DATE:

EX.NO: 8(a) **IMPLEMENTATION OF BASIC CONVOLUTION NETWORK FOR MNIST DATASETS FROM SCRATCH**

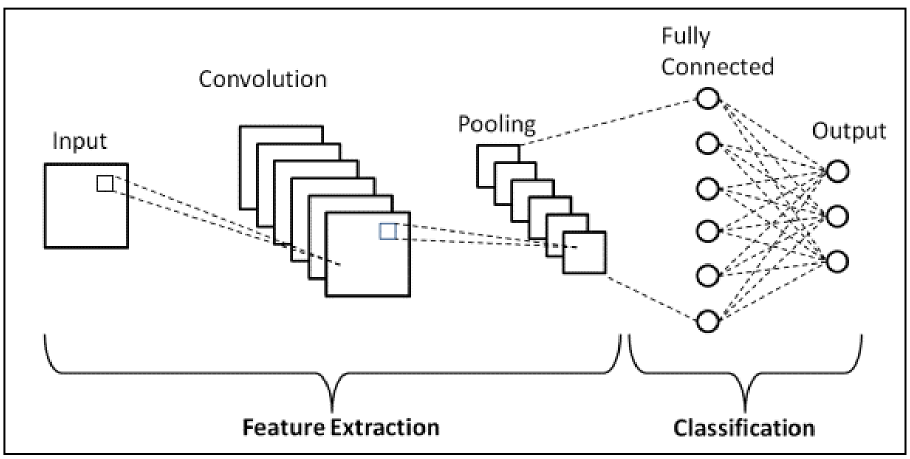
AIM:

To implement a basic Convolution Neural Network for MNIST datasets from scratch

DESCRIPTION:

- A **Convolutional Neural Network (CNN)** is a type of **artificial neural network** designed specifically for processing structured grid data, such as images and videos.

- CNNs have revolutionized computer vision tasks and have been widely used in various applications, including image classification, object detection, facial recognition, and more.



PROGRAM:

```
import tensorflow as tf
from tensorflow.keras import layers,models
from tensorflow.keras.datasets import mnist
#Load and preprocess the MNIST datasets
(train_images,train_labels),(test_images,test_labels)=mnist.load_data()
```

OUTPUT:

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

```
#Normalize pixel values to be between 0 and 1
train_images,test_images=train_images/255.0,test_images/255.0
#Create CNN model
model=models.Sequential([
        layers.Conv2D(32,(3,3),activation='relu',input_shape=(28,28,1)),
```

```
                    layers.MaxPooling2D((2,2)), layers.Conv2D(64,(3,3),activation='relu'),
                    layers.MaxPooling2D((2,2)), layers.Flatten(),
                    layers.Dense(64,activation='relu'), layers.Dense(10,activation='softmax'),
                                  ])
#Compile the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
#Print the model summary
model.summary()
#Train the model
model.fit(train_images.reshape(1,28,28,1),train_labels,epochs=5,validation_data=(test_images.reshape(1,28,28,1),test_labels))
```

OUTPUT:

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, 64) 18496 max_pooling2d_1 (MaxPoolin (None, 5, 5, 64) 0 g2D) flatten (Flatten) (None, 1600) 0 dense (Dense) (None, 64) 102464 dense_1 (Dense) (None, 10) 650 === Total params: 121930 (476.29 KB) Trainable params: 121930 (476.29 KB) Non-trainable params: 0 (0.00 Byte) ____Epoch 1/5 1875/1875 [==============================] - 54s 28ms/step - loss: 0.0150 - accuracy: 0.9950 - val_loss: 0.0288 - val_accuracy: 0.9908 Epoch 2/5 1875/1875 [==============================] - 53s 28ms/step - loss: 0.0104 - accuracy: 0.9968 - val_loss: 0.0313 - val_accuracy: 0.9915 Epoch 3/5 1875/1875 [==============================] - 52s 28ms/step - loss: 0.0096 - accuracy: 0.9969 - val_loss: 0.0349 - val_accuracy: 0.9908 Epoch 4/5 1875/1875 [==============================] - 53s 28ms/step - loss: 0.0067 - accuracy: 0.9979 - val_loss: 0.0367 - val_accuracy: 0.9910 Epoch 5/5 1875/1875 [==============================] - 53s 28ms/step - loss: 0.0075 - accuracy: 0.9974 - val_loss: 0.0314 - val_accuracy: 0.9925 313/313 [==============================] - 3s 8ms/step - loss: 0.0314 - accuracy: 0.9925 Test accuracy:(test_accuracy*100:.2f)%

```
#Evaluate the model on the test dataset
test_loss,test_accuracy=model.evaluate(test_images.reshape(-1,28,28,1),test_labels)
print(f"Test accuracy:{test_accuracy*100:.2f}%")
```

