Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

Лабораторная работа №4

По дисциплине «МРЗИС»

Тема: “Сверточные сети”

Выполнил:

Студент 3 курса

Группы ИИ-21

Романко Н.А.

Проверил:

Туз И.С.

Брест 2024

Цель:реализовать и обучить сверточную нейронную сеть для классификации изображений.

Начальные данные: датасет MNIST

Код программы:

import numpy as np

import keras

from keras.utils import to\_categorical

from scipy.signal import correlate2d

from sklearn.metrics import accuracy\_score

class Convolution:

def \_\_init\_\_(self, input\_shape, filter\_size, num\_filters):

input\_height, input\_width = input\_shape

self.num\_filters = num\_filters

self.input\_shape = input\_shape

self.filter\_shape = (num\_filters, filter\_size, filter\_size)

self.output\_shape = (num\_filters, input\_height - filter\_size + 1, input\_width - filter\_size + 1)

self.filters = np.random.randn(\*self.filter\_shape)

self.biases = np.random.randn(\*self.output\_shape)

def forward(self, input\_data):

self.input\_data = input\_data

output = np.zeros(self.output\_shape)

for i in range(self.num\_filters):

output[i] = correlate2d(self.input\_data, self.filters[i], mode="valid")

output = np.maximum(output, 0)

return output

def backward(self, dL\_dout, lr):

dL\_dinput = np.zeros\_like(self.input\_data)

dL\_dfilters = np.zeros\_like(self.filters)

for i in range(self.num\_filters):

dL\_dfilters[i] = correlate2d(self.input\_data, dL\_dout[i],mode="valid")

dL\_dinput += correlate2d(dL\_dout[i],self.filters[i], mode="full")

self.filters -= lr \* dL\_dfilters

self.biases -= lr \* dL\_dout

return dL\_dinput

class MaxPool:

def \_\_init\_\_(self, pool\_size):

self.pool\_size = pool\_size

def forward(self, input\_data):

self.input\_data = input\_data

self.num\_channels, self.input\_height, self.input\_width = input\_data.shape

self.output\_height = self.input\_height // self.pool\_size

self.output\_width = self.input\_width // self.pool\_size

self.output = np.zeros((self.num\_channels, self.output\_height, self.output\_width))

for c in range(self.num\_channels):

for i in range(self.output\_height):

for j in range(self.output\_width):

start\_i = i \* self.pool\_size

start\_j = j \* self.pool\_size

end\_i = start\_i + self.pool\_size

end\_j = start\_j + self.pool\_size

patch = input\_data[c, start\_i:end\_i, start\_j:end\_j]

self.output[c, i, j] = np.max(patch)

return self.output

def backward(self, dL\_dout, lr):

dL\_dinput = np.zeros\_like(self.input\_data)

for c in range(self.num\_channels):

for i in range(self.output\_height):

for j in range(self.output\_width):

start\_i = i \* self.pool\_size

start\_j = j \* self.pool\_size

end\_i = start\_i + self.pool\_size

end\_j = start\_j + self.pool\_size

patch = self.input\_data[c, start\_i:end\_i, start\_j:end\_j]

mask = patch == np.max(patch)

dL\_dinput[c,start\_i:end\_i, start\_j:end\_j] = dL\_dout[c, i, j] \* mask

return dL\_dinput

class Fully\_Connected:

def \_\_init\_\_(self, input\_size, output\_size):

self.input\_size = input\_size

self.output\_size = output\_size

self.weights = np.random.randn(output\_size, self.input\_size)

self.biases = np.random.rand(output\_size, 1)

def softmax(self, z):

shifted\_z = z - np.max(z)

exp\_values = np.exp(shifted\_z)

sum\_exp\_values = np.sum(exp\_values, axis=0)

log\_sum\_exp = np.log(sum\_exp\_values)

probabilities = exp\_values / sum\_exp\_values

return probabilities

def softmax\_derivative(self, s):

return np.diagflat(s) - np.dot(s, s.T)

def forward(self, input\_data):

self.input\_data = input\_data

flattened\_input = input\_data.flatten().reshape(1, -1)

