МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

КАФЕДРА ПРИКЛАДНОЇ МАТЕМАТИКИ І ІНФОРМАТИКИ

Практична робота №8

з дисципліни «Методи та системи штучного інтелекту»

на тему «Створення простої нейронної мережі для класифікації зображень»

Варіант 4

Виконав:

студент гр. КІБ-21

Андрій Севастьянов

Перевірив:

асистент кафедри ПМІ

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 до виконання:**

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 80% 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 No1 (опціонально): імплементуйте нейронну мережу за допомогою бібліотек tensorflow або pytorch. Порівняйте реалізацію нейронних мереж між собою.

Додаткове завдання No2 (опціонально): реалізуйте «confusion matrix» для розробленої нейронної мережі. Поясніть її результати.

**Завдання згідно до варіанту:**

Класифікація категорій кухонних предметів: Використовуйте набір даних Food-101(svhn\_cropped) для класифікації кухонних предметів на зображеннях, таких як фрукти, овочі,страви тощо. Нейронна мережа повинна складатись з 5-х шарів. Функції активації для прихованих шарів: relu, tanh

Лістинг коду простої нейромережі на numpy:

import tensorflow as tf

import tensorflow\_datasets as tfds

import numpy as np

import matplotlib.pyplot as plt

# завантажуємо датасет

ds\_train, ds\_info\_train = tfds.load('svhn\_cropped', split='train', as\_supervised=True, with\_info=True)

ds\_test, ds\_info\_test = tfds.load('svhn\_cropped', split='test', as\_supervised=True, with\_info=True)

# Преобразование TensorFlow Dataset в NumPy arrays

def dataset\_to\_numpy(dataset):

images = []

labels = []

for image, label in dataset:

images.append(image.numpy())

labels.append(label.numpy())

return np.array(images), np.array(labels)

# Завантаження даних в NumPy arrays

X\_train, y\_train = dataset\_to\_numpy(ds\_train)

X\_test, y\_test = dataset\_to\_numpy(ds\_test)

# Normalize pixel values to [0, 1]

X\_train = X\_train.astype('float32') / 255

X\_test = X\_test.astype('float32') / 255

# Перетворення міток класів в one-hot encoding

def to\_categorical(y, num\_classes):

return np.eye(num\_classes)[y]

y\_train = to\_categorical(y\_train, num\_classes=10)

y\_test = to\_categorical(y\_test, num\_classes=10)

class NeuralNetwork:

def \_\_init\_\_(self, input\_size, hidden\_size1, hidden\_size2, hidden\_size3, hidden\_size4, 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, hidden\_size3) / np.sqrt(hidden\_size2)

self.b3 = np.zeros((1, hidden\_size3))

self.W4 = np.random.randn(hidden\_size3, hidden\_size4) / np.sqrt(hidden\_size3)

self.b4 = np.zeros((1, hidden\_size4))

self.W5 = np.random.randn(hidden\_size4, output\_size) / np.sqrt(hidden\_size4)

self.b5 = np.zeros((1, output\_size))

def forward\_propagation(self, X): # пряме поширення

X = X.reshape(X.shape[0], -1)

self.z1 = np.dot(X, self.W1) + self.b1

self.a1 = np.maximum(0, self.z1)

self.z2 = np.dot(self.a1, self.W2) + self.b2

self.a2 = np.maximum(0, self.z2)

self.z3 = np.dot(self.a2, self.W3) + self.b3

self.a3 = np.maximum(0, self.z3)

self.z4 = np.dot(self.a3, self.W4) + self.b4

self.a4 = np.maximum(0, self.z4)

self.z5 = np.dot(self.a4, self.W5) + self.b5

self.z5 -= np.max(self.z5, axis=1, keepdims=True)

self.a5 = np.exp(self.z5) / np.sum(np.exp(self.z5), axis=1, keepdims=True)

return self.a5

def backward\_propagation(self, X, y): # Зворотне поширення

m = X.shape[0]

delta5 = self.a5 - y

self.dW5 = np.dot(self.a4.T, delta5) / m

self.db5 = np.sum(delta5, axis=0, keepdims=True) / m

delta4 = np.dot(delta5, self.W5.T) \* np.where(self.a4 > 0, 1, 0)

self.dW4 = np.dot(self.a3.T, delta4) / m

self.db4 = np.sum(delta4, axis=0, keepdims=True) / m

delta3 = np.dot(delta4, self.W4.T) \* np.where(self.a3 > 0, 1, 0)

