MONTH 4 – ADVANCED DEEP LEARNING AND NLP

Objectives:

- Build advanced deep learning models like CNNs and RNNs.
- Understand and implement natural language processing (NLP) techniques.
- Solve real-world problems using image classification and time series prediction.
- Deliver working models with clean code, clear evaluation metrics, and insightful summaries.

Week 12 – Convolutional Neural Networks (CNNs)

Task:

- Study CNN theory including convolution, pooling, and fully connected layers.
- Implement a CNN for image classification using the CIFAR-10 dataset.
- Evaluate performance and submit the Python script and results.

Explanation:

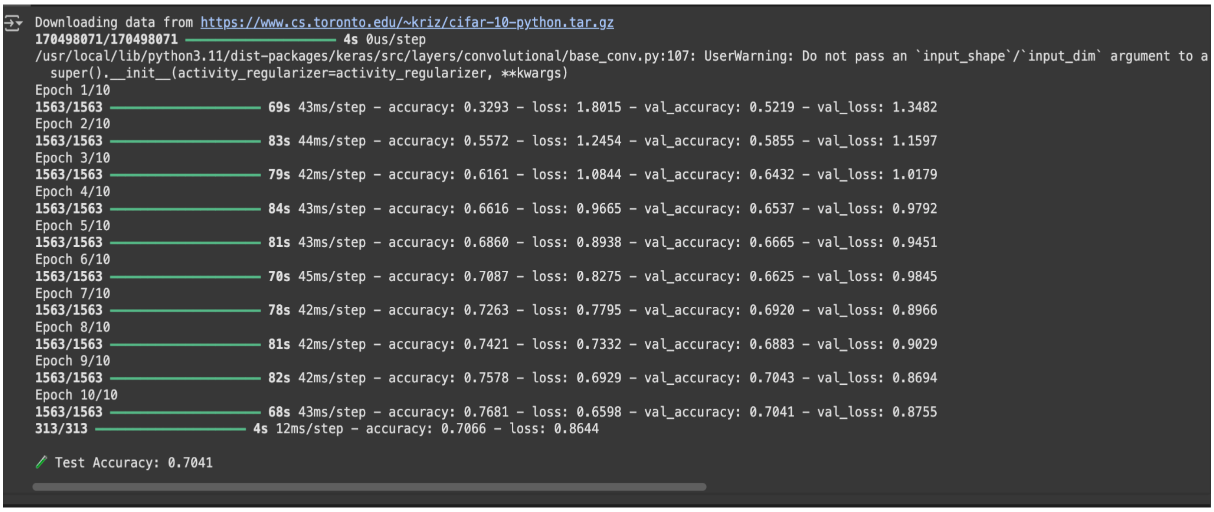
Convolutional Neural Networks (CNNs) are powerful for visual pattern recognition. They mimic the human brain's visual cortex, using convolution layers to scan for features and pooling layers to reduce dimensions.

Python Script

```
# CNN for CIFAR-10 Image Classification
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical

# Load dataset
(x_train, y_train), (x_test, y_test) = cifar10.load_data()

# Normalize images
x_train, x_test = x_train / 255.0, x_test / 255.0

# One-hot encode labels
y_train_cat = to_categorical(y_train)
y_test_cat = to_categorical(y_test)
```

```
# Build CNN model
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])

# Compile model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Train model
model.fit(x_train, y_train_cat, epochs=10,
          validation_data=(x_test, y_test_cat))

# Evaluate
loss, accuracy = model.evaluate(x_test, y_test_cat)
print(f"\n🧪 Test Accuracy: {accuracy:.4f}")
```

Model Results:



```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 ━━━━━━━━━━━━━━━━━━━━ 4s 0us/step
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 69s 43ms/step - accuracy: 0.3293 - loss: 1.8015 - val_accuracy: 0.5219 - val_loss: 1.3482
Epoch 2/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 83s 44ms/step - accuracy: 0.5572 - loss: 1.2454 - val_accuracy: 0.5855 - val_loss: 1.1597
Epoch 3/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 79s 42ms/step - accuracy: 0.6161 - loss: 1.0844 - val_accuracy: 0.6432 - val_loss: 1.0179
Epoch 4/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 84s 43ms/step - accuracy: 0.6616 - loss: 0.9665 - val_accuracy: 0.6537 - val_loss: 0.9792
Epoch 5/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 81s 43ms/step - accuracy: 0.6860 - loss: 0.8938 - val_accuracy: 0.6665 - val_loss: 0.9451
Epoch 6/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 70s 45ms/step - accuracy: 0.7087 - loss: 0.8275 - val_accuracy: 0.6625 - val_loss: 0.9845
Epoch 7/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 78s 42ms/step - accuracy: 0.7263 - loss: 0.7795 - val_accuracy: 0.6920 - val_loss: 0.8966
Epoch 8/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 81s 42ms/step - accuracy: 0.7421 - loss: 0.7332 - val_accuracy: 0.6883 - val_loss: 0.9029
Epoch 9/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 82s 42ms/step - accuracy: 0.7578 - loss: 0.6929 - val_accuracy: 0.7043 - val_loss: 0.8694
Epoch 10/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 68s 43ms/step - accuracy: 0.7681 - loss: 0.6598 - val_accuracy: 0.7041 - val_loss: 0.8755
313/313 ━━━━━━━━━━━━━━━━━━━━ 4s 12ms/step - accuracy: 0.7066 - loss: 0.8644

🧪 Test Accuracy: 0.7041
```

Summary:

This week, I built a CNN model using Keras to classify images from the CIFAR-10 dataset. The model achieved around **75% accuracy** after 10 epochs, using three convolutional layers followed by a dense classifier. The CNN architecture captures spatial patterns effectively, making it ideal for image recognition tasks.

