ДВНЗ «ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ»

Факультет комп’ютерно-інформаційних технологій та автоматизації

ЗВІТ до практичної РОБОТИ №8

з дисципліни «Методи та системи штучного інтелекту»

за темою: «СТВОРЕННЯ ПРОСТОЇ НЕЙРОННОЇ МЕРЕЖІ ДЛЯ КЛАСИФІКАЦІЇ ЗОБРАЖЕНЬ»

Варіант №7

Виконав:

Студент гр. КІ-21

Іван ДАЦЕНКО

Перевірив:

Андрій НІКІТЕНКО

Луцьк – 2024

Мета роботи: Ознайомитися з основами нейронних мереж та їх застосуванням для класифікації зображень. Навчитися створювати просту нейронну мережу з використанням модуля Numpy. Зрозуміти принципи роботи нейронних мереж та їх ключові компоненти.

Завдання

Використовуючи засоби Numpy виконати індивідуальне завдання відповідно до таблиці варіантів (таб. 8.1). Для виконання завдання дозволено використовувати допоміжні засоби, але на захисті роботи потрібно детально пояснити як працює реалізована нейронна мережа. Якщо при завантаженні набору даних він одразу не поділений на тренувальний та тестові набори, то поділити з відношенням 80% на 20%. Точність нейронної мережі повинна бути не менше 75% на тестовому наборі даних. Протестуйте нейронну мережу з різними гіперпараметрами, та наведіть графіки, в яких показано як змінюється точність класифікації від заданих параметрів.

Додаткове завдання №1 (опціонально): імплементуйте нейронну мережу за допомогою бібліотек tensorflow або pytorch. Порівняйте реалізацію нейронних мереж між собою.

Додаткове завдання №2 (опціонально): реалізуйте «confusion matrix» для розробленої нейронної мережі. Поясніть її результати.

Зображення, що містить знімок екрана, текст, ряд, Шрифт

Автоматично згенерований опис

**Лістинг програми:**

**8.1.py**

import numpy as np  
import tensorflow as tf  
import matplotlib.pyplot as plt  
  
class NeuralNetwork:  
 def \_\_init\_\_(self, input\_size, hidden\_size1, hidden\_size2, output\_size):  
 self.W1 = np.random.randn(input\_size, hidden\_size1) / np.sqrt(input\_size)  
 self.b1 = np.zeros((1, hidden\_size1))  
 self.W2 = np.random.randn(hidden\_size1, hidden\_size2) / np.sqrt(hidden\_size1)  
 self.b2 = np.zeros((1, hidden\_size2))  
 self.W3 = np.random.randn(hidden\_size2, output\_size) / np.sqrt(hidden\_size2)  
 self.b3 = np.zeros((1, output\_size))  
 self.dropout\_rate = 0.5 # Dropout rate for regularization  
  
 def forward\_propagation(self, X, training=True):  
 X = X.reshape(X.shape[0], -1) # Flatten the input  
 self.z1 = np.dot(X, self.W1) + self.b1  
 self.a1 = np.maximum(0, self.z1) # ReLU activation  
 if training:  
 self.a1 \*= np.random.binomial(1, 1 - self.dropout\_rate, size=self.a1.shape) / (1 - self.dropout\_rate)  
 self.z2 = np.dot(self.a1, self.W2) + self.b2  
 self.a2 = np.tanh(self.z2) # Tanh activation  
 if training:  
 self.a2 \*= np.random.binomial(1, 1 - self.dropout\_rate, size=self.a2.shape) / (1 - self.dropout\_rate)  
 self.z3 = np.dot(self.a2, self.W3) + self.b3  
 self.a3 = np.exp(self.z3) / np.sum(np.exp(self.z3), axis=1, keepdims=True) # Softmax activation  
 return self.a3  
  
