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

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Искусственные нейронные сети»

Тема: «Распознавание рукописных символов»

Студент гр. 7381 _____ Тарасенко Е.А.

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;
- Создать модель;
- Настроить параметры обучения;
- Написать функцию, позволяющую загружать изображение пользователем 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.

Сначала необходимо добиться максимальной точности модели на тестовых данных. Для этого исследуем то, как на результатах обучения сети сказывается изменение количества нейронных слоев и самих нейронов на этих слоях. Также можно увеличить количество эпох. Результаты модели, архитектура которой не подвергалась изменениям (такая же, как в листинге работы; потери и точность для 5, 7 и 10 эпох соответственно):

```
test_loss: 0.073920648831781     test_acc: 0.9778
```

```
test_loss: 0.06518236029858235   test_acc: 0.9797
test_loss: 0.0695036103330378    test_acc: 0.9784
```

Изменение количества эпох не сильно повлияло на результаты обучения сети, поэтому далее количество эпох будет равно 5. Добавим в модель слой с 128 нейронами, затем изменим это число на 256:

```
test_loss: 0.07757177699232197   test_acc: 0.9797
test_loss: 0.0857729448779981    test_acc: 0.9787
```

Добавление еще одного слоя не сильно сказалось на результатах. Дальнейшие попытки улучшить (и так удовлетворяющий требованиям результат) ни к чему не привели – точность продолжала приближаться к значению в 98%, однако не становилась больше этого числа. Т. о., было решено оставить архитектуру сети такой, какой она была представлена в условии лабораторной работы.

Далее необходимо исследовать влияние различных оптимизаторов на результаты обучения. Результаты приведены на рис. 1 – 7.

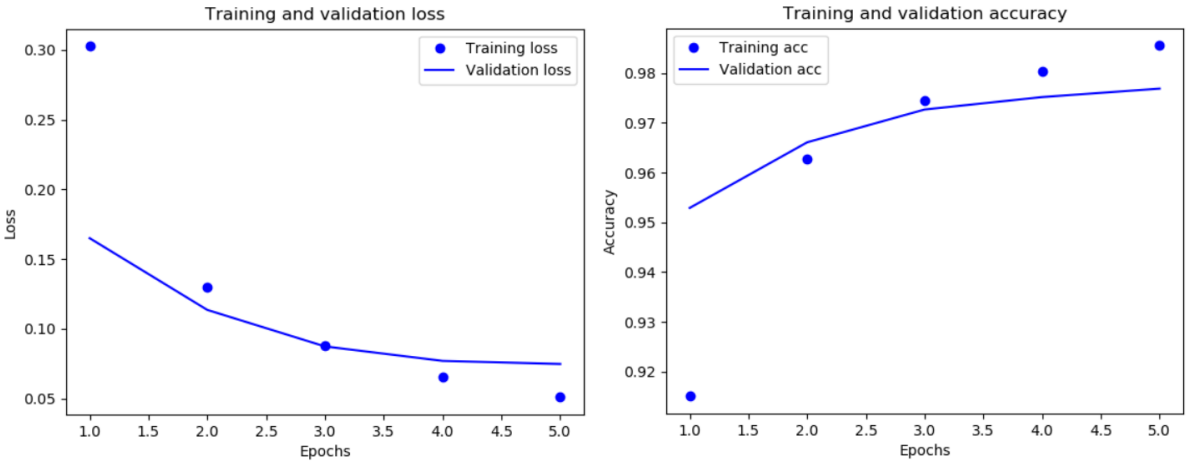

Рис. 1 – Графики ошибки и точности модели с оптимизатором Adam (test_loss: 0.07483100843783468 test_acc: 0.9769)

