МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Искусственные нейронные сети»

Тема: Распознавание рукописных символов

Студентка гр. 8382 _____ Ефимова М.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2021

Цель работы.

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

Порядок выполнения работы.

1. Ознакомиться с представлением графических данных.
2. Ознакомиться с простейшим способом передачи графических данных нейронной сети.
3. Создать модель.
4. Настроить параметры обучения.
5. Написать функцию, позволяющая загружать изображение пользователи и классифицировать его.

Требования к выполнению задания.

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.

Основные теоретические положения.

MINIST – база данных рукописных цифр, имеющая подготовленный набор обучающих значений в размере 60000 примеров и тестовых значений из 10000 примеров. Это подмножество более широкого набора, доступное из NIST. Наш набор состоит из изображений размером 28x28, каждый пиксель которого представляет собой оттенок серого, цифры нормализированы по размеру и имеют фиксированный размер изображения. Таким образом, есть 10 цифр (от 0 до 9) или 10 классов для прогнозирования. Результаты сообщаются с использованием ошибки прогнозирования, которая является не чем иным, как инвертированной точностью классификации.

Ход работы.

Была создана и обучена модель искусственной нейронной сети. Код предоставлен в приложении А.

Оптимизатор adam.

Скорость обучения = 0.001, инициализация весов – normal, Epochs = 5, batch_size = 128, loss = categorical_crossentropy.

Слои:

```
model.add(Flatten())

model.add(Dense(64,activation='relu'))

model.add(Dense(32, activation='relu'))

model.add(Dense(10, activation='softmax'))
```

Данная архитектура дает точность ~ 96%. Графики точности и ошибки предоставлены на рис. 1 и рис. 2 соответственно.

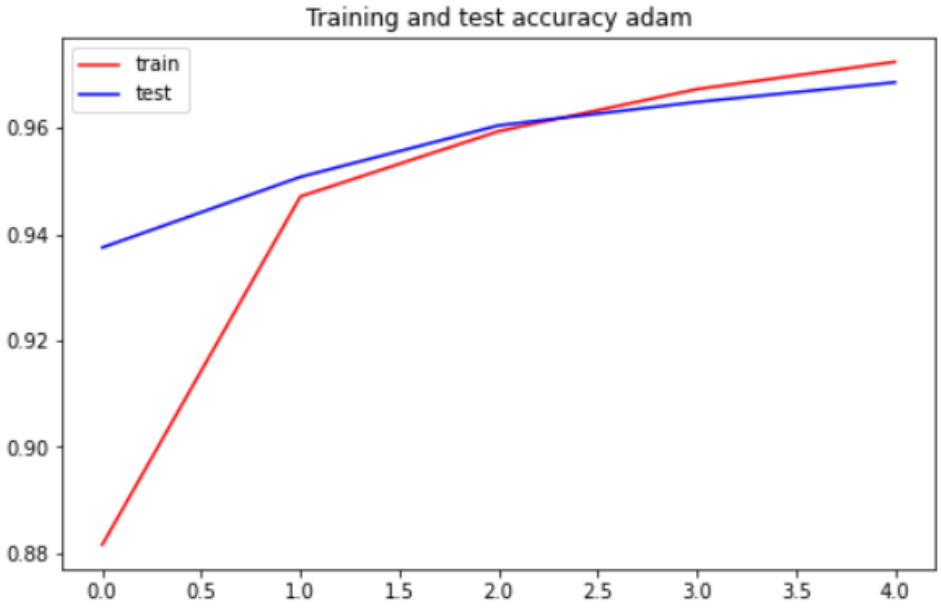

Рисунок 1 – График точности для оптимизатора adam

Рисунок 2 – График потерь для оптимизатора adam

Оптимизатор SGD.

Данная архитектура дает точность ~ 90%. Графики точности и ошибки предоставлены на рис. 1 и рис. 2 соответственно.

Рисунок 3 – График точности для оптимизатора SGD

Рисунок 4 – График потерь для оптимизатора SGD

Оптимизатор Adagrad.

Можно заметить, что обучение происходит медленнее, чем в предыдущих двух случаях. Графики точности и ошибкипредоставлены на рис. 1 и рис. 2 соответственно.