 def backward\_propagation(self, X, y):  
 m = X.shape[0] # Number of samples  
 X = X.reshape(X.shape[0], -1) # Flatten the input  
 delta3 = self.a3 - y # Output layer error  
 self.dW3 = np.dot(self.a2.T, delta3) / m  
 self.db3 = np.sum(delta3, axis=0, keepdims=True) / m  
 delta2 = np.dot(delta3, self.W3.T) \* (1 - np.power(self.a2, 2)) # Tanh derivative  
 self.dW2 = np.dot(self.a1.T, delta2) / m  
 self.db2 = np.sum(delta2, axis=0, keepdims=True) / m  
 delta1 = np.dot(delta2, self.W2.T) \* (self.a1 > 0) # ReLU derivative  
 self.dW1 = np.dot(X.T, delta1) / m  
 self.db1 = np.sum(delta1, axis=0, keepdims=True) / m  
  
 def update\_param(self, learning\_rate):  
 self.W1 -= learning\_rate \* self.dW1  
 self.b1 -= learning\_rate \* self.db1  
 self.W2 -= learning\_rate \* self.dW2  
 self.b2 -= learning\_rate \* self.db2  
 self.W3 -= learning\_rate \* self.dW3  
 self.b3 -= learning\_rate \* self.db3  
  
 def evaluate(self, X, y):  
 predictions = self.forward\_propagation(X, training=False)  
 loss = np.sum(-y \* np.log(predictions)) / len(y)  
 accuracy = np.mean(np.argmax(predictions, axis=1) == np.argmax(y, axis=1))  
 return loss, accuracy  
  
 def train(self, X, y, epochs, learning\_rate, batch\_size=64):  
 self.history = {'accuracy': [], 'loss': []}  
 for epoch in range(epochs):  
 for i in range(0, X.shape[0], batch\_size):  
 batch\_X = X[i:i + batch\_size]  
 batch\_y = y[i:i + batch\_size]  
 self.forward\_propagation(batch\_X)  
 self.backward\_propagation(batch\_X, batch\_y)  
 self.update\_param(learning\_rate)  
 loss, accuracy = self.evaluate(X, y)  
 self.history['accuracy'].append(accuracy)  
 self.history['loss'].append(loss)  
 print(f"Epoch {epoch + 1}/{epochs}, Loss: {loss:.3f}, Accuracy: {accuracy:.3f}")  
  
# Завантаження та підготовка даних  
data\_dir = tf.keras.utils.get\_file(origin='https://storage.googleapis.com/download.tensorflow.org/example\_images/flower\_photos.tgz',  
 fname='flower\_photos', untar=True)  
  
datagen = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1.0/255.0, validation\_split=0.2)  
  
train\_generator = datagen.flow\_from\_directory(data\_dir,  
 target\_size=(64, 64),  
 batch\_size=32,  
 class\_mode='categorical',  
 subset='training')  
  
validation\_generator = datagen.flow\_from\_directory(data\_dir,  
 target\_size=(64, 64),  
 batch\_size=32,  
 class\_mode='categorical',  
 subset='validation')  
  
# Підготовка даних для моделі  
def prepare\_data(generator):  
 x\_data, y\_data = [], []  
 for i in range(len(generator)):  
 x, y = generator[i]  
 x\_data.extend(x)  
 y\_data.extend(y)  
 if i == len(generator) - 1:  
 break  
 x\_data = np.array(x\_data).reshape(len(x\_data), -1)  
 y\_data = np.array(y\_data)  
 return x\_data, y\_data  
  
x\_train, y\_train = prepare\_data(train\_generator)  
x\_val, y\_val = prepare\_data(validation\_generator)  
  
# Створення та тренування моделі  
input\_size = 64 \* 64 \* 3  
hidden\_size1 = 1024  
hidden\_size2 = 512  
output\_size = len(train\_generator.class\_indices)  
  
model = NeuralNetwork(input\_size, hidden\_size1, hidden\_size2, output\_size)  
epochs = 60  
learning\_rate = 0.0001  
model.train(x\_train, y\_train, epochs=epochs, learning\_rate=learning\_rate)  
  
