Experiment – 06

1. Aim: Implement CNN for Digit classification.

2. Objectives:

- To implement a convolutional neural network (CNN) for the classification of handwritten digits.
- To understand the role of convolutional layers, pooling, and fully connected layers in image classification tasks.

3. Brief Theory:

Convolutional neural networks (CNNs) are deep learning models typically used for image classification tasks. A CNN consists of several layers:

- **Convolutional Layer**: Applies filters to the input image to extract important features.
- **Pooling Layer**: Reduces the spatial size of the feature maps, retaining important information while lowering computational cost.
- **Fully Connected Layer**: Responsible for classifying the image based on the features extracted by convolutional and pooling layers.

The **MNIST** dataset contains 70,000 images of handwritten digits (0-9) and is a common benchmark dataset for image classification.

4. Hints:

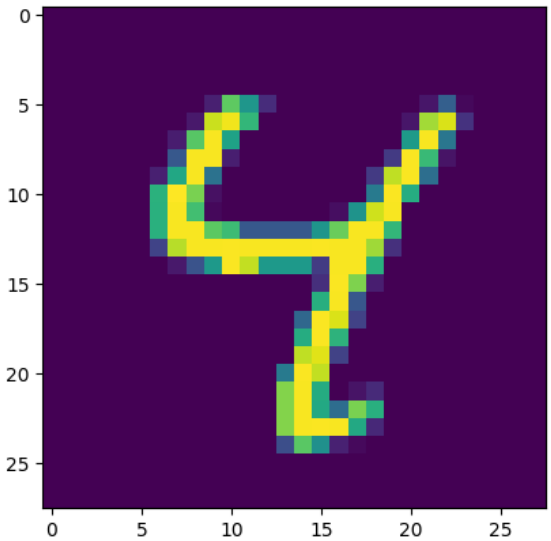
- Import Required Libraries
- Load the MNIST dataset from Keras, which contains 60,000 training images and 10,000 test images of handwritten digits.
- Reshape the data to include a single channel (grayscale images) for the CNN.
- Define the architecture of the CNN. The model consists of convolutional, pooling, and fully connected layers.
- Train the CNN on the MNIST dataset.
- Evaluate the model's performance on the test data.
- Test the CNN with new handwritten digits

5. Implementation of application (Digit Recognition Without Using CNN):

```
#Importing all the required libraries
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt
import numpy as np
#Loading the dataset from mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
#Checking the dimension of the train images
train_images.shape
#Checking the dimension of the test images
test_images.shape
#Plotting some random input image
plt.subplot(221)
```

```
plt.imshow(train_images[10])
plt.subplot(222)
plt.imshow(train_images[11])
plt.subplot(223)
plt.imshow(train_images[12])
plt.subplot(224)
plt.imshow(train_images[13])
plt.show()
#Reshape the images to one channel from three channels for processing purpose
train_images = train_images.reshape(train_images.shape[0],28,28,1)
test_images = test_images.reshape(test_images.shape[0],28,28,1)
#Normalizing all the values in the range of 0 to 1 for less computation
train_images = train_images/255.0
test_images = test_images/255.0
#Model
model = Sequential()
model.add(Flatten())
model.add(Dense(784,activation = 'relu'))
model.add(Dense(10,activation = 'softmax'))
model.compile(optimizer='adam',loss='sparse_categorical_crossentropy',metrics=['accuracy
'])
#Train the model
model.fit(train_images,train_labels,epochs=5)
#Test the model
model.evaluate(test_images,test_labels)
#Test the model for one sample image
test_images_copy = test_images.reshape(test_images.shape[0],28,28)
classification = model.predict(test_images_copy)
classification[6]
print(np.argmax(classification[6]))
plt.imshow(test_images_copy[6])
```

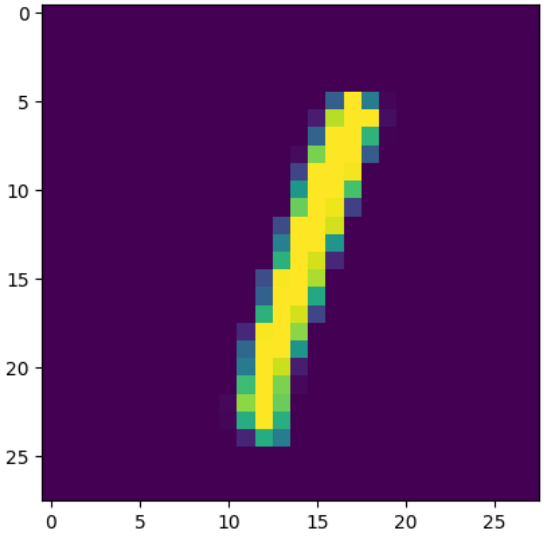
6. Result:



