Name : Durva Kadam Roll No . : 23

St. Francis Institute of Technology, Mumbai-400 103
Department Of Information Technology

A.Y. 2025-2026
Class: BE-ITA/B, Semester: VII
Subject: Data Science Lab

Experiment – 5

1. **Aim:** To design a simple LSTM Network.

2. **Objectives:** Students should be familiarize with Learning Architectures and Frameworks using LSTM and CNN.
3. **Prerequisite:** Python basics

4. **Pre-Experiment Exercise:**
Theory:
Long Short Term Memory Network(LSTM). In concept, an LSTM recurrent unit tries to “remember” all the past knowledge that the network is seen so far and to “forget” irrelevant data. This is done by introducing different activation function layers called “gates” for different purposes. Each LSTM recurrent unit also maintains a vector called the Internal Cell State which conceptually describes the information that was chosen to be retained by the previous LSTM recurrent unit. A Long Short Term Memory Network consists of four different gates for different purposes as described below:-

Forget Gate(f): It determines to what extent to forget the previous data.
Input Gate(i): It determines the extent of information be written onto the Internal Cell State.
Input Modulation Gate(g): It is often considered as a sub-part of the input gate and much literature on LSTM’s does not even mention it and assume it is inside the Input gate. It is used to modulate the information that the Input gate will write onto the Internal State Cell by adding non-linearity to the information and making the information Zero-mean. This is done to reduce the learning time as Zero-mean input has faster convergence. Although this gate’s actions are less important than the others and are often treated as a finesse-providing concept, it is good practice to include this gate in the structure of the LSTM unit.
Output Gate(o): It determines what output(next Hidden State) to generate from the current Internal Cell State.

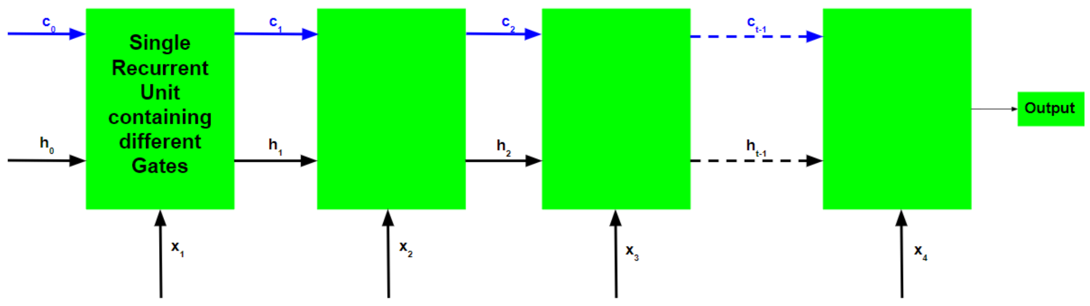


Fig:6.1 LSTM Network

CNN:
A convolutional neural network (CNN, or ConvNet) is a class of artificial neural network (ANN), most commonly applied to analyze visual imagery. CNNs are also known as Shift Invariant or Space Invariant Artificial Neural Networks (SIANN), based on the shared-weight architecture of the convolution kernels or filters that slide along input features and provide

translation-equivariant responses known as feature maps. Counter-intuitively, most convolutional neural networks are not invariant to translation, due to the down sampling operation they apply to the input. They have applications in image and video recognition, recommender systems, image classification, image segmentation, medical image analysis, natural language processing, brain–computer interfaces, and financial time series.

6. Laboratory Exercise

Procedure

- Take input the current input, the previous hidden state, and the previous internal cell state.
- Calculate the values of the four different gates by following the below steps:-
- For each gate, calculate the parameterized vectors for the current input and the previous hidden state by element-wise multiplication with the concerned vector with the respective weights for each gate.
- Apply the respective activation function for each gate element-wise on the parameterized vectors.
- Calculate the current internal cell state by first calculating the element-wise multiplication vector of the input gate and the input modulation gate, then calculate the element-wise multiplication vector of the forget gate and the previous internal cell state and then adding the two vectors.
- c_{t} = i\odot g + f\odot c_{t-1}

Post-Experiments Exercise:

A. Extended Theory:

a. Explain Architecture of CNN.

