SR UNIVERSITY

School of Computer Science and Artificial Intelligence

Course- B. Tech
Course Code-21CS102
Year- 2024-2025
Date – 28-08-24
Name : Siddhartha Namilikonda
Roll no. 2203A52110

Type- Elective Course
Course Name- Generative AI
Semester- odd
Batch- 36

Lab Assignment - 4

Aim: Design a CNN architecture to implement the image classification task over an image dataset.

Steps:

1. **Understand the Dataset**

- **Dataset Type:** Determine the type of images (e.g., grayscale, RGB).
- **Image Size:** Note the dimensions of the images.
- **Number of Classes:** Identify the number of classes for classification.
- **Dataset Examples:** Visualize a few samples to understand variations in the dataset.

2. Prepare the Environment

- **Install Necessary Libraries:** Ensure you have TensorFlow, Keras, or PyTorch installed.
- **Load the Dataset:** Load the image dataset, which could be something like CIFAR-10, MNIST, or a custom dataset.

3. Preprocess the Data

- **Normalization:** Scale the pixel values (typically between 0 and 1).
- **Data Augmentation (Optional):** Apply transformations like rotations, flips, and shifts to increase dataset diversity.

4. Design the CNN Architecture

- **Input Layer:** Define the input shape matching the image dimensions.
- **Convolutional Layers:** Add convolutional layers with filters, kernel size, and activation function (usually ReLU).
- **Pooling Layers:** Add pooling layers (e.g., MaxPooling) to reduce spatial dimensions.
- **Dropout Layers (Optional):** Include dropout to prevent overfitting.
- **Flatten Layer:** Flatten the 2D outputs to 1D before passing to fully connected layers.
- **Fully Connected (Dense) Layers:** Add one or more dense layers, typically with ReLU activation.
- **Output Layer:** The final dense layer should match the number of classes, with softmax activation for classification.

5. Compile the Model

- **Optimizer:** Choose an optimizer like Adam or SGD.
- **Loss Function:** Use a loss function appropriate for classification, such as sparse_categorical_crossentropy or categorical_crossentropy.
- **Metrics:** Define metrics like accuracy to monitor the model's performance.

6. Train the Model

- **Batch Size:** Choose an appropriate batch size (e.g., 32 or 64).
- **Epochs:** Determine the number of epochs based on the dataset size and model performance.
- **Validation:** Use a validation set to monitor the model's performance on unseen data during training.

7. Evaluate the Model

- **Test Set Evaluation:** Evaluate the trained model on the test dataset to check its generalization performance.
- **Performance Metrics:** Record the accuracy, loss, and other relevant metrics.

8. Fine-tune the Model (Optional)

- **Hyperparameter Tuning:** Adjust the learning rate, number of layers, and units in each layer to improve performance.
- **Data Augmentation:** Further augment the data if the model is overfitting.

9. Save the Model

Save the trained model for future inference or further fine-tuning.

10. Analyze and Visualize the Results

- **Plot Accuracy and Loss Curves:** Visualize the training and validation accuracy/loss over epochs.
- **Confusion Matrix:** Analyze the confusion matrix to understand the model's performance on individual classes.

Data Set Descriptions:

MNIST: Handwritten digit recognition (10 classes)
CIFAR-10: Object recognition (10 classes)
Iris: Flower classification (3 classes)
Fashion-MNIST: Clothing item classification (10 classes)
Reuters Newswire: Text categorization (46 classes)

Software/ Hardware Descriptions:

TensorFlow/Keras, NumPy, Matplotlib, Python 3.x

Theory:

Convolutional Neural Networks (CNNs) are a type of deep learning used for classifying images. They effectively recognize patterns in images by using convolutional layers that apply filters to find features like edges and textures. The ReLU activation function adds non-linearity by changing negative values to zero, which helps the network learn more complex patterns. Pooling layers, like MaxPooling, reduce the size of feature maps, which lowers the amount of computation needed and helps prevent overfitting. Dropout is an optional technique aiming to avoid overfitting. In training, it randomly eliminates units.

The Flatten layer changes multi-dimensional arrays from earlier layers into 1D vectors. These vectors are then used by fully connected (dense) layers for classification. The last dense layer uses softmax activation for tasks with many classes. During training, CNNs use categorical cross-entropy to check how accurate their predictions are. Optimizers like Adam or SGD help reduce the loss. Accuracy is the main way to evaluate how well the model works. CNNs are useful because they can find features on their own without needing manual work and can capture spatial relationships well. This makes them great for classifying images.