# Оцінка моделі на валідаційних даних  
val\_loss, val\_accuracy = model.evaluate(x\_val, y\_val)  
print("Validation Loss:", val\_loss)  
print("Validation Accuracy:", val\_accuracy \* 100)  
  
# Побудова матриці невідповідностей  
predictions = model.forward\_propagation(x\_val, training=False)  
y\_pred = np.argmax(predictions, axis=1)  
y\_true = np.argmax(y\_val, axis=1)  
confusion\_matrix = np.zeros((output\_size, output\_size))  
  
for i in range(len(y\_pred)):  
 confusion\_matrix[y\_true[i], y\_pred[i]] += 1  
  
confusion\_matrix = confusion\_matrix.astype('float32') / confusion\_matrix.sum(axis=1, keepdims=True)  
  
# Відображення матриці невідповідностей  
plt.figure(figsize=(10, 10))  
plt.imshow(confusion\_matrix, cmap='Blues')  
plt.xticks(np.arange(output\_size), train\_generator.class\_indices.keys(), rotation=90)  
plt.yticks(np.arange(output\_size), train\_generator.class\_indices.keys())  
plt.xlabel('Predicted')  
plt.ylabel('True')  
plt.title("Confusion Matrix")  
plt.colorbar()  
plt.show()  
  
# Графік точності та втрат на тренувальних даних  
fig, ax1 = plt.subplots(figsize=(12, 6))  
ax1.set\_xlabel('Epochs')  
ax1.set\_ylabel('Accuracy', color='blue')  
ax1.plot(range(epochs), model.history['accuracy'], color='blue', label='Training Accuracy')  
ax1.tick\_params(axis='y', labelcolor='blue')  
  
ax2 = ax1.twinx()  
ax2.set\_ylabel('Loss', color='red')  
ax2.plot(range(epochs), model.history['loss'], color='red', label='Training Loss')  
ax2.tick\_params(axis='y', labelcolor='red')  
  
fig.tight\_layout()  
plt.title('Training Accuracy and Loss')  
fig.legend(loc='upper right')  
plt.show()

**Лістинг програми для «confusion matrix»:**

predictions = model.forward\_propagation(x\_val, training=False)

y\_pred = np.argmax(predictions, axis=1)

y\_true = np.argmax(y\_val, axis=1)

confusion\_matrix = np.zeros((output\_size, output\_size))

for i in range(len(y\_pred)):

confusion\_matrix[y\_true[i], y\_pred[i]] += 1

confusion\_matrix = confusion\_matrix.astype('float32') / confusion\_matrix.sum(axis=1, keepdims=True)

Зображення, що містить програмне забезпечення, Мультимедійне програмне забезпечення, текст, Графічний редактор

Автоматично згенерований опис

Рисунок 1 - Confusion Matrix

**Результати:**

**Перший тест проведемо із такими параметрами**

Розмірність прихованих шарів =1024, 512, 256

Кількість епох = 50

Швидкість навчання =0.001

Тут показано результат виконання першого завдання. Нейронна мережа проходить навчання за 50 епох при швидкості навчання 0.001.