B. Post Lab Program:

a. Write a Python program to demonstrate any real life application using CNN .

C. Conclusion:

1. Write what was performed in the program (s) .
2. What is the significance of program and what Objective is achieved?

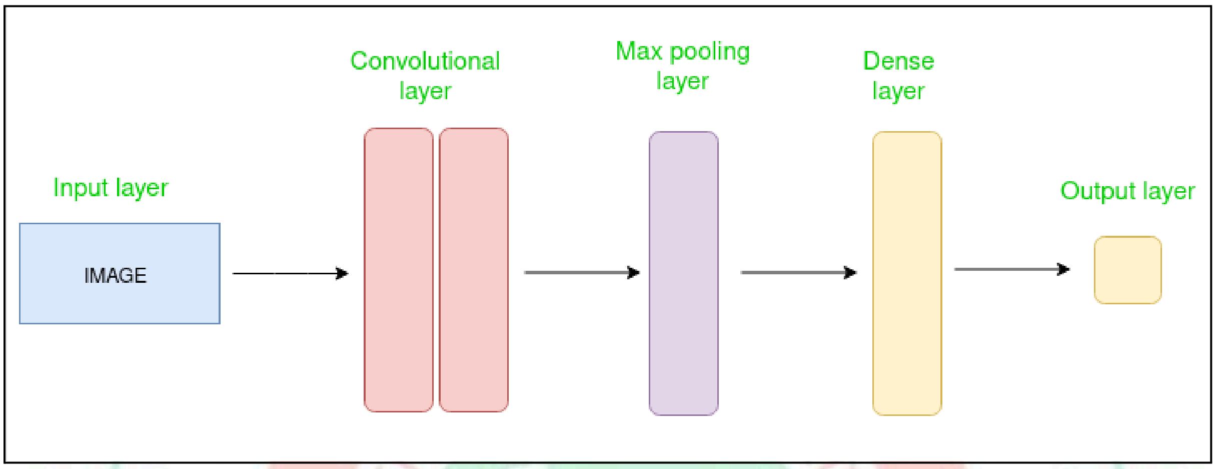
D. References:

[1] https://stackabuse.com/image-recognition-in-python-with-tensorflow-and-keras/
[2] . https://www.simplilearn.com/tutorials/deep-learning-tutorial/rnn

Post-Experiments Exercise:

- **Extended Theory:**

1. Explain Architecture of CNN.



1. Input Layer

- Takes the image as input.
- For example, a colored image might be represented as `height × width × 3` (RGB channels).
- No computation here—just data feeding into the network.

2. Convolutional Layer

- The first major processing step.
- Applies multiple **filters/kernels** to the image to detect features like edges, textures, and simple shapes.
- Each filter creates a **feature map**.
- Usually followed by an **activation function (ReLU)** to introduce non-linearity.

3. Max Pooling Layer

- Reduces the size of the feature maps while keeping the important features.
- Example: A 2×2 max pooling reduces a 28×28 feature map to 14×14.
- Helps with:
  - Lower computational cost
  - Preventing overfitting
  - Capturing only dominant features

4. Dense Layer (Fully Connected Layer)

- After flattening the pooled feature maps into a **1D vector**, it is fed into dense layers.
- Each neuron is connected to all outputs from the previous layer.
- Learns high-level combinations of features (e.g., eyes + nose + mouth = face).

5. Output Layer

- Final layer gives the classification output.
- Uses **Softmax** (for multi-class) or **Sigmoid** (for binary classification).
- Example: If classifying digits (0–9), output layer has 10 neurons, each representing one digit.

