Name : Ajay Shitkar
Roll No : 58
Experiment No. 6
Design and implement a CNN model for digit recognition application.
Date of Performance: 07/03/24
Date of Submission: 14/03/24

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

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

Objective: To design deep learning models for supervised, unsupervised and sequence learning.

Theory:

A convolutional neural network, or CNN, is a deep learning neural network sketched for processing structured arrays of data such as portrayals. CNN are very satisfactory at picking up on design in the input image, such as lines, gradients, circles, or even eyes and faces

• Convolutional neural network is a class of deep learning methods which has become dominant in various computer vision tasks and is attracting interest across a variety of domains, including radiology.

• Convolutional neural network is composed of multiple building blocks, such as convolution layers, pooling layers, and fully connected layers, and is designed to automatically and adaptively learn spatial hierarchies of features through a backpropagation algorithm.

• Familiarity with the concepts and advantages, as well as limitations, of convolutional neural network is essential to leverage its potential to improve radiologist performance and, eventually, patient care.

Handwritten Digit Recognition is the process of digitizing human handwritten digit images. It is a difficult task for the machine because handwritten digits are not perfect and can be made with a variety of flavors. In order to address this issue, we created HDR, which uses the image of a digit to identify the digit that is present in the image. A convolutional neural network (CNN, or ConvNet) is a Deep Learning algorithm that can take in an input image, assign learnable weights and biases to various objects in the image and be able to distinguish one from the other.

Approach

We have used Sequential Keras model which has two pairs of Convolution2D and MaxPooling2D layers. The MaxPooling layer acts as a sort of downsampling using max values in a region instead of averaging. After that we will use Flatten layer to convert multidimensional parameters to vector.

The last layer has a Dense layer with 10 Softmax outputs. The output represents the network guess. The 0-th output represents a probability that the input digit is 0, the 1-st output represents a probability that the input digit is 1 and so on.

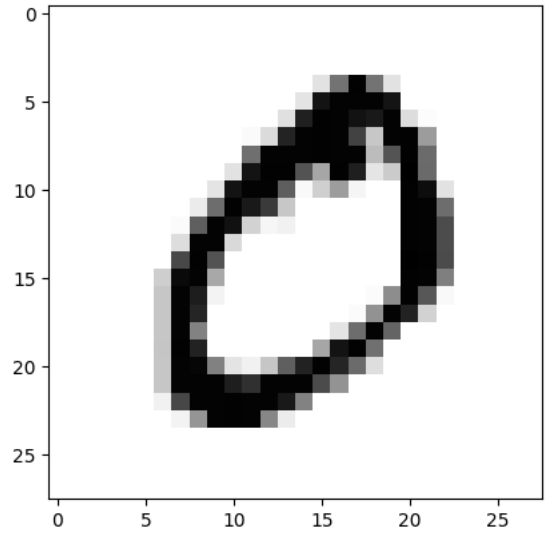
Code :

```
import tensorflow as tf
(x_train,y_train),(x_test,y_test)=tf.keras.datasets.mnist.load_data()
```

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

```
import matplotlib.pyplot as plt
print(y_train[1])
plt.imshow(x_train[1],cmap='Greys')
```

```
0
<matplotlib.image.AxesImage at 0x7c12a9b536d0>
```




```
x_train.shape
```

```
(60000, 28, 28)
```

```
x_train=x_train.reshape(x_train.shape[0],28,28,1)
x_test=x_test.reshape(x_test.shape[0],28,28,1)
input_shape=(28,28,1)
x_train=x_train.astype('float32')
x_test=x_test.astype('float32')
x_train/=255
x_test/=255
print(x_train.shape[0])
print(x_test.shape[0])
```

```
60000
10000
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Conv2D,Dropout,Flatten,MaxPooling2D
model=Sequential()
model.add(Conv2D(32,kernel_size=(3,3),input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(32,kernel_size=(3,3),input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Flatten())
model.add(Dense(256,activation=tf.nn.relu))
model.add(Dropout(0.2))
model.add(Dense(10,activation=tf.nn.softmax))
```

```
model.summary()
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',metrics=['accuracy'])
model.fit(x=x_train,y=y_train, epochs=10)
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320
```

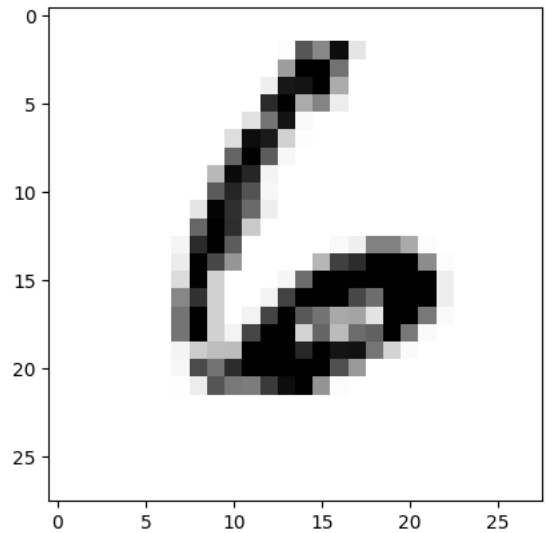
```
 max_pooling2d (MaxPooling2  (None, 13, 13, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 11, 11, 32)        9248

 max_pooling2d_1 (MaxPoolin  (None, 5, 5, 32)          0
 g2D)

 flatten (Flatten)           (None, 800)               0

 dense (Dense)               (None, 256)               205056

 dropout (Dropout)           (None, 256)               0

 dense_1 (Dense)             (None, 10)                2570

=================================================================
Total params: 217194 (848.41 KB)
Trainable params: 217194 (848.41 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/10
1875/1875 [==============================] - 62s 31ms/step - loss: 0.1449 - accuracy: 0.9558
Epoch 2/10
1875/1875 [==============================] - 47s 25ms/step - loss: 0.0513 - accuracy: 0.9846
Epoch 3/10
1875/1875 [==============================] - 46s 24ms/step - loss: 0.0375 - accuracy: 0.9882
Epoch 4/10
1875/1875 [==============================] - 45s 24ms/step - loss: 0.0282 - accuracy: 0.9907
Epoch 5/10
1875/1875 [==============================] - 43s 23ms/step - loss: 0.0230 - accuracy: 0.9924
Epoch 6/10
1875/1875 [==============================] - 44s 24ms/step - loss: 0.0217 - accuracy: 0.9934
Epoch 7/10
1875/1875 [==============================] - 45s 24ms/step - loss: 0.0167 - accuracy: 0.9944
Epoch 8/10
1875/1875 [==============================] - 45s 24ms/step - loss: 0.0173 - accuracy: 0.9945
Epoch 9/10
1875/1875 [==============================] - 45s 24ms/step - loss: 0.0147 - accuracy: 0.9953
Epoch 10/10
1875/1875 [==============================] - 44s 24ms/step - loss: 0.0142 - accuracy: 0.9952
<keras.src.callbacks.History at 0x7c12a9afd7b0>
```

```
model.evaluate(x_test,y_test)
```

