**Lab Assignment 7**

**Neural Network & Deep Learning**

**Build and train a CNN and know the application of different layers**

**Step 1**: Load the MNIST dataset into your notebook

**Step 2**: Pre-processing and prepare the data for giving to the CNN.

1. Encoding the classes using one hot encoder.
2. Normalize the features.

**Step 3**: Building the convolutional network model.

1. You may choose the layers.
2. Print the summary and note the number of neurons and parameters of the model.
3. Compile the model and train it using the training data.

**Step 4:** Vary the number of layers and repeat step 3.

**Step 5:** Implement the architecture of LeNet 5.

**Step 6:** Finally note which network gives you the best performance.

PART B

|  |  |
| --- | --- |
| Roll No: C0009 | Name: Samarth Borade |
| Class : B | Batch : EB1 |
| Date of Experiment: 16/02/24 | Date of Submission |
| Grade : |  |

**B.1 Software Code written by student:**

#Samarth Borade

#C009

#BTI SEM 10

#EXP 7: Build and train a CNN.

import tensorflow as tf

from tensorflow.keras.datasets import mnist

from sklearn.preprocessing import OneHotEncoder

import numpy as np

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

import matplotlib.pyplot as plt

import ssl

ssl.\_create\_default\_https\_context = ssl.\_create\_unverified\_context

# Load MNIST dataset

(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()

# Encoding the classes using one hot encoder

one\_hot\_encoder = OneHotEncoder(*sparse*=False)

y\_train\_encoded = one\_hot\_encoder.fit\_transform(y\_train.reshape(-1, 1))

y\_test\_encoded = one\_hot\_encoder.transform(y\_test.reshape(-1, 1))

# Normalize the features

x\_train\_normalized = x\_train.astype('float32') / 255.0

x\_test\_normalized = x\_test.astype('float32') / 255.0

CNN:

# Building the convolutional network model

model = Sequential([

Conv2D(32, (3, 3), *activation*='relu', *input\_shape*=(28, 28, 1)),

MaxPooling2D((2, 2)),

Conv2D(64, (3, 3), *activation*='relu'),

MaxPooling2D((2, 2)),

Flatten(),

Dense(64, *activation*='relu'),

Dense(10, *activation*='softmax')

])

# Print the summary

model.summary()

# Compile the model

model.compile(*optimizer*='adam', *loss*='categorical\_crossentropy', *metrics*=['accuracy'])

