ТРАНСПОРТНІ ПОТОКИ. АНАЛІЗ ТА ПОБУДОВА

Ольховський В.М., студент МК-51-24
Науковий керівник – ***Попова А.В.,*** доц., к.т.н.
Харківський національний автомобільно-дорожній університет

Вступ

Машинне навчання – це галузь штучного інтелекту, яка застосовує статистичні методи, дозволяючи комп'ютерним програмам навчатися на основі даних без необхідності явного програмування. Ця технологія стрімко набирає популярності та постійно вдосконалюється. Її використовують у різних сферах, таких як онлайн-перекладачі, розпізнавання тексту, пошук інформації, фільтрація електронної пошти, персоналізована реклама, чат-боти, автопілот для автомобілів тощо.

Одним із важливих досягнень у машинному навчанні стали голосові помічники, як-от Apple Siri, які здатні розпізнавати мовлення. Саме такі помічники суттєво розширили можливості застосування машинного навчання у повсякденному житті. Сьогодні нейронні мережі вже можуть виконувати більш складні завдання, наприклад, керувати автомобілем.

Огляд існуючих аналогів розробки

Сьогодні багато компаній і програм використовують технології розпізнавання тексту. Однією з найвідоміших є компанія "ABBYY", яка розробила програму "FineReader".

Цей застосунок здатний розпізнавати текст на 192 мовах і має вбудовану перевірку орфографії для 48 мов. За інформацією з офіційного сайту, технології "ABBYY FineReader" дозволяють розпізнавати текст із файлів або безпосередньо з камери смартфона за допомогою мобільного додатку.

"ABBYY FineReader" позиціюють як "універсальне рішення для роботи з паперовими та PDF-документами будь-якого типу, що об'єднує систему оптичного розпізнавання тексту (OCR) та інструменти для роботи з PDF". Це дозволяє ефективно вирішувати різні завдання, пов'язані з обробкою документів. Алгоритм розпізнавання починається з аналізу структури документа, під час якого визначаються таблиці, блоки тексту та інші елементи. Далі ці елементи розбиваються на слова, а слова – на окремі символи, які порівнюються зі зразками у програмі, що дозволяє сформувати гіпотези щодо розпізнаних символів.

Обґрунтування вибору засобів розробки

Дивлячись на те, що створювати нейронну мережу ми будемо без використання додаткових модулів, мною була обрана мова розробки Java через її простоту, роботу з класами та наявність стандартних бібліотек для побудови користувацького інтерфейсу.

Для роботи з функціями активації та деактивації було використано стандартний інтерфейс Unaryoperator він дає змогу дуже просто працювати з такими функціями.

Для зчитування файлів було використано клас File, це стандартний клас Java для зчитування та запису файлів.

Для обробки зображення та конвертування їх в масиви було використано клас BufferedImage, який наслідується від класу Image.

Для розробки користувацького інтерфейсу було використано бібліотеку javax.swing та java.awt

Створення моделі

Першим кроком ми створимо клас, який буде відповідати за один шар нейронів та назвемо його Layer. В ньому зберігаються такі дані: розмір шару, список всіх нейронів шару, список всіх нейронів зміщення та список всіх ваг для наступного шару. Нижче приведено конструктор для даного класу.

```
public Layer(int sizeOfLayer, int sizeOfNextLayer) {
    this.layer_length = sizeOfLayer;
    neurons = new double[sizeOfLayer];
    wt = new double[sizeOfLayer][sizeOfNextLayer];
    displacementNeuron = new double[sizeOfLayer];
}
```

Наступний крок це створення нейронної мережі за допомогою раніше створеного класу шару. Нейронна мережа зберігає такі дані: список всіх шарів нашої мережі та значення швидкості навчання. Нижче приведено конструктор.

```
public NeuralNetwork(double speedLearn, int... sizesOfLayers) {
    this.speedLearn = speedLearn;
    this.allLayers = new Layer[sizesOfLayers.length];
    for (int i = 0; i < sizesOfLayers.length; i++) {
        int next = 0;
        if (i < sizesOfLayers.length - 1) {
            next = sizesOfLayers[i + 1];
        }
        this.allLayers[i] = new Layer(sizesOfLayers[i], next);
        for (int j = 0; j < sizesOfLayers[i]; j++) {
            this.allLayers[i].displacementNeuron[j] = Math.random() * 2.0 - 1.0;
            for (int k = 0; k < next; k++) {
                this.allLayers[i].wt[j][k] = Math.random() * 2.0 - 1.0;
            }
        }
    }
}
```

Значення нейронів зміщення та ваг коливаються в межах проміжку [−1; 1].