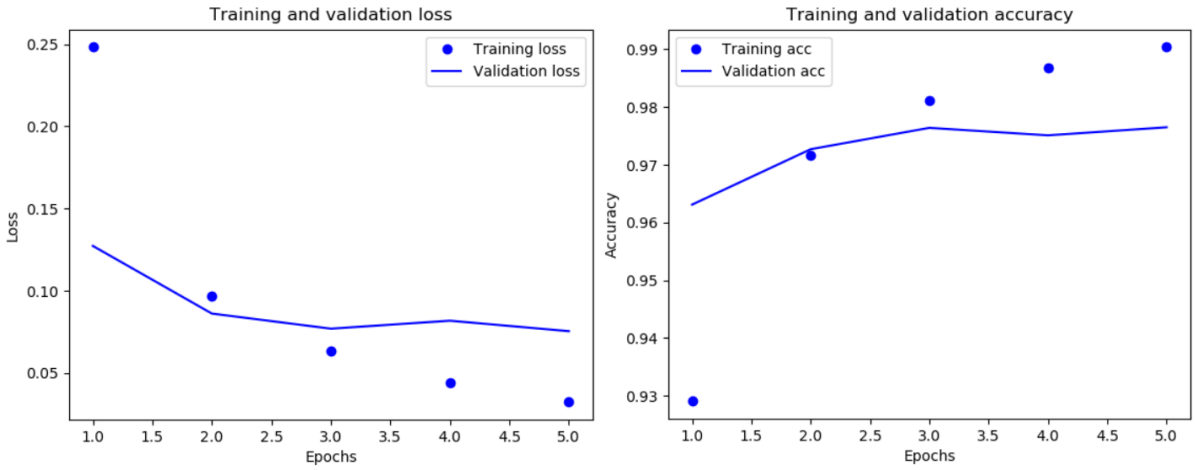

Рис. 2 – Графики ошибки и точности модели с оптимизатором Nadam

(test_loss: 0.0754362354881363 test_acc: 0.9765)

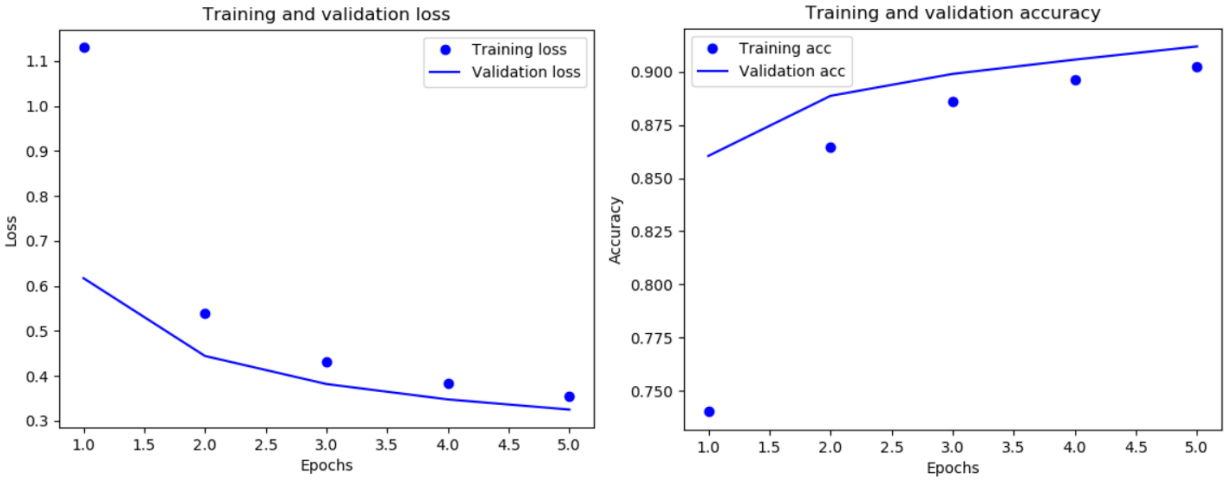

Рис. 3 – Графики ошибки и точности модели с оптимизатором SGD

(test_loss: 0.3250110074877739 test_acc: 0.9118)

