Assignment No-09

Name- Dumbre Chirayu Rohidas
Roll No- 16
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- Dumbre Chirayu Rohidas
Roll No- 16

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, MaxPooling2Dfrom
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
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-15-75bb0166727e>:40: UserWarning: `Model.fit_generator` is deprecated andwill 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 169ms/step - loss: nan - accuracy:1.0000 - val_loss: nan - val_accuracy: 1.0000
Epoch 3/100
1562/1562 [==============================] - 255s 163ms/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 161ms/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 16/100
1562/1562 [==============================] - 241s 154ms/step - loss: nan - accuracy:1.0000 - val_loss: nan - val_accuracy: 1.0000

Assignment No-11

Name- Dumbre Chirayu Rohidas
Roll No- 16

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_splitfrom
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- Dumbre Chirayu Rohidas
Roll No- 16

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, Densefrom
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- Dumbre Chirayu Rohidas
Roll No- 16

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
```