Experiment No. 6
Design and implement a CNN model for digit recognition application
Date of Performance: 12/09/23
Date of Submission: 26/09/23

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

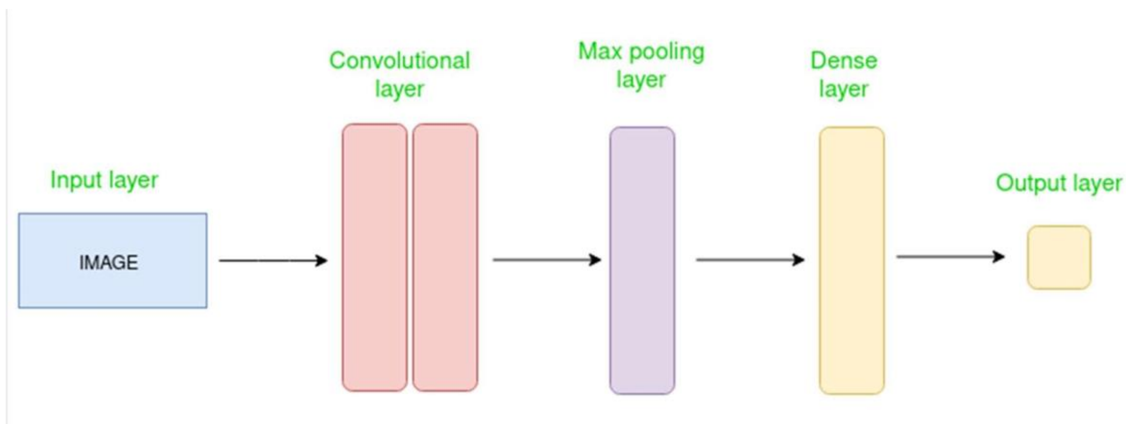
called feedforward, we then calculate the error using an error function, some common error functions are cross-entropy, square loss error, etc. The error function measures how well the network is performing. After that, we backpropagate into the model by calculating the derivatives. This step is called 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.

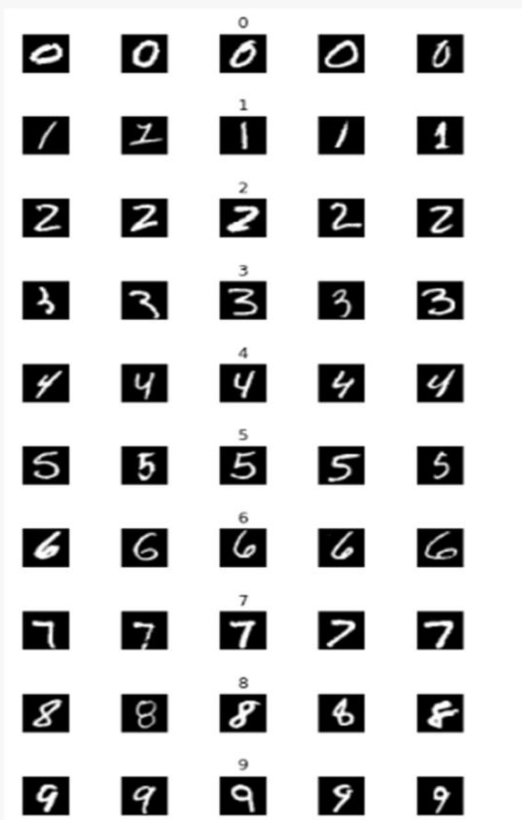
Fully Connected Layers: It takes the input from the previous layer and computes the final classification or regression task.

