

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. 7381

Алясова А.Н.

Преподаватель

Жукова Н.А.

Санкт-Петербург

2020

Цель работы.

Реализовать классификацию черно-белых изображений рукописных цифр (28x28) по 10 категориям (от 0 до 9). Набор данных содержит 60 000 изображений для обучения и 10 000 изображений для тестирования.

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

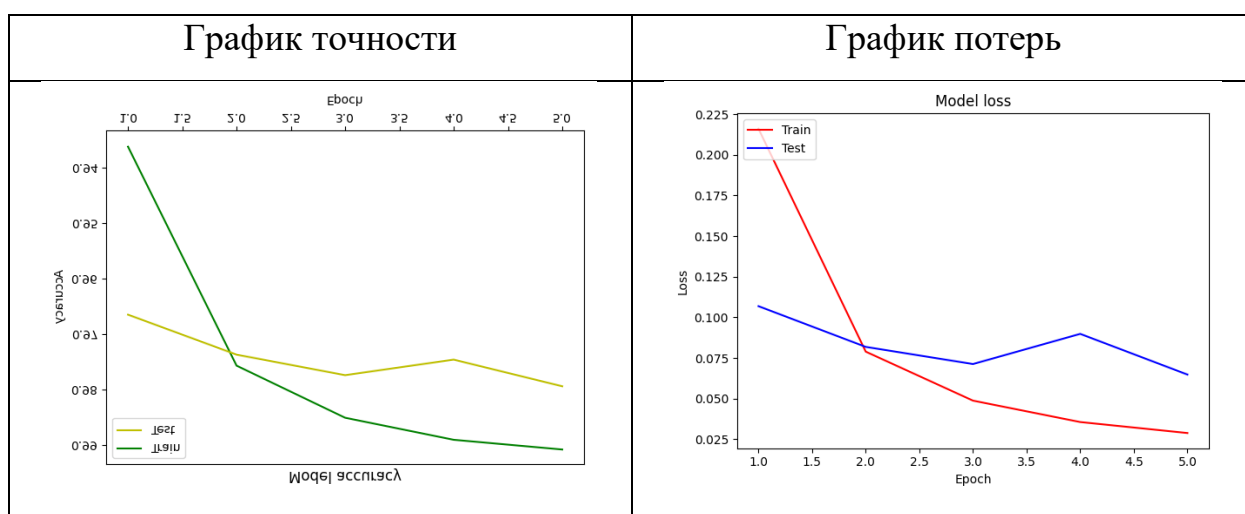
Требования.

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

Ход работы.

В ходе работы была создана и обучена модель искусственной нейронной сети в соответствии с условиями (код представлен в приложении).

Результатом поиска наилучшей архитектуры, при которой точность классификации сети более 95%, оказалась модель с 2мя скрытыми слоями на 512 и 216 нейронов. На самом деле добавление слоев не сильно влияет на результат, но при таком строении потери наименьшие. Эти результаты достигнуты при исходном количестве эпох = 5, однако в процессе тестирования я выяснила, что точнее результаты при количестве эпох = 20.



Результаты обучения модели при использовании различных оптимизаторов, а также их параметров представлены в таблицах ниже. Чтобы определить какой оптимизатор и с какими параметрами подходит нам больше всего, я рассмотрела каждый оптимизатор с различными параметрами. Далее из них выбирала наилучший и фиксировала, что менялось.

Название оптимизатора	Его параметры	Точность
Adadelata	'learning_rate': 0.001, 'decay': 0.0, 'rho': 0.95, 'epsilon': 1e-07}:	0.8683
	'learning_rate': 0.001, 'decay': 0.0, 'rho': 0.5, 'epsilon': 1e-07}:	0.8038
	'learning_rate': 0.01, 'decay': 0.0, 'rho': 0.95, 'epsilon': 1e-07}:	0.9411

С увеличением 'learning_rate' увеличивается точность, поэтому наилучший вариант это:

```
{'name': 'Adadelta', 'learning_rate': 0.01, 'decay': 0.0, 'rho': 0.95, 'epsilon': 1e-07}: 0.9411
```