7. Laboratory Exercise

1. Import the required libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

2. Import the training dataset

```
train = pd.read_csv("DailyDelhiClimateTrain.csv")
print(train.head())
training_set = train[['meantemp']].values
```

```
         date   meantemp   humidity  wind_speed  meanpressure
0  2013-01-01  10.000000  84.500000    0.000000   1015.666667
1  2013-01-02   7.400000  92.000000    2.980000   1017.800000
2  2013-01-03   7.166667  87.000000    4.633333   1018.666667
3  2013-01-04   8.666667  71.333333    1.233333   1017.166667
4  2013-01-05   6.000000  86.833333    3.700000   1016.500000
```

3. Perform feature scaling to transform the data

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range=(0,1))
training_set_scaled = scaler.fit_transform(training_set)
```

4. Create a data structure with 60-time steps and 1 output

```
X_train = []
y_train = []

for i in range(60, len(training_set_scaled)):
    X_train.append(training_set_scaled[i-60:i, 0])
    y_train.append(training_set_scaled[i, 0])

X_train, y_train = np.array(X_train), np.array(y_train)

X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
```

5. Import Keras library and its packages

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dropout, Dense
```

6. Initialize the RNN

```
model = Sequential()
```

7. Add the LSTM layers and some dropout regularization.

```
model.add(LSTM(units=50, return_sequences=True, input_shape=(X_train.shape[1], 1)))
model.add(Dropout(0.2))
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/rnn/rnn.py:199: UserWarning: Do
  super().__init__(**kwargs)
```

```
model.add(LSTM(units=50, return_sequences=True))
model.add(Dropout(0.2))
```

```
model.add(LSTM(units=50, return_sequences=True))
model.add(Dropout(0.2))
```

```
model.add(LSTM(units=50))
model.add(Dropout(0.2))
```

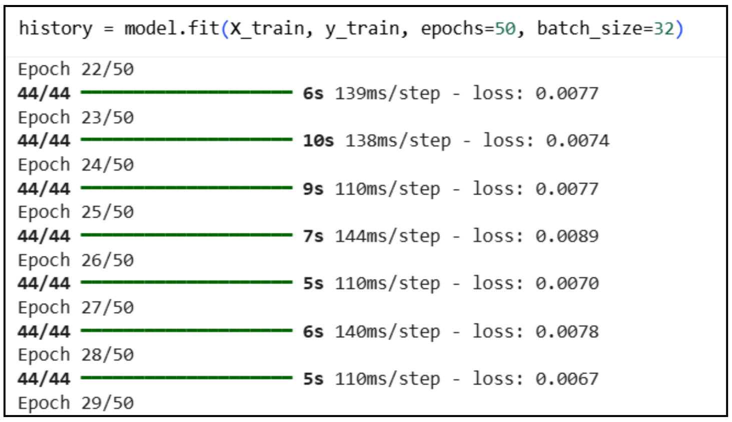
8. Add the output layer.

```
model.add(Dense(units=1))
```

9. Compile the RNN

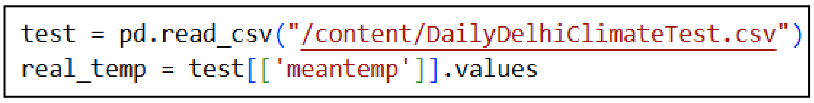
```
model.compile(optimizer='adam', loss='mean_squared_error')
```