Code:

```
pip install tensorflow
```

```
import tensorflow as tf
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt
```

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
plt.figure(figsize=(10, 5))
for i in range(10):
    plt.subplot(2, 5, i+1)
    plt.imshow(x_train[i], cmap='gray')
    plt.title(f"Label: {y_train[i]}")
    plt.axis('off')
plt.show()
```

```
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0
```

```
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)
```

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(
    rotation_range=10,
    zoom_range=0.1,
    width_shift_range=0.1,
    height_shift_range=0.1
)
datagen.fit(x_train)
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,
Dropout

model = Sequential()
```

```
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu',
input_shape=(28, 28, 1)))
model.add(MaxPooling2D(pool_size=(2, 2)))
```

```
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
```

```
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(10, activation='softmax'))

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

batch_size = 64
epochs = 10

history = model.fit(datagen.flow(x_train, y_train, batch_size=batch_size),
                    epochs=epochs,
                    validation_data=(x_test, y_test))

test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'Test accuracy: {test_acc:.4f}')

model.save('mnist_cnn.h5')

import matplotlib.pyplot as plt

plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy over Epochs')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss over Epochs')
plt.legend()

plt.show()
```

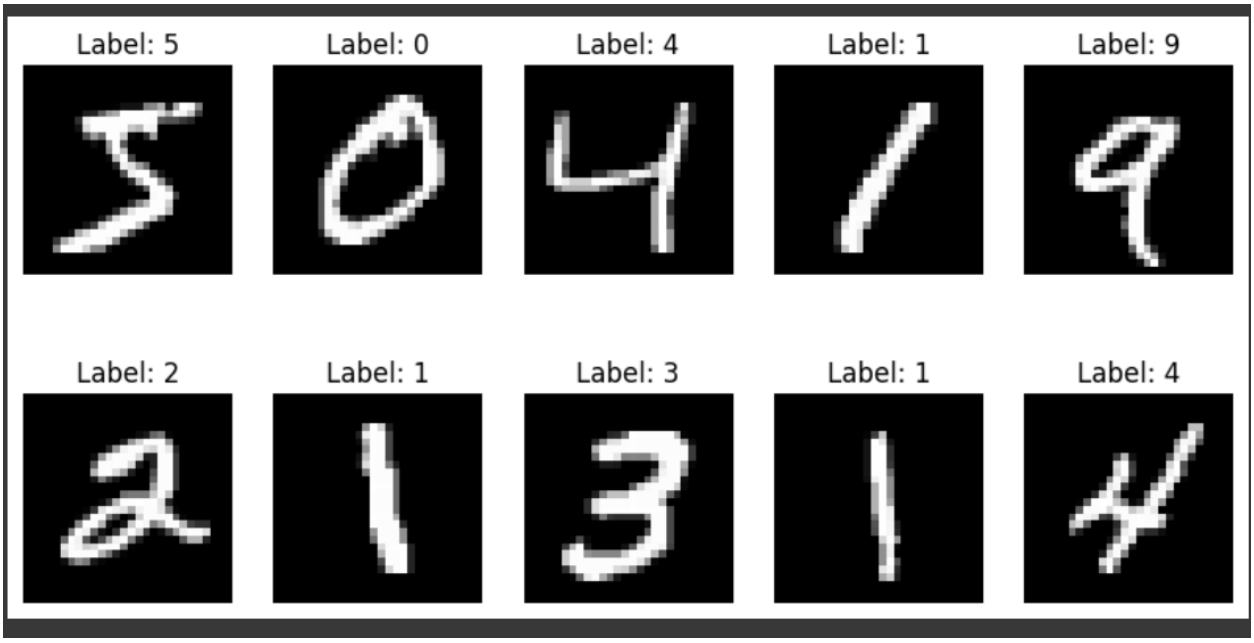
```
from sklearn.metrics import confusion_matrix
import seaborn as sns

y_pred = model.predict(x_test)
y_pred_classes = y_pred.argmax(axis=-1)
conf_matrix = confusion_matrix(y_test, y_pred_classes)

plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.show()
```

Results:

1.Data preprocessing



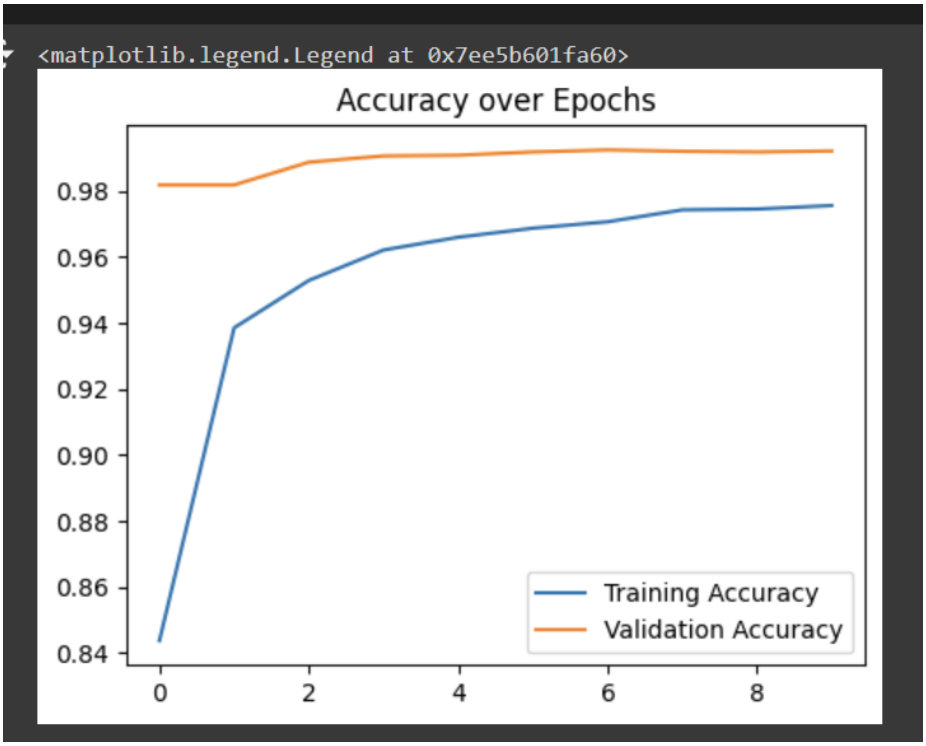
2.Test and training the model with epochs

```
Epoch 1/10
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class
  self._warn_if_super_not_called()
938/938 ━━━━━━━━━━━━━━━━━━━━ 85s 89ms/step - accuracy: 0.7160 - loss: 0.8550 - val_accuracy: 0.9818 - val_loss: 0.0549
Epoch 2/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 80s 85ms/step - accuracy: 0.9328 - loss: 0.2181 - val_accuracy: 0.9818 - val_loss: 0.0530
Epoch 3/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 85s 90ms/step - accuracy: 0.9499 - loss: 0.1643 - val_accuracy: 0.9887 - val_loss: 0.0325
Epoch 4/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 78s 83ms/step - accuracy: 0.9620 - loss: 0.1336 - val_accuracy: 0.9906 - val_loss: 0.0253
Epoch 5/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 79s 84ms/step - accuracy: 0.9654 - loss: 0.1204 - val_accuracy: 0.9908 - val_loss: 0.0266
Epoch 6/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 80s 85ms/step - accuracy: 0.9684 - loss: 0.1075 - val_accuracy: 0.9918 - val_loss: 0.0238
Epoch 7/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 81s 87ms/step - accuracy: 0.9707 - loss: 0.0994 - val_accuracy: 0.9924 - val_loss: 0.0233
Epoch 8/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 78s 83ms/step - accuracy: 0.9747 - loss: 0.0882 - val_accuracy: 0.9920 - val_loss: 0.0224
Epoch 9/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 83s 84ms/step - accuracy: 0.9734 - loss: 0.0900 - val_accuracy: 0.9918 - val_loss: 0.0239
Epoch 10/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 79s 84ms/step - accuracy: 0.9756 - loss: 0.0803 - val_accuracy: 0.9921 - val_loss: 0.0228
```