Название оптимизатора	Его параметры	Точность
Adagrad	'learning_rate': 0.001, 'decay': 0.0, 'initial_accumulator_value': 0.1, 'epsilon': 1e-07}	0.9285
	'learning_rate': 0.01, 'decay': 0.0, 'initial_accumulator_value': 0.1, 'epsilon': 1e-07}	0.9736

С увеличением 'learning_rate' увеличивается точность, поэтому наилучший вариант это:

```
{'name': 'Adagrad', 'learning_rate': 0.01, 'decay': 0.0, 'initial_accumulator_value': 0.1, 'epsilon': 1e-07}: 0.9736
```

Название оптимизатора	Его параметры	Точность
Adam	'learning_rate': 0.001, 'decay': 0.0, 'beta_1': 0.9, 'beta_2': 0.999, 'epsilon': 1e-07, 'amsgrad': False}	0.9838
	'learning_rate': 0.01, 'decay': 0.0, 'beta_1': 0.9, 'beta_2': 0.999, 'epsilon': 1e-07, 'amsgrad': False}:	0.9749
	{'name': 'Adamax', 'learning_rate': 0.001, 'decay': 0.0, 'beta_1': 0.9, 'beta_2': 0.999, 'epsilon': 1e-07}:	0.9825

С увеличением 'learning_rate' увеличивается потери, поэтому наилучший вариант это:

```
{'name': 'Adam', 'learning_rate': 0.001, 'decay': 0.0, 'beta_1': 0.9, 'beta_2': 0.999, 'epsilon': 1e-07, 'amsgrad': False}: 0.9838
```

Название оптимизатора	Его параметры	Точность
RMSprop	'learning_rate': 0.001, 'decay': 0.0, 'rho': 0.9, 'momentum': 0.0, 'epsilon': 1e-07, 'centered': False}	0.9828
	'learning_rate': 0.01, 'decay': 0.0, 'rho': 0.9, 'momentum': 0.0, 'epsilon': 1e-07, 'centered': False}	0.9644

	'learning_rate': 0.001, 'decay': 0.0, 'rho': 0.5, 'momentum': 0.0, 'epsilon': 1e-07, 'centered': False}	0.9823
	'learning_rate': 0.001, 'decay': 0.0, 'rho': 0.9, 'momentum': 0.9, 'epsilon': 1e-07, 'centered': False}:	0.9768

С увеличением параметров 'learning_rate' и 'momentum' увеличивается потери, при изменении параметра 'rho' ничего не меняется, поэтому наилучший вариант это:

{'name': 'RMSprop', 'learning_rate': 0.001, 'decay': 0.0, 'rho': 0.9, 'momentum': 0.0, 'epsilon': 1e-07, 'centered': False}: 0.9828

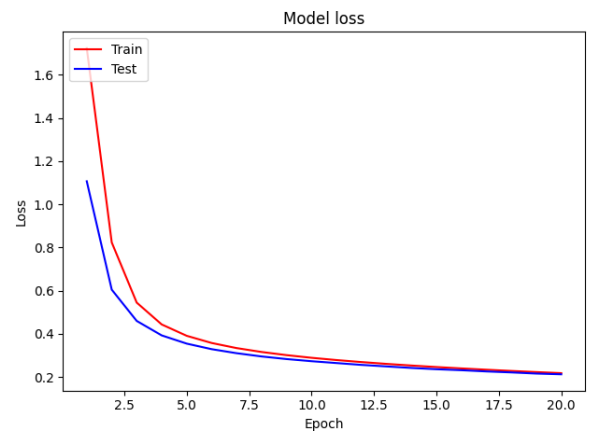
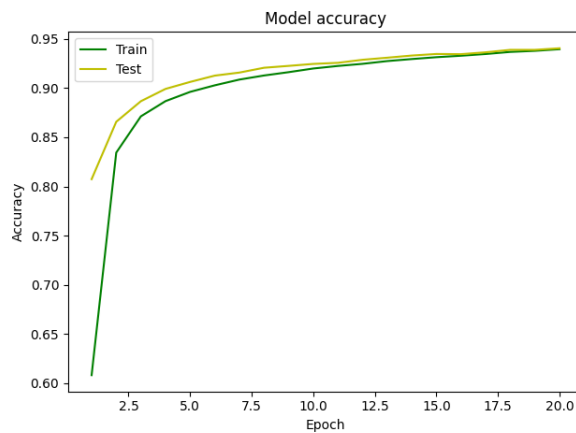
Название оптимизатора	Его параметры	Точность
SGD	'learning_rate': 0.01, 'decay': 0.0, 'momentum': 0.0, 'nesterov': False}	0.9568
	'learning_rate': 0.001, 'decay': 0.0, 'momentum': 0.0, 'nesterov': False}	0.9014
	'learning_rate': 0.01, 'decay': 0.0, 'momentum': 0.9, 'nesterov': False}:	0.9799

