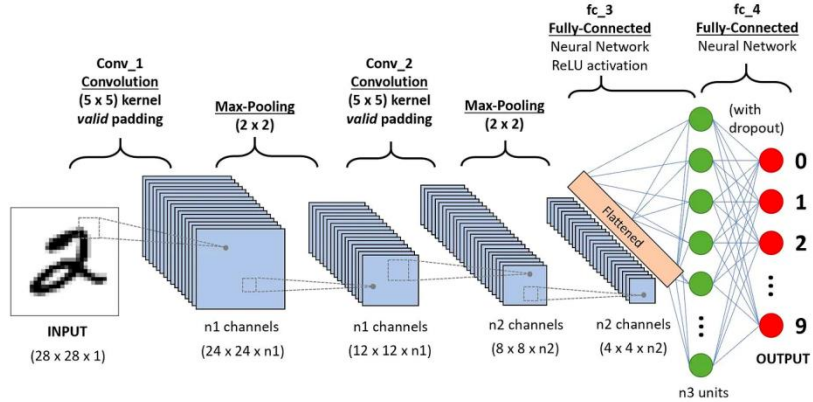


Experiment No. 7																				
BE (AI&DS)		ROLL NO : 9742																		
Date of Implementation: 20/09/2024																				
Aim: Implement CNN model																				
Programming Language Used : Python																				
Upon completion of this experiment, students will be able to LO3: Build and train deep learning models for given problem																				
<table border="1"> <tr> <td>Indicator</td> <td></td> <td></td> </tr> <tr> <td>Timeline</td> <td></td> <td></td> </tr> <tr> <td>Maintains submission deadline (1)</td> <td>On time (1)</td> <td>Otherwise (0)</td> </tr> <tr> <td>Completion and Organization (2)</td> <td>Completed in LAB (2)</td> <td>Otherwise(1)</td> </tr> <tr> <td>Analysis of output and conclusion(2)</td> <td>Properly done (2)</td> <td>Otherwise (0)</td> </tr> <tr> <td>Viva (10)</td> <td></td> <td></td> </tr> </table>			Indicator			Timeline			Maintains submission deadline (1)	On time (1)	Otherwise (0)	Completion and Organization (2)	Completed in LAB (2)	Otherwise(1)	Analysis of output and conclusion(2)	Properly done (2)	Otherwise (0)	Viva (10)		
Indicator																				
Timeline																				
Maintains submission deadline (1)	On time (1)	Otherwise (0)																		
Completion and Organization (2)	Completed in LAB (2)	Otherwise(1)																		
Analysis of output and conclusion(2)	Properly done (2)	Otherwise (0)																		
Viva (10)																				
Assessment Marks : <table border="1"> <tr> <td>Timeline(1)</td> <td></td> </tr> <tr> <td>Completion and Organization (2)</td> <td></td> </tr> <tr> <td>Analysis of output and conclusion(2)</td> <td></td> </tr> <tr> <td>Viva (10)</td> <td></td> </tr> <tr> <td>Total (15)</td> <td></td> </tr> </table>			Timeline(1)		Completion and Organization (2)		Analysis of output and conclusion(2)		Viva (10)		Total (15)									
Timeline(1)																				
Completion and Organization (2)																				
Analysis of output and conclusion(2)																				
Viva (10)																				
Total (15)																				

EXPERIMENT	7
Aim	To Implement CNN model
Tools	PYTHON
Theory	<p>A Convolutional Neural Network (CNN or ConvNet) is a deep learning algorithm specifically designed for any task where object recognition is crucial such as image classification, detection, and segmentation. Many real-life applications, such as self-driving cars, surveillance cameras, and more, use CNNs. CNNs' architecture tries to mimic the structure of neurons in the human visual system composed of multiple layers, where each one is responsible for detecting a specific feature in the data. The typical CNN is made of a combination of four main layers:</p> <ul style="list-style-type: none"> • Convolutional layers • Rectified Linear Unit (ReLU for short) • Pooling layers • Fully connected layers  <p>Convolution layers</p> <p>This is the first building block of a CNN. As the name suggests, the main mathematical task performed is called convolution, which is the application of a sliding window function to a matrix of pixels representing an image. The sliding function applied to the matrix is called kernel or filter, and both can be used interchangeably. In the convolution layer, several filters of equal size are applied, and each filter is used to recognize a specific pattern from the image, such as the curving of the digits, the edges, the whole shape of the digits, and more.</p> <p>Activation function</p> <p>A ReLU activation function is applied after each convolution operation. This function helps the network learn non-linear relationships between the features in the image, hence making the network more robust for identifying different patterns.</p> <p>Pooling layer</p> <p>The goal of the pooling layer is to pull the most significant features from the convoluted matrix. This is done by applying some aggregation</p>