10. Fit the RNN to the training set



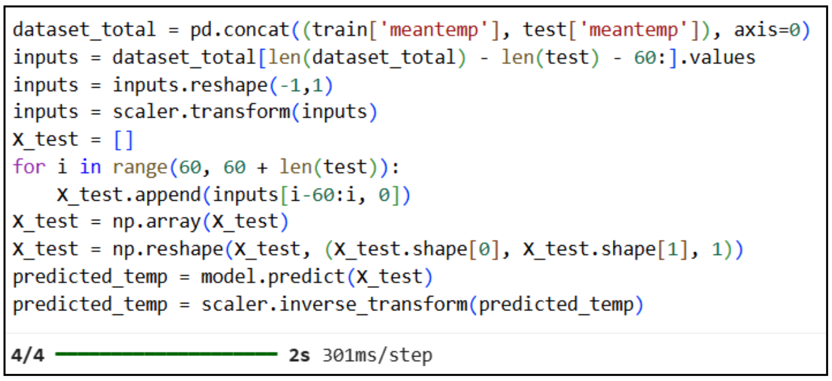
```
history = model.fit(X_train, y_train, epochs=50, batch_size=32)

Epoch 22/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 6s 139ms/step - loss: 0.0077
Epoch 23/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 10s 138ms/step - loss: 0.0074
Epoch 24/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 9s 110ms/step - loss: 0.0077
Epoch 25/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 7s 144ms/step - loss: 0.0089
Epoch 26/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 5s 110ms/step - loss: 0.0070
Epoch 27/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 6s 140ms/step - loss: 0.0078
Epoch 28/50
44/44 ━━━━━━━━━━━━━━━━━━━━ 5s 110ms/step - loss: 0.0067
Epoch 29/50
```

11. Load the stock price test data for 2017



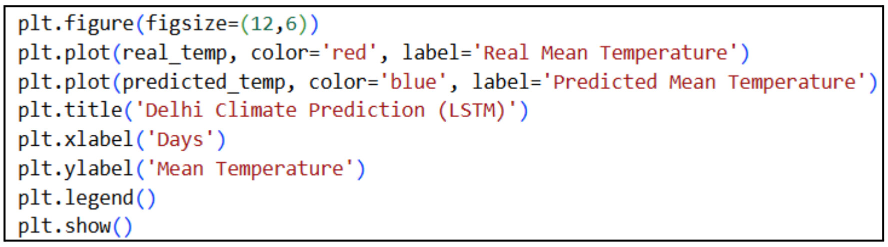
```
test = pd.read_csv("/content/DailyDelhiClimateTest.csv")
real_temp = test[['meantemp']].values
```

12. Get the predicted stock price for 2017

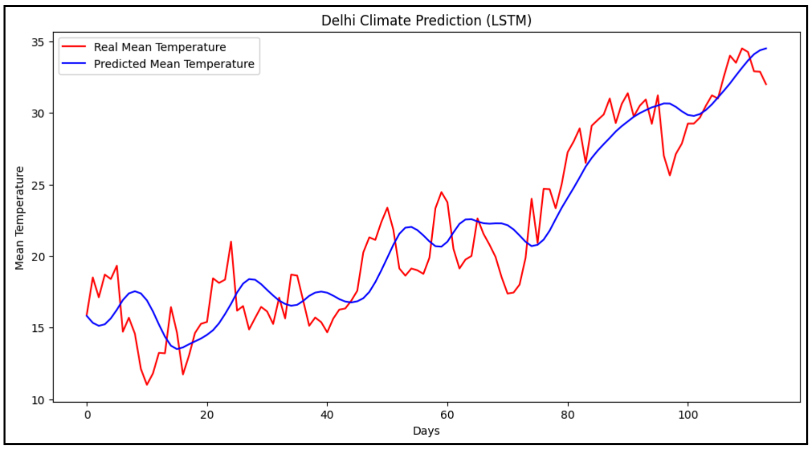


```
dataset_total = pd.concat((train['meantemp'], test['meantemp']), axis=0)
inputs = dataset_total[len(dataset_total) - len(test) - 60:].values
inputs = inputs.reshape(-1,1)
inputs = scaler.transform(inputs)
X_test = []
for i in range(60, 60 + len(test)):
    X_test.append(inputs[i-60:i, 0])
X_test = np.array(X_test)
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
predicted_temp = model.predict(X_test)
predicted_temp = scaler.inverse_transform(predicted_temp)

4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 301ms/step
```

13. Visualize the results of predicted and real stock price



```
plt.figure(figsize=(12,6))
plt.plot(real_temp, color='red', label='Real Mean Temperature')
plt.plot(predicted_temp, color='blue', label='Predicted Mean Temperature')
plt.title('Delhi Climate Prediction (LSTM)')
plt.xlabel('Days')
plt.ylabel('Mean Temperature')
plt.legend()
plt.show()
```