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) \* np.where(self.a2 > 0, 1, 0)

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) \* np.where(self.a1 > 0, 1, 0)

self.dW1 = np.dot(X.reshape(X.shape[0], -1).T, delta1) / m

self.db1 = np.sum(delta1, axis=0, keepdims=True) / m

def update\_param(self, learning\_rate):

# Оновлення вагів і зсувів для всіх шарів

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

self.W4 -= learning\_rate \* self.dW4

self.b4 -= learning\_rate \* self.db4

self.W5 -= learning\_rate \* self.dW5

self.b5 -= learning\_rate \* self.db5

def train(self, X, y, epochs, learning\_rate, X\_val, y\_val, batch\_size=64): # Тренування

train\_accuracies = []

val\_accuracies = []

for epoch in range(epochs): # епохи

for i in range(0, X.shape[0], batch\_size): # батчі

batch\_X = X[i:i + batch\_size] # Виділяємо батч з X

batch\_y = y[i:i + batch\_size]

self.forward\_propagation(batch\_X)

self.backward\_propagation(batch\_X, batch\_y)

self.update\_param(learning\_rate)

train\_loss, train\_accuracy = self.evaluate(X, y)

val\_loss, val\_accuracy = self.evaluate(X\_val, y\_val)

train\_accuracies.append(train\_accuracy)

val\_accuracies.append(val\_accuracy)

print(f"Epoch {epoch + 1}/{epochs}, Loss: {train\_loss:.3f}, Accuracy: {train\_accuracy:.3f}, Val\_Loss: {val\_loss:.3f}, Val\_Accuracy: {val\_accuracy:.3f}")

return train\_accuracies, val\_accuracies

def evaluate(self, X, y): # Ф-ція для перевірки

predictions = self.forward\_propagation(X)

loss = -np.sum(y \* np.log(predictions)) / len(y) # обчислення втрат

accuracy = np.mean(np.argmax(predictions, axis=1) == np.argmax(y, axis=1)) # обчислення точності

return loss, accuracy

model = NeuralNetwork(3072, 512, 256, 128, 64, 10)

train\_accuracies, val\_accuracies = model.train(X\_train, y\_train, epochs=25, learning\_rate=0.01, X\_val=X\_test, y\_val=y\_test)

test\_loss, test\_accuracy = model.evaluate(X\_test, y\_test)

print("Test Loss:", test\_loss)

print("Test Accuracy:", test\_accuracy \* 100)

# Функція для виведення графіку точності

def plot\_accuracies(train\_accuracies, val\_accuracies):

plt.plot(train\_accuracies, label='Training Accuracy')

plt.plot(val\_accuracies, label='Validation Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.show()

# Виводимо графік точності

plot\_accuracies(train\_accuracies, val\_accuracies)

Перший тест проведемо з такими параметрами:

* Розмірність прихованих шарів: 1024, 512, 256, 128, 10 ;
* Кількість епох: 25;

