Experiment No. 6
Design and implement a CNN model for digit recognition application
Date of Performance: 26/09/23
Date of Submission: 03/09/23

Aim: Design and implement a CNN model for digit recognition application.

Objective: Ability to design convolution neural network to solve the given problem

Theory:

A Convolutional Neural Network (CNN) is a type of Deep Learning neural network architecture commonly used in Computer Vision. Computer vision is a field of Artificial Intelligence that enables a computer to understand and interpret the image or visual data.
When it comes to Machine Learning, Artificial Neural Networks perform really well. Neural Networks are used in various datasets like images, audio, and text. Different types of Neural Networks are used for different purposes, for example for predicting the sequence of words we use Recurrent Neural Networks more precisely an LSTM, similarly for image classification we use Convolution Neural networks. In this blog, we are going to build a basic building block for CNN.

In a regular Neural Network there are three types of layers:

Input Layers: It's the layer in which we give input to our model. The number of neurons in this layer is equal to the total number of features in our data (number of pixels in the case of an image).
Hidden Layer: The input from the Input layer is then feed into the hidden layer. There can be many hidden layers depending upon our model and data size. Each hidden layer can have different numbers of neurons which are generally greater than the number of features. The output from each layer is computed by matrix multiplication of output of the previous layer with learnable weights of that layer and then by the addition of learnable biases followed by activation function which makes the network nonlinear.
Output Layer: The output from the hidden layer is then fed into a logistic function like sigmoid or softmax which converts the output of each class into the probability score of each class.

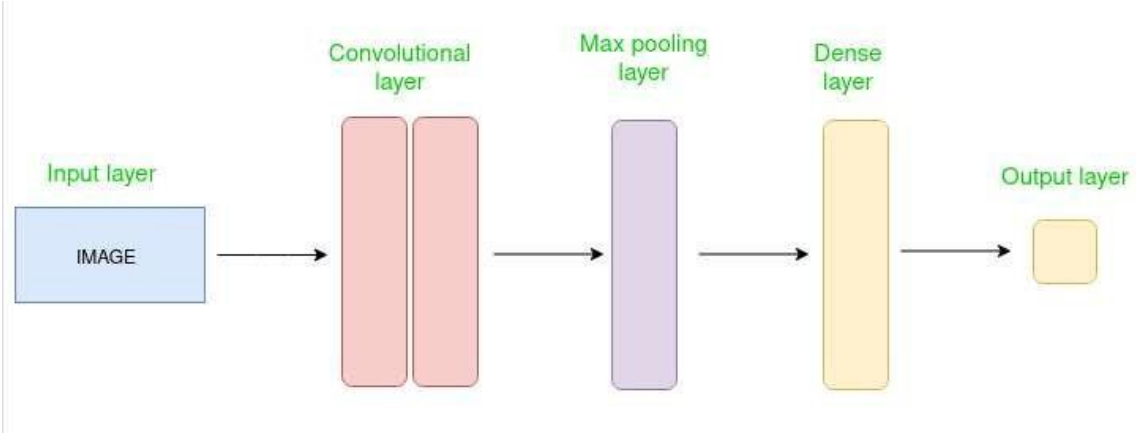
The data is fed into the model and output from each layer is obtained from the above step is called feedforward, we then calculate the error using an error function, some common error functions are cross-entropy, square loss error, etc. The error function measures how well the network is performing. After that, we backpropagate into the model by calculating the derivatives. This step is called Backpropagation which basically is used to minimize the loss.

Convolution neural network:

Convolutional Neural Network (CNN) is the extended version of artificial neural networks (ANN) which is predominantly used to extract the feature from the grid-like matrix dataset. For example visual datasets like images or videos where data patterns play an extensive role.

CNN architecture

Convolutional Neural Network consists of multiple layers like the input layer, Convolutional layer, Pooling layer, and fully connected layers.



The Convolutional layer applies filters to the input image to extract features, the Pooling layer downsamples the image to reduce computation, and the fully connected layer makes the final prediction. The network learns the optimal filters through backpropagation and gradient descent.