Рисунок 5 – График точности для оптимизатора Adagrad

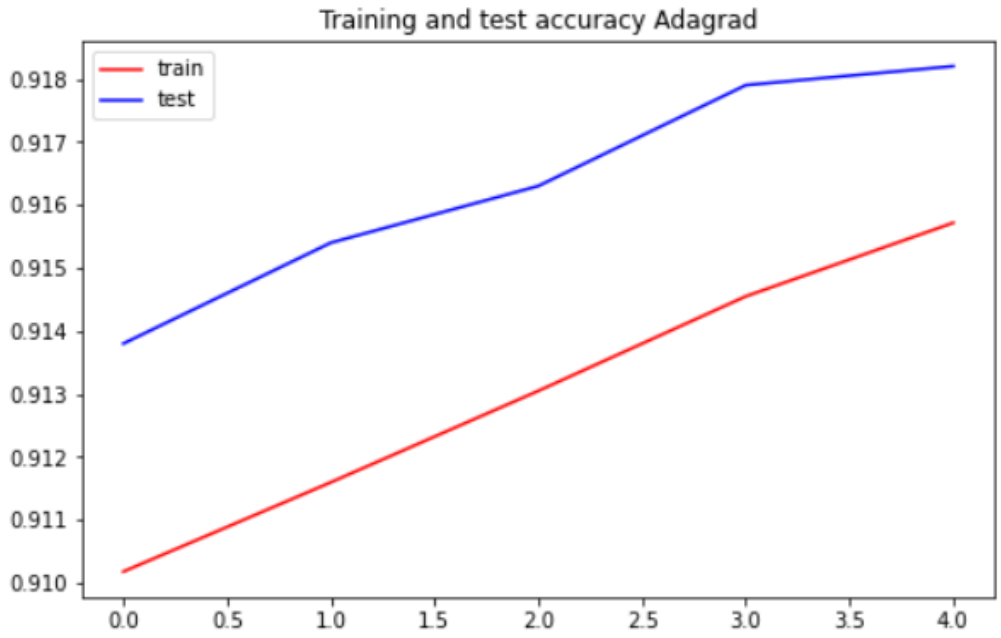

Рисунок 6 – График потерь для оптимизатора Adagrad

Из графиков можем сделать вывод, что оптимизатор Adam обучается лучше всего, в то время как Adagrad и SGD заметно отстают.

Напишем функцию, которая позволит загружать пользовательское изображение не из датасета:

```
def openImage(path):
    return numpy.asarray(Image.open(path))
```

Посмотрим, как выглядят первые 25 изображений с помощью функции imshow:

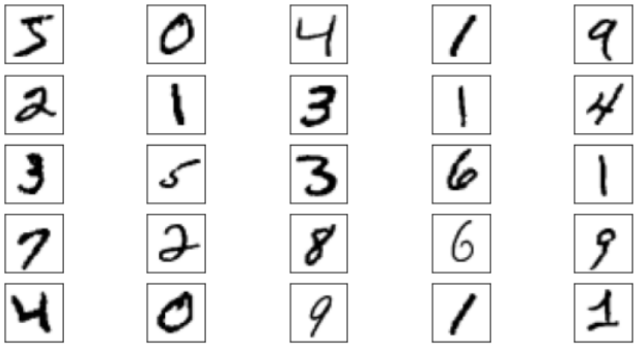

Рисунок 7 – Вывод первых 25 изображений

В первом случае используем 4 слоя, во втором случае используем три слоя.

```
Flatten(input_shape=(28, 28, 1))
Dense(128, activation='relu')
Dense(10, activation='softmax')
```

Первый слой должен преобразовывать изображение 28 на 28 пикселей в вектор из 784 элементов, следующий слой создадим с помощью уже известного нам класса Dense, который свяжет все 784 входа со всеми 128 нейронами. И такой же последний слой из 10 нейронов, который будет связан со всеми 128 нейронами предыдущего слоя.

Компиляция нейронной сети: оптимизация - Adam и критерием - категориальная кросс-энтропия.

Проверим распознавание отдельных цифр:

```
[ ] n = 0
    x = np.expand_dims(x_test[n], axis=0)
    res = model.predict(x)
    print(res)
    print(f"The number is: {np.argmax(res)}") #индекс макс значения (в данном случае это будет 7)

    plt.imshow(x_test[n], cmap = plt.cm.binary)
    plt.show()
```

```
[[4.8481719e-10 1.7535955e-10 2.4516129e-07 1.0127516e-05 6.9586700e-15
  1.9180276e-12 2.3376444e-14 9.9998963e-01 1.1937073e-08 7.2264039e-09]]
The number is: 7
```

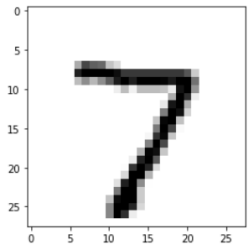

Рисунок 8 – Проверка распознавания цифры 7

Также выведем варианты, которые сеть распознала неверно:
Изначально происходит поэлементное сравнение значений, далее выделение неверно распознанных вариантов и запись их количества.
Выведем первые 2 варианта:

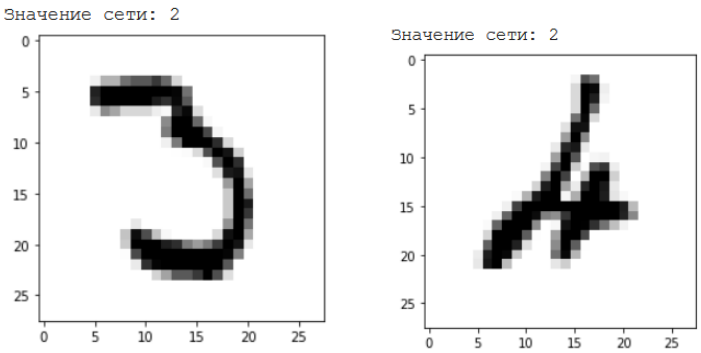

Рисунок 9 – Вывод неверного распознавания первых двух цифр

Выводы.