Наступним кроком була розроблена функція для заповнення всіх шарів мережі значеннями, а саме значення нейрона дорівнює значення вхідного нейрона з попереднього шару помножено на відповідні вхідні ваги. Потім додається нейрон зміщення та застосовується функція активації (сигмоїда) для того, щоб вихідне значення нейрона було в межах проміжку [−1; 1].

Нижче приведена реалізація даної функції.

```
public double[] outputLayer(double[] inputs) {
    UnaryOperator<Double> sigmoidActiovationFunction = x -> 1 / (1 + Math.exp(-x));
    //копіюємо значення пікселів в перший шар нейронів
    System.arraycopy(inputs, srcPos: 0, allLayers[0].neurons, destPos: 0, inputs.length);
    //рахуємо значення для всіх вихідних шарів з урахуванням нейрона зміщення
    for (int i = 1; i < allLayers.length; i++) {
        Layer layer = allLayers[i];
        Layer previousLayer = allLayers[i - 1];
        for (int j = 0; j < layer.layer_length; j++) {
            layer.neurons[j] = 0;
            //новий шар нейронів = множимо значення вихідних нейронів на ваги
            for (int k = 0; k < previousLayer.layer_length; k++) {
                layer.neurons[j] += previousLayer.neurons[k] * previousLayer.wt[k][j];
            }
            //застосовуємо формулу активації та додаємо нейрон зміщення
            layer.neurons[j] += layer.displacementNeuron[j];
            layer.neurons[j] = sigmoidActiovationFunction.apply(layer.neurons[j]);
        }
    }
    return allLayers[allLayers.length - 1].neurons; //повертаємо останній шар вихідних нейронів
}
```

Наступний крок це реалізація методу для алгоритму зворотного розповсюдження помилки. В якому ми вираховуємо величину помилки для кожного нейрона вихідного шару, а потім за допомогою алгоритму, описаного вище, корегуємо значення ваг та нейронів зміщення для кожного шару нашої мережі.

```
//метод зворотного розповсюдження помилки
public void backpropagationAlgorithm(double[] aims) {
    UnaryOperator<Double> dsigmoidDeactionvationFunction = y -> y * (1 - y);
    double[] allErr = new double[allLayers[allLayers.length - 1].layer_length];
    //рахуємо помилку для кожного нейрону
    for (int index = 0; index < allLayers[allLayers.length - 1].layer_length; index++) {
        //помилка для кожного нейрону = потрібне значення - значення нейрона
        allErr[index] = aims[index] - allLayers[allLayers.length - 1].neurons[index];
    }
    for (int k = allLayers.length - 2; k >= 0; k--) {
        Layer previousLayer = allLayers[k];
        Layer layer = allLayers[k + 1];
        double[] nextErr = new double[previousLayer.layer_length];
        double[] deactivateNeurons = new double[layer.layer_length];
        //коефіцієнт для кожного нейрона в вихідному шарі
        for (int i = 0; i < layer.layer_length; i++) {
            deactivateNeurons[i] = allErr[i] * dsigmoidDeactionvationFunction.apply(allLayers[k + 1].neurons[i]);
            deactivateNeurons[i] *= speedLearn;
        }
        //рахуємо коефіцієнт без врахування ваг для кожного нейрона, який входив в вихідний
        double[][] koef = new double[layer.layer_length][previousLayer.layer_length];
        for (int index = 0; index < layer.layer_length; index++) {
            for (int j = 0; j < previousLayer.layer_length; j++) {
                koef[index][j] = deactivateNeurons[index] * previousLayer.neurons[j];
            }
        }
        //помилка для кожного нейрону не вихідного шару
        for (int index = 0; index < previousLayer.layer_length; index++) {
            nextErr[index] = 0;
            for (int jndex = 0; jndex < layer.layer_length; jndex++) {
                nextErr[index] += previousLayer.wt[index][jndex] * allErr[jndex];
            }
        }
```

```
        allErr = new double[previousLayer.layer_length];
        System.arraycopy(nextErr, srcPos: 0, allErr, destPos: 0, previousLayer.layer_length);
        //корекція ваг для кожного вхідного шару
        double[][] weightsNew = new double[previousLayer.wt.length][previousLayer.wt[0].length];
        for (int index = 0; index < layer.layer_length; index++) {
            for (int jndex = 0; jndex < previousLayer.layer_length; jndex++) {
                weightsNew[jndex][index] = previousLayer.wt[jndex][index] + koef[index][jndex];
            }
        }
        previousLayer.wt = weightsNew;
        //корекція нейрону зміщення
        for (int i = 0; i < layer.layer_length; i++) {
            layer.displacementNeuron[i] += deactivateNeurons[i];
        }
    }
}
```