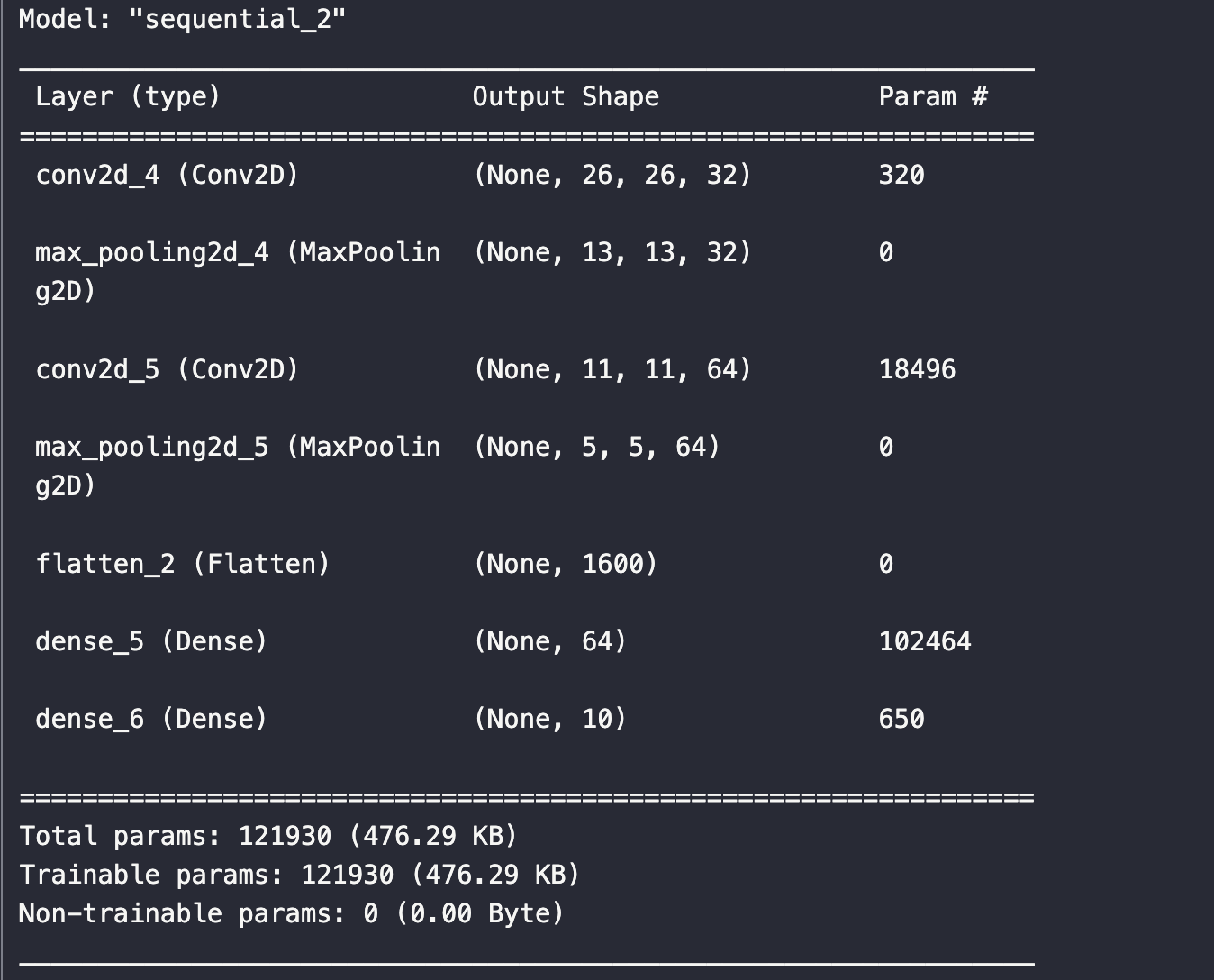
# Reshape data for CNN

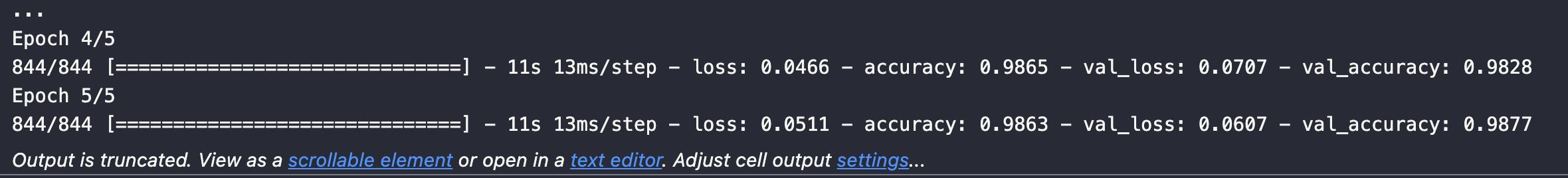
x\_train\_reshaped = np.expand\_dims(x\_train\_normalized, *axis*=-1)

x\_test\_reshaped = np.expand\_dims(x\_test\_normalized, *axis*=-1)

# Train the model

history\_cnn = model.fit(x\_train\_reshaped, y\_train\_encoded, *epochs*=5, *batch\_size*=64, *validation\_split*=0.1)





LeNet-5:

# Building LeNet-5 architecture

model\_lenet5 = Sequential([

Conv2D(6, *kernel\_size*=(5, 5), *strides*=(1, 1), *activation*='relu', *input\_shape*=(28, 28, 1)),

MaxPooling2D(*pool\_size*=(2, 2), *strides*=(2, 2)),

Conv2D(16, *kernel\_size*=(5, 5), *strides*=(1, 1), *activation*='relu'),

MaxPooling2D(*pool\_size*=(2, 2), *strides*=(2, 2)),

Flatten(),

Dense(120, *activation*='relu'),

Dense(84, *activation*='relu'),

Dense(10, *activation*='softmax')

])