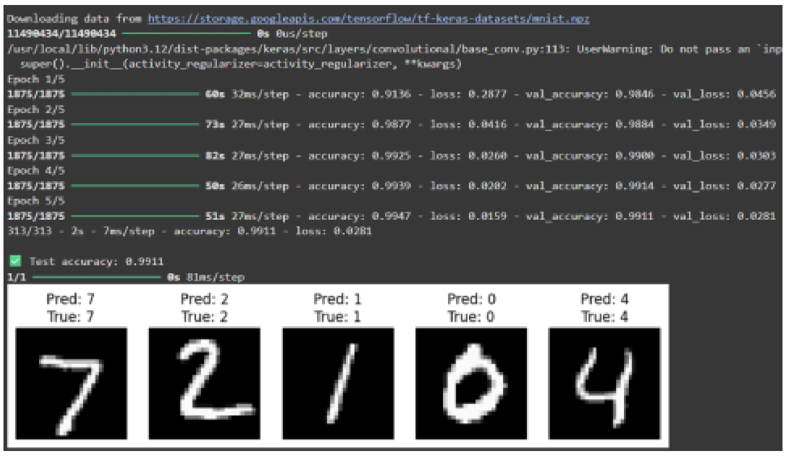
Post Experiment Exercise:

A. Write a Python program to demonstrate any real life application using CNN .

```
# CNN for MNIST Handwritten Digit Recognition
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
import numpy as np
# 1. Load dataset
(x_train, y_train), (x_test, y_test) = datasets.mnist.load_data()
# Normalize images (0–255 -> 0–1) and reshape for CNN input
x_train = x_train.reshape((x_train.shape[0], 28, 28, 1)) / 255.0
x_test = x_test.reshape((x_test.shape[0], 28, 28, 1)) / 255.0
# 2. Build CNN model
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(10, activation='softmax')  # 10 output classes (digits 0–9)
])
# 3. Compile model
model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
# 4. Train model
history = model.fit(
    x_train, y_train,
    epochs=5,
    validation_data=(x_test, y_test)
)
# 5. Evaluate
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
```

```
print(f"\n✅ Test accuracy: {test_acc:.4f}")
# 6. Predict on test samples
predictions = model.predict(x_test[:5])
predicted_labels = np.argmax(predictions, axis=1)
# Show first 5 test images with predictions
plt.figure(figsize=(10, 5))
for i in range(5):
   plt.subplot(1, 5, i + 1)
   plt.imshow(x_test[i].reshape(28, 28), cmap="gray")
   plt.title(f"Pred: {predicted_labels[i]}\nTrue: {y_test[i]}")
   plt.axis("off")
plt.show()
```

Output:



```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
/usr/local/lib/python3.12/dist-packages/keras/src/layers/convolutional/base_conv.py:113: UserWarning: Do not pass an `inp
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 60s 32ms/step - accuracy: 0.9136 - loss: 0.2877 - val_accuracy: 0.9846 - val_loss: 0.0456
Epoch 2/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 73s 27ms/step - accuracy: 0.9877 - loss: 0.0416 - val_accuracy: 0.9884 - val_loss: 0.0349
Epoch 3/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 82s 27ms/step - accuracy: 0.9925 - loss: 0.0260 - val_accuracy: 0.9900 - val_loss: 0.0303
Epoch 4/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 50s 26ms/step - accuracy: 0.9939 - loss: 0.0202 - val_accuracy: 0.9914 - val_loss: 0.0277
Epoch 5/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 51s 27ms/step - accuracy: 0.9947 - loss: 0.0159 - val_accuracy: 0.9911 - val_loss: 0.0281
313/313 - 2s - 7ms/step - accuracy: 0.9911 - loss: 0.0281

✅ Test accuracy: 0.9911
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 81ms/step
```