Vivekanand Education Society's Institute of Technology
Approved by AICTE & Affiliated to University of Mumbai

Artificial Intelligence and Data Science Department
Deep Learning/Odd Sem 2023-23/Experiment 4a

Name: Subrato Tapaswi	**Class/Roll No.: D16AD/60**	**Grade:**

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

Objective of Experiment: The objective of the experiment is to design and implement a Convolutional Neural Network (CNN) model for the purpose of digit recognition in an application. This involves creating a deep learning architecture capable of precisely identifying and categorizing handwritten or printed digits into their corresponding numeric values.

Outcome of Experiment: The outcome of the experiment is the successful design and implementation of a Convolutional Neural Network (CNN) model customized for digit recognition applications. The CNN is anticipated to precisely classify hand-drawn or printed digits, showcasing its effectiveness in performing image recognition tasks.

Problem Statement: To design and implement a CNN model for digit recognition application. Use MNIST dataset for the same.

Description / Theory:
CNN:
A Convolutional Neural Network, or CNN for short, is a type of neural network designed for working with data that has a grid-like topology, such as images. Digital images are essentially a way to represent visual information in a binary format. They consist of a grid of tiny units called pixels, with each pixel having values that determine its brightness and color.

CNN Architecture:
A CNN typically has three layers: a convolutional layer, a pooling layer, and a fully connected layer.

1. Convolutional layer:
   This layer conducts a mathematical operation known as a dot product between two matrices. One matrix comprises learnable parameters referred to as a kernel, while the other matrix is a restricted part of the receptive field. The kernel is smaller in terms of spatial dimensions compared to an image but extends in-depth. This means that if an image has three (RGB) channels, the kernel will have small spatial height and width but cover all three channels in terms of depth.

   To calculate the size of the output volume for an input of dimensions W x W x D, with Dout kernels of spatial size F, a stride of S, and padding of P, you can use the following formula:
   $$\mathbf{W_{out} = [(W - F + 2P) / S] + 1}$$

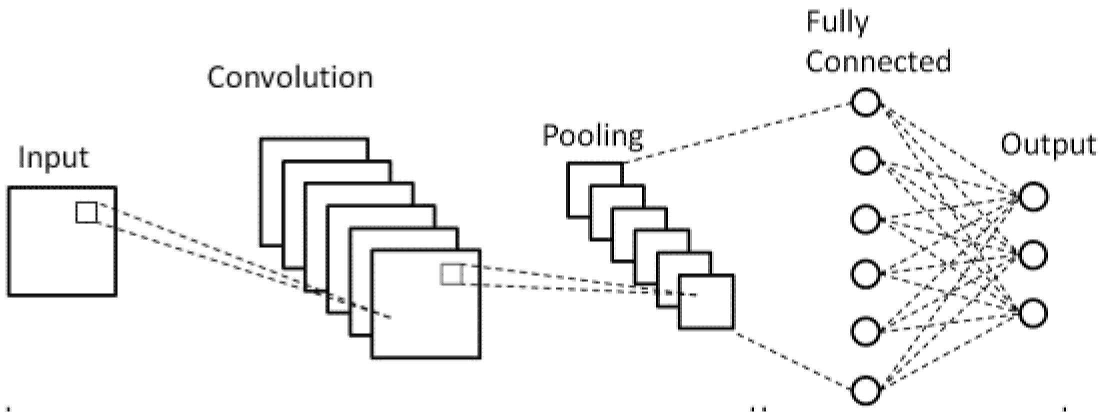
2. Pooling layer:

   The pooling layer takes the place of certain network outputs by calculating a summary statistic of nearby outputs. This is done to reduce the spatial dimensions of the representation, which in turn reduces the computational workload and the number of parameters needed. The pooling operation is performed independently on each slice of the representation.