С увеличением 'learning_rate' и 'momentum' увеличивается точность, поэтому наилучший вариант это:

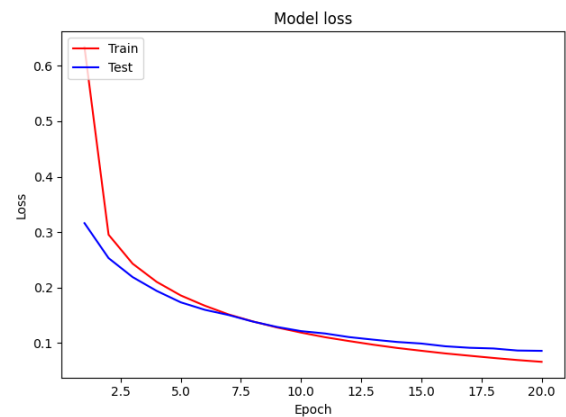
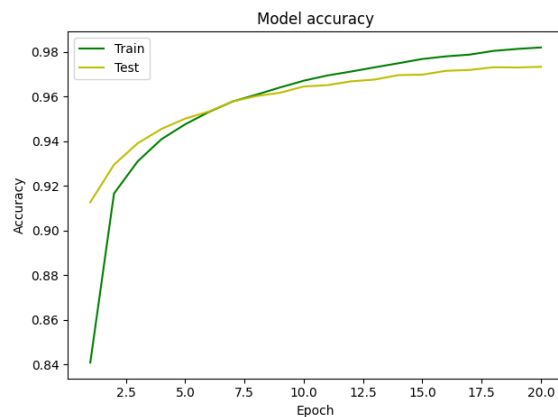
{'name': 'SGD', 'learning_rate': 0.01, 'decay': 0.0, 'momentum': 0.9, 'nesterov': False}: 0.9799

Теперь из наилучших вариаций оптимизаторов выберем самый подходящий:

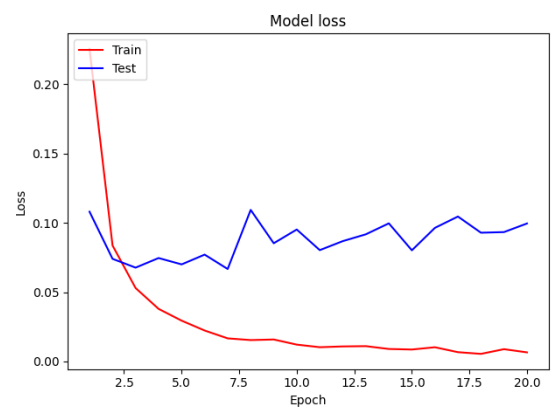
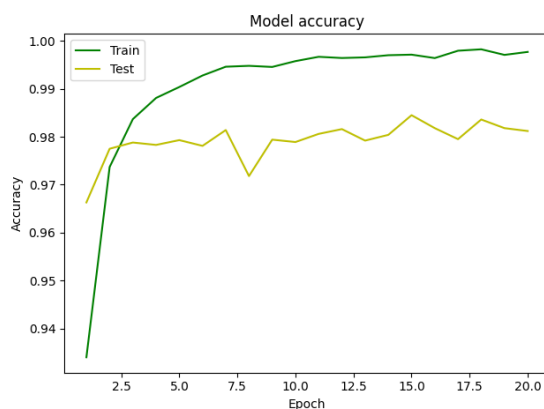
```
{'name': 'Adadelata', 'learning_rate': 0.01, 'decay': 0.0, 'rho': 0.95, 'epsilon': 1e-07}: 0.9404
```