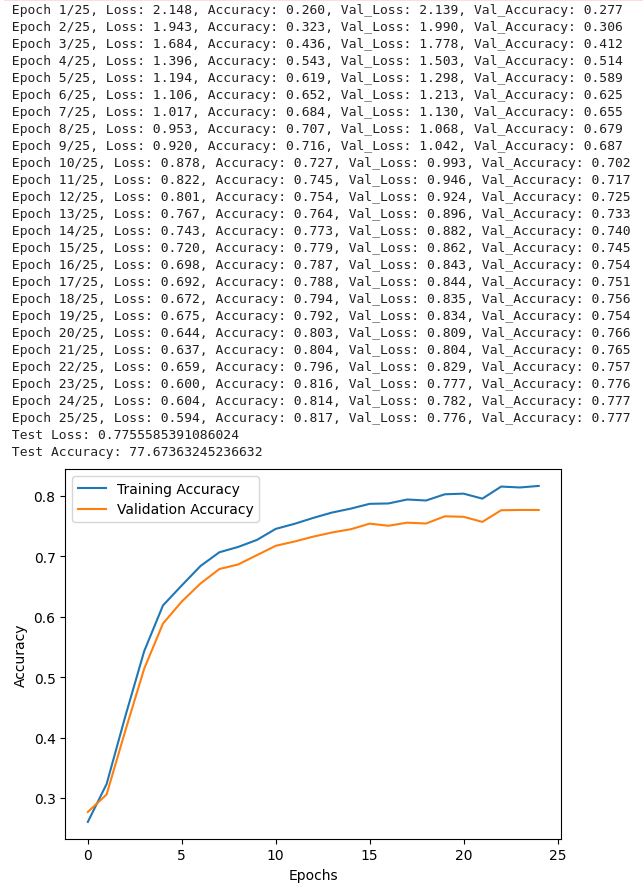


Рисунок 1 – графік точності наборів train/test та процес навчання

Другий тест проведемо з такими параметрами:

* Розмірність прихованих шарів: 512, 256, 128, 64, 10 ;
* Кількість епох: 25;

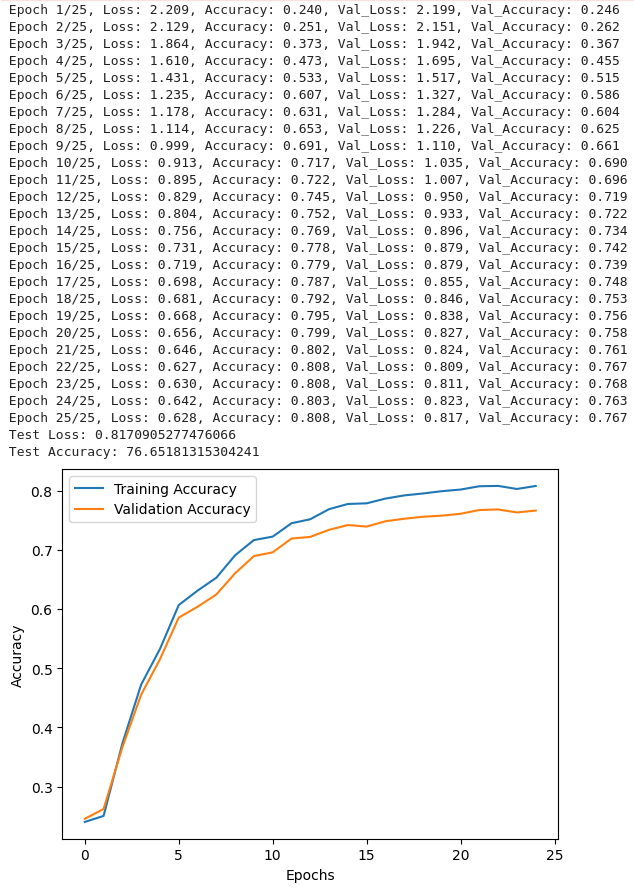


Рисунок 2 – графік точності наборів train/test та процес навчання

Третій тест проведемо з такими параметрами:

* Розмірність прихованих шарів: 1024, 512, 256, 64, 10 ;
* Кількість епох: 25;

