**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

лабораторная работа №4

**по дисциплине «Искусственные нейронные сети»**

**Тема: «Распознавание рукописных символов»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 7383 |  | Зуев Д.В. |
| Преподаватель |  | Жукова Н. А. |

Санкт-Петербург

2020

**Цели.**

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 пользователи и классифицировать его

**Ход работы.**

1. Для определения оптимальной архитектуры искусственной нейронной сети с точностью более 95% была взята к сведению статья [1]. Получившаяся архитектура представлена на рис. 1.

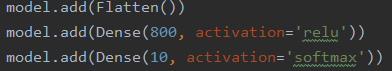


Рисунок 1 - Архитектура сети

Подтверждение того, что точность данной архитектуры более 95% подтверждена ниже при исследовании результатов обучения на различных оптимизаторах.

1. Для нахождения наиболее точной модель нейронной сети были исследованы различные оптимизаторы с различными параметрами. Исследуемые оптимизаторы представлены на рис. 2.

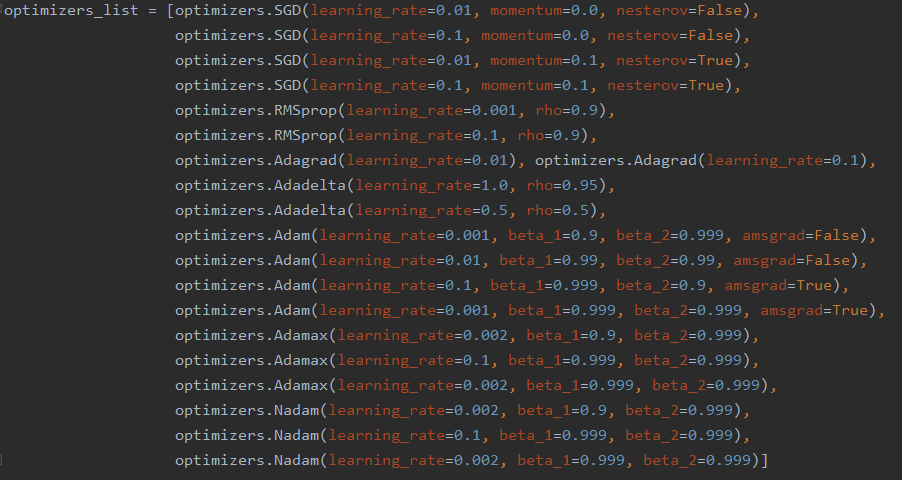


Рисунок 2 - Исследуемые оптимизаторы

Сравнение потерь и точности моделей, построенных при использовании представленных оптимизаторов, представлено на рис. 3 и рис. 4 соответственно.

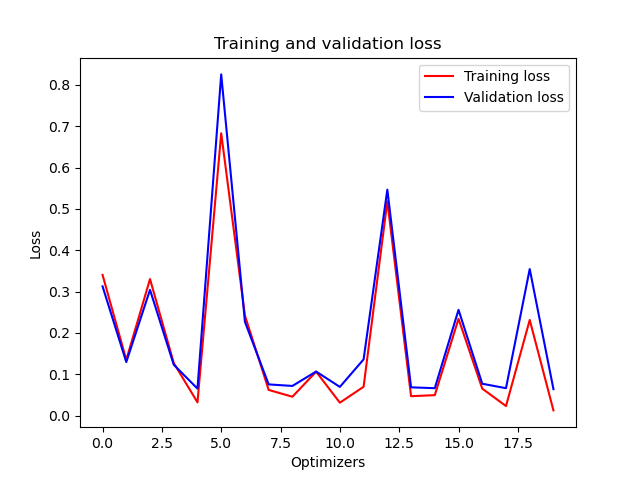


Рисунок 3 - Потери оптимизаторов

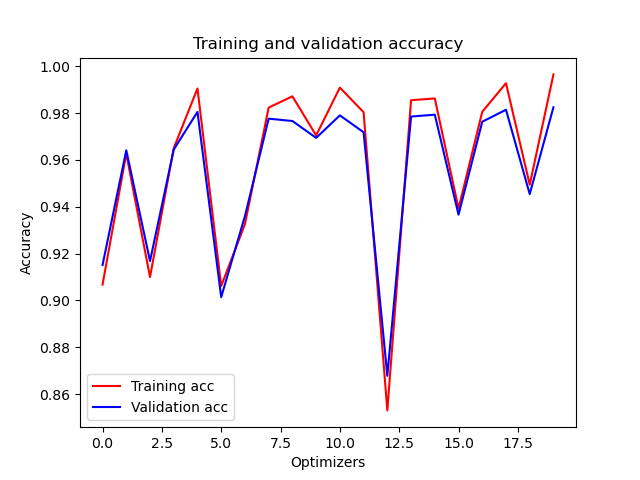


Рисунок 4 - Точности оптимизаторов

По рис. 3 и рис. 4, а так же по выводу программы <tensorflow.python.keras.optimizer\_v2.nadam.Nadam object at 0x000001D4A4613CF8> 19 (оптимизатор и его номер в массиве оптимизаторов) видно, что наибольшей точностью обладают следующие оптимизаторы:

