**Name-Mohit Shailesh Deo**

**Roll No-13**

**PRN No-12210623**

**Problem Statement:-**

Convolutional neural network (CNN) Use any dataset of plant disease and design a plant disease detection system using CNN. **Without Buildin Library**

Code:-

import numpy as np

import cv2

from sklearn.utils import shuffle

from sklearn.metrics import accuracy\_score

# ------------------ Utils ------------------

def relu(x):

return np.maximum(0, x)

def relu\_deriv(x):

return (x > 0).astype(float)

def softmax(x):

exp\_x = np.exp(x - np.max(x))

return exp\_x / exp\_x.sum(axis=1, keepdims=True)

def cross\_entropy(y\_pred, y\_true):

return -np.sum(y\_true \* np.log(y\_pred + 1e-8)) / y\_pred.shape[0]

# ------------------ Conv Layer ------------------

class Conv3x3:

def \_\_init\_\_(self, num\_filters):

self.num\_filters = num\_filters

self.filters = np.random.randn(num\_filters, 3, 3) / 9

def iterate\_regions(self, image):

h, w = image.shape

for i in range(h - 2):

for j in range(w - 2):

yield i, j, image[i:i+3, j:j+3]

def forward(self, input):

self.last\_input = input

h, w = input.shape

output = np.zeros((h - 2, w - 2, self.num\_filters))

for i, j, region in self.iterate\_regions(input):

output[i, j] = np.sum(region \* self.filters, axis=(1, 2))

return relu(output)

# ------------------ Max Pool Layer ------------------

class MaxPool2:

def iterate\_regions(self, image):

h, w, \_ = image.shape

for i in range(0, h, 2):

for j in range(0, w, 2):

yield i, j, image[i:i+2, j:j+2]

def forward(self, input):

self.last\_input = input

h, w, f = input.shape

output = np.zeros((h // 2, w // 2, f))

for i, j, region in self.iterate\_regions(input):

output[i//2, j//2] = np.amax(region, axis=(0, 1))

return output

# ------------------ Fully Connected ------------------

class SoftmaxLayer:

def \_\_init\_\_(self, input\_len, nodes):

self.weights = np.random.randn(input\_len, nodes) / input\_len

self.biases = np.zeros(nodes)

def forward(self, input):

self.last\_input\_shape = input.shape

input\_flat = input.flatten()

self.last\_input = input\_flat

total = np.dot(input\_flat, self.weights) + self.biases

self.last\_output = softmax(total.reshape(1, -1))

return self.last\_output

# ------------------ Model ------------------

class CNN:

def \_\_init\_\_(self):

self.conv = Conv3x3(8)

self.pool = MaxPool2()

self.softmax = SoftmaxLayer(13 \* 13 \* 8, 3) # Adjust 13\*13 if image shape changes

def forward(self, image):

out = self.conv.forward(image)

out = self.pool.forward(out)

out = self.softmax.forward(out)

return out

def predict(self, X):

return np.array([self.forward(x).argmax() for x in X])

# ------------------ Training Loop ------------------

def train(X, y, epochs=3):

model = CNN()

for epoch in range(epochs):

X, y = shuffle(X, y)

loss = 0

correct = 0

for i in range(len(X)):

output = model.forward(X[i])

loss += cross\_entropy(output, y[i])

if np.argmax(output) == np.argmax(y[i]):

correct += 1

print(f"Epoch {epoch+1}, Loss: {loss/len(X):.4f}, Accuracy: {correct/len(X):.4f}")

return model

# ------------------ Data Loader Example ------------------

def load\_dataset(img\_size=(28, 28)):

# Simulated 3-class, grayscale plant leaf images

X = []

y = []

for label in range(3):

for \_ in range(20): # 20 images per class

img = np.random.rand(\*img\_size).astype(np.float32)

X.append(img)

y\_vec = np.zeros(3)

y\_vec[label] = 1

y.append(y\_vec)

X = np.array(X)

y = np.array(y)

return X, y

# ------------------ Main ------------------

# Load dummy data (replace this with real image loading)

X\_train, y\_train = load\_dataset()

# Train the model

model = train(X\_train, y\_train, epochs=5)

# Predict on training set

preds = model.predict(X\_train)

true\_labels = np.argmax(y\_train, axis=1)

# Accuracy

acc = accuracy\_score(true\_labels, preds)

print("Final Training Accuracy:", acc)

# Show few predictions

for i in range(5):

print(f"Image {i+1} - Predicted: {preds[i]}, Actual: {true\_labels[i]}")