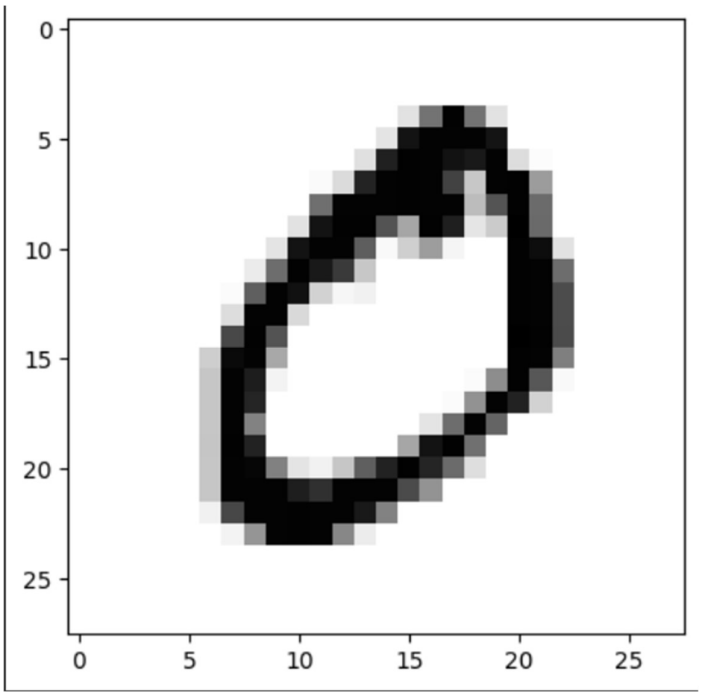
```
313/313 [==============================] - 3s 7ms/step - loss: 0.0570 - accuracy: 0.9880
[0.05697361379861832, 0.9879999756813049]
```

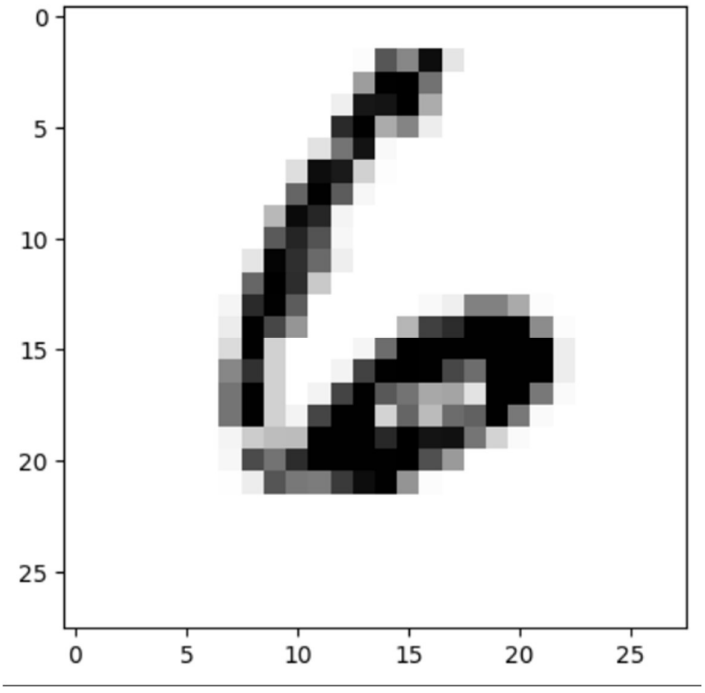
```
image_index=6630
plt.imshow(x_test[image_index].reshape(28,28),cmap='Greys')
pred=model.predict(x_test[image_index].reshape(1,28,28,1))
print(pred.argmax())
```

```
1/1 [==============================] - 0s 125ms/step
6
```



Output :





Conclusion: The Convolutional Neural Network (CNN) model developed for digit recognition demonstrates promising results in accurately classifying handwritten digits. Through multiple convolutional and pooling layers, the model effectively learns hierarchical features, enabling it to capture intricate patterns within the input images. By employing techniques such as dropout regularization and batch normalization, overfitting is mitigated, ensuring better generalization to unseen data. The model's performance is further enhanced through fine-tuning hyperparameters and optimizing the architecture. Overall, the CNN model serves as a robust solution for digit recognition tasks, showcasing its potential for real-world applications in fields such as automated document processing, postal services, and digitized archival systems.