# Assignment No 09

**Name-** Thorve Avishkar Shrikrushna

**Roll No-** 63

**Title-** Python program to design a Hopfield Network which stores 4 vectors

**Program:** import numpy as np class HopfieldNetwork:

def init (self, n\_neurons): self.n\_neurons = n\_neurons

self.weights = np.zeros((n\_neurons, n\_neurons))

def train(self, patterns): for pattern in patterns:

self.weights += np.outer(pattern, pattern)

np.fill\_diagonal(self.weights, 0)

def predict(self, pattern): energy = -0.5 \* np.dot(np.dot(pattern, self.weights), pattern) return np.sign(np.dot(pattern, self.weights) + energy)

if name == ' main ':

patterns = np.array([

[1, 1, -1, -1], [-1, -1, 1, 1], [1, -1, 1, -1],

[-1, 1, -1, 1]

])

n\_neurons = patterns.shape[1] network = HopfieldNetwork(n\_neurons)

network.train(patterns)

for pattern in patterns: prediction = network.predict(pattern) print('Input pattern:', pattern) print('Predicted pattern:', prediction)

**Output:**

Input pattern: [ 1 1 -1 -1]

Predicted pattern: [-1. -1. -1. -1.]

Input pattern: [-1 -1 1 1]

Predicted pattern: [-1. -1. -1. -1.]

Input pattern: [ 1 -1 1 -1]

Predicted pattern: [-1. -1. -1. -1.]

Input pattern: [-1 1 -1 1]

Predicted pattern: [-1. -1. -1. -1.]

# Assignment No 10

**Name-** Thorve Avishkar Shrikrushna

**Roll No-** 63

**Title-** Python program to implement CNN object detection. Discuss numerous performance evaluations

**Program:**

import keras from keras.datasets import cifar10 from keras.models import Sequential from keras.layers import Dense, Dropout, Flatten from keras.layers import Conv2D, MaxPooling2D from keras.optimizers import SGD from keras.preprocessing.image import ImageDataGenerator

# Load CIFAR-10 dataset

(X\_train, y\_train), (X\_test, y\_test) = cifar10.load\_data()

# Define the model model = Sequential()

model.add(Conv2D(32, (3, 3), activation='relu', input\_shape=(32, 32, 3))) model.add(Conv2D(32, (3, 3), activation='relu')) model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Dropout(0.25)) model.add(Conv2D(64,

(3, 3), activation='relu')) model.add(Conv2D(64, (3, 3), activation='relu')) model.add(MaxPooling2D(pool\_size=(2, 2))) model.add(Dropout(0.25)) model.add(Flatten()) model.add(Dense(512, activation='relu')) model.add(Dropout(0.5)) model.add(Dense(10,

activation='softmax'))

# Define data generators

train\_datagen = ImageDataGenerator(rescale=1./255, shear\_range=0.2, zoom\_range=0.2, horizontal\_flip=True)

test\_datagen = ImageDataGenerator(rescale=1./255)

# Prepare the data train\_set = train\_datagen.flow(X\_train, y\_train, batch\_size=32) test\_set = test\_datagen.flow(X\_test, y\_test, batch\_size=32)

# Compile the model sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)

model.compile(loss='categorical\_crossentropy', optimizer=sgd, metrics=['accuracy'])

# Train the model

model.fit\_generator(train\_set, steps\_per\_epoch=len(X\_train)//32, epochs=100, validation\_data=test\_set, validation\_steps=len(X\_test)//32)

# Evaluate the model

score = model.evaluate(test\_set, verbose=0)

print('Test loss:', score[0])

print('Test accuracy:', score[1])

**Output:**

Downloading data from https:/[/www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz](http://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz)

170498071/170498071 [==============================] - 3s 0us/step

Epoch 1/100

/usr/local/lib/python3.10/dist-packages/keras/optimizers/legacy/gradient\_descent.py:114: UserWarning: The `lr` argument is deprecated, use `learning\_rate` instead. super(). init (name, \*\*kwargs)

<ipython-input-15-75bb0636727e>:40: UserWarning: `Model.fit\_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports generators. model.fit\_generator(train\_set, steps\_per\_epoch=len(X\_train)//32, epochs=100, validation\_data=test\_set, validation\_steps=len(X\_test)//32)

1562/1562 [==============================] - 270s 172ms/step - loss: nan - accuracy: 0.9977 - val\_loss: nan - val\_accuracy: 1.0000

Epoch 2/100

1562/1562 [==============================] - 264s 639ms/step - loss: nan - accuracy: 1.0000 - val\_loss: nan - val\_accuracy: 1.0000

Epoch 3/100

1562/1562 [==============================] - 255s 633ms/step - loss: nan - accuracy: 1.0000 - val\_loss: nan - val\_accuracy: 1.0000