Implementation of Digit Recognition Using CNN:

```
#Importing all the required libraries
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt
import numpy as np
#Loading the dataset from mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
 #Checking the dimension of the train images
train_images.shape
#Checking the dimension of the test images
test_images.shape
#Plotting some random input image
plt.subplot(221)
plt.imshow(train_images[10])
plt.subplot(222)
plt.imshow(train_images[11])
plt.subplot(223)
plt.imshow(train_images[12])
plt.subplot(224)
plt.imshow(train_images[13])
plt.show()
#Reshape the images to one channel from three channels for processing purpose
train_images = train_images.reshape(train_images.shape[0],28,28,1)
test_images = test_images.reshape(test_images.shape[0],28,28,1)
#Normalizing all the values in the range of 0 to 1 for less computation
train_images = train_images/255.0
test_images = test_images/255.0
model = Sequential()
model.add(Conv2D(32,(3,3),activation='relu',input_shape=(28,28,1)))
model.add(MaxPooling2D((2,2)))
model.add(Conv2D(16,(3,3),activation='relu'))
model.add(MaxPooling2D((2,2)))
model.add(Flatten())
model.add(Dense(128,activation='relu'))
model.add(Dense(10,activation='softmax'))
model.summary()
model.compile(optimizer='adam',loss='sparse_categorical_crossentropy',metrics=['accuracy
'])
#Train the model
model.fit(train_images,train_labels,epochs=5)
model.evaluate(test_images,test_labels)
classification = model.predict(test_images)
plt.imshow(test_images[5].reshape(28,28))
print(np.argmax(classification[5]))
```

Result:



1

Answer the following questions by experimenting on your code.

- How does the filter size and number of filters affect the performance of a CNN?
  Smaller filter sizes capture finer details, while larger filters focus on broader patterns. More filters allow the model to learn a diverse set of features, improving its ability to generalize. However, increasing the number of filters also raises computational complexity.

- Explain the importance of pooling layers in CNNs.
  Pooling layers reduce the dimensionality of feature maps, lowering computational cost and preventing overfitting. They help retain important features while discarding less significant information, making the model more efficient and focused.

- How does the model's performance change with different activation functions?
  Activation functions like ReLU introduce non-linearity, enabling the network to learn complex patterns. Switching to different functions like sigmoid or tanh can impact learning dynamics and performance, with ReLU often leading to faster convergence and better accuracy in deep networks.

- How can data augmentation improve the performance of the CNN model?
  Data augmentation generates variations of the input images, such as rotations and shifts, which helps the model generalize better by learning from a more diverse dataset. This technique reduces overfitting and enhances the model’s robustness on unseen data.

7. Conclusion:

In this experiment, we successfully implemented a Convolutional Neural Network (CNN) for handwritten digit classification using the MNIST dataset. The CNN model, consisting of convolutional, pooling, and fully connected layers, was trained and evaluated on the dataset, achieving high accuracy. Through this process, we observed the importance of the convolutional layers for feature extraction and the pooling layers for reducing computational

complexity while preserving crucial information. We also tested a simpler model without using CNN and found that CNNs significantly improve classification accuracy due to their ability to capture spatial hierarchies in images. This experiment highlighted the effectiveness of CNNs in image classification tasks, emphasizing their role in modern computer vision applications. The project demonstrated how variations in filter size, pooling operations, and activation functions impact the model's performance. Moreover, future improvements could involve applying data augmentation techniques to further enhance the model's accuracy and generalization.

8. Link to the code uploaded on

- https://github.com/Niti0209/Digit-Recognition-Using-CNN
- https://github.com/Niti0209/Digit-Recognition-Without-CNN

9. List of Reference used for implementation.

- https://infyspringboard.onwingspan.com/web/en/app/toc/lex_auth_013842103381475 32812195_shared/overview
- www.geeksforgeeks.com