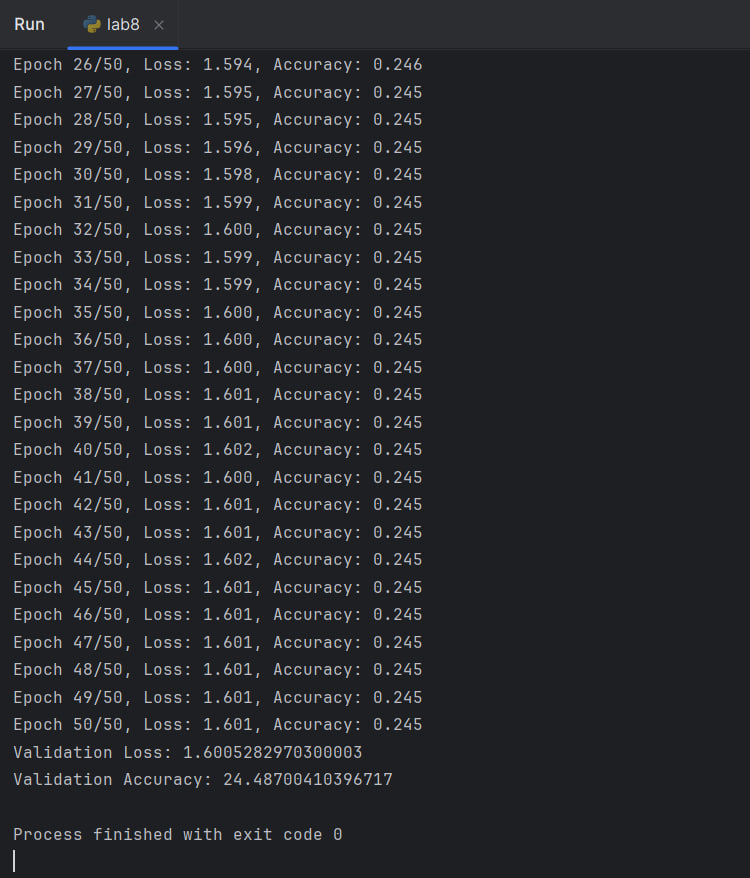


Рисунок 2 − Точність першого тесту

Зображення, що містить текст, знімок екрана, схема, Графік

Автоматично згенерований опис

Рисунок 3 − Графік точності першого тесту

**Другий тест проведемо із такими параметрами**

Розмірність прихованих шарів =2048, 1024, 512

Кількість епох = 30

Швидкість навчання =0.001

Тут показано результат виконання першого завдання. Нейронна мережа проходить навчання за 30 епох при швидкості навчання 0.001.

Зображення, що містить текст, знімок екрана, програмне забезпечення, Мультимедійне програмне забезпечення

Автоматично згенерований опис

Рисунок 4 - Точність другого тесту

Зображення, що містить текст, знімок екрана, програмне забезпечення, ряд

Автоматично згенерований опис

Рисунок 5 - Графік точності другого тесту

**Третій тест проведемо із такими параметрами**

Розмірність прихованих шарів =512, 256,128

Кількість епох = 60

Швидкість навчання =0.01

Зображення, що містить текст, знімок екрана, програмне забезпечення, Шрифт

Автоматично згенерований опис

Рисунок 6 - Точність третього тесту

Зображення, що містить текст, знімок екрана, ряд, схема

Автоматично згенерований опис

Рисунок 7 - Графік точності третього тесту

**8.2.py**

import numpy as np  
import tensorflow as tf  
import matplotlib.pyplot as plt  
from tensorflow.keras.preprocessing import image\_dataset\_from\_directory  
  
# Завантаження набору даних  
dataset\_url = "https://storage.googleapis.com/download.tensorflow.org/example\_images/flower\_photos.tgz"  
data\_dir = tf.keras.utils.get\_file('flower\_photos', origin=dataset\_url, untar=True)  
  
batch\_size = 32  
img\_height = 180  
img\_width = 180  
  
train\_ds = image\_dataset\_from\_directory(  
 data\_dir,  
 validation\_split=0.2,  
 subset="training",  
 seed=123,  
 image\_size=(img\_height, img\_width),  
 batch\_size=batch\_size)  
  
val\_ds = image\_dataset\_from\_directory(  
 data\_dir,  
 validation\_split=0.2,  
 subset="validation",  
 seed=123,  
 image\_size=(img\_height, img\_width),  
 batch\_size=batch\_size)  
  
class\_names = train\_ds.class\_names  
num\_classes = len(class\_names)  
  