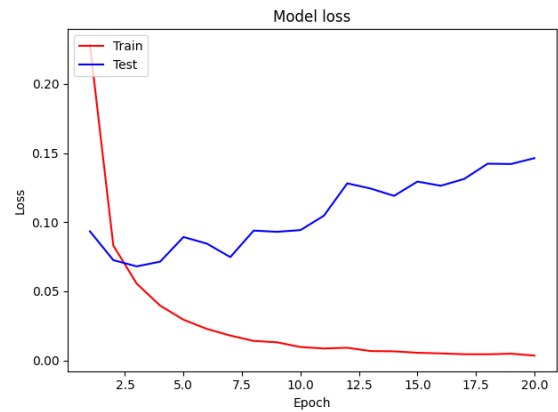
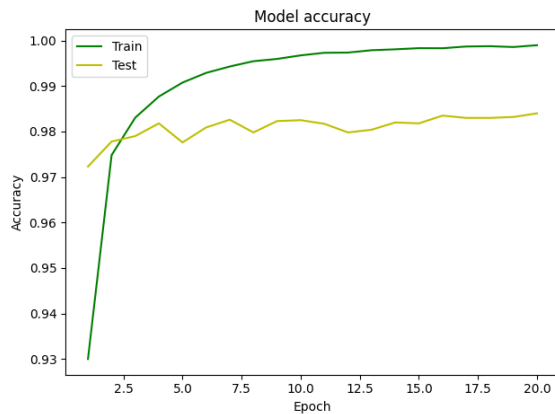
```
{'name': 'Adagrad', 'learning_rate': 0.01, 'decay': 0.0, 'initial_accumulator_value': 0.1, 'epsilon': 1e-07}: 0.9733
```



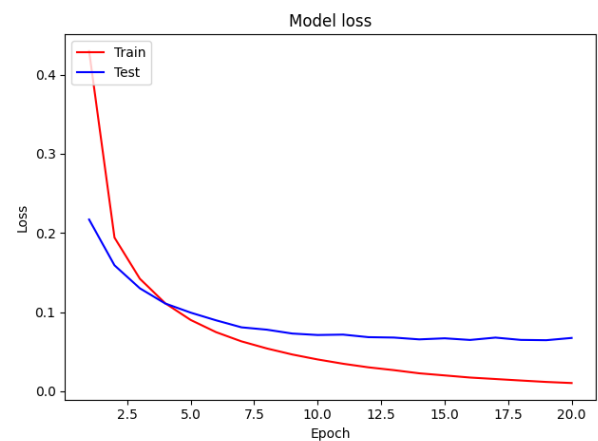
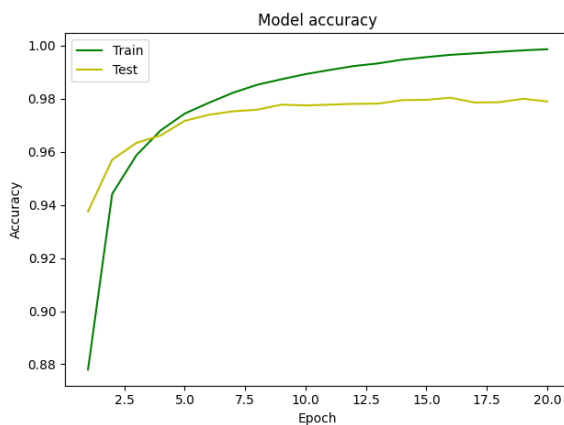
```
{'name': 'Adam', 'learning_rate': 0.001, 'decay': 0.0, 'beta_1': 0.9, 'beta_2': 0.999, 'epsilon': 1e-07, 'amsgrad': False}: 0.9812
```



```
{'name': 'RMSprop', 'learning_rate': 0.001, 'decay': 0.0, 'rho': 0.9, 'momentum': 0.0, 'epsilon': 1e-07, 'centered': False}: 0.984
```



```
{'name': 'SGD', 'learning_rate': 0.01, 'decay': 0.0, 'momentum': 0.9, 'nesterov': False}: 0.9789
```



Сравнивая полученные цифры и смотря на графики, можно увидеть, что хуже всех себя оптимизатор 'Adadelata', так как у нее самая маленькая точность. Лучше всех справился оптимизатор 'RMSprop'.