Тренування моделі

Спочатку завантажуємо дані з бази даних рукописних цифр Mnist в якій містяться 60000 картинок різних цифр для тренування.

Потім розбиваємо всі файли на два масиви даних: картинки та відповіді

```
private static void trainFiles(int picturesNumber, BufferedImage[] images, int[] numbers_names) throws IOException {
    File[] files = new File( pathname: "train").listFiles(); //всі картинки
    for (int i = 0; i < picturesNumber; i++) {
        images[i] = ImageIO.read(files[i]); //масив картинок
        numbers_names[i] = Integer.parseInt( s: files[i].getName().charAt(10) + ""); //масив відповідей
    }
}
```

Далі нам потрібно обробити зображення та конвертувати їх у двовимірні масиви пікселів та зберегти в новому масиві даних.

```
//значення для першого шару
double[][] inputs = new double[picturesNumber][784];
//конвертуємо картинки в масиви пікселів
for (int i = 0; i < picturesNumber; i++) {
    for (int j = 0; j < 28; j++) {
        for (int k = 0; k < 28; k++) {
            inputs[i][j+k*28] = (allImages[i].getRGB(j, k) & 0xff)/255.0;
        }
    }
}
```

На наступному етапі проводиться навчання нейронної мережі протягом 3000 епох. У кожній епосі мережа отримує на вхід по 100 випадкових зображень з бази даних. Для кожного зображення на кожній ітерації його пікселі подаються на вхідний шар, який містить 784 нейрони (відповідно до кількості пікселів у картинці). Ці значення використовуються для заповнення всієї мережі за допомогою раніше написаної функції. Після цього визначається максимальне значення на вихідному шарі, який складається з 10 нейронів, кожен з яких відповідає за певну цифру. Це максимальне значення є гіпотезою щодо того, яка цифра зображена на картинці. Якщо мережа видає правильну відповідь, кількість правильних відповідей збільшується.

Далі застосовується алгоритм зворотного розповсюдження помилки, який було реалізовано раніше, для коригування значень нейронів (рис. 1).

