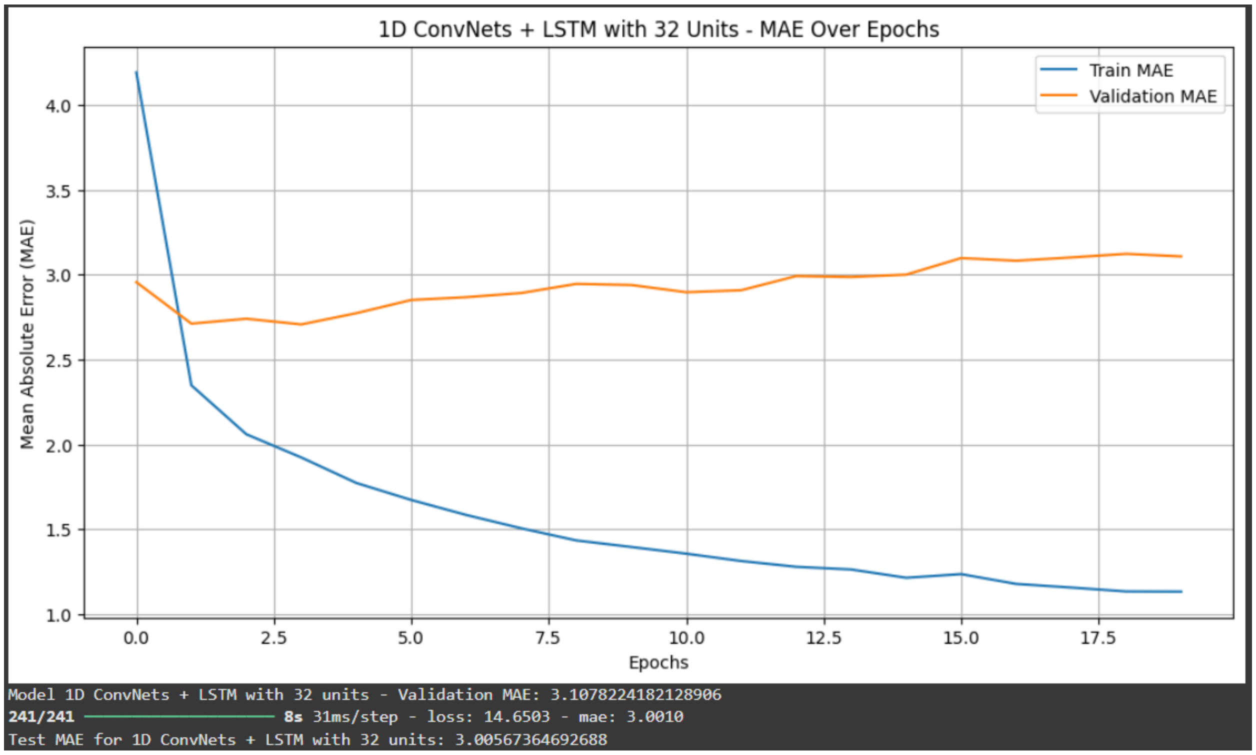
Lstm:



```python
test_results = []  # To store test results for all models

# Iterate over model configurations
for rnn_type in ['lstm', 'gru']:
    # LSTM only has 32 units
    if rnn_type == 'lstm':
        rnn_units_list = [32]
    else:  # GRU
        rnn_units_list = [32, 64]

    for rnn_units in rnn_units_list:
        model = create_model(rnn_type=rnn_type, rnn_units=rnn_units)
        history = model.fit(train_dataset, validation_data=val_dataset, epochs=20, verbose=1)

        # Plot the training and validation MAE
        plt.figure(figsize=(12, 6))
        plt.plot(history.history['mae'], label='Train MAE')
        plt.plot(history.history['val_mae'], label='Validation MAE')
        plt.title(f'{rnn_type.upper()} with {rnn_units} Units - MAE Over Epochs')
        plt.xlabel('Epochs')
        plt.ylabel('Mean Absolute Error (MAE)')
        plt.legend()
        plt.grid(True)
        plt.show()

        # Store validation MAE for comparison
        val_mae = history.history['val_mae'][-1]
        results.append((rnn_type, rnn_units, val_mae))
        print(f"Model {rnn_type} with {rnn_units} units - Validation MAE: {val_mae}")

        # Evaluate the model on the test set
        test_loss, test_mae = model.evaluate(test_dataset)
        test_results.append((rnn_type, rnn_units, test_mae))
        print(f"Test MAE for model {rnn_type} with {rnn_units} units: {test_mae}")
```

```
1147/1147 ━━━━━━━━━━━━━━━━━━━━ 51s 44ms/step - loss: 2.9009 - mae: 1.2939 - val_loss: 15.5184 - val_mae: 3.0949
Epoch 13/20
1147/1147 ━━━━━━━━━━━━━━━━━━━━ 83s 45ms/step - loss: 2.7118 - mae: 1.2523 - val_loss: 15.1435 - val_mae: 3.0817
Epoch 14/20
1147/1147 ━━━━━━━━━━━━━━━━━━━━ 82s 45ms/step - loss: 2.6475 - mae: 1.2353 - val_loss: 15.6850 - val_mae: 3.1414
Epoch 15/20
1147/1147 ━━━━━━━━━━━━━━━━━━━━ 53s 46ms/step - loss: 2.4907 - mae: 1.1982 - val_loss: 16.2879 - val_mae: 3.2052
Epoch 16/20
1147/1147 ━━━━━━━━━━━━━━━━━━━━ 81s 45ms/step - loss: 2.4336 - mae: 1.1859 - val_loss: 15.4715 - val_mae: 3.1286
Epoch 17/20
1147/1147 ━━━━━━━━━━━━━━━━━━━━ 50s 43ms/step - loss: 2.3230 - mae: 1.1588 - val_loss: 15.9743 - val_mae: 3.1733
Epoch 18/20
1147/1147 ━━━━━━━━━━━━━━━━━━━━ 52s 45ms/step - loss: 2.2604 - mae: 1.1448 - val_loss: 17.0232 - val_mae: 3.2662
Epoch 19/20
1147/1147 ━━━━━━━━━━━━━━━━━━━━ 51s 44ms/step - loss: 2.1793 - mae: 1.1228 - val_loss: 16.5119 - val_mae: 3.2226
Epoch 20/20
1147/1147 ━━━━━━━━━━━━━━━━━━━━ 82s 44ms/step - loss: 2.0866 - mae: 1.0997 - val_loss: 16.1569 - val_mae: 3.1844
```

Combination of 1dconvnets and RNN:



```
Model 1D ConvNets + LSTM with 32 units - Validation MAE: 3.1078224182128906
241/241 ━━━━━━━━━━━━━━━━━━━━ 8s 31ms/step - loss: 14.6503 - mae: 3.0010
Test MAE for 1D ConvNets + LSTM with 32 units: 3.00567364692688
```

Result Summary:

Model	Hidden Units	Validation mae	Test mae
Recurrent layer with 32 hidden units	32	3.07	2.89
Recurrent layer with 64 hidden units	64	3.03	2.86
LSTM	32	3.08	2.82
1dconvents and RNN	32	3.10	3

Conclusion:

The LSTM model with 32 hidden units achieved the best test performance, showing that its structure is well-suited to the task. The GRU models also performed reasonably well, but increasing model complexity by adding convolutional layers did not yield improvements. These findings suggest that, for this dataset, simpler RNN architectures (LSTM) with moderate hidden units offer the most effective balance of accuracy and computational efficiency.