Layers In CNN:

Input Layers: It's the layer in which we give input to our model. In CNN, Generally, the input will be an image or a sequence of images. This layer holds the raw input of the image with width 32, height 32, and depth 3.

Convolutional Layers: This is the layer, which is used to extract the feature from the input dataset. It applies a set of learnable filters known as the kernels to the input images. The filters/kernels are smaller matrices usually 2×2, 3×3, or 5×5 shape. it slides over the input image data and computes the dot product between kernel weight and the corresponding input image patch. The output of this layer is referred ad feature maps. Suppose we use a total of 12 filters for this layer we'll get an output volume of dimension 32 x 32 x 12.

Activation Layer: By adding an activation function to the output of the preceding layer, activation layers add nonlinearity to the network. it will apply an element-wise activation function to the output of the convolution layer. Some common activation functions are RELU: max(0, x), Tanh, Leaky RELU, etc. The volume remains unchanged hence output volume will have dimensions 32 x 32 x 12.

Pooling layer: This layer is periodically inserted in the covnets and its main function is to reduce the size of volume which makes the computation fast reduces memory and also prevents overfitting. Two common types of pooling layers are max pooling and average pooling. If we use a max pool with 2 x 2 filters and stride 2, the resultant volume will be of dimension 16x16x12.

Flattening: The resulting feature maps are flattened into a one-dimensional vector after the convolution and pooling layers so they can be passed into a completely linked layer for categorization or regression.

Fully Connected Layers: It takes the input from the previous layer and computes the final classification or regression task.

Output Layer: The output from the fully connected layers is then fed into a logistic function for classification tasks like sigmoid or softmax which converts the output of each class into the probability score of each class.

Code:

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

```
[2] (x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_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

    y_train = tf.keras.utils.to_categorical(y_train, 10)
    y_test = tf.keras.utils.to_categorical(y_test, 10)
```

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

```
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')
])
```

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

```
[5]  model.fit(x_train, y_train, epochs=5, batch_size=64, validation_data=(x_test, y_test))

Epoch 1/5
938/938 [==============================] - 58s 60ms/step - loss: 0.1710 - accuracy: 0.9496 - val_loss: 0.0590 - val_accuracy: 0.9816
Epoch 2/5
938/938 [==============================] - 52s 55ms/step - loss: 0.0507 - accuracy: 0.9845 - val_loss: 0.0419 - val_accuracy: 0.9867
Epoch 3/5
938/938 [==============================] - 52s 56ms/step - loss: 0.0351 - accuracy: 0.9889 - val_loss: 0.0383 - val_accuracy: 0.9870
Epoch 4/5
938/938 [==============================] - 52s 55ms/step - loss: 0.0267 - accuracy: 0.9917 - val_loss: 0.0352 - val_accuracy: 0.9884
Epoch 5/5
938/938 [==============================] - 52s 55ms/step - loss: 0.0213 - accuracy: 0.9930 - val_loss: 0.0323 - val_accuracy: 0.9890
<keras.src.callbacks.History at 0x7c2977b49ae0>

test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'Test accuracy: {test_acc*100:.2f}%')

313/313 [==============================] - 3s 10ms/step - loss: 0.0323 - accuracy: 0.9890
Test accuracy: 98.90%
```



```
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'Test accuracy: {test_acc*100:.2f}%')

313/313 [==============================] - 3s 10ms/step - loss: 0.0323 - accuracy: 0.9890
Test accuracy: 98.90%

[7]  predictions = model.predict(x_test[:10])
     predicted_labels = [tf.argmax(prediction).numpy() for prediction in predictions]
     print(f'Predicted labels: {predicted_labels}')

     1/1 [==============================] - 0s 114ms/step
     Predicted labels: [7, 2, 1, 0, 4, 1, 4, 9, 5, 9]
```

Conclusion:

The designed CNN architecture for digit recognition demonstrates promising results. It effectively learns hierarchical features through convolutional layers, reduces spatial dimensions with max pooling, and makes accurate predictions with fully connected layers. However, further fine-tuning and experimentation with hyperparameters could potentially yield even better performance. Additionally, considerations for potential overfitting and real-world applicability should be taken into account.