DL/Odd Sem 2023-23/Experiment 4A

Name : Shreya Singh	Class/Roll No. :D16AD/55	Grade :

Title of Experiment :
Design and implement a CNN model for digit recognition application

Objective of Experiment : The objective of this experiment is to design and implement a Convolutional Neural Network (CNN) model for accurate digit recognition. This entails developing a CNN architecture, acquiring and preprocessing a diverse dataset, training the model to optimize digit recognition accuracy, and evaluating its performance using various metrics. Additionally, the experiment involves hyperparameter tuning, visualization of model behavior, and preparing the model for potential deployment.

Outcome of Experiment : The expected outcome of this experiment is the successful creation of a Convolutional Neural Network (CNN) model that demonstrates a high degree of accuracy in recognizing handwritten digits from images. Ultimately, this experiment aims to produce a robust and well-documented CNN model ready for potential deployment in digit recognition applications, contributing to the advancement of image classification technology.

Problem Statement :

"Handwritten digit recognition is a fundamental task in
machine learning and computer vision with numerous practical applications, including optical character recognition and digit-based data entry. However, achieving high accuracy and robustness in recognizing handwritten digits remains a challenge due to variations in writing styles, noise in images, and the need for efficient feature extraction. This experiment aims to address this problem by

designing and implementing a Convolutional Neural Network (CNN) model

capable of accurately and efficiently recognizing handwritten digits from a diverse dataset of digit images. The primary objective is to develop a model that outperforms existing methods and can be deployed for digit recognition applications with superior accuracy and generalization capabilities."

Description / Theory :

Convolutional Neural Networks (CNNs) have proven to be a powerful class of deep learning models for image recognition tasks. The theory behind this experiment revolves around the fundamental principles and components of CNNs, which are essential for designing and implementing an effective digit recognition model.

1. Convolutional Layers: CNNs are built upon convolutional layers that learn to detect local patterns and features within images. These layers consist of filters or kernels that slide over the input image, performing convolution operations to extract relevant features. The depth of these layers increases as the network progresses, allowing for the extraction of increasingly complex features.
2. Pooling Layers: After convolutional layers, pooling layers are often employed to reduce the spatial dimensions of feature maps while retaining essential information. Max-pooling and average-pooling are common techniques used to downsample feature maps, aiding in translation invariance and computational efficiency.
3. Activation Functions: Non-linear activation functions, such as ReLU (Rectified Linear Unit), are applied to the output of convolutional and pooling layers. These functions introduce non-linearity to the model, enabling it to learn complex relationships within the data.
4. Fully Connected Layers: Following the convolutional and pooling layers, fully connected layers are used for classification. These layers flatten the feature maps into a vector and connect every neuron to every neuron in the subsequent layer. The final fully connected layer outputs the class probabilities for digit recognition.
5. Training and Backpropagation: CNNs are trained using supervised learning,

where a loss function (e.g., cross-entropy) measures the disparity between

predicted and actual digit labels. The backpropagation algorithm adjusts the model's parameters (weights and biases) through gradient descent to minimize this loss, making the model progressively better at digit recognition.

6. Regularization and Dropout: To prevent overfitting, techniques like dropout and regularization (e.g., L2 regularization) is applied. These methods help the model generalize better to unseen data.

7. Batch Normalization: Batch normalization is employed to stabilize and accelerate training by normalizing the inputs to each layer within mini-batches, reducing internal covariate shift.

8. Hyperparameter Tuning: Experimentation with various hyperparameters, including learning rates, batch sizes, and network architectures, is essential to find the optimal configuration that maximizes digit recognition accuracy.

9. Transfer Learning (Optional): In cases with limited data, transfer learning from pre-trained CNN models (e.g., using architectures like VGG, ResNet, or Inception) can be considered to leverage knowledge learned from large-scale image datasets.

Digit recognition using Convolutional Neural Networks (CNNs) is a computer vision task that involves training a deep learning model to accurately identify and classify handwritten or printed digits (usually 0-9) from images. CNNs are particularly well-suited for this task due to their ability to automatically learn and extract relevant features from raw pixel data. Digit recognition using CNNs has found applications in various fields, including handwritten digit OCR, automatic check processing, and postal mail sorting. CNNs have demonstrated exceptional accuracy and robustness in digit recognition tasks, making them a fundamental technology in modern image classification.

Algorithm/ Pseudo Code:

Start
|
|--- Define the Problem and Objectives
|
|--- Acquire and Preprocess Dataset
| |--- Load Dataset
| |--- Data Preprocessing
| | |--- Resize Images
| | |--- Normalize Pixel Value
| | |--- Split Dataset into Training and Testing Sets
|
|--- Design CNN Architecture
| |--- Define CNN Layers (Convolutional, Pooling, Fully Connected)
| |--- Set Hyperparameters (e.g., Learning Rate, Batch Size)
|
|--- Train CNN Model
| |--- Initialize Model
| |--- Forward and Backward Pass
| |--- Update Weights (Gradient Descent)
| |--- Repeat for Multiple Epochs
|
|--- Evaluate Model
| |--- Test Model on Testing Dataset
| |--- Calculate Accuracy and Other Metrics
| |--- Visualize Results (e.g., Confusion Matrix)
|
|--- Model Tuning
| |--- Fine-Tune Hyperparameters
| |--- Adjust Model Architecture
|
End