* RMSprop(learning\_rate=0.001, rho=0.9)
* Adagrad(learning\_rate=0.01)
* Adam(learning\_rate=0.001, beta\_1=0.9, beta\_2=0.999, amsgrad=False)
* Adamax(learning\_rate=0.002, beta\_1=0.9, beta\_2=0.999)
* Nadam(learning\_rate=0.002, beta\_1=0.9, beta\_2=0.999)
* Nadam(learning\_rate=0.002, beta\_1=0.999, beta\_2=0.999)

Среди них оптимизатором, с помощью которого была построена самая точная модель является последний. Графики точности и потерь при обучении с использованием последнего оптимизатора представлены на рис. 5 и рис. 6. соответственно.

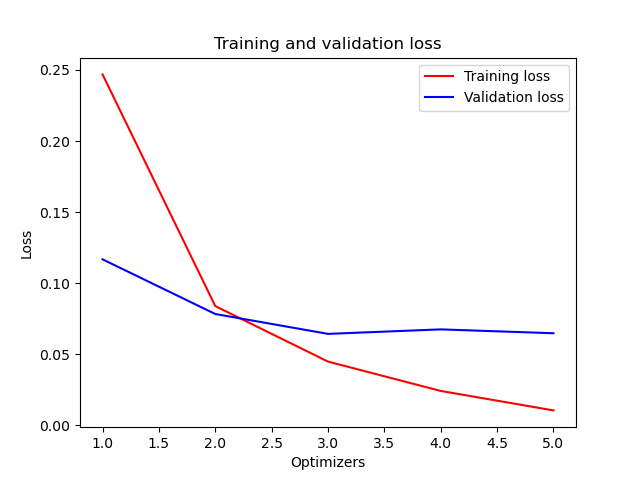


Рисунок 5 - Потери при использовании Nadam

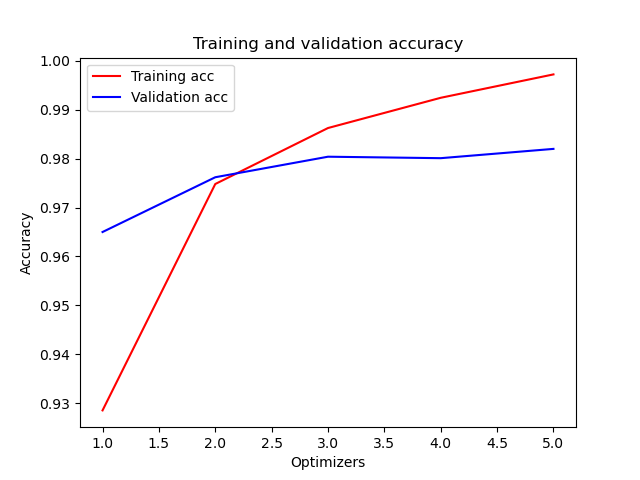


Рисунок 6 - Точность при использовании Nadam

1. Была написана функция для считывания картинки, содержащей цифру. Код представлен в приложении А. Тестовые рисунки представлены на рис. 7.



Рисунок 7 - Тестовые рисунки

Вывод результата работы нейросети на этих картинках представлен на рис. 8.



Рисунок 8 - Вывод программы

По выводу видно, что несмотря на достаточно большую точность, нейросеть может ошибиться при определении цифры на картинке. Это может быть связано с тем что стиль написания цифры на картинке может сильно отличаться от стиля написания цифр на тренировочных картинках.

**Вывод.**

В ходе выполнения данной лабораторной работы было изучено представление и обработка графических данных, был выявлен лучший оптимизатор для построения модели искусственной нейронной сети, распознающей рукописные цифры, была построена и протестирована на пользовательских изображениях модель.

**Список использованных источников**

1. Patrice Y. Simard. Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis / Dave Steinkraus, John C. Platt // Seventh International Conference on Document Analysis and Recognition. – 2003.

**ПРИЛОЖЕНИЕ А**

import tensorflow as tf

import matplotlib.pyplot as plt

import numpy as np

from tensorflow.keras import optimizers

from tensorflow.keras.utils import to\_categorical

from tensorflow.keras.layers import Dense, Activation, Flatten

from tensorflow.keras.models import Sequential

from PIL import Image

mnist = tf.keras.datasets.mnist

(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)

optimizers\_list = [optimizers.SGD(learning\_rate=0.01, momentum=0.0, nesterov=False),