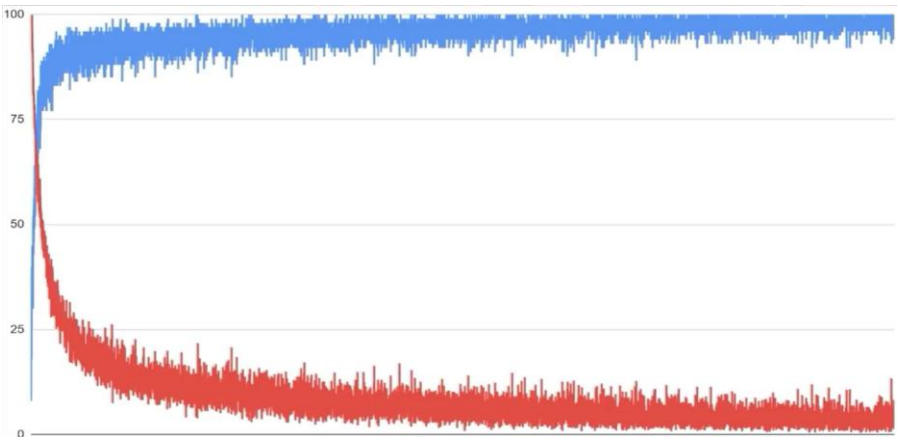

Рисунок 1

На графіку (рис. 1) показано процес навчання мережі: синя лінія ілюструє кількість правильних відповідей, а червона – величину помилки. Видно, що нейронна мережа швидко адаптувалася, проте вона все ще не повністю впевнена у своїх результатах.

Після успішного завершення навчання ми записуємо значення нашої навченої нейронної мережі в файл. Потім при кожному новому запуску програми нейронна мережа зчитується з файлу і не навчається кожного разу.

Тестування програми

При запуску програми відкривається вікно, де є полотно на якому можна малювати. Малювання здійснюється за допомогою лівої кнопки мишки, для того щоб очистити полотно використовується клавіша пробіл. На рис. 2 можна побачити як це виглядає.

Через те що розпізнавання проходить в режимі реального часу, нейронна мережа сприймає чорний екран як цифру 5.

На рис. 3 ми бачимо як програма розпізнає цифри на прикладі двійки в різних варіантах.

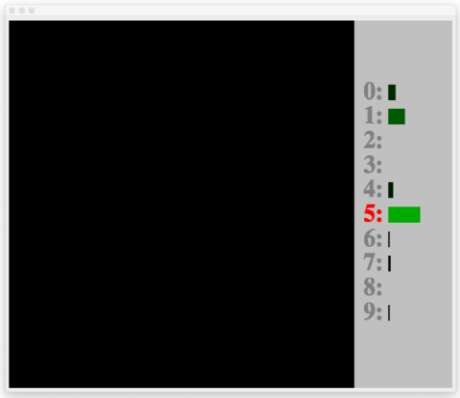

Рисунок 2

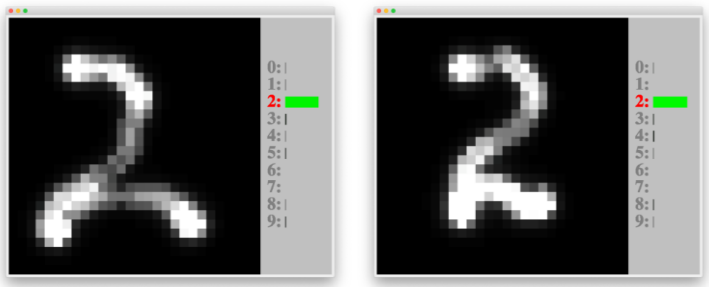

Рисунок 3

На рис. 4 ми можемо бачити випадок, коли програма не на 100% впевнена, що на малюнку зображена цифра 9.

Не вирішеною проблемою залишаються випадки, коли цифра розташована під кутом (рис. 5). На даному прикладі цифру 4 розпізнає як 5.

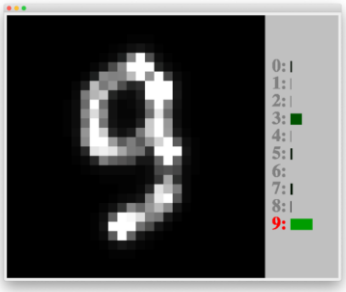

Рисунок 4

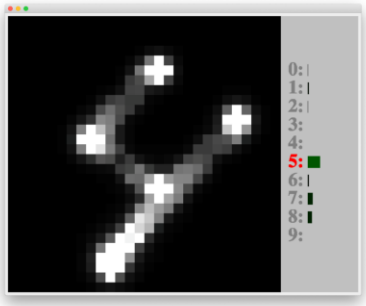

Рисунок 5

Побудува моделі TensorFlow

Перший крок - це імпорт необхідних бібліотек. TensorFlow - це бібліотека для глибокого навчання, але часто вона використовується разом з іншими бібліотеками для роботи з даними та візуалізації. Наприклад:

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, Flatten, MaxPooling2D import matplotlib.pyplot as plt

Створення моделі в TensorFlow може бути виконано різними способами. Один з найпоширеніших - використання Sequential, який дозволяє додавати шари в послідовному порядку.

model = Sequential()

Потім можна додавати шари до моделі, наприклад, згорткові шари, пулінгові шари, плоский шар і Dense шари. Наприклад:

model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(64, 64, 3)))
model.add(MaxPooling2D(pool_size=(2, 2))) model.add(Flatten())
model.add(Dense(128, activation='relu')) model.add(Dense(10, activation='softmax'))

Цей код додає згортковий шар з 32 фільтрами та активацією ReLU, пулінговий шар, плоский шар (для перетворення зображення в одновимірний вектор) і два Dense шари, останній з активацією Softmax для класифікації.

Після створення моделі її потрібно "скомпілювати". Це включає в себе вибір функції втрати, оптимізатора і метрик для оцінки моделі під час навчання. Наприклад:

model.compile(loss='categorical_crossentropy', optimizer='adam', etrics= ['accuracy'])

Тут ми використовуємо функцію втрати "categorical_crossentropy" для задачі класифікації, оптимізатор "adam" і метрику "accuracy".

Для тренування моделі потрібно завантажити дані навчання і мітки, і викликати метод fit. Наприклад:

history = model.fit(train_images, train_labels, epochs=10, validation_data= (val_images, val_labels))

Це тренує модель на навчальних даних, інструкція epochs вказує кількість повторень навчання.

Після навчання можна оцінити модель на тестових даних та здійснити передбачення на нових даних. Наприклад:

test_loss, test_accuracy = model.evaluate(test_images, test_labels) predictions = model.predict(new_data)

Це дозволяє вам визначити точність моделі на тестових даних і отримати передбачені значення для нових даних.

Модель можна зберегти на диск для подальшого використання: model.save (’my_model.h5’)

А потім завантажити її для використання без повторного навчання: loaded_model = tf.keras.models.load_model(’my_model.h5’)

Підготовка моделі OCR для українських літер

Для початку нам треба розуміти що таке OCR.

OCR (Optical Character Recognition) - це технологія, яка використовується для автоматичного розпізнавання тексту на зображеннях або сканах. Головною метою OCR є перетворення тексту, зображеного на папері або в іншому графічному форматі, текст, який можна аналізувати, редагувати та зберігати в електронному вигляді.

Основні завдання OCR включають в себе:

- Зчитування тексту з зображень: OCR аналізує зображення або скани та намагається виявити на них текстові об’єкти.
- Визначення локальзації тексту: OCR може визначити, де на зображенні розміщений текст, визначаючи координати окремих символів або рядків.
- Розпізнавання символів: OCR аналізує форму та структуру символів і спробовує визначити, які саме символи або слова присутні на зображенні.
- Перетворення на текст: Основним результатом роботи OCR є перетворення знайденого тексту на машинночитабельний текст у вигляді послідовності символів.

OCR використовується в багатьох областях, таких як сканування документів, розпізнавання рукописного тексту, автоматизація обробки документів, розпізнавання номерів та тексту на зображеннях, оптичний розпізнавання символів в банкнотах і багато інших додатках.

OCR важливий інструмент для конвертації фізичних документів в електронний формат та подальшого аналізу, обробки і збереження текстової інформації.

Для побудови OCR моделі скористаємось TensorFlow та навчання нейронної мережі для розпізнавання букв українського алфавіту (літери від Л до Р). Ця задача включає в себе кілька важливих кроків:

- Підготовка даних:

Спершу необхідно зібрати набір даних, який містить зображення літер українського алфавіту від Л до Р. Цей набір даних потрібно розподілити на навчальну та тестову вибірку.

- Передпроцесинг даних:

Зображення літер потрібно перевести у відповідний формат для подальшого використання нейронною мережею. Зазвичай це включає в себе зміну розміру зображень, масштабування пікселів до діапазону [0, 1] і може вимагати перетворення в чорно-білий колір (якщо зображення не чорно-білі).