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:

```
# Imports import numpy as np import matplotlib.pyplot as plt import

keras from keras.datasets import mnist from keras.models import

Sequential from keras.layers import Dense   from keras.optimizers

import Adam from keras.utils.np_utils import to_categorical import

random

# To get same data whenever called np.random.seed(0)

# importing training data to obtain the parameters and test data to evaluate the performance of the neural
network.
(X_train, y_train), (X_test, y_test) = mnist.load_data()
# (dataset size, width, height) is the output print(X_train.shape)
print(X_test.shape) print(y_train.shape[0]) # no.of labels
# Conditions to be satisfied: assert(X_train.shape[0] == y_train.shape[0]), "The number of images is not
equal to the number of labels." assert(X_test.shape[0] == y_test.shape[0]), "The number of images is not
equal to the number of labels." assert(X_train.shape[1:] == (28,28)), "The dimensions of the images are
not 28x28" assert(X_test.shape[1:] == (28,28)), "The dimensions of the images are not 28x28"
# Visulalize the no.of images in each class (from 0 to 9)
# array to record no.of images in each of our ten categories num_of_samples = [] cols =
5 num_classes = 10
```

```
# subplots allow you to display multiple plots on the same figure. It also returns tuples which contains 2 values, an instance of our figure and plot axis.
fig, axs = plt.subplots(nrows=num_classes, ncols = cols, figsize=(5, 10))
fig.tight_layout()  # To avoid overlapping of plots

# creating a nested for loop arrangement that cycles through our data and counts it up.
for i in range(cols):
    for j in range(num_classes):
        x_selected = X_train[y_train == j]
        axs[j][i].imshow(x_selected[random.randint(0, (len(x_selected) - 1)), :, :], cmap=plt.get_cmap('gray')) # random images from the dataset are shown to see how different the digits are in the same class.
        axs[j][i].axis("off") # To remove axis
#           Adding titles to each row like 0,1,2,3......9
        if i == 2:
            axs[j][i].set_title(str(j))
            num_of_samples.append(len(x_selected))
```



```
print("No.of Samples:", num_of_samples) # shows the no.of images
```

```
belonging to each class

# Lets visualize this with bar plots
plt.figure(figsize=(12, 4))
plt.bar(range(0, num_classes), num_of_samples)
plt.title("Distribution of the training dataset")
plt.xlabel("Class number")
plt.ylabel("Number of images")
```

```
No.of Samples: [5923, 6742, 5958, 6131, 5842, 5421, 5918, 6265, 5851, 5949]
Text(0, 0.5, 'Number of images')
```


```
# adding depth

X_train = X_train.reshape(60000, 28, 28, 1)
X_test = X_test.reshape(10000, 28, 28, 1)

# First perform One hot encoding on train and test data, which is
necessary for multi class classification.
y_train = to_categorical(y_train, 10) # (labels to encode, total no.of
classes)
y_test = to_categorical(y_test, 10)
# Normalize the data
X_train = X_train/255
X_test = X_test/255

# Creating the model
from keras.layers import Flatten      # To flatten our data
from keras.layers.convolutional import Conv2D  # for Convolutional
layers
from keras.layers.convolutional import MaxPooling2D # for pooling
layers
```

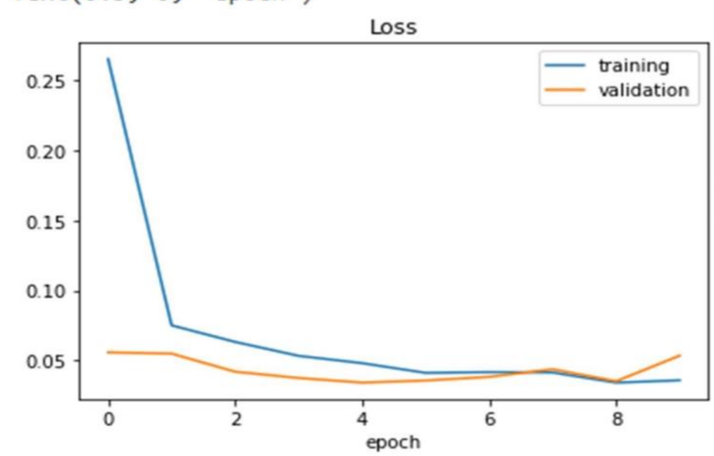
```
from keras.layers import Dropout
from keras.models import Model

# define LeNet func
def leNet_model():
  model = Sequential()
  model.add(Conv2D(30, (5, 5), input_shape=(28, 28, 1),
activation='relu')) # Note 1
  model.add(MaxPooling2D(pool_size=(2,2))) # Note 2
  model.add(Conv2D(15, (3, 3), activation='relu')) # Note 3
  model.add(MaxPooling2D(pool_size=(2, 2)))
  model.add(Flatten())  # Note 4
  model.add(Dense(500, activation='relu')) # Note 5
  model.add(Dropout(0.5)) # Have a look at the plots below and comment
this dropout layer to see the change in the plots.
  model.add(Dense(num_classes, activation='softmax'))  # output layer
with no.of nodes = no.of classes.
  model.compile(Adam(learning_rate=0.01),
loss="categorical_crossentropy", metrics=["accuracy"])
  return model

# Seeing the summary gives us an overview of our Convolutional model

model = leNet_model()
print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 24, 24, 30)        780
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 12, 12, 30)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 10, 10, 15)        4065
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 15)          0
_________________________________________________________________
flatten (Flatten)            (None, 375)               0
_________________________________________________________________
dense (Dense)                (None, 500)               188000
_________________________________________________________________
dropout (Dropout)            (None, 500)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                5010
=================================================================
Total params: 197,855
Trainable params: 197,855
Non-trainable params: 0
_________________________________________________________________
None
```

```
# Train the model usiing model.fit. Remember that model.fit gives the
```