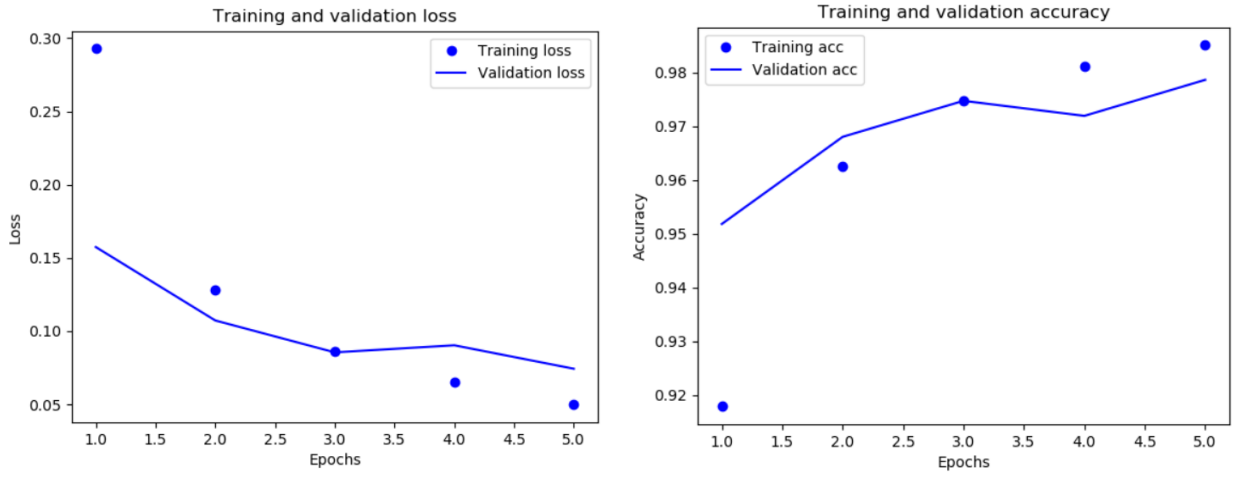

Рис. 4 – Графики ошибки и точности модели с оптимизатором RMSprop

(test_loss: 0.07434772649402731 test_acc: 0.9786)

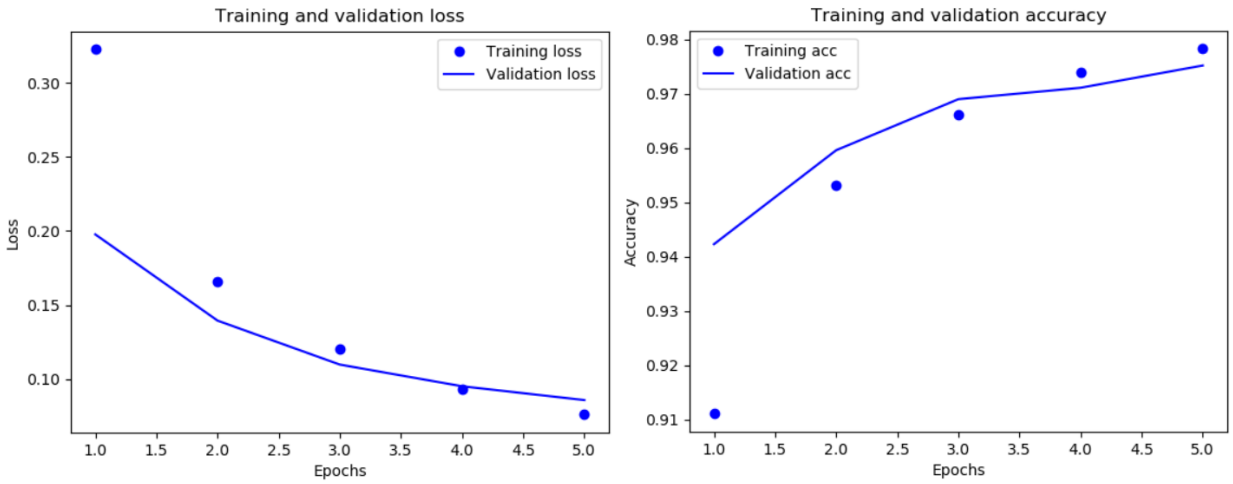

Рис. 5 – Графики ошибки и точности модели с оптимизатором Adamax

(test_loss: 0.08578065742477775 test_acc: 0.9752)