OUTPUT:

313/313 [==============================] - 3s 8ms/step - loss: 0.0314 - accuracy: 0.9925 Test

accuracy:99.25%

RESULT:

Thus the CNN architecture has been implemented successfully.

DATE:

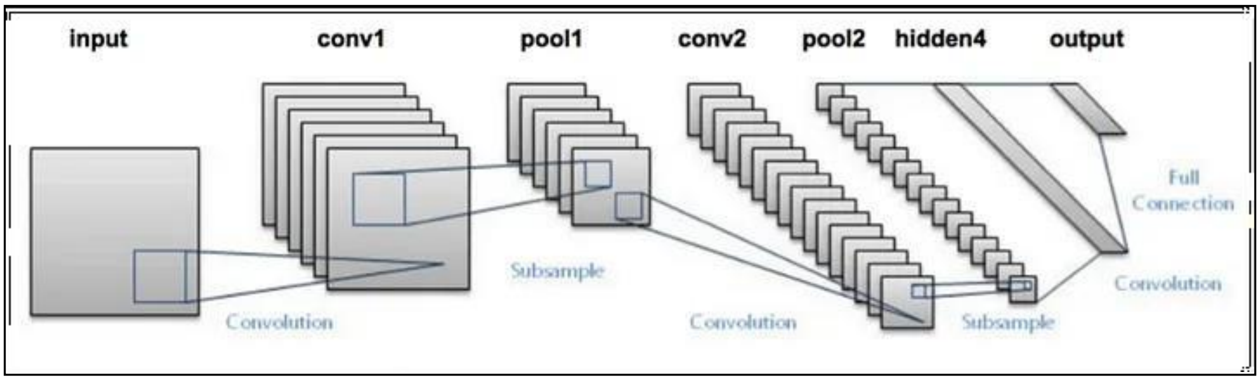
EX.NO: 8(b) **IMPLEMENTATION OF LENET ARCHITECTURE FROM SCRATCH**

AIM:

To use the LeNet architecture to perform handwritten digit detection using Tensorflow.

DESCRIPTION:

- LeNet-5, a pioneering 7-level convolutional network by LeCun et al in 1998, that classifies digits, was applied by several banks to recognise hand-written numbers on checks (cheques) digitized in 32x32 pixel grayscale input images.
- The ability to process higher resolution images requires larger and more convolutional layers, so this technique is constrained by the availability of computing resources



PROGRAM:

```
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras import datasets,layers,models,losses

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

OUTPUT:

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

```
x_train=tf.pad(x_train,[[0,0],[2,2],[2,2]])/255
x_test=tf.pad(x_test,[[0,0],[2,2],[2,2]])/255
x_train=tf.expand_dims(x_train,axis=3,name=None)
x_test=tf.expand_dims(x_test,axis=3,name=None)
x_val=x_train[-2000:,:,:]
y_val=y_train[-2000:]
```

```
x_train=x_train[:-2000,:,:,:]
y_train=y_train[:-2000]

model=models.Sequential([
        layers.Conv2D(6,5,activation='tanh',input_shape=x_train.shape[1:]),
        layers.AveragePooling2D(2),
        layers.Activation('sigmoid'),
        layers.Conv2D(16,5,activation='tanh'),
        layers.AveragePooling2D(2),
        layers.Activation('sigmoid'),
        layers.Conv2D(120,5,activation='tanh'),
        layers.Flatten(),
        layers.Dense(84,activation='tanh'),
        layers.Dense(10,activation='softmax')
    ])
model.summary()
```

OUTPUT:

Model: "sequential"____Layer (type) Output Shape Param # === conv2d (Conv2D) (None, 28, 28, 6) 156 average_pooling2d (Average (None, 14, 14, 6) 0 Pooling2D) activation (Activation) (None, 14, 14, 6) 0 conv2d_1 (Conv2D) (None, 10, 10, 16) 2416 average_pooling2d_1 (Avera (None, 5, 5, 16) 0 gePooling2D) activation_1 (Activation) (None, 5, 5, 16) 0 conv2d_2 (Conv2D) (None, 1, 1, 120) 48120 flatten (Flatten) (None, 120) 0 dense (Dense) (None, 84) 10164 dense_1 (Dense) (None, 10) 850 === Total params: 61706 (241.04 KB) Trainable params: 61706 (241.04 KB) Non-trainable params: 0 (0.00 Byte)

____