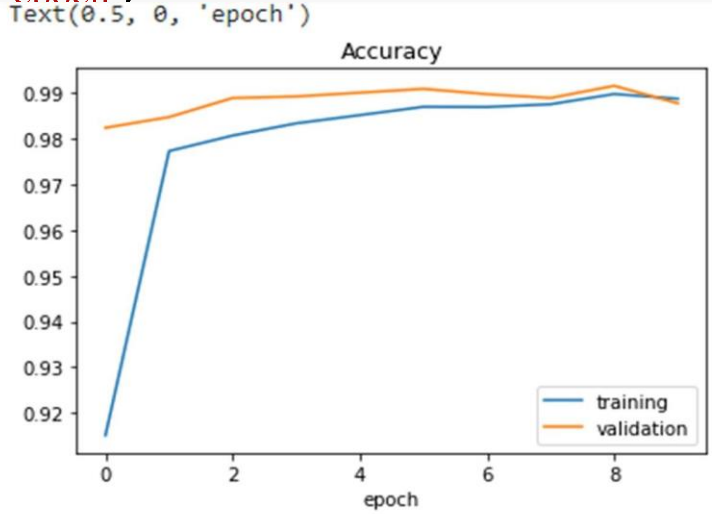
```
history
history = model.fit(X_train, y_train, epochs =10,
validation_split=0.1, batch_size=400, verbose=1, shuffle=1)

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(['training', 'validation'])
plt.title('Loss')
plt.xlabel('epoch')
```

```
Text(0.5, 0, 'epoch')
```



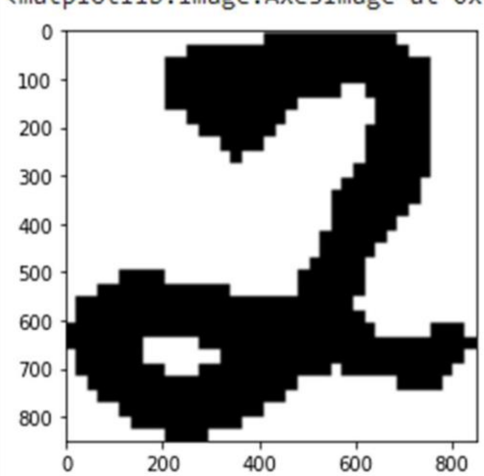

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.legend(['training','validation'])
plt.title('Accuracy')
plt.xlabel('epoch')
```

```
Text(0.5, 0, 'epoch')
```



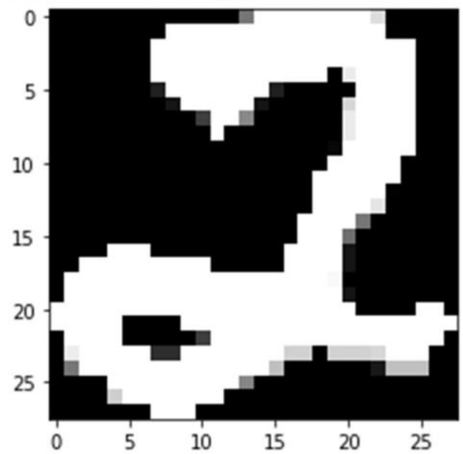
```
# Testing our model on new external image
# url for number 2
https://www.researchgate.net/profile/Jose_Sempere/publication/221258631/figure/fig1/AS:305526891139075@1449854695342/Handwritten-digit-2.png

import requests
from PIL import Image

url =
'https://www.researchgate.net/profile/Jose_Sempere/publication/221258631/figure/fig1/AS:305526891139075@1449854695342/Handwritten-digit-2.png'
response = requests.get(url, stream=True)
img = Image.open(response.raw)
plt.imshow(img, cmap=plt.get_cmap('gray'))
```