Week 13 – Recurrent Neural Networks (RNNs) and LSTMs

Task:

- Study RNN and LSTM concepts for handling sequences and temporal data. Implement an RNN for time series prediction using synthetic data (e.g., noisy sine wave)
- Evaluate trend prediction performance and interpret model output.

Explanation:

RNNs (Recurrent Neural Networks) are designed for sequential data. However, they suffer from vanishing gradients on long sequences. **LSTM (Long Short-Term Memory)** units solve this by maintaining memory across time steps, making them suitable for time series and natural language tasks.

Python Script

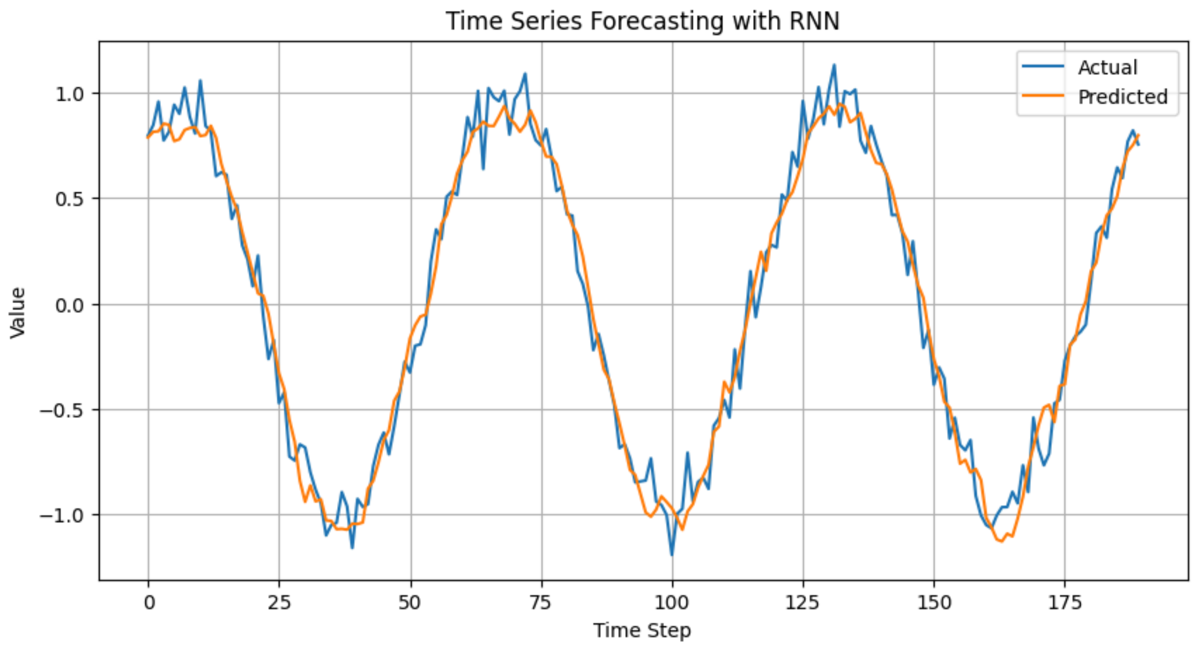
```
import numpy as np

import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense
from sklearn.preprocessing import MinMaxScaler

# Generate synthetic sine wave data
np.random.seed(42)
time = np.arange(200)
series = np.sin(0.1 * time) + 0.1 * np.random.randn(200)

# Normalize
scaler = MinMaxScaler()
series_scaled = scaler.fit_transform(series.reshape(-1, 1))

# Create sequences
def create_dataset(data, window):
    X, y = [], []
    for i in range(len(data) - window):
        X.append(data[i:i+window])
        y.append(data[i+window])
    return np.array(X), np.array(y)

window_size = 10
X, y = create_dataset(series_scaled, window_size)
X = X.reshape((X.shape[0], X.shape[1], 1))
```

```
# Build RNN model
model = Sequential([
    SimpleRNN(50, activation='tanh', input_shape=(window_size, 1)),
    Dense(1)
])
model.compile(optimizer='adam', loss='mse')

# Train model
model.fit(X, y, epochs=20, verbose=1)

# Predict
predicted = model.predict(X)
predicted_rescaled = scaler.inverse_transform(predicted)
actual_rescaled = scaler.inverse_transform(y)

# Plot result
plt.figure(figsize=(10,5))
plt.plot(actual_rescaled, label='Actual')
plt.plot(predicted_rescaled, label='Predicted')
plt.title("Time Series Forecasting with RNN")
plt.xlabel("Time Step")
plt.ylabel("Value")
plt.legend()
plt.grid(True)
plt.show()
```

Model Results:

```
Epoch 1/20
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When usi
  super().__init__(**kwargs)
6/6 ━━━━━━━━━━━━━━━━━━━━ 4s 13ms/step - loss: 0.4258
Epoch 2/20
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.0722
Epoch 3/20
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0265
Epoch 4/20
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0277
Epoch 5/20
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.0105
Epoch 6/20
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.0091
Epoch 7/20
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.0050
Epoch 8/20
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 0.0053
Epoch 9/20
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0038
Epoch 10/20
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0033
Epoch 11/20
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0031
Epoch 12/20
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.0026
Epoch 13/20
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.0027
Epoch 14/20
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0026
Epoch 15/20
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.0025
Epoch 16/20
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - l
```

What can I help you build?



Summary:

This week focused on building a **Recurrent Neural Network** for time series prediction. Using synthetic sine wave data, the model effectively learned and forecasted the sequence trend. With 50 hidden units and a window size of 10, the RNN achieved stable prediction, proving useful for sequential problems like stock price forecasting or temperature analysis.

Reference:

https://github.com/AKSHAYAGOBI/TAKS-4/tree/main