

Цель

Реализовать классификацию черно-белых изображений рукописных цифр (28x28) по 10 категориям (от 0 до 9).

Задачи

- Ознакомиться с представлением графических данных
- Ознакомиться с простейшим способом передачи графических данных нейронной сети
- Создать модель
- Настроить параметры обучения
- Написать функцию, позволяющая загружать изображение пользователя и классифицировать его

Требования

1. Найти архитектуру сети, при которой точность классификации будет не менее 95%
2. Исследовать влияние различных оптимизаторов, а также их параметров, на процесс обучения
3. Написать функцию, которая позволит загружать пользовательское изображение не из датасета

Ход работы

1. Использование алгоритма *Adam* со стандартными параметрами Результат: 0.97649997472763

2. Использование алгоритма *RMSprop* со стандартными параметрами Результат: 0.978600025177002. Данный результат всего на 0.0015 лучше, чем у *Adam*.

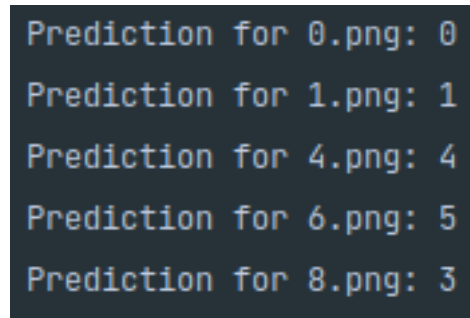
3. Использование алгоритма *SGD* со стандартными параметрами Результат: 0.908200025558471
Данный результат не удовлетворяет требуемой точности.

4. Использование алгоритма *Adam* с параметрами, отличными от стандартных Результат с $\beta_1 = 0.5, \beta_2 = 0.999$: 0.9761999845504761. Данный результат почти не отличается от результата работы оптимизатора со значениями по умолчанию (0.9 и 0.999).

Результат с $\beta_1 = 0.9, \beta_2 = 0.5$: 0.97079998254776. Данный результат хуже работы оптимизатора со значениями по умолчанию.

Результат с $\beta_1 = 0.999999, \beta_2 = 0.999$: 0.9607999920845032. Данный результат так же хуже работы оптимизатора со значениями по умолчанию.

5. Результат распознавания Результат работы программы приведён на рис.1. Ошибки распознавания 6 и 8, возможно, можно объяснить оригинальностью автора картинок.



```
Prediction for 0.png: 0
Prediction for 1.png: 1
Prediction for 4.png: 4
Prediction for 6.png: 5
Prediction for 8.png: 3
```

Рис. 1: Результат работы программы

Вывод

В ходе выполнения лабораторной работы были получены практические навыки построения нейронных сетей для распознавания рукописных символов (цифр). Так же были сравнены алгоритмы оптимизации.

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД

```
import sys

from typing import Tuple, List

import numpy as np
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras import layers
from tensorflow.keras.datasets import mnist
from tensorflow.python.keras import models
from tensorflow.keras.optimizers import Adam, RMSprop, SGD
from tensorflow.python.keras.callbacks import History
```

```

from tensorflow.python.keras.utils.np_utils import to_categorical

def load_data() -> Tuple[Tuple[np.array, np.array], Tuple[np.array, np.array]]:
    (train_images, train_labels), (test_images, test_labels) = mnist.load_data()

    train_images = train_images / 255.0
    test_images = test_images / 255.0

    train_labels = to_categorical(train_labels)
    test_labels = to_categorical(test_labels)
    return (train_images, train_labels), (test_images, test_labels)

def create_model() -> models.Model:
    model = models.Sequential()
    model.add(layers.Flatten())
    model.add(layers.Dense(256, activation="relu"))
    model.add(layers.Dense(10, activation="softmax"))

    optimizer = Adam()
    model.compile(optimizer=optimizer, loss="categorical_crossentropy",
                  metrics=["accuracy"])
    return model

def train_model(model: models.Model, train_data: np.array, train_labels: np.array,
               return model.fit(train_data, train_labels, batch_size=batch_size, epochs=epochs)

def load_image_to_array(path: str) -> np.array:
    image = load_img(path, color_mode="grayscale", target_size=(28, 28))
    input_arr = img_to_array(image)

```

```

input_arr -= 255
input_arr /= -255

ret_array = np.array([input_arr])
return ret_array

def main(images: List[str]):
    (train_data, train_labels), (test_data, test_labels) = load_data()

    model = create_model()
    train_model(model, train_data, train_labels, 128, 5)

    results = model.evaluate(test_data, test_labels)
    print(results)

    for image_path in images:
        image = load_image_to_array(image_path)
        print(f"Prediction for {image_path}: {np.argmax(model.predict(image))}")

if __name__ == "__main__":
    main(sys.argv[1:])

```