# Print the summary

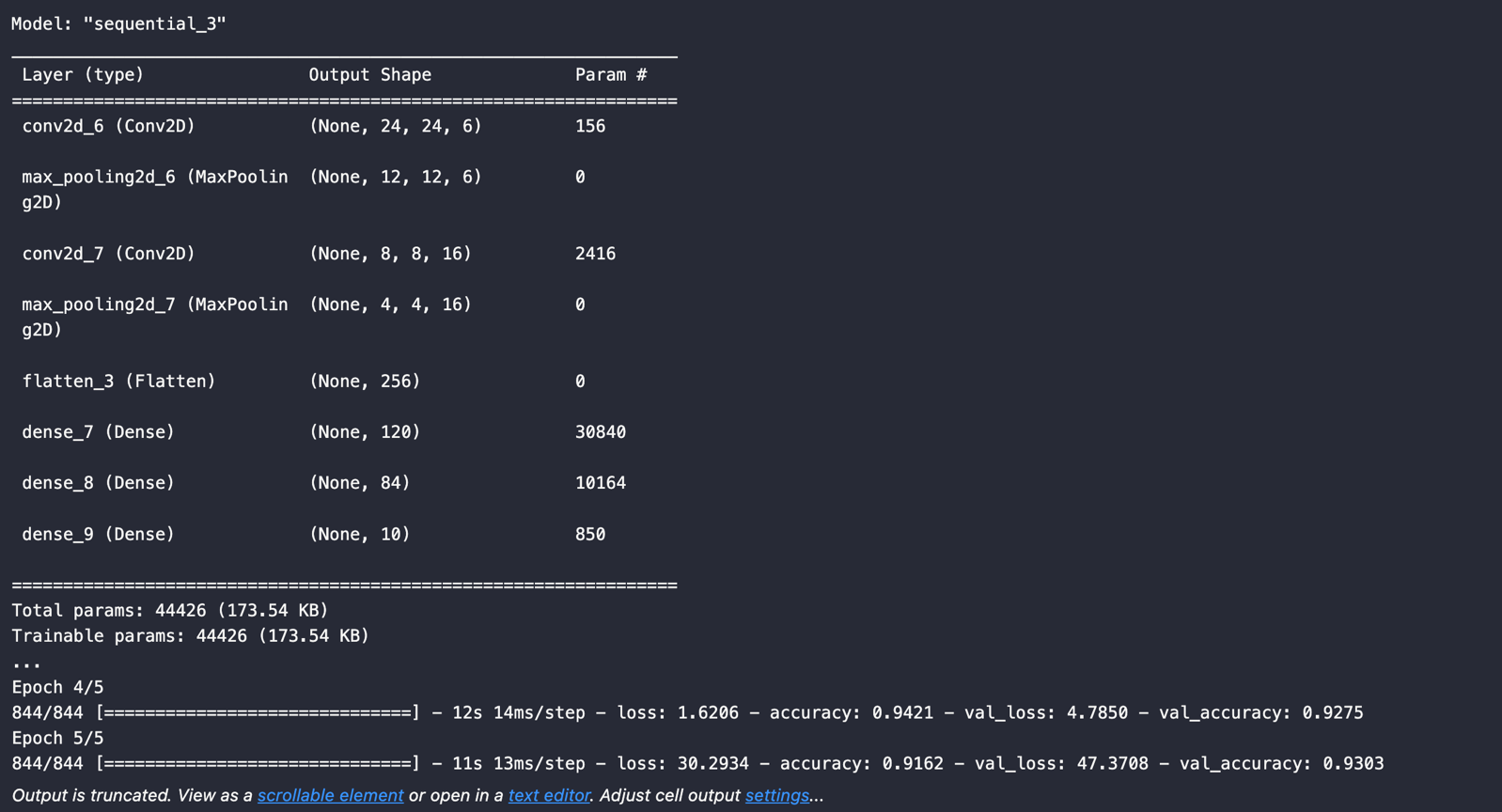
model\_lenet5.summary()

# Compile the model

model\_lenet5.compile(*optimizer*='adam', *loss*='categorical\_crossentropy', *metrics*=['accuracy'])

# Train the model

history\_lenet5 = model\_lenet5.fit(x\_train\_reshaped, y\_train\_encoded, *epochs*=5, *batch\_size*=64, *validation\_split*=0.1)



Plot:

# Plotting the training and validation accuracy for CNN and LeNet-5

plt.figure(*figsize*=(12, 8))

plt.subplot(2, 1, 1)

plt.plot(history\_cnn.history['accuracy'], *label*='CNN Training Accuracy', *color*='blue')

plt.plot(history\_cnn.history['val\_accuracy'], *label*='CNN Validation Accuracy', *color*='orange')

plt.plot(history\_lenet5.history['accuracy'], *label*='LeNet-5 Training Accuracy', *color*='green')

plt.plot(history\_lenet5.history['val\_accuracy'], *label*='LeNet-5 Validation Accuracy', *color*='red')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.title('Training and Validation Accuracy Comparison')

plt.legend()

# Plotting the training and validation loss for CNN and LeNet-5

plt.subplot(2, 1, 2)

plt.plot(history\_cnn.history['loss'], *label*='CNN Training Loss', *color*='blue')

plt.plot(history\_cnn.history['val\_loss'], *label*='CNN Validation Loss', *color*='orange')

plt.plot(history\_lenet5.history['loss'], *label*='LeNet-5 Training Loss', *color*='green')

plt.plot(history\_lenet5.history['val\_loss'], *label*='LeNet-5 Validation Loss', *color*='red')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.title('Training and Validation Loss Comparison')

plt.legend()

plt.tight\_layout()