- Побудова моделі:

Потрібно створити архітектуру нейронної мережі для розпізнавання літер. Модель може містити згорткові (Conv2D) та пулінгові (MaxPooling2D) шари, плоский шар (Flatten) та Dense шари. Кількість шарів і їх параметри можуть бути налаштовані залежно від завдання та доступних ресурсів.

- Компіляція моделі:

Модель потрібно скомпілювати, визначивши функцію втрати (наприклад, категоріальна крос-ентропія), оптимізатор (наприклад, Adam) та метрики (наприклад, точність) для оцінки її ефективності.

- Тренування моделі:

Модель треба тренувати на навчальних даних з використанням методу fit. Процес тренування варіюється в залежності від обсягу даних і ресурсів, і може займати певний час.

- Оцінка та налагодження:

Після тренування модель потрібно оцінити на тестових даних для визначення її точності та продуктивності. Якщо результати не задовільняють, можливо, буде потрібно налаштувати архітектуру моделі, параметри навчання або зібрати більше даних.

- Використання моделі:

Після успішного навчання і налагодження можна використовувати модель для розпізнавання літер українського алфавіту. Подавайте нові зображення на вхід моделі та отримуйте передбачені класи або ймовірності.

Нижче на рис.16 відбувається підготовка даних для тренування моделі. Я використовував окремі зображення.

Рисунок 6 – Підготовка даних для тренування моделі

Нижче на рис. 7 наведено код побудови моделі OCR з використанням TenserFlow:

main.py × tensortest.py tendortest2.py Dockerfile

```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, Flatten, MaxPooling2D, Dropout
from tensorflow.keras.preprocessing.image import ImageDataGenerator
# Параметри моделі
input_shape = (28, 28, 1)  # Розмір вхідного зображення
num_classes = 6  # Кількість класів (Л, М, Н, О, П, Р)
# Створення моделі
model = Sequential([
    Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=input_shape),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(64, kernel_size=(3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(num_classes, activation='softmax')
])
# Компіляція моделі
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

# Виведення підсумку моделі
model.summary()
# Підготовка даних для тренування моделі
train_data_gen = ImageDataGenerator(rescale=1./255)
train_generator = train_data_gen.flow_from_directory(
    'data/train',
    target_size=(28, 28),
    batch_size=32,
    color_mode='grayscale',
    class_mode='categorical')
# Тренування моделі
model.fit(train_generator, epochs=10)
# Збереження моделі
model.save('letter_recognition_model.h5')
```

Рисунок 7– Побудова моделі OCR

Тепер можна запустити скрипт, переконавшись, що всі залежності та бібліотеки встановлені, для цього я використовую Docker контейнер (рис. 8).

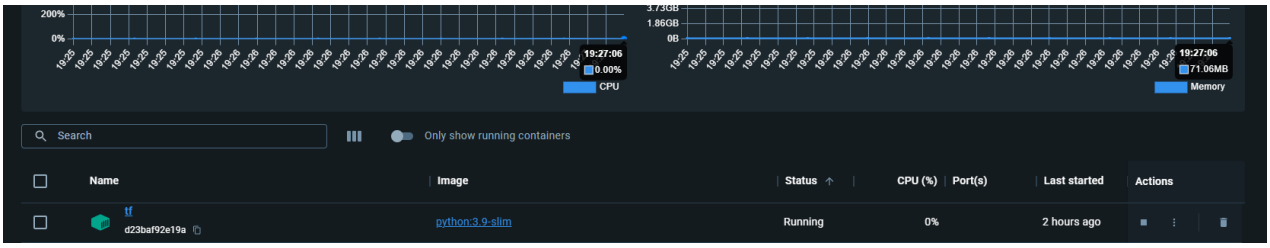

Рисунок 8 – Docker контейнер для виконання задачі

За результатами (рис. 9) тренування можна відзначити:

Епоха 1 має дуже низьку точність (accuracy: 0.1667), це може бути пов’язано з недостатньою кількістю даних або слабкими параметрами моделі.

Після 3-ї епохи точність починає покращуватися і залишається на рівні близько 0.6667 протягом решти епох.