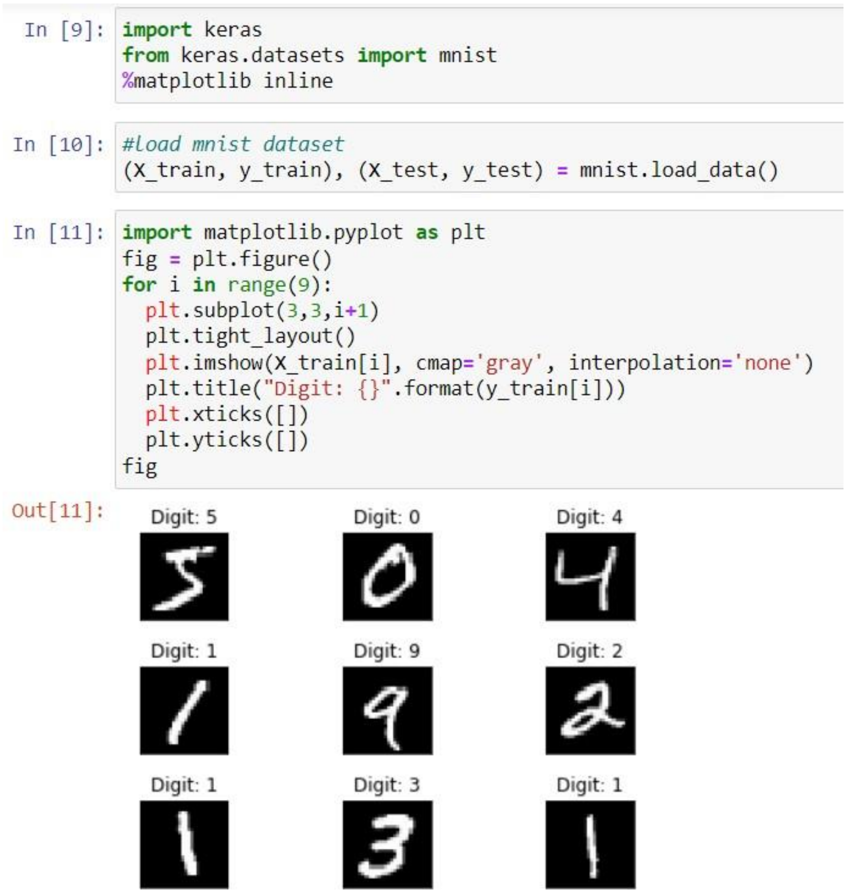
   If you have an activation map with dimensions W x W x D, use a pooling kernel of size F and a stride of S, you can calculate the size of the output volume using the following formula:
   $$\mathbf{W_{out} = [(W - F) / S] + 1}$$

3. Fully connected layer:
   In this layer, neurons are fully connected to all neurons in both the previous and the next layer, just like in a typical Fully Connected Neural Network (FCNN). This means that you can compute the output of this layer through standard matrix multiplication, followed by the addition of bias values.

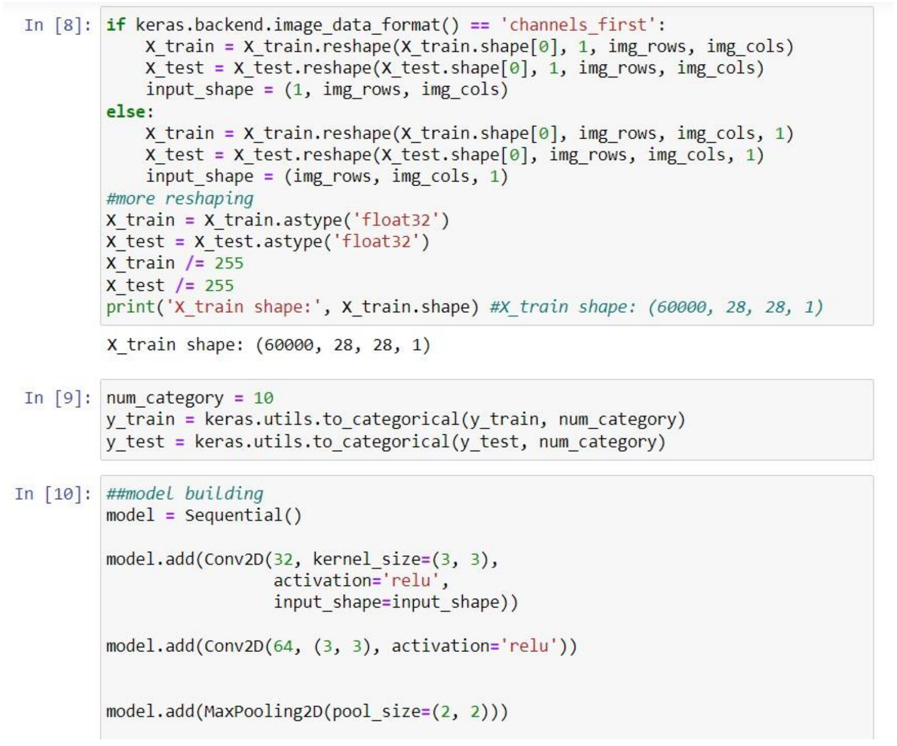
   The Fully Connected (FC) layer plays a crucial role in mapping the representation from the input to the output, allowing for complex relationships and patterns to be learned and captured in the network.