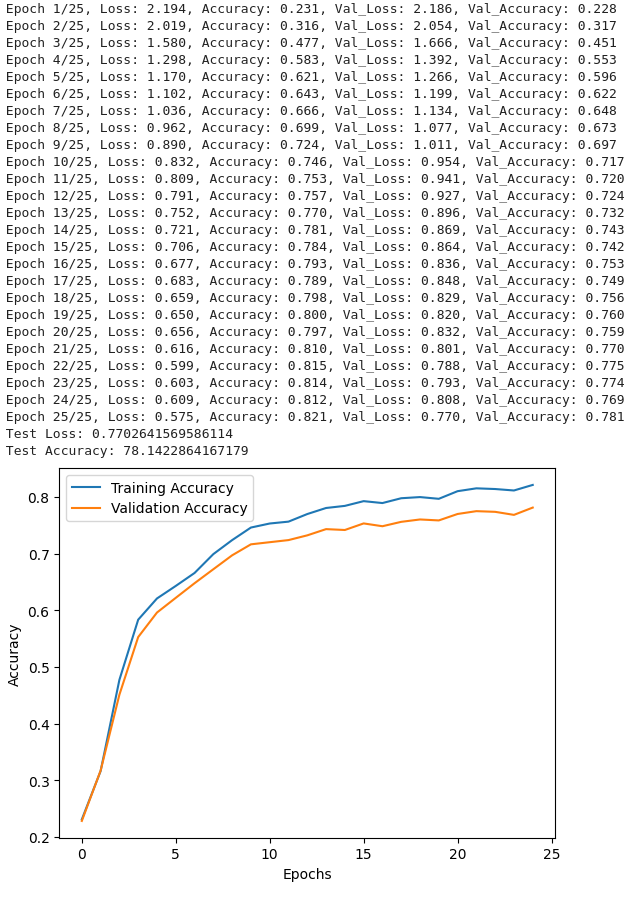


Рисунок 3 – графік точності наборів train/test та процес навчання

Четвертий тест проведемо з такими параметрами:

* Розмірність прихованих шарів: 256, 128, 64, 32, 10 ;
* Кількість епох: 25;

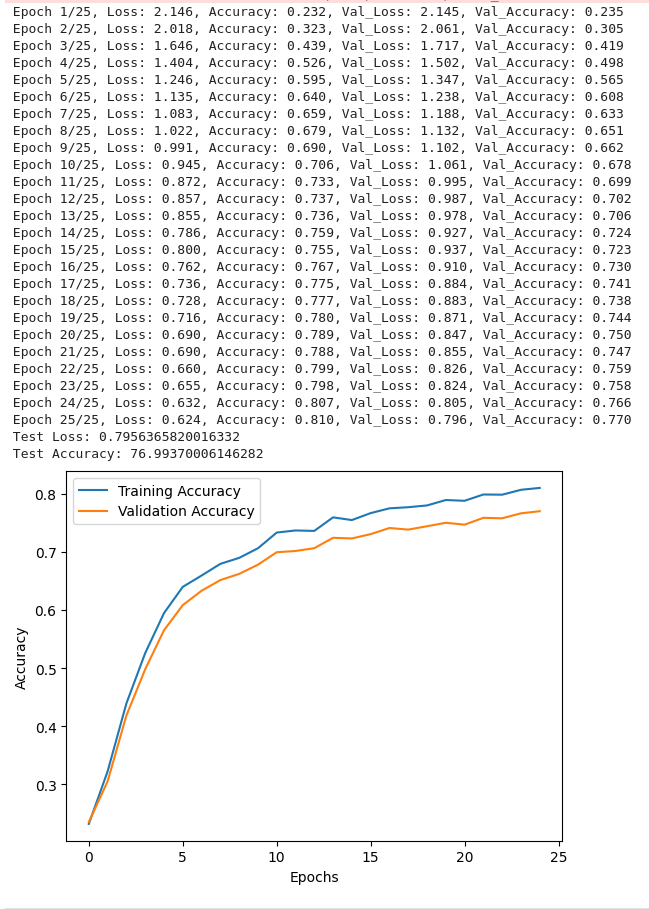


Рисунок 4 – графік точності наборів train/test та процес навчання

Далі реалізуємо дану нейронну мережу засобами tensorflow і порівняємо точність за схожих параметрів:

Лістинг коду:

import tensorflow as tf

import tensorflow\_datasets as tfds

import matplotlib.pyplot as plt

import numpy as np

# Load the SVHN dataset from TensorFlow Datasets

ds\_train, ds\_info\_train = tfds.load('svhn\_cropped', split='train', as\_supervised=True, with\_info=True)

ds\_test, ds\_info\_test = tfds.load('svhn\_cropped', split='test', as\_supervised=True, with\_info=True)

# Define preprocessing functions

def preprocess\_img(image, label):

image = tf.cast(image, tf.float32) / 255.0 # Normalize pixel values to [0, 1]

return image, tf.one\_hot(label, 10) # One-hot encode the labels

# Apply preprocessing to the datasets

ds\_train = ds\_train.map(preprocess\_img).cache().shuffle(10000).batch(64).prefetch(tf.data.experimental.AUTOTUNE)

ds\_test = ds\_test.map(preprocess\_img).batch(64).prefetch(tf.data.experimental.AUTOTUNE)

# Define model architecture

model = tf.keras.Sequential([

tf.keras.layers.Flatten(input\_shape=(32, 32, 3)),

tf.keras.layers.Dense(512, activation='relu'),

tf.keras.layers.Dense(256, activation='relu'),

tf.keras.layers.Dense(128, activation='relu'),

tf.keras.layers.Dense(64, activation='relu'),

tf.keras.layers.Dense(10, activation='softmax')

])

# Compile the model

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

# Train the model and save the training history

history = model.fit(ds\_train, epochs=25, validation\_data=ds\_test)

# Evaluate the model on the test set

test\_loss, test\_accuracy = model.evaluate(ds\_test)

print('Test accuracy:', test\_accuracy \* 100)

# Plot the training and validation accuracy

def plot\_accuracy(history):

plt.plot(history.history['accuracy'], label='Training Accuracy')

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.show()

# Display the accuracy plot

plot\_accuracy(history)