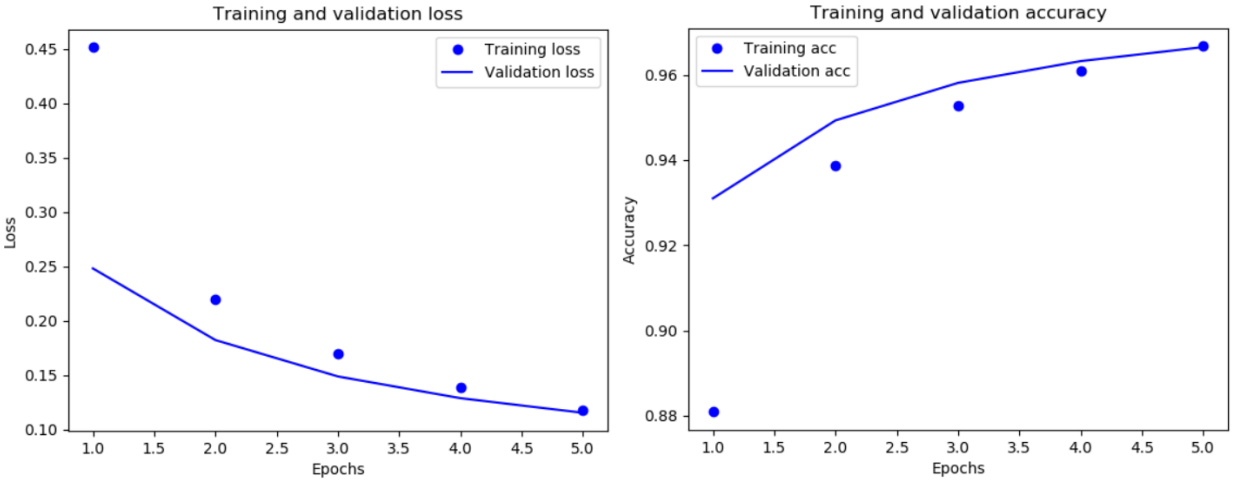

Рис. 6 – Графики ошибки и точности модели с оптимизатором Adadelta

(test_loss: 0.11520541351810097 test_acc: 0.9665)

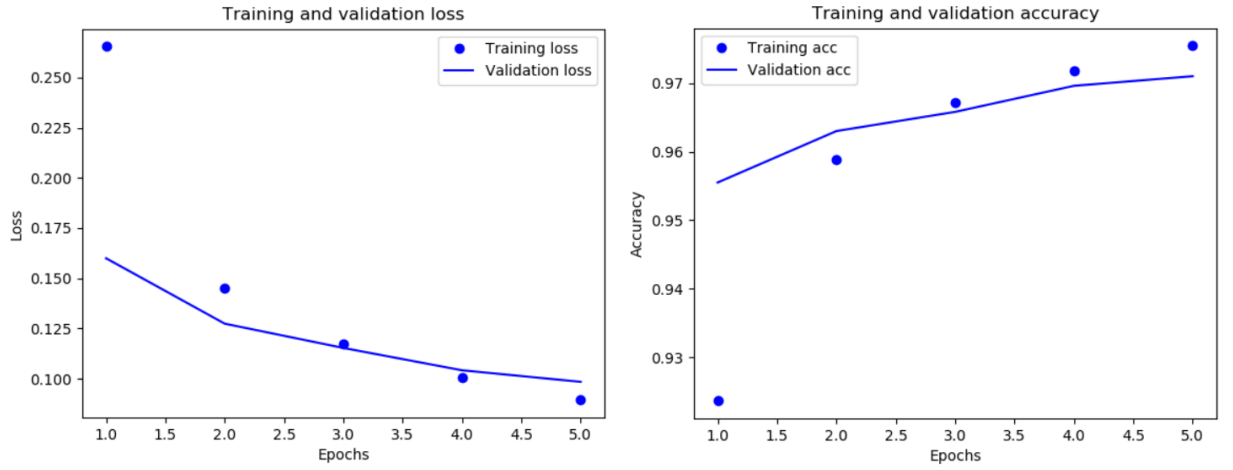

Рис. 7 – Графики ошибки и точности модели с оптимизатором Adagrad

(test_loss: 0.09840184226594866 test_acc: 0.971)