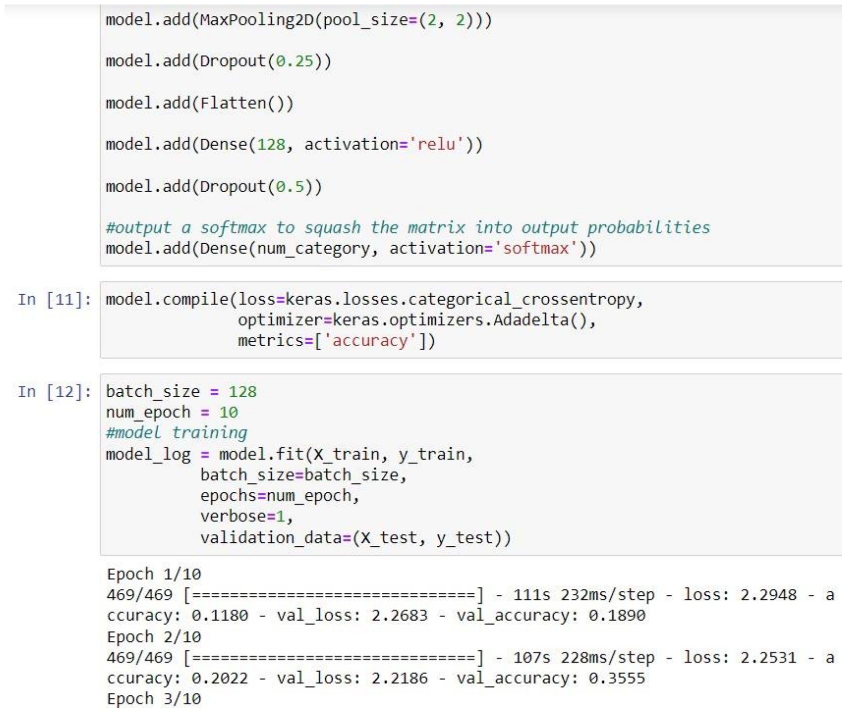
Program:



```
In [9]:  import keras
         from keras.datasets import mnist
         %matplotlib inline

In [10]: #load mnist dataset
         (X_train, y_train), (X_test, y_test) = mnist.load_data()

In [11]: import matplotlib.pyplot as plt
         fig = plt.figure()
         for i in range(9):
           plt.subplot(3,3,i+1)
           plt.tight_layout()
           plt.imshow(X_train[i], cmap='gray', interpolation='none')
           plt.title("Digit: {}".format(y_train[i]))
           plt.xticks([])
           plt.yticks([])
         fig

Out[11]:
```



```
In [8]: if keras.backend.image_data_format() == 'channels_first':
            X_train = X_train.reshape(X_train.shape[0], 1, img_rows, img_cols)
            X_test = X_test.reshape(X_test.shape[0], 1, img_rows, img_cols)
            input_shape = (1, img_rows, img_cols)
        else:
            X_train = X_train.reshape(X_train.shape[0], img_rows, img_cols, 1)
            X_test = X_test.reshape(X_test.shape[0], img_rows, img_cols, 1)
            input_shape = (img_rows, img_cols, 1)
        #more reshaping
        X_train = X_train.astype('float32')
        X_test = X_test.astype('float32')
        X_train /= 255
        X_test /= 255
        print('X_train shape:', X_train.shape) #X_train shape: (60000, 28, 28, 1)

        X_train shape: (60000, 28, 28, 1)

In [9]: num_category = 10
        y_train = keras.utils.to_categorical(y_train, num_category)
        y_test = keras.utils.to_categorical(y_test, num_category)

In [10]: ##model building
         model = Sequential()

         model.add(Conv2D(32, kernel_size=(3, 3),
                          activation='relu',
                          input_shape=input_shape))

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


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



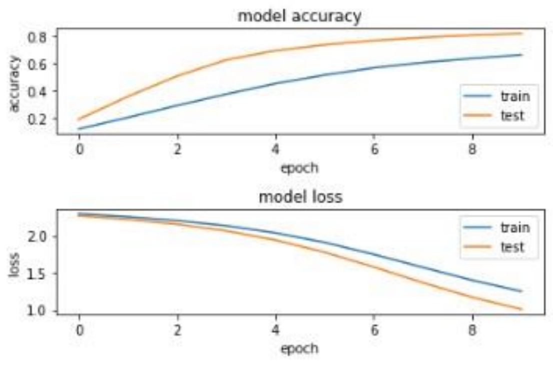
```
         model.add(MaxPooling2D(pool_size=(2, 2)))