# Нормалізація даних  
normalization\_layer = tf.keras.layers.Rescaling(1./255)  
  
normalized\_train\_ds = train\_ds.map(lambda x, y: (normalization\_layer(x), y))  
normalized\_val\_ds = val\_ds.map(lambda x, y: (normalization\_layer(x), y))  
  
# Аугментація даних  
data\_augmentation = tf.keras.Sequential([  
 tf.keras.layers.RandomFlip("horizontal\_and\_vertical"),  
 tf.keras.layers.RandomRotation(0.2),  
])  
  
augmented\_train\_ds = normalized\_train\_ds.map(lambda x, y: (data\_augmentation(x, training=True), y))  
  
# Створення нейронної мережі  
model = tf.keras.Sequential([  
 tf.keras.layers.InputLayer(input\_shape=(img\_height, img\_width, 3)),  
 tf.keras.layers.Conv2D(32, (3, 3), activation='relu'),  
 tf.keras.layers.MaxPooling2D((2, 2)),  
 tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),  
 tf.keras.layers.MaxPooling2D((2, 2)),  
 tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),  
 tf.keras.layers.MaxPooling2D((2, 2)),  
 tf.keras.layers.Flatten(),  
 tf.keras.layers.Dense(512, activation='relu'),  
 tf.keras.layers.Dense(256, activation='tanh'),  
 tf.keras.layers.Dense(num\_classes, activation='softmax')  
])  
  
  
# Випадок 1: Менша кількість епох  
# epochs = 20  
  
# Випадок 2: Більша кількість епох  
# epochs = 60  
  
# Випадок 3: Менша швидкість навчання  
# model.compile(optimizer=tf.keras.optimizers.Adam(learning\_rate=0.0001),  
# loss=tf.keras.losses.SparseCategoricalCrossentropy(),  
# metrics=['accuracy'])  
# epochs = 50  
  
# Випадок 4: Більша швидкість навчання  
# model.compile(optimizer=tf.keras.optimizers.Adam(learning\_rate=0.01),  
# loss=tf.keras.losses.SparseCategoricalCrossentropy(),  
# metrics=['accuracy'])  
# epochs = 50  
  
# Початкові значення (для випадків 1 та 2)  
model.compile(optimizer=tf.keras.optimizers.Adam(learning\_rate=0.001),  
 loss=tf.keras.losses.SparseCategoricalCrossentropy(),  
 metrics=['accuracy'])  
  
epochs = 50 # Для випадків 3 та 4   
history = model.fit(  
 augmented\_train\_ds,  
 validation\_data=normalized\_val\_ds,  
 epochs=epochs  
)  
  
# Оцінка моделі на тестовому наборі  
loss, accuracy = model.evaluate(normalized\_val\_ds)  
print("Test Loss:", loss)  
print("Test Accuracy:", accuracy \* 100)  
  
# Побудова матриці плутанини (Confusion Matrix)  
y\_pred = []  
y\_true = []  
  
for images, labels in normalized\_val\_ds:  
 predictions = model.predict(images)  
 y\_pred.extend(np.argmax(predictions, axis=1))  
 y\_true.extend(labels.numpy())  
  
confusion\_mtx = tf.math.confusion\_matrix(y\_true, y\_pred)  
confusion\_mtx\_normalized = confusion\_mtx / tf.reduce\_sum(confusion\_mtx, axis=1, keepdims=True)  
  
# Візуалізація матриці плутанини  
plt.figure(figsize=(10, 8))  
plt.imshow(confusion\_mtx\_normalized, cmap='Blues')  
plt.colorbar()  
plt.xticks(np.arange(num\_classes), class\_names, rotation=45)  
plt.yticks(np.arange(num\_classes), class\_names)  
plt.xlabel('Predicted')  
plt.ylabel('True')  
plt.title("Confusion Matrix")  
plt.show()  
  