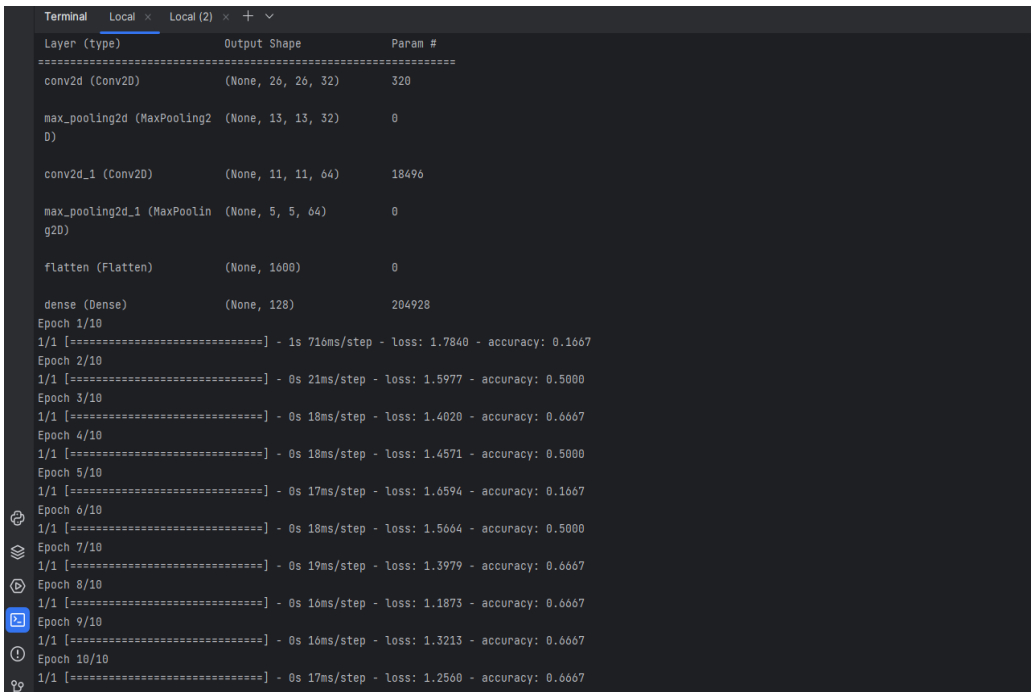

Рисунок 9 – Успішне створення моделі OCR

Тестування моделі OCR

Тестування допомагає перевірити, наскільки точно може класифікувати букви на нових зображеннях, які не використовувались під час тренування. Це важливо для переконання, що модель буде ефективно працювати в реальних умовах, а не тільки на даних, на яких вона була навчена. Для цього необхідно зробити: імпорт необхідних бібліотек:

Tensorflow, ImageDataGenerator, numpy.

Завантажую попередньо навчену модель letter_recognition_model.h5, яку створив для розпізнавання букв. Використовуватиму ImageDataGenerator для підготовки тестових даних. Це включає масштабування зображень (нормалізацію пікселів до діапазону від 0 до 1), вказання розміру зображення, що очікується моделлю (28x28 пікселів), та встановлення color_mode на ’grayscale’, оскільки модель, ймовірно, була навчена на чорно-білих зображеннях. Для кожного прогнозу виводиться індекс класу, який модель вважає найбільш ймовірним для даного зображення. Це робиться за допомогою np.argmax(pred), яке визначає індекс найбільшого значення в прогнозованому векторі, що відповідає класу, на який модель вказує.

Нижче, на Рисунку 10 можна побачити код та як це працює.