         model.add(Dropout(0.25))

         model.add(Flatten())

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

         model.add(Dropout(0.5))

         #output a softmax to squash the matrix into output probabilities
         model.add(Dense(num_category, activation='softmax'))

In [11]: model.compile(loss=keras.losses.categorical_crossentropy,
                       optimizer=keras.optimizers.Adadelta(),
                       metrics=['accuracy'])

In [12]: batch_size = 128
         num_epoch = 10
         #model training
         model_log = model.fit(X_train, y_train,
                   batch_size=batch_size,
                   epochs=num_epoch,
                   verbose=1,
                   validation_data=(X_test, y_test))

         Epoch 1/10
         469/469 [==============================] - 111s 232ms/step - loss: 2.2948 - a
         ccuracy: 0.1180 - val_loss: 2.2683 - val_accuracy: 0.1890
         Epoch 2/10
         469/469 [==============================] - 107s 228ms/step - loss: 2.2531 - a
         ccuracy: 0.2022 - val_loss: 2.2186 - val_accuracy: 0.3555
         Epoch 3/10
```

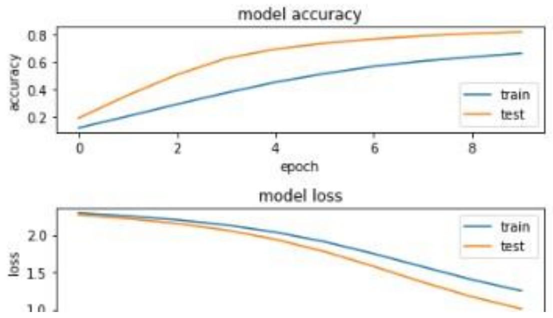
Output:

```
Test loss: 1.0082017183303833
Test accuracy: 0.8167999982833862
```

```
In [15]: import os
         # plotting the metrics
         fig = plt.figure()
         plt.subplot(2,1,1)
         plt.plot(model_log.history['accuracy'])
         plt.plot(model_log.history['val_accuracy'])
         plt.title('model accuracy')
         plt.ylabel('accuracy')
         plt.xlabel('epoch')
         plt.legend(['train', 'test'], loc='lower right')
         plt.subplot(2,1,2)
         plt.plot(model_log.history['loss'])
         plt.plot(model_log.history['val_loss'])
         plt.title('model loss')
         plt.ylabel('loss')
         plt.xlabel('epoch')
         plt.legend(['train', 'test'], loc='upper right')
         plt.tight_layout()
         fig
```

Out[15]:





Results and Discussions: To sum it up, we successfully created and used a Convolutional Neural Network (CNN) to recognize digits. The CNN performed really well in recognizing digits, showing that deep learning is quite effective for tasks like identifying digits in images.