Выводы.

В ходе выполнения данной работы была изучена задача распознавания рукописных цифр и исследовано влияние различных оптимизаторов на обучение моделей. Также была произведена работа по работе и обработке изображений.

ПРИЛОЖЕНИЕ

Исходный код

```
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np

from keras.utils import to_categorical
from tensorflow.keras.layers import Dense, Activation, Flatten
from tensorflow.keras.models import Sequential
from PIL import Image
from tensorflow.keras import optimizers

NUM_EPOCHS = 20

optimizers = [optimizers.Adadelta(learning_rate=0.01),
               optimizers.Adagrad(learning_rate=0.01),
               optimizers.Adam(),
               optimizers.RMSprop(),
               optimizers.Ftrl(),
               optimizers.SGD(momentum=0.9)]

#####
#####
def load_data():
    #MNIST - набор данных
    mnist = tf.keras.datasets.mnist
    (train_images, train_labels), (test_images, test_labels) =
mnist.load_data()

    #преобразование изображений в массив чисел из интервала [0, 1]
    train_images = train_images / 255.0
    test_images = test_images / 255.0

    #кодирование меток категорий
    train_labels = to_categorical(train_labels)
    test_labels = to_categorical(test_labels)

    return train_images, train_labels, test_images, test_labels
#####
#####
#####
#####
# Функция для загрузки изображения не из датасета
```

```

def transform_Image(filename):
    img = Image.open(filename).convert('L')
    img = img.resize((28, 28))
    img = np.array(img)
    img = 1 - img
    img = img / 255.0
    img = np.expand_dims(img, axis=0)
    return img

#####

#####

# Задание базовой архитектуры сети
def build_model(opt):
    model = Sequential()
    model.add(Flatten())
    model.add(Dense(512, activation='relu'))
    model.add(Dense(256, activation='relu'))
    model.add(Dense(10, activation='softmax'))

    #задание ф-ции потерь, оптимизатора, метрики
    model.compile(optimizer=opt, loss='categorical_crossentropy',
metrics=['accuracy'])

    return model

#####

#####

def create_graphics(history):
    # графики потерь
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    epochs = range(1, len(loss) + 1)
    plt.plot(epochs, loss, 'r')
    plt.plot(epochs, val_loss, 'b')
    plt.title('Model loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Test'], loc='upper left')
    plt.show()

    # графики точности

```

```

    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    plt.plot(epochs, acc, 'g')
    plt.plot(epochs, val_acc, 'y')
    plt.title('Model accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Test'], loc='upper left')
    plt.show()
#####

#####

#####
def train_model():
    model = build_model('adam')
    # обучение сети
    history = model.fit(train_images, train_labels, epochs=NUM_EPOCHS,
batch_size=128, validation_data=(test_images, test_labels))
    create_graphics(history)

    # проверка, как модель распознает контрольный набор
    test_loss, test_acc = model.evaluate(test_images, test_labels)
    print('test_acc:', test_acc)

    return model
#####

#####

#####
def test (opt):
    opt_config = opt.get_config()
    print ("Researching with optimizer ")
    print(opt_config)

    model = build_model(opt)
    history = model.fit(train_images, train_labels, epochs=NUM_EPOCHS,
batch_size=128, validation_data=(test_images, test_labels))

    test_loss, test_acc = model.evaluate(test_images, test_labels)

    create_graphics(history)

    result["%s" % (opt_config)] = test_acc

```

```

    print(test_loss)
    return model
#####

#####

#####
#####

if __name__ == '__main__':
    train_images, train_labels, test_images, test_labels = load_data()

    print("Введите число:")
    print("0 - Запустить модель с наилучшими параметрами")
    print("1 - Запустить все")
    num = input()
    if num == '0':
        model = train_model()
        image = transform_Image('77.png')
        predictions = model.predict(image)
        print(np.argmax(predictions))

    if num == '1':
        result = dict()
        for opt in optimizers:
            test(opt)

    #результаты тестирования
    for res in result:
        print("%s: %s" % (res, result[res]))

```