За результатами виконання цього коду можно побачити індекси класів для перших 5 зображень у тестовому наборі. Наприклад, "Зображення 0: Клас 5" означає, що перше зображення класифікувалося як клас 5 за попередньо навченою моделлю (рис. 11).

```python
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import numpy as np

# Завантаження моделі
model = tf.keras.models.load_model('letter_recognition_model.h5')

# Підготовка тестових даних
test_data_gen = ImageDataGenerator(rescale=1./255)
test_generator = test_data_gen.flow_from_directory(
    'data/test',  # шлях до папки з тестовими даними
    target_size=(28, 28),
    batch_size=1,
    color_mode='grayscale',
    class_mode='categorical',
    shuffle=False)

# Прогнозування
predictions = model.predict(test_generator, steps=len(test_generator))

# Оцінка результатів
# Для цього вам потрібно мати мітки для порівняння
# Наприклад, вивести перші кілька прогнозів
for i, pred in enumerate(predictions[:5]):
    print(f"Зображення {i}: Клас {np.argmax(pred)}")
```

Рисунок 10 – Код тестування моделі

```
1/1 [==============================] - 0s 102ms/step
Зображення 0: Клас 5
root@d23baf92e19a:/home#
EasyOCR > tensortest.py
```

Рисунок 11 – Результат тестування моделі

Висновок

У процесі збору та аналізу інформації було прийнято рішення створити програму для розпізнавання цифр, яку можна буде в подальшому модифікувати для роботи з будь-якими графічними об’єктами.

В результаті роботи мета проекту була досягнута. Було детально вивчено нейронні мережі, зокрема архітектуру «перцептрон», і створено програмну реалізацію. Крім того, розроблено користувацький інтерфейс для демонстрації роботи нейронної мережі.

Переваги використання TensorFlow для розпізнавання букв українського алфавіту:

1. Гнучкість у моделюванні: TensorFlow надає можливість створювати різні архітектури нейронних мереж, які можна оптимізувати для конкретних завдань, таких як розпізнавання букв.

2. Обробка зображень: Використання конволюційних нейронних мереж (CNN) в TensorFlow є ефективним підходом для задач розпізнавання образів, оскільки такі моделі добре виявляють шаблони та особливості на зображеннях.

3. Можливість масштабування: TensorFlow підтримує розподілене навчання та працює ефективно як на CPU, так і на GPU, що дозволяє масштабувати обробку великих наборів даних.

4. Висока точність: З правильно підготовленими даними та належним тренуванням, моделі, створені за допомогою TensorFlow, можуть досягати високої точності у розпізнаванні букв, що є важливим для таких завдань, як автоматичний ввід тексту або допоміжні технології.

5. Необхідність оптимізації та доопрацювання: Для досягнення високої продуктивності може знадобитися значна робота з оптимізації архітектури мережі, налаштування гіперпараметрів та обробки даних.

6. Важливість якісного набору даних: Успіх навчання моделі значною мірою залежить від якості та різноманітності даних. Важливо, щоб у наборі даних були представлені всі букви від Л до Р у різних шрифтах та контекстах.

7. Перспективи розвитку: Завдяки постійному прогресу в області машинного навчання є значні перспективи для подальшого вдосконалення таких систем, включно з автоматизованим розпізнаванням рукописного тексту, глибшим розумінням контексту та впровадженням навчання з підкріпленням.

Література

1. ABBYY FineReader [Електронний ресурс]: Матеріал з сайту ABBYY – режим доступу: https://www.abbyy.com/en-eu/finereader/

2. Docker [Електронний ресурс]: Матеріал з сайту Docker – режим доступу: – https://www.docker.com/

3. PyCharm [Електронний ресурс]: Матеріал з сайту JetBrains – режим доступу: – https://www.jetbrains.com/pycharm/

4. JetBrains Idea for Java [Електронний ресурс]: Матеріал з сайту JetBrains режим доступу: – https://www.jetbrains.com/idea/

5. OCR Guide [Електронний ресурс]: Матеріал з сайту JetBrains режим доступу: – https://nanonets.com/blog/ocr-with-tesseract/

6. Tensorflow Guide [Електронний ресурс]: Матеріал з сайту Tensorflow режим доступу: – https://www.tensorflow.org/learn

7. Paperspace Guide [Електронний ресурс]: Матеріал з сайту Paperspace режим доступу: – https://blog.paperspace.com/absolute-guide-to-tensorflow/

8. Курс Плєхова Г.А.«Моделювання транспортних потоків», ХНАДУ: – https://dl2022.khadi-kh.com/course/view.php?id=5092