В данном случае хуже всех себя показал оптимизатор SGD. Adadelta – немного хуже остальных (Adam, Nadam, RMSprop, Adamax и Adagrad), которые давали практически одинаковые результаты. Следовательно, после проделанных испытаний, решено было оставить оптимизатор Adam. Теперь необходимо обучить модель с различными значениями learning_rate (скорости обучения). Результаты приведены на рис. 8 – 10.

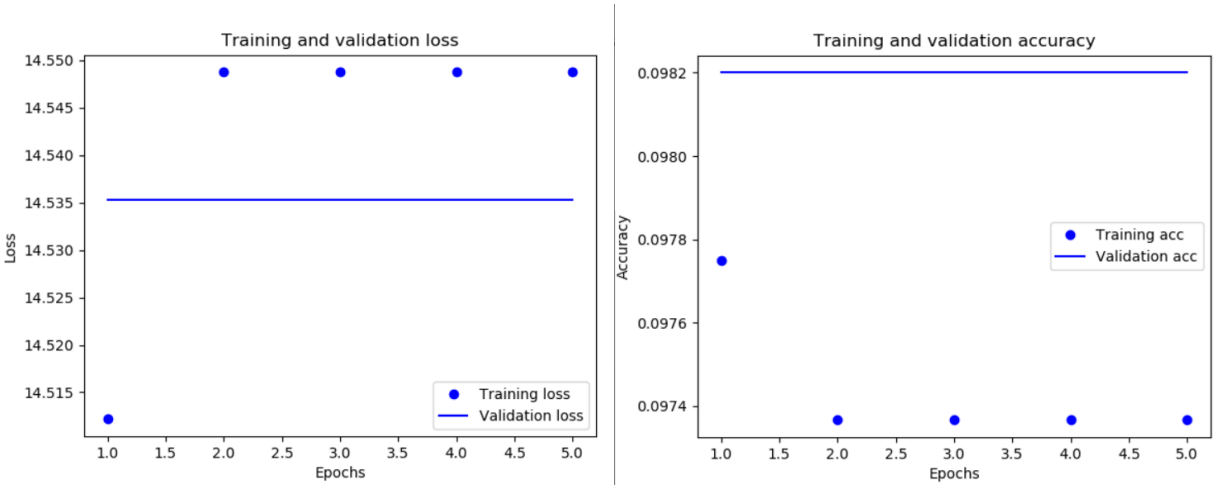

Рис. 7 – Графики ошибки и точности модели с learning_rate = 0.1
(test_loss: 14.535298265075683 test_acc: 0.0982)