optimizers.SGD(learning\_rate=0.1, momentum=0.0, nesterov=False),

optimizers.SGD(learning\_rate=0.01, momentum=0.1, nesterov=True),

optimizers.SGD(learning\_rate=0.1, momentum=0.1, nesterov=True),

optimizers.RMSprop(learning\_rate=0.001, rho=0.9),

optimizers.RMSprop(learning\_rate=0.1, rho=0.9),

optimizers.Adagrad(learning\_rate=0.01), optimizers.Adagrad(learning\_rate=0.1),

optimizers.Adadelta(learning\_rate=1.0, rho=0.95),

optimizers.Adadelta(learning\_rate=0.5, rho=0.5),

optimizers.Adam(learning\_rate=0.001, beta\_1=0.9, beta\_2=0.999, amsgrad=False),

optimizers.Adam(learning\_rate=0.01, beta\_1=0.99, beta\_2=0.99, amsgrad=False),

optimizers.Adam(learning\_rate=0.1, beta\_1=0.999, beta\_2=0.9, amsgrad=True),

optimizers.Adam(learning\_rate=0.001, beta\_1=0.999, beta\_2=0.999, amsgrad=True),

optimizers.Adamax(learning\_rate=0.002, beta\_1=0.9, beta\_2=0.999),

optimizers.Adamax(learning\_rate=0.1, beta\_1=0.999, beta\_2=0.999),

optimizers.Adamax(learning\_rate=0.002, beta\_1=0.999, beta\_2=0.999),

optimizers.Nadam(learning\_rate=0.002, beta\_1=0.9, beta\_2=0.999),

optimizers.Nadam(learning\_rate=0.1, beta\_1=0.999, beta\_2=0.999),

optimizers.Nadam(learning\_rate=0.002, beta\_1=0.999, beta\_2=0.999)]

def load\_image(path):

image = Image.open(path).convert('L')

image = 255 - np.array(image) # т.к. черная цифра на белом фоне

image = image/255

return np.expand\_dims(image, axis=0)

def build\_model(optimizer):

model = Sequential()

model.add(Flatten())

model.add(Dense(800, activation='relu'))

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

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

print(model.optimizer)

h = model.fit(train\_images, train\_labels, epochs=5, batch\_size=128)

test\_loss, test\_acc = model.evaluate(test\_images, test\_labels)

train\_acc = h.history['accuracy'][-1]

train\_loss = h.history['loss'][-1]

return train\_acc, train\_loss, test\_acc, test\_loss

def test\_optimizers():

best\_optimizer = optimizers.SGD(learning\_rate=0.01, momentum=0.0, nesterov=False)

index = 0

min\_test\_accuracy = 0

train\_acc\_list = []

test\_acc\_list = []

train\_loss\_list = []

test\_loss\_list = []

for optimizer in optimizers\_list:

train\_acc, train\_loss, test\_acc, test\_loss = build\_model(optimizer)

train\_acc\_list.append(train\_acc)

train\_loss\_list.append(train\_loss)

test\_acc\_list.append(test\_acc)

test\_loss\_list.append(test\_loss)

if test\_acc > min\_test\_accuracy:

best\_optimizer = optimizer

index = optimizers\_list.index(optimizer)

plt.plot(range(len(optimizers\_list)), train\_loss\_list, 'r', label='Training loss')

plt.plot(range(len(optimizers\_list)), test\_loss\_list, 'b', label='Validation loss')

plt.title('Training and validation loss')

plt.xlabel('Optimizers')

plt.ylabel('Loss')

plt.legend()

plt.show()

plt.clf()

plt.plot(range(len(optimizers\_list)), train\_acc\_list, 'r', label='Training acc')

plt.plot(range(len(optimizers\_list)), test\_acc\_list, 'b', label='Validation acc')

plt.title('Training and validation accuracy')

plt.xlabel('Optimizers')

plt.ylabel('Accuracy')

plt.legend()

plt.show()

return best\_optimizer, index

optimizer, index = test\_optimizers()

print(optimizer, index)

model = Sequential()

model.add(Flatten())

model.add(Dense(800, activation='relu'))

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

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

h = model.fit(train\_images, train\_labels, epochs=5, batch\_size=128, validation\_data=(test\_images, test\_labels))

loss = h.history['loss']

val\_loss = h.history['val\_loss']

acc = h.history['accuracy']

val\_acc = h.history['val\_accuracy']

print(val\_acc[-1])

image = load\_image('1.png')

print('1:', model.predict\_classes(image))

image = load\_image('7.png')

print('7:', model.predict\_classes(image))

image = load\_image('6.png')

print('6:', model.predict\_classes(image))

image = load\_image('0.png')

print('0:', model.predict\_classes(image))

epochs = range(1, len(loss) + 1)

plt.plot(epochs, loss, 'r', label='Training loss')

plt.plot(epochs, val\_loss, 'b', label='Validation loss')

plt.title('Training and validation loss')

plt.xlabel('Optimizers')

plt.ylabel('Loss')

plt.legend()

plt.show()

plt.clf()

plt.plot(epochs, acc, 'r', label='Training acc')

plt.plot(epochs, val\_acc, 'b', label='Validation acc')

plt.title('Training and validation accuracy')

plt.xlabel('Optimizers')

plt.ylabel('Accuracy')

plt.legend()

plt.show()