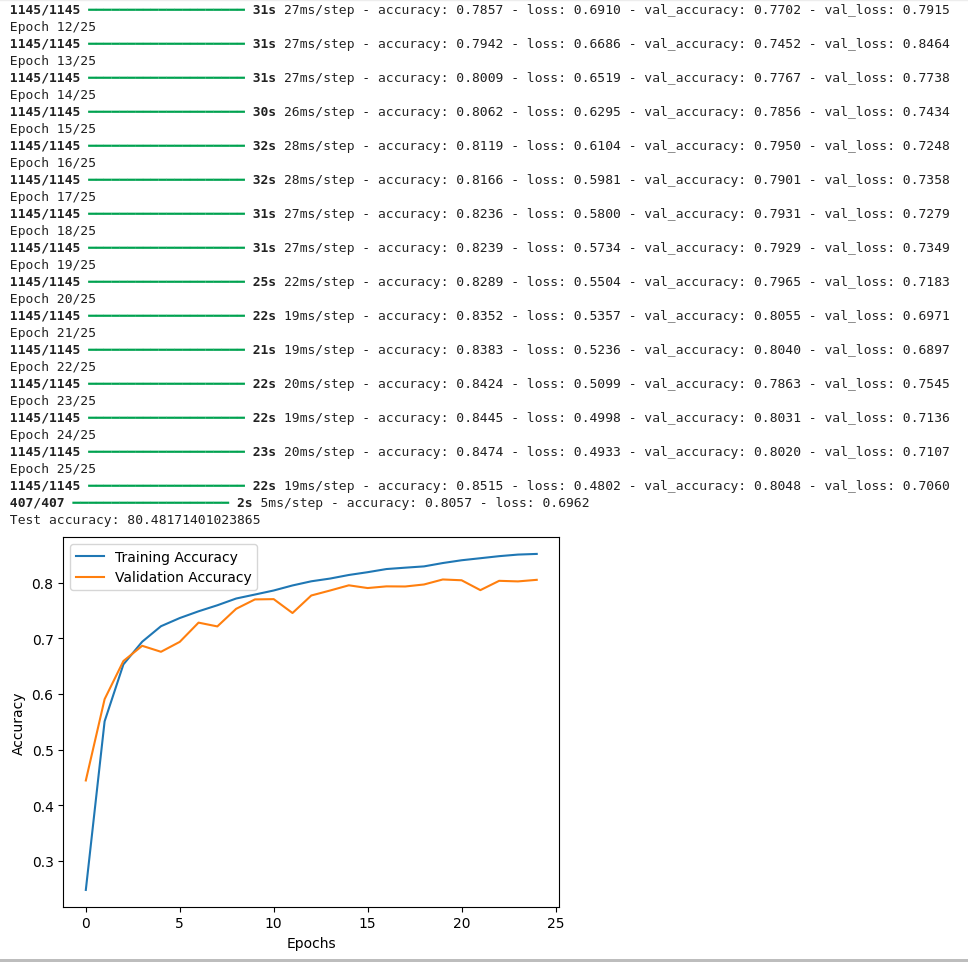


Рисунок 5 – графік точності наборів train/test та процес навчання через бібліотеку tensorflow

При порівнянні двох нейронних мереж можливо спостерігати кращу точність під час тренування. Також бачимо, що тренування відбувається краще, але різниця на тестовому наборі не є великою.

Для нейронної мережі на numpy додамо код який створить “confusion matrix”. Для візуалізації скористаємося функціоналом matplotlib/

Лістинг коду для “confusion matrix”:

# Виводимо confusion matrix

plot\_accuracies(train\_accuracies, val\_accuracies)

# Побудова confusion matrix

predictions = model.forward\_propagation(X\_test)

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

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

confusion = np.zeros((10, 10))

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

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

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

plt.figure(figsize=(8, 8))

plt.imshow(confusion, cmap='Blues')

plt.xticks(range(10))

plt.yticks(range(10))

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

plt.title("Confusion Matrix")

plt.colorbar()

plt.show()

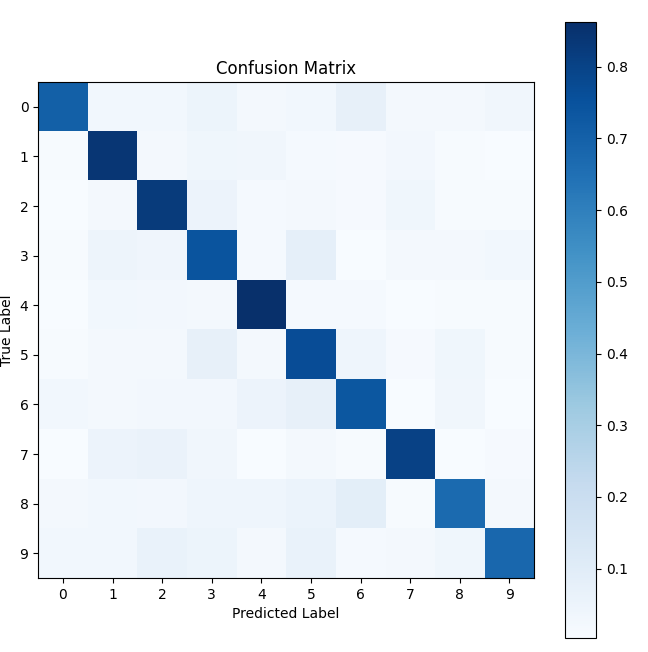


Рисунок 6 – Confusion matrix

Матриця плутанини – це метод вимірювання продуктивності для класифікації машинного навчання. Це свого роду таблиця, яка допомагає вам дізнатися про ефективність моделі класифікації на наборі тестових даних, для яких відомі справжні значення. Термін «матриця плутанини» сам по собі дуже простий, але пов’язана з ним термінологія може дещо заплутати. Тут наведено просте пояснення цієї техніки.

Висновок:

Я ознайомився з основами нейронних мереж та їх застосуванням для класифікації зображень. Навчився створювати просту нейронну мережу з використанням модуля Numpy. Зрозумв принципи роботи нейронних мереж та їх ключові компоненти.