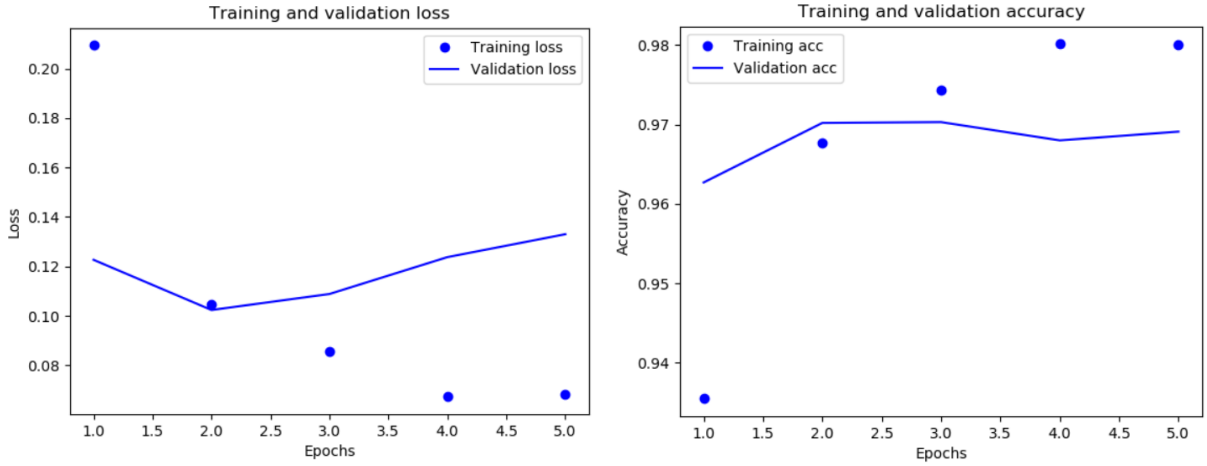

Рис. 8 – Графики ошибки и точности модели с learning_rate = 0.01
(test_loss: 0.13301685354665388 test_acc: 0.9691)

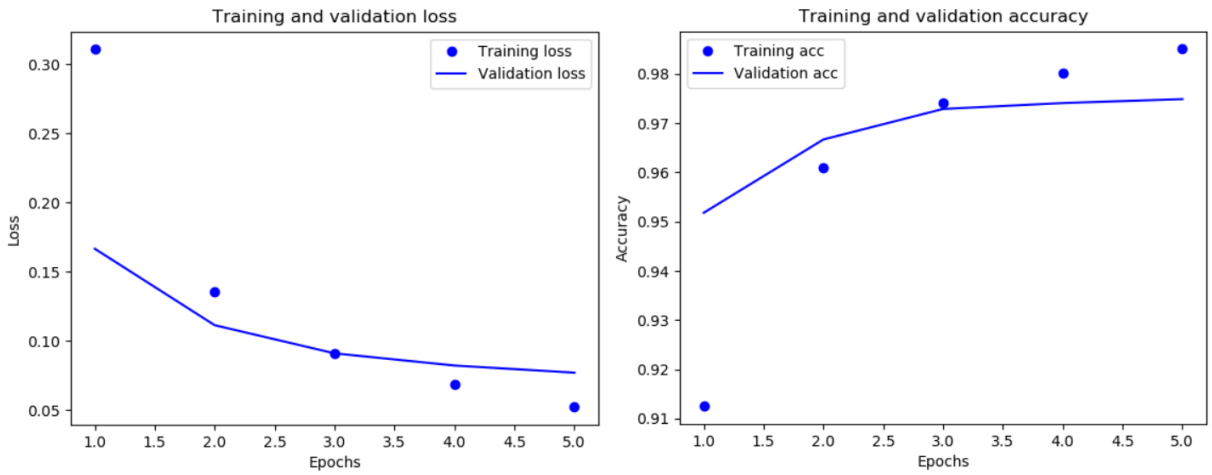

Рис. 9 – Графики ошибки и точности модели с learning_rate = 0.001

(test_loss: 0.07698409154331312 test_acc: 0.9749)