В ходе работы была изучена задача классификации рукописных цифр с помощью базы данных MINIST. Подобрана архитектура, дающая точность свыше 96%, таковой оказалась adam. Сеть была обучена в первом случае – на четырех слоях, во втором случае – на трех слоях. Также была написана функция загрузки изображения в память программы. Отдельно были рассмотрены неверно распознанные значения при обучении нейронной сети на трех слоях.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

```
from keras.layers import Dense, Flatten

from keras.models import Sequential

from keras.datasets import mnist

from keras.utils import to_categorical

from keras import optimizers

from PIL import Image

import matplotlib.pyplot as plt

import numpy as np

from tensorflow.keras.datasets import mnist

from tensorflow import keras

from tensorflow.keras.layers import Dense, Flatten

%matplotlib inline


def openImage(path):

    return numpy.asarray(Image.open(path))

(x_train, y_train),(x_test, y_test) = mnist.load_data()


x_train = x_train / 255

x_test = x_test / 255


y_train_cat = keras.utils.to_categorical(y_train, 10)

y_test_cat = keras.utils.to_categorical(y_test, 10)


plt.figure(figsize=(10,5))

for i in range(25):

    plt.subplot(5,5,i+1)

    plt.xticks([])

    plt.yticks([])

    plt.imshow(x_train[i], cmap=plt.cm.binary)  #берем первые 25 изображений с пом. ф-ции
imshow и отображаем на экране
```

```
plt.show()


model = Sequential()

model.add(Flatten())

model.add(Dense(64, activation='relu'))

model.add(Dense(32, activation='relu'))

model.add(Dense(10, activation='softmax'))


def compile_fit_print(optimizer, name):

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

    H = model.fit(x_train, y_train_cat, epochs=5, batch_size=128, validation_data=(x_test,
y_test_cat))


    plt.figure(1, figsize=(8, 5))

    plt.title('Training and test accuracy ' + name)

    plt.plot(H.history['accuracy'], 'r', label='train')

    plt.plot(H.history['val_accuracy'], 'b', label='test')

    plt.legend()

    plt.show()

    plt.clf()


    plt.figure(1, figsize=(8, 5))

    plt.title('Training and test loss ' + name)

    plt.plot(H.history['loss'], 'r', label='train')

    plt.plot(H.history['val_loss'], 'b', label='test')

    plt.legend()

    plt.show()

    plt.clf()



compile_fit_print(optimizers.Adam(), 'adam')


model = keras.Sequential([

    Flatten(input_shape=(28, 28, 1)), #Первый слой должен преобразовывать изображение 28x28
```

пикселей в вектор из 784 элементов, 1 - 1 bite of grey(from 255)

Dense(128, activation='relu'), #Следующий слой создадим с помощью уже известного нам класса Dense, который свяжет все 784 входа со всеми 128 нейронами.

Dense(10, activation='softmax') #И такой же последний слой из 10 нейронов, который будет связан со всеми 128 нейронами предыдущего слоя.

])

```
print(model.summary())

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

model.fit(x_train, y_train_cat, batch_size=32, epochs=10, validation_split=0.2)

model.evaluate(x_test, y_test_cat)

n = 0
x = np.expand_dims(x_test[n], axis=0)
res = model.predict(x)
print(res)
print(f"The number is: {np.argmax(res)}") #индекс макс значения (в данном случае это будет 7)

plt.imshow(x_test[n], cmap = plt.cm.binary)
plt.show()

n = 1
x = np.expand_dims(x_test[n], axis=0)
res = model.predict(x)
print(res)
print(f"The number is: {np.argmax(res)}") #индекс макс значения (в данном случае это будет 2)

plt.imshow(x_test[n], cmap = plt.cm.binary)
plt.show()
```

```
pred = model.predict(x_test)  #пропускаем всю тестовую выборку
pred = np.argmax(pred, axis=1)


print(pred.shape)


print(pred[:20])
print(y_test[:20])


mask = pred == y_test   #поэлементно сравниваем значения
print(mask[:10])


x_false = x_test[~mask] #выделаем неверные результаты распознавания
p_false = pred[~mask] #их значения


print(x_false.shape)


for i in range(5):
  print("Значение сети: "+str(p_false[i]))
  plt.imshow(x_false[i], cmap=plt.cm.binary)
  plt.show()
```