```
<matplotlib.image.AxesImage at 0x7fb69cf47990>
```




```
import cv2

array_img = np.asarray(img)
resized_img = cv2.resize(array_img, (28, 28))
print("resized image shape:", resized_img.shape)
gray_img = cv2.cvtColor(resized_img, cv2.COLOR_BGR2GRAY)
print("Grayscale image shape:", gray_img.shape)
image = cv2.bitwise_not(gray_img)
plt.imshow(image, cmap=plt.get_cmap('gray'))
```

```
resized image shape: (28, 28, 4)
Grayscale image shape: (28, 28)
<matplotlib.image.AxesImage at 0x7fb69cd70310>
```

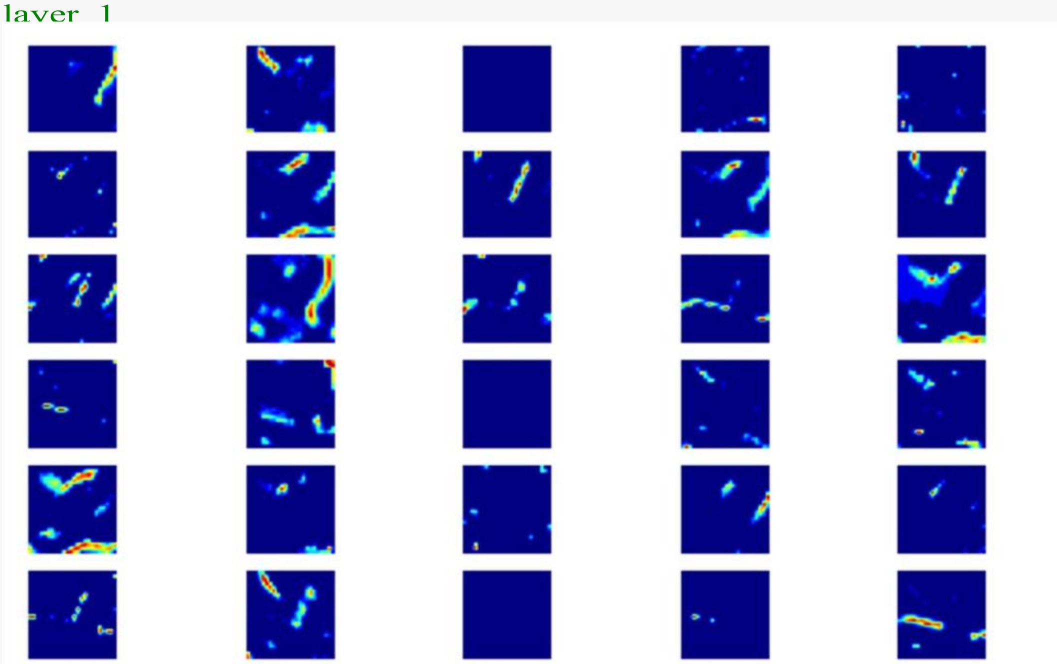