self.z = np.dot(self.weights, flattened\_input.T) + self.biases

self.output = self.softmax(self.z)

return self.output

def backward(self, dL\_dout, lr):

dL\_dy = np.dot(self.softmax\_derivative(self.output), dL\_dout)

dL\_dw = np.dot(dL\_dy, self.input\_data.flatten().reshape(1, -1))

dL\_db = dL\_dy

dL\_dinput = np.dot(self.weights.T, dL\_dy)

dL\_dinput = dL\_dinput.reshape(self.input\_data.shape)

self.weights -= lr \* dL\_dw

self.biases -= lr \* dL\_db

return dL\_dinput

def cross\_entropy\_loss(predictions, targets):

num\_samples = 10

epsilon = 1e-7

predictions = np.clip(predictions, epsilon, 1 - epsilon)

loss = -np.sum(targets \* np.log(predictions)) / num\_samples

return loss

def cross\_entropy\_loss\_gradient(actual\_labels, predicted\_probs):

num\_samples = actual\_labels.shape[0]

gradient = -actual\_labels / (predicted\_probs + 1e-7) / num\_samples

return gradient

def train\_network(X, y, conv, pool, full, epochs, lr=0.01):

for epoch in range(epochs):

total\_loss = 0.0

correct\_predictions = 0

for i in range(len(X)):

conv\_out = conv.forward(X[i])

pool\_out = pool.forward(conv\_out)

full\_out = full.forward(pool\_out)

loss = cross\_entropy\_loss(full\_out.flatten(), y[i])

total\_loss += loss

one\_hot\_pred = np.zeros\_like(full\_out)

one\_hot\_pred[np.argmax(full\_out)] = 1

one\_hot\_pred = one\_hot\_pred.flatten()

num\_pred = np.argmax(one\_hot\_pred)

num\_y = np.argmax(y[i])

if num\_pred == num\_y:

correct\_predictions += 1

gradient = cross\_entropy\_loss\_gradient(y[i], full\_out.flatten()).reshape((-1, 1))

full\_back = full.backward(gradient, lr)

pool\_back = pool.backward(full\_back, lr)

conv\_back = conv.backward(pool\_back, lr)

average\_loss = total\_loss / len(X)

accuracy = correct\_predictions / len(X\_train) \* 100.0

print(f"Epoch {epoch + 1}/{epochs} - Loss: {average\_loss:.4f} - Accuracy: {accuracy:.2f}%")

def predict(input\_sample, conv, pool, full):

conv\_out = conv.forward(input\_sample)

pool\_out = pool.forward(conv\_out)

flattened\_output = pool\_out.flatten()

predictions = full.forward(flattened\_output)

return predictions

(train\_images, train\_labels), (test\_images, test\_labels) = keras.datasets.mnist.load\_data()

X\_train = train\_images[:5000] / 255.0

y\_train = train\_labels[:5000]

X\_test = train\_images[5000:10000] / 255.0

y\_test = train\_labels[5000:10000]

y\_train = to\_categorical(y\_train)

y\_test = to\_categorical(y\_test)

conv = Convolution(X\_train[0].shape, 6, 1)

pool = MaxPool(2)

full = Fully\_Connected(121, 10)

train\_network(X\_train, y\_train, conv, pool, full, epochs=10)

predictions = []

for data in X\_test:

pred = predict(data, conv, pool, full)

one\_hot\_pred = np.zeros\_like(pred)

one\_hot\_pred[np.argmax(pred)] = 1

predictions.append(one\_hot\_pred.flatten())

predictions = np.array(predictions)

print(accuracy\_score(predictions, y\_test))

total\_predictions = min(100, len(y\_test))

for i in range(total\_predictions):

predicted\_label = np.argmax(predictions[i])

true\_label = np.argmax(y\_test[i])

print(f"{'Верно!' if predicted\_label == true\_label else 'Неверно!'} - Предсказание: {predicted\_label}, Истинное значение: {true\_label}")

Вывод программы:

Изображение выглядит как текст, снимок экрана, шаблон, ткань

Автоматически созданное описание

Вывод: реализовал и обучил сверточную нейронную сеть для классификации изображений.