	<p>operations, which reduces the dimension of the feature map (convoluted matrix), hence reducing the memory used while training the network.</p> <p>Fully connected layers</p> <p>These layers are in the last layer of the convolutional neural network, and their inputs correspond to the flattened one-dimensional matrix generated by the last pooling layer. ReLU activations functions are applied to them for non-linearity.</p> <p>Finally, a softmax prediction layer is used to generate probability values for each of the possible output labels, and the final label predicted is the one with the highest probability score.</p>
Implementation	<p>One research paper is attached. Students are requested to go through the research paper and implement the proposed methodology described in it. Also use the same database mentioned in the paper.</p>
Conclusion	<p>Compare the performance of CNN with respect to different parameters.</p> <p>The primary objective of this study was to develop a model capable of recognizing digits with high accuracy. This research implemented a Convolutional Neural Network (CNN) for digit recognition, contrasting with other studies that have utilized machine learning algorithms such as KNN, SVM, and RFC. The CNN achieved a remarkable validation accuracy of 99.30% and a training accuracy of 99.17%, with low error rates of 0.83% for training and 0.70% for validation, demonstrating its effectiveness.</p> <p>The model was trained for 15 epochs, which allowed for better convergence without overfitting. The training loss was recorded at 0.0264, and the validation loss at 0.0267, indicating robust generalization to unseen data. The utilization of GPU acceleration further enhanced training efficiency, enabling faster computations.</p> <p>The results suggest that CNNs provide superior accuracy and lower error rates compared to traditional machine learning methods, emphasizing their potential for applications in image classification tasks.</p>

```

import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

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

# Normalize the data
x_train = x_train.reshape((x_train.shape[0], 28, 28, 1)).astype('float32') / 255
x_test = x_test.reshape((x_test.shape[0], 28, 28, 1)).astype('float32') / 255

# Build the CNN model
model = models.Sequential()

# Input layer and first Conv2D layer
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
# First MaxPooling layer
model.add(layers.MaxPooling2D((2, 2)))
# First Dropout layer
model.add(layers.Dropout(0.3))

/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape` /