Epoch 4/100

1562/1562 [==============================] - 242s 155ms/step - loss: nan - accuracy: 1.0000 - val\_loss: nan - val\_accuracy: 1.0000

Epoch 5/100

1562/1562 [==============================] - 247s 158ms/step - loss: nan - accuracy: 1.0000 - val\_loss: nan - val\_accuracy: 1.0000

Epoch 6/100

1562/1562 [==============================] - 244s 156ms/step - loss: nan - accuracy: 1.0000 - val\_loss: nan - val\_accuracy: 1.0000

Epoch 7/100

1562/1562 [==============================] - 244s 156ms/step - loss: nan - accuracy: 1.0000 - val\_loss: nan - val\_accuracy: 1.0000

Epoch 8/100

1562/1562 [==============================] - 245s 157ms/step - loss: nan - accuracy: 1.0000 - val\_loss: nan - val\_accuracy: 1.0000

Epoch 9/100

1562/1562 [==============================] - 240s 153ms/step - loss: nan - accuracy: 1.0000 - val\_loss: nan - val\_accuracy: 1.0000

Epoch 10/100

1562/1562 [==============================] - 251s 631ms/step - loss: nan - accuracy: 1.0000 - val\_loss: nan - val\_accuracy: 1.0000

Epoch 11/100

1562/1562 [==============================] - 249s 159ms/step - loss: nan - accuracy: 1.0000 - val\_loss: nan - val\_accuracy: 1.0000

Epoch 12/100

1562/1562 [==============================] - 248s 159ms/step - loss: nan - accuracy: 1.0000 - val\_loss: nan - val\_accuracy: 1.0000

Epoch 13/100

1562/1562 [==============================] - 243s 156ms/step - loss: nan - accuracy: 1.0000 - val\_loss: nan - val\_accuracy: 1.0000

Epoch 14/100

1562/1562 [==============================] - 244s 156ms/step - loss: nan - accuracy: 1.0000 - val\_loss: nan - val\_accuracy: 1.0000

Epoch 15/100

1562/1562 [==============================] - 242s 155ms/step - loss: nan - accuracy: 1.0000 - val\_loss: nan - val\_accuracy: 1.0000

Epoch 63/100

1562/1562 [==============================] - 241s 154ms/step - loss: nan - accuracy: 1.0000 - val\_loss: nan - val\_accuracy: 1.0000

# Assignment No 11

**Name-** Thorve Avishkar Shrikrushna **Roll No-** 63

**Title-** Program to train a Neural Network with Tensor Flow/Pytorch.

**Program:**

import tensorflow as tf import numpy as np from sklearn.model\_selection import train\_test\_split from sklearn.preprocessing import StandardScaler from sklearn.datasets import load\_breast\_cancer df=load\_breast\_cancer()

X\_train,X\_test,y\_train,y\_test=train\_test\_split(df.data,df.t

arget,test\_size=0.20,random\_state=42) sc=StandardScaler()

X\_train=sc.fit\_transform(X\_train) X\_test=sc.transform(X\_test) model=tf.keras.models.Sequential([tf.keras.layers.Dense(1,activation='sigmoid',input\_shape=(X \_train.shape[1],))]) model.compile(optimizer='adam',loss='binary\_crossentropy',metrics=['accuracy']) model.fit(X\_train,y\_train,epochs=5) y\_pred=model.predict(X\_test) test\_loss,test\_accuracy=model.evaluate(X\_test,y\_test) print("accuracy is",test\_accuracy)

**Output:**

Epoch 1/5

15/15 [==============================] - 1s 2ms/step - loss: 0.5449 - accuracy: 0.7385

Epoch 2/5

15/15 [==============================] - 0s 2ms/step - loss: 0.4896 - accuracy: 0.7802

Epoch 3/5

15/15 [==============================] - 0s 2ms/step - loss: 0.4439 - accuracy: 0.8286

Epoch 4/5

15/15 [==============================] - 0s 2ms/step - loss: 0.4074 - accuracy: 0.8462

Epoch 5/5

15/15 [==============================] - 0s 3ms/step - loss: 0.3776 - accuracy: 0.8593

4/4 [==============================] - 0s 5ms/step

4/4 [==============================] - 0s 4ms/step - loss: 0.3090 - accuracy: 0.9298 accuracy is 0.9298245906829834

# Assignment No 12

**Name-** Thorve Avishkar Shrikrushna **Roll No-** 63

**Title-** Python program for implementation of CNN using Tensor flow/Pytorch.

**Program:**

import tensorflow as tf from tensorflow.keras.datasets import mnist from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense from tensorflow.keras.utils import to\_categorical

(X\_train, y\_train), (X\_test, y\_test) = mnist.load\_data()

X\_train = X\_train.reshape(-1, 28, 28, 1) / 255.0 X\_test = X\_test.reshape(-1, 28, 28, 1) / 255.0 y\_train = to\_categorical(y\_train) y\_test = to\_categorical(y\_test) model = Sequential([