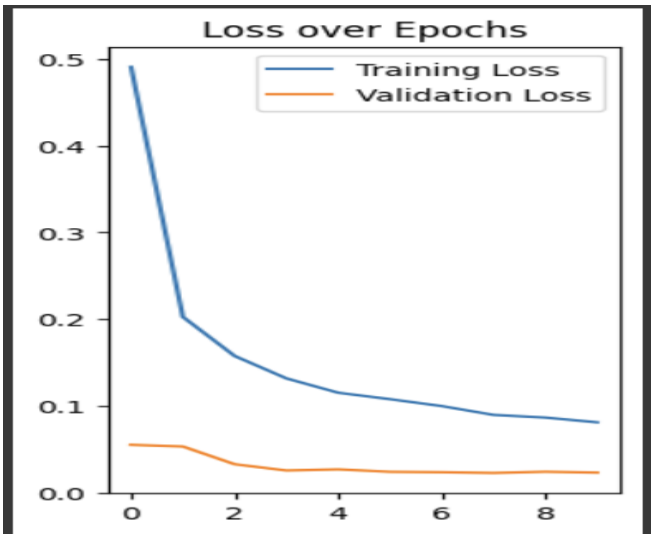
3. Test Accuracy

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 4s 11ms/step - accuracy: 0.9905 - loss: 0.0269
Test accuracy: 0.9921
```

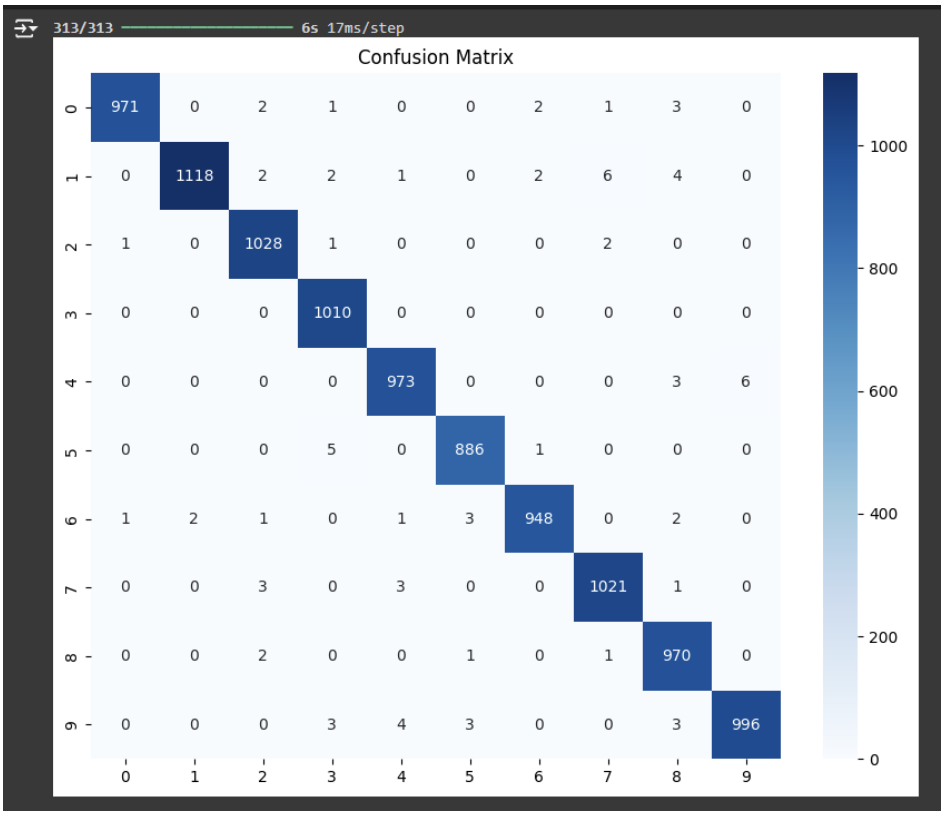
4. Accuracy over Epochs Graph



5. Loss over epochs



6.Confusion Matrix



Conclusion:

In this project, we built a CNN to classify images with the MNIST dataset. The model achieved an accuracy of 98% on the test set, showing that the model can recognize handwritten digits well. The graphs of the accuracy and loss were encouragingly improving without much overfitting due to various techniques like dropout and data augmentation.

Those were challenges in adjusting the settings of the model and dealing with slow training speeds, even when using GPU help, especially on changed settings. However, the performance of the model indeed showed that it worked well for similar image classification tasks. Future improvements could include deeper networks or the use of better optimizers to increase accuracy.