super().__init__(activity_regularizer=activity_regularizer, **kwargs)

# Second Conv2D layer
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
# Second MaxPooling layer
model.add(layers.MaxPooling2D((2, 2)))
# Second Dropout layer
model.add(layers.Dropout(0.3))

# Third Conv2D layer
model.add(layers.Conv2D(64, (3, 3), activation='relu'))

# Flatten layer
model.add(layers.Flatten())

# First Dense layer
model.add(layers.Dense(64, activation='relu'))
# Third Dropout layer
model.add(layers.Dropout(0.3))
# Output layer
model.add(layers.Dense(10, activation='softmax'))

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

# Train the model and store the history
history = model.fit(x_train, y_train, epochs=15, batch_size=64, validation_data=(x_test, y_test))

Epoch 1/15
938/938 ————— 56s 58ms/step - accuracy: 0.7868 - loss: 0.6419 - val_accuracy: 0.9830 - val_loss: 0.0579
Epoch 2/15
938/938 ————— 54s 57ms/step - accuracy: 0.9680 - loss: 0.1044 - val_accuracy: 0.9877 - val_loss: 0.0375
Epoch 3/15
938/938 ————— 54s 57ms/step - accuracy: 0.9773 - loss: 0.0756 - val_accuracy: 0.9867 - val_loss: 0.0385
Epoch 4/15
938/938 ————— 84s 60ms/step - accuracy: 0.9827 - loss: 0.0561 - val_accuracy: 0.9913 - val_loss: 0.0279
Epoch 5/15
938/938 ————— 80s 58ms/step - accuracy: 0.9851 - loss: 0.0483 - val_accuracy: 0.9918 - val_loss: 0.0305
Epoch 6/15
938/938 ————— 55s 58ms/step - accuracy: 0.9856 - loss: 0.0451 - val_accuracy: 0.9923 - val_loss: 0.0284
Epoch 7/15
938/938 ————— 82s 58ms/step - accuracy: 0.9880 - loss: 0.0382 - val_accuracy: 0.9917 - val_loss: 0.0240
Epoch 8/15
938/938 ————— 84s 60ms/step - accuracy: 0.9898 - loss: 0.0353 - val_accuracy: 0.9921 - val_loss: 0.0263
Epoch 9/15
938/938 ————— 80s 58ms/step - accuracy: 0.9894 - loss: 0.0323 - val_accuracy: 0.9929 - val_loss: 0.0244

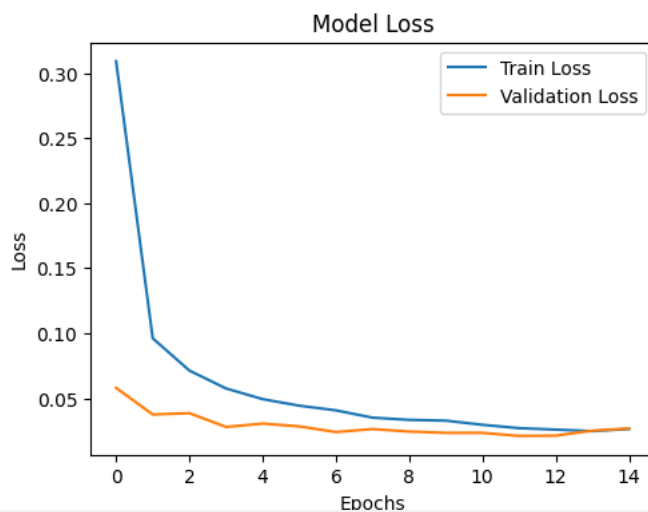
```

```
Epoch 10/15
938/938 55s 59ms/step - accuracy: 0.9909 - loss: 0.0303 - val_accuracy: 0.9932 - val_loss: 0.0234
Epoch 11/15
938/938 81s 58ms/step - accuracy: 0.9910 - loss: 0.0300 - val_accuracy: 0.9935 - val_loss: 0.0234
Epoch 12/15
938/938 56s 59ms/step - accuracy: 0.9914 - loss: 0.0278 - val_accuracy: 0.9938 - val_loss: 0.0210
Epoch 13/15
938/938 80s 58ms/step - accuracy: 0.9927 - loss: 0.0249 - val_accuracy: 0.9937 - val_loss: 0.0212
Epoch 14/15
938/938 55s 58ms/step - accuracy: 0.9919 - loss: 0.0259 - val_accuracy: 0.9928 - val_loss: 0.0250
Epoch 15/15
938/938 81s 57ms/step - accuracy: 0.9921 - loss: 0.0247 - val_accuracy: 0.9930 - val_loss: 0.0267
```

```
# Plotting the loss and accuracy curves
```