```
image = image/255
image = image.reshape(1, 28, 28, 1)

prediction = np.argmax(model.predict(image), axis=-1)
print("predicted digit:", str(prediction))
```

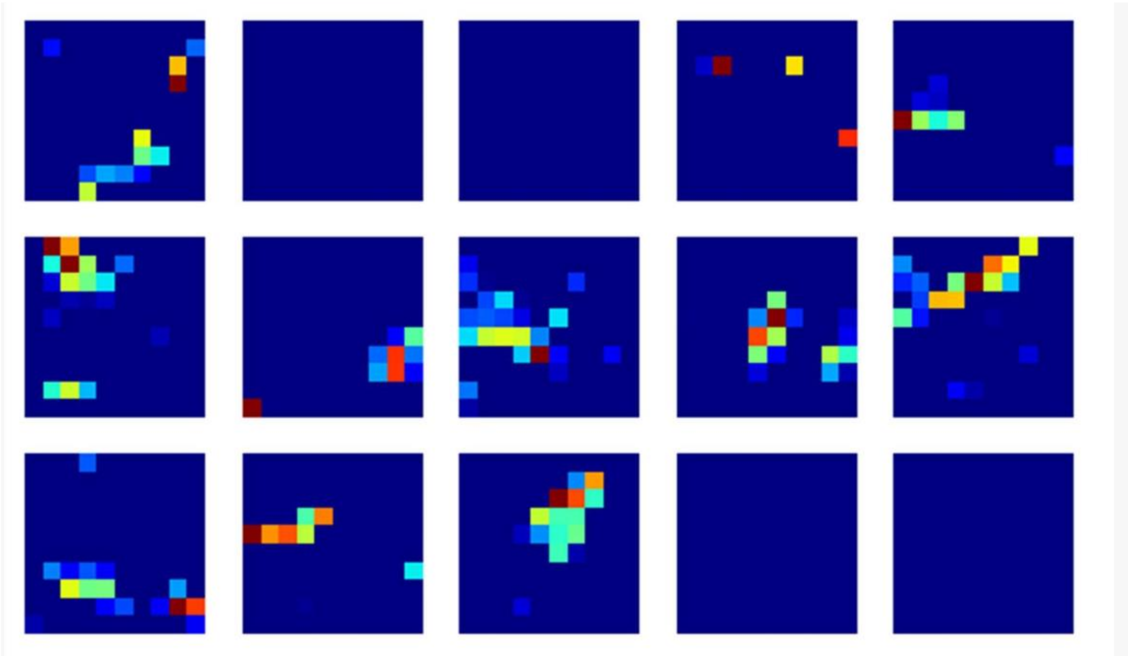
```
predicted digit: [2]
```

```
# Testing data

score = model.evaluate(X_test, y_test, verbose=0)
print(type(score))
print('Test score:', score[0])
print('Test accuracy:', score[1])

# To visualize what happens in convolutional layers using model class
API

layer1 = Model(inputs=model.layers[0].input,
outputs=model.layers[0].output)
layer2 = Model(inputs=model.layers[0].input,
outputs=model.layers[2].output)

# run a prediciton
visual_layer1, visual_layer2 = layer1.predict(image),
layer2.predict(image)
```

```
print(visual_layer1.shape) # indicates 30 outputs one for each filter of 24 by 24 dimention
print(visual_layer2.shape) # indicates 15 outputs one for each filter of 10 by 10 dimention

plt.figure(figsize=(10, 6))
# for 30 filters
for  i in range(30):
   plt.subplot(6, 5, i+1) # 6 rows 5 cols
   plt.imshow(visual_layer1[0, :, :, i], cmap = plt.get_cmap('jet'))
   plt.axis('off')
# we can see various features extracted by 30 filters in Convolutional layer 1
```



```
plt.figure(figsize=(10, 6))
# for 15 filters
for  i in range(15):
   plt.subplot(3, 5, i+1) # 3 rows 5 cols
   plt.imshow(visual_layer2[0, :, :, i], cmap = plt.get_cmap('jet'))
   plt.axis('off')
# we can see various features extracted by 15 filters in Convolutional layer 2
```



Conclusion:

The architecture of a Convolutional Neural Network (CNN) model for digit recognition typically consists of multiple convolutional layers followed by pooling layers, fully connected layers, and an output layer. The convolutional layers are designed to extract hierarchical features from the input images, and the pooling layers reduce spatial dimensions to increase computational efficiency. The fully connected layers help to make high-level predictions, and the output layer provides digit classifications. The network is trained on a dataset of labeled digit images to learn the discriminative features. The results of a well-designed CNN for digit recognition are highly accurate digit classifications, making it a crucial technology for applications such as optical character recognition (OCR) and automated digit identification in various fields, from finance to healthcare.