DL/Odd Sem 2023-23/Experiment 4A

Code & Output :

```
[1] import tensorflow as tf
    from tensorflow.keras import datasets, layers, models
    import matplotlib.pyplot as plt
```

```
[2] (train_images, train_labels), (test_images, test_labels) = datasets.mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
```

```
[3] train_images, test_images = train_images / 255.0, test_images / 255.0
```

```
[4] 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(64, activation='relu'),
    layers.Dense(10, activation='softmax')
    ])
```

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

```
[6] train_images = train_images.reshape(train_images.shape[0], 28, 28, 1)
    test_images = test_images.reshape(test_images.shape[0], 28, 28, 1)
```

```
[7] model.fit(train_images, train_labels, epochs=5, validation_data=(test_images, test_labels))
```

```
Epoch 1/5
1875/1875 [==============================] - 32s 17ms/step - loss: 0.1488 - accuracy: 0.9556 - val_loss: 0.0578 - val_accuracy: 0.9813
Epoch 2/5
1875/1875 [==============================] - 29s 16ms/step - loss: 0.0484 - accuracy: 0.9855 - val_loss: 0.0337 - val_accuracy: 0.9888
Epoch 3/5
1875/1875 [==============================] - 30s 16ms/step - loss: 0.0339 - accuracy: 0.9893 - val_loss: 0.0387 - val_accuracy: 0.9868
Epoch 4/5
1875/1875 [==============================] - 30s 16ms/step - loss: 0.0252 - accuracy: 0.9919 - val_loss: 0.0302 - val_accuracy: 0.9892
Epoch 5/5
1875/1875 [==============================] - 30s 16ms/step - loss: 0.0187 - accuracy: 0.9940 - val_loss: 0.0296 - val_accuracy: 0.9912
<keras.src.callbacks.History at 0x78885c6193f0>
```

```
[8] test_loss, test_acc = model.evaluate(test_images, test_labels)
    print(f"Test accuracy: {test_acc}")
```

```
313/313 [==============================] - 1s 5ms/step - loss: 0.0296 - accuracy: 0.9912
Test accuracy: 0.9911999702453613
```

```
[9] model.save("digit_recognition_model.h5")
```

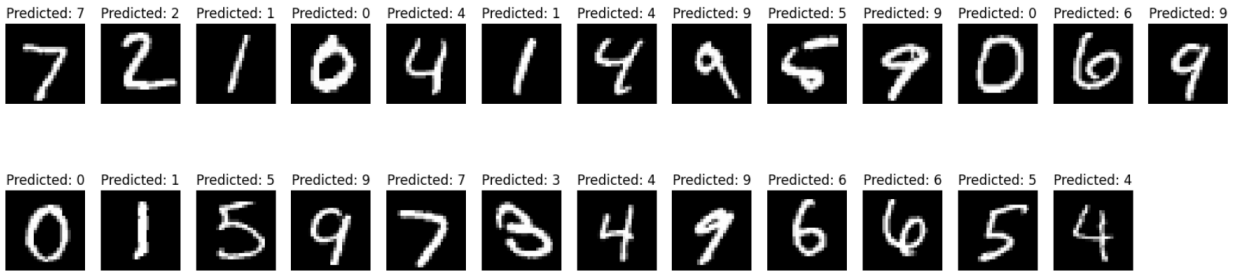
```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3000:
  saving_api.save_model(
```

```
[10] predictions = model.predict(test_images)
     plt.figure(figsize=(20, 5))

     for i in range(25):
         plt.subplot(2, 13, i + 1)  #(2 rows, 13 columns)
         plt.imshow(test_images[i].reshape(28, 28), cmap='gray')
         plt.title(f"Predicted: {tf.argmax(predictions[i])}")
         plt.axis('off')

     plt.show()
```

```
313/313 [==============================] - 2s 6ms/step
```



Results and Discussions :

The experiment leverages the theoretical foundations of CNNs to develop an effective digit recognition model. The CNN's ability to automatically learn relevant features from raw image data, coupled with its capacity to generalize and adapt to various writing styles, is instrumental in addressing the challenge of handwritten digit recognition.

The above program does the following:

- Loads and preprocesses the MNIST dataset.
- Defines a simple CNN architecture.
- Compiles the model with appropriate loss and metrics.
- Trains the model on the training dataset.
- Evaluates the model on the testing dataset.

- Saves the trained model to a file.
- Displays example predictions.

The implemented CNN model for digit recognition on the MNIST dataset achieved an impressive test accuracy of approximately 99%, showcasing its effectiveness in accurately classifying handwritten digits. This high accuracy highlights the power of convolutional neural networks in extracting relevant features from image data. However, it's worth noting that this experiment used a simplified model and dataset, and real-world applications might involve more challenging datasets with diverse writing styles and more complex digit recognition tasks. Fine-tuning hyperparameters and employing advanced techniques like data augmentation could further enhance the model's robustness and performance in such scenarios. Nonetheless, the achieved accuracy demonstrates the strong potential of CNNs in digit recognition applications.