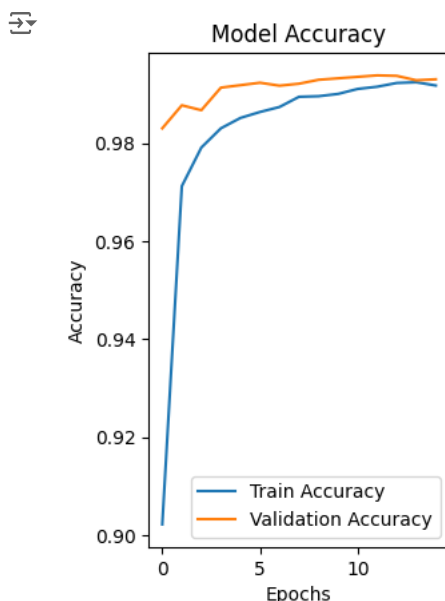
```
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7c4be385ece0>
```



```
plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
```

```
plt.show()
```

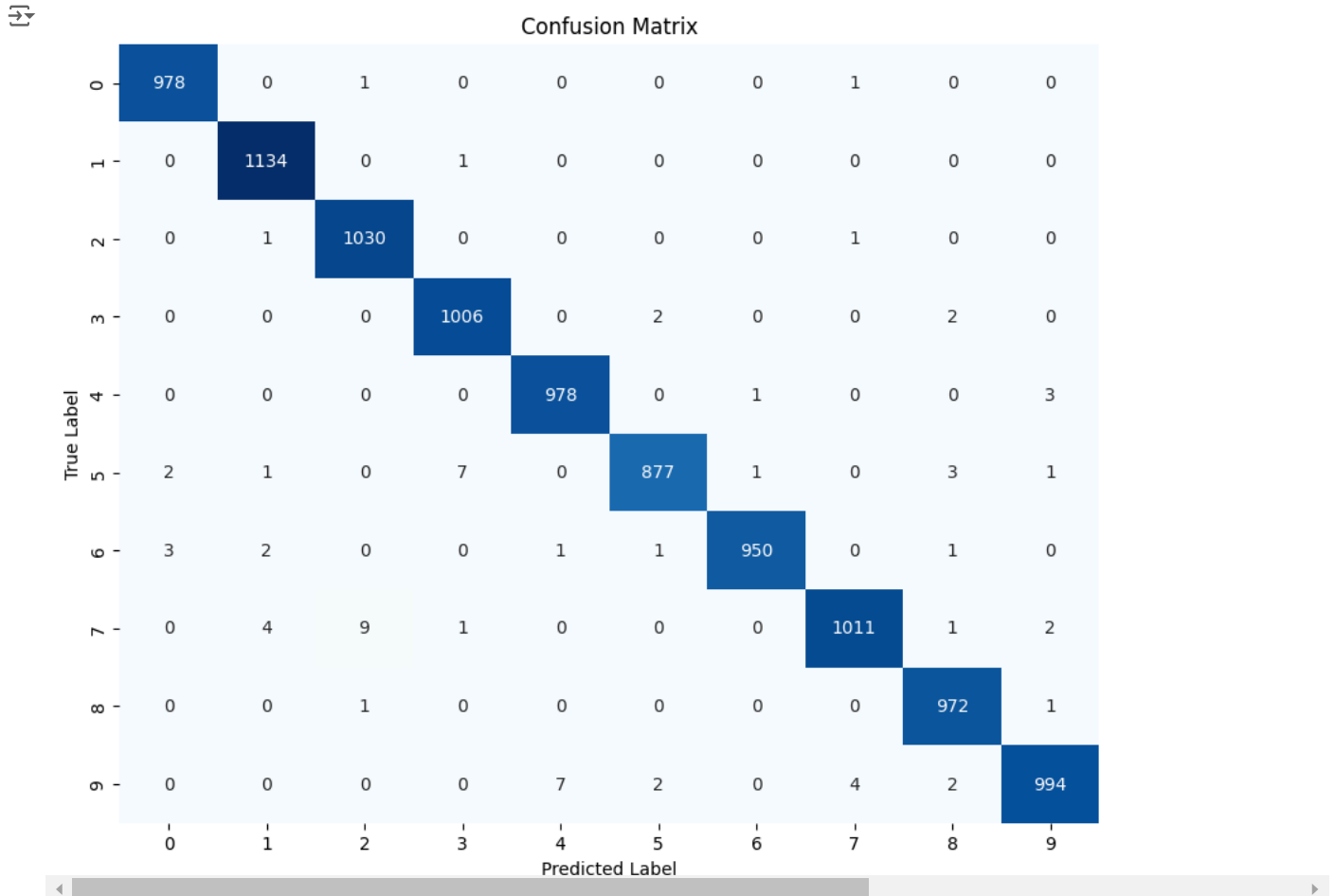


```
# Generate predictions for confusion matrix
y_pred = np.argmax(model.predict(x_test), axis=1)
```

313/313 ————— 2s 8ms/step

```
# Create confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
```

```
# Plot the confusion matrix using seaborn
plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", cbar=False, xticklabels=range(10), yticklabels=range(10))
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```



```
# Get the final training and validation accuracy and loss after the last epoch
```

```
train_accuracy = history.history['accuracy'][-1]
train_loss = history.history['loss'][-1]
val_accuracy = history.history['val_accuracy'][-1]
val_loss = history.history['val_loss'][-1]
```

```
# Calculate error rates (Error Rate = 1 - Accuracy)
```

```
train_error_rate = 1 - train_accuracy
val_error_rate = 1 - val_accuracy
```

```
# Print out the results as described in the paper
```

```
print(f"Train Accuracy: {train_accuracy * 100:.2f}%")
print(f"Validation Accuracy: {val_accuracy * 100:.2f}%")
print(f"Train Error Rate: {train_error_rate * 100:.2f}%")
print(f"Validation Error Rate: {val_error_rate * 100:.2f}%")
print(f"Train Loss: {train_loss:.4f}")
print(f"Validation Loss: {val_loss:.4f}")
```

Train Accuracy: 99.17%
 Validation Accuracy: 99.30%
 Train Error Rate: 0.83%
 Validation Error Rate: 0.70%
 Train Loss: 0.0264
 Validation Loss: 0.0267