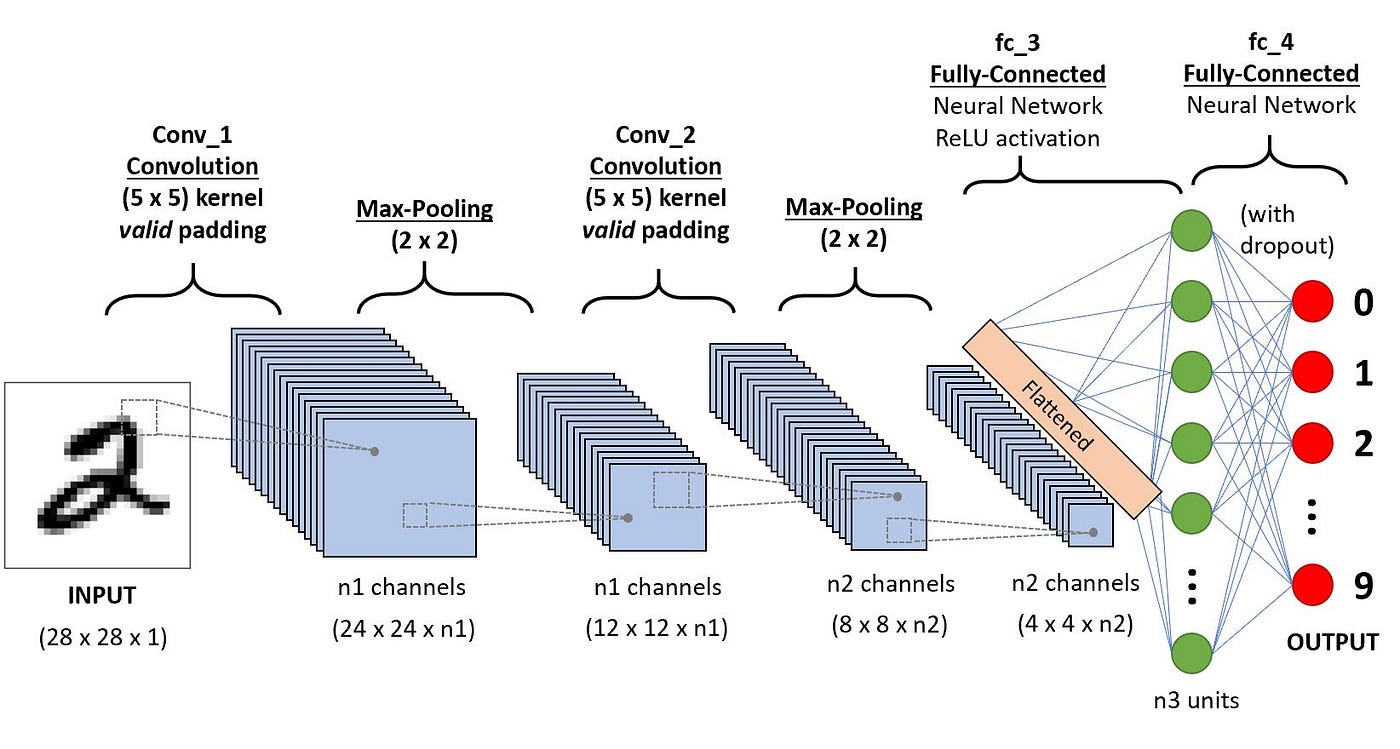
plt.show()

A graph of different colored lines

Description automatically generated

**B.3 Observations and learning:**

1. **The CNN achieved an accuracy of 98.63% on the training set and 98.77% on the validation set after 5 epochs, with a slight increase in loss from 0.0466 to 0.0511.**
2. **LeNet-5 achieved an accuracy of 91.62% on the training set and 93.03% on the validation set after 5 epochs, but it struggled with high loss values, indicating possible overfitting.**



1. **Convolutional Layer**: Extracts features from input data, enabling the network to recognize patterns like edges, textures, and shapes in images.
2. **Pooling Layer**: Reduces the spatial dimensions of feature maps, helping to control overfitting, increase computational efficiency, and introduce translation invariance.
3. **Activation Layer (ReLU)**: Introduces non-linearity into the network, enabling it to learn complex relationships between features and improving training efficiency.
4. **Fully Connected (Dense) Layer**: Learns to classify features extracted by previous layers into output classes, providing global information aggregation and mapping capabilities.
5. **Flatten Layer**: Reshapes the input tensor into a 1D vector, facilitating the transition between convolutional and fully connected layers.

Each layer type plays a specific role in feature extraction, non-linearity introduction, dimensionality reduction, and classification, enabling CNNs to effectively learn and represent complex patterns in data, particularly in tasks like image classification and object recognition.

**B.4 Conclusion:**

The CNN model showed consistent improvement in accuracy and maintained lower loss values, indicating better generalization ability compared to LeNet-5. Despite LeNet-5's historical significance, its performance on this task suggests limitations in handling complex datasets like MNIST, where deeper architectures like CNNs tend to perform better.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*