```
from keras.src.engine.training import optimizer
model.compile(optimizer='adam',loss=losses.sparse_categorical_crossentropy,metrics=['accuracy'])
history=model.fit(x_train,y_train,batch_size=64,epochs=30,validation_data=(x_val,y_val))
```

OUTPUT:

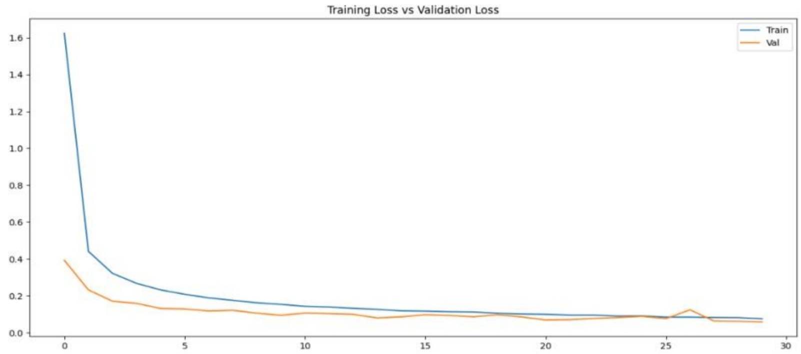
Epoch 27/30 907/907 [==============================] - 31s 35ms/step - loss: 0.0845 - accuracy: 0.9727 - val_loss: 0.1241 - val_accuracy: 0.9685 Epoch 28/30 907/907 [==============================] - 30s 33ms/step - loss: 0.0823 - accuracy: 0.9739 - val_loss: 0.0636 - val_accuracy: 0.9875 Epoch 29/30 907/907 [==============================] - 30s 33ms/step - loss: 0.0815 - accuracy: 0.9740 - val_loss: 0.0621 - val_accuracy: 0.9870 Epoch 30/30 907/907 [==============================] - 35s 39ms/step - loss: 0.0755 - accuracy: 0.9767 - val_loss: 0.0582 - val_accuracy: 0.9855

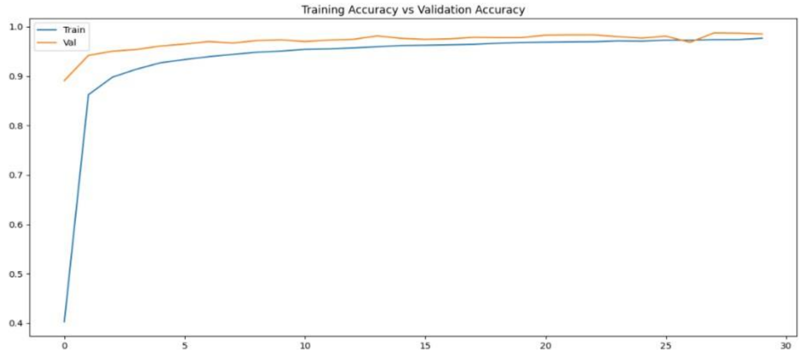
```
fig, axs = plt.subplots(2, 1, figsize=(15,15))

axs[0].plot(history.history['loss'])
axs[0].plot(history.history['val_loss'])
axs[0].title.set_text('Training Loss vs Validation Loss')
axs[0].legend(['Train', 'Val'])
axs[1].plot(history.history['accuracy'])
axs[1].plot(history.history['val_accuracy'])
axs[1].title.set_text('Training Accuracy vs Validation Accuracy')
axs[1].legend(['Train', 'Val'])
```

```
results = model.evaluate(x_test, y_test)
print("Loss = {}, Accuray = {}".format(results[0], results[1]))
```

OUTPUT:





RESULT:

Thus the LeNet architecture has been implemented successfully