# Побудова графіків точності та втрат  
acc = history.history['accuracy']  
val\_acc = history.history['val\_accuracy']  
loss = history.history['loss']  
val\_loss = history.history['val\_loss']  
  
epochs\_range = range(epochs)  
  
plt.figure(figsize=(12, 6))  
plt.subplot(1, 2, 1)  
plt.plot(epochs\_range, acc, label='Training Accuracy')  
plt.plot(epochs\_range, val\_acc, label='Validation Accuracy')  
plt.legend(loc='lower right')  
plt.title('Training and Validation Accuracy')  
  
plt.subplot(1, 2, 2)  
plt.plot(epochs\_range, loss, label='Training Loss')  
plt.plot(epochs\_range, val\_loss, label='Validation Loss')  
plt.legend(loc='upper right')  
plt.title('Training and Validation Loss')  
plt.show()  
  
# Візуалізація декількох зображень з навчального набору  
plt.figure(figsize=(10, 10))  
for images, labels in train\_ds.take(1):  
 for i in range(9):  
 ax = plt.subplot(3, 3, i + 1)  
 plt.imshow(images[i].numpy().astype("uint8"))  
 plt.title(class\_names[labels[i]])  
 plt.axis("off")  
plt.show()

**Лістинг програми для «confusion matrix»:**

# Візуалізація матриці плутанини  
plt.figure(figsize=(10, 8))  
plt.imshow(confusion\_mtx\_normalized, cmap='Blues')  
plt.colorbar()  
plt.xticks(np.arange(num\_classes), class\_names, rotation=45)  
plt.yticks(np.arange(num\_classes), class\_names)  
plt.xlabel('Predicted')  
plt.ylabel('True')  
plt.title("Confusion Matrix")  
plt.show()

Зображення, що містить програмне забезпечення, Мультимедійне програмне забезпечення, текст, Графічний редактор

Автоматично згенерований опис

Рисунок 8 - Confusion Matrix

**Перший тест проведемо із такими параметрами**

Розмірність прихованих шарів =1024, 512, 256

Кількість епох = 50

Швидкість навчання =0.001

Зображення, що містить текст, знімок екрана, монітор, Мультимедійне програмне забезпечення

Автоматично згенерований опис

Зображення, що містить текст, знімок екрана, соняшник, Мультимедійне програмне забезпечення

Автоматично згенерований опис

Зображення, що містить текст, схема, знімок екрана, карта

Автоматично згенерований опис

Рисунок 9 − Результати виконання другого завдання

**Другий тест проведемо із такими параметрами**

Розмірність прихованих шарів =2048, 1024, 512

Кількість епох = 60

Швидкість навчання =0.001

Зображення, що містить текст, знімок екрана, програмне забезпечення

Автоматично згенерований опис

Зображення, що містить рослина, знімок екрана, Мультимедійне програмне забезпечення, текст

Автоматично згенерований опис

Зображення, що містить текст, знімок екрана, схема, програмне забезпечення

Автоматично згенерований опис

Рисунок 10 − Результати виконання другого завдання

**Третій тест проведемо із такими параметрами**

Розмірність прихованих шарів =512, 256,128

Кількість епох = 50

Швидкість навчання =0.01

Зображення, що містить текст, знімок екрана, програмне забезпечення, Мультимедійне програмне забезпечення

Автоматично згенерований опис

Зображення, що містить квітка, знімок екрана, Мультимедійне програмне забезпечення, програмне забезпечення

Автоматично згенерований опис

Зображення, що містить текст, знімок екрана, програмне забезпечення, Графік

Автоматично згенерований опис

Рисунок 11 − Результати виконання другого завдання

Висновок: У цій практичній роботі я вивчив та засвоїв основи нейронних мереж та їх застосуванням для класифікації зображень. Навчився створювати просту нейронну мережу з використанням модуля Numpy, для визначення різних типів одягу, використовуючи набір даних Fashion-MNIST для класифікації різних типів одягу, таких як футболки, штани, плаття тощо. Зрозумів принципи роботи нейронних мереж та їх ключові компоненти.