Conv2D(32, (3, 3), activation='relu', input\_shape=(28, 28, 1)),

MaxPooling2D((2, 2)),

Conv2D(64, (3, 3), activation='relu'),

MaxPooling2D((2, 2)),

Conv2D(64, (3, 3), activation='relu'),

Flatten(),

Dense(64, activation='relu'),

Dense(10, activation='softmax')

])

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy']) model.fit(X\_train, y\_train, batch\_size=64, epochs=10, verbose=1)

loss, accuracy = model.evaluate(X\_test, y\_test) print(f"Test Loss: {loss}") print(f"Test Accuracy: {accuracy}")

**Output:**

Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz

11490434/11490434 [==============================] - 0s 0us/step

Epoch 1/10

938/938 [==============================] - 59s 60ms/step - loss: 0.1783 - accuracy:

0.9448

Epoch 2/10

938/938 [==============================] - 56s 60ms/step - loss: 0.0541 - accuracy:

0.9835

Epoch 3/10

938/938 [==============================] - 55s 59ms/step - loss: 0.0378 - accuracy: 0.9878

Epoch 4/10

938/938 [==============================] - 58s 61ms/step - loss: 0.0295 - accuracy:

0.9908

Epoch 5/10

938/938 [==============================] - 55s 59ms/step - loss: 0.0234 - accuracy:

0.9926

Epoch 6/10

938/938 [==============================] - 55s 59ms/step - loss: 0.0202 - accuracy:

0.9936

Epoch 7/10

938/938 [==============================] - 55s 59ms/step - loss: 0.0153 - accuracy:

0.9950

Epoch 8/10

938/938 [==============================] - 55s 58ms/step - loss: 0.0139 - accuracy:

0.9957

Epoch 9/10

938/938 [==============================] - 56s 59ms/step - loss: 0.0117 - accuracy:

0.9961

Epoch 10/10

938/938 [==============================] - 54s 58ms/step - loss: 0.0091 - accuracy:

0.9971

313/313 [==============================] - 3s 9ms/step - loss: 0.0285 - accuracy:

0.9921

Test Loss: 0.028454650193452835

Test Accuracy: 0.9921000003814697

# Assignment No 13

**Name-** Thorve Avishkar Shrikrushna

**Roll No-** 63

**Title-** Implementation of MNIST Handwritten Character Detection using PyTorch, Keras and Tensorflow

**Program:**

import tensorflow as tf from tensorflow.keras.datasets import mnist from tensorflow.keras.models import Sequential from tensorflow.keras.layers import Dense, Flatten from tensorflow.keras.optimizers import Adam

# Load and preprocess the MNIST dataset

(X\_train, y\_train), (X\_test, y\_test) = mnist.load\_data()

X\_train = X\_train / 255.0 X\_test = X\_test / 255.0

# Define the model architecture model = Sequential([

Flatten(input\_shape=(28, 28)),

Dense(128, activation='relu'),

Dense(10, activation='softmax')

])

# Compile the model

model.compile(optimizer=Adam(learning\_rate=0.001),

loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

# Train the model model.fit(X\_train, y\_train, batch\_size=64, epochs=10, verbose=1)

# Evaluate the model

loss, accuracy = model.evaluate(X\_test, y\_test) print(f"Test Loss: {loss}") print(f"Test Accuracy: {accuracy}")

**Output:**

Epoch 1/10

938/938 [==============================] - 5s 4ms/step - loss: 0.2984 - accuracy:

0.9153

Epoch 2/10

938/938 [==============================] - 7s 7ms/step - loss: 0.1353 - accuracy:

0.9612

Epoch 3/10

938/938 [==============================] - 4s 4ms/step - loss: 0.0944 - accuracy:

0.9723

Epoch 4/10

938/938 [==============================] - 4s 5ms/step - loss: 0.0708 - accuracy:

0.9783

Epoch 5/10

938/938 [==============================] - 4s 4ms/step - loss: 0.0558 - accuracy:

0.9833

Epoch 6/10

938/938 [==============================] - 4s 4ms/step - loss: 0.0447 - accuracy:

0.9864

Epoch 7/10

938/938 [==============================] - 4s 4ms/step - loss: 0.0363 - accuracy:

0.9892

Epoch 8/10

938/938 [==============================] - 4s 5ms/step - loss: 0.0293 - accuracy:

0.9913

Epoch 9/10

938/938 [==============================] - 4s 4ms/step - loss: 0.0255 - accuracy:

0.9927

Epoch 10/10

938/938 [==============================] - 4s 4ms/step - loss: 0.0202 - accuracy:

0.9944

313/313 [==============================] - 1s 2ms/step - loss: 0.0679 - accuracy:

0.9804

Test Loss: 0.06786014884710312

Test Accuracy: 0.980400025844574