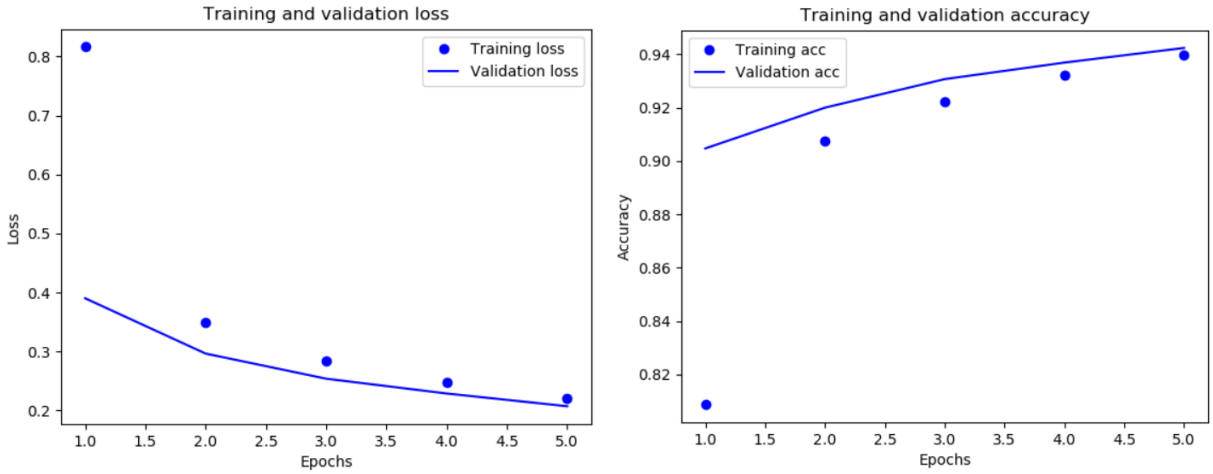

Рис. 10 – Графики ошибки и точности модели с learning_rate = 0.0001

(test_loss: 0.20726175293326377 test_acc: 0.9424)

Очень сильно отличаются результаты обучения для скорости, равной 0.1, от остальных (очень низкая точность и очень большие потери). Точность возрастает с уменьшением learning_rate до значения в 0.001, после чего начал наблюдаться спад. Так для learning_rate = 0.00001, например, (этот результат уже не проиллюстрирован на рисунках) результаты обучения оказались следующими:

```
test_loss: 0.5710290570259094   test_acc: 0.8798
```

Т. о., лучше выставить скорость обучения на значение 0.001.

Теперь напишем функции тестирования и считывания пользовательских изображений, после чего протестируем сеть на приведенных на рис. 11 примерах. Результат работы программы представлен на рис. 12.

```
def user_image_test(image_file_path):
    image_file_path = './' + image_file_path
    image = numpy.array(PIL.Image.open(image_file_path).convert('L').resize((28,
28)))
    image = (255 - image) / 255
    return numpy.expand_dims(image, axis=0)


def testing(curr_model, image):
    print(numpy.argmax(curr_model.predict_on_batch(image)))
```

Рис. 11 – Примеры пользовательских изображений (все в формате png)

```
Educate (Press ENTER) / Testing (print an image file path) OR print `q` for quit?
1.png
1
Educate (Press ENTER) / Testing (print an image file path) OR print `q` for quit?
2.png
2
Educate (Press ENTER) / Testing (print an image file path) OR print `q` for quit?
3.png
3
Educate (Press ENTER) / Testing (print an image file path) OR print `q` for quit?
8.png
8
```

Рис. 12 – Результат работы программы

Выводы.

В ходе выполнения данной лабораторной работы были получены некоторые навыки по построению сетей для классификации изображений. В рамках исследования была создана и обучена модель, распознающая рукописные символы (цифры от 0 до 9). Также программой поддерживается возможность тестирования сети на пользовательских изображениях.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД

```
import tensorflow, numpy, PIL.Image
import matplotlib.pyplot as plt


def user_image_test(image_file_path):
    image_file_path = './' + image_file_path
    image = numpy.array(PIL.Image.open(image_file_path).convert('L').resize((28,
28)))
    # белым по черному или черным по белому?))) + нормализация
    image = (255 - image) / 255
    return numpy.expand_dims(image, axis=0)


def testing(curr_model, image):
    print(numpy.argmax(curr_model.predict_on_batch(image)))


def create_model():
    # загрузка тренировочных и проверочных данных
    mnist = tensorflow.keras.datasets.mnist
    (train_images, train_labels),(test_images, test_labels) = mnist.load_data()

    # проверка корректности загрузки
    plt.imshow(train_images[0], cmap=plt.cm.binary)
    plt.show()
    print(train_labels[0])

    # нормализация входных данных
    train_images = train_images / 255.0
    test_images = test_images / 255.0

    # перевод правильных ответов в категориальные вектора
    train_labels = tensorflow.keras.utils.to_categorical(train_labels)
    test_labels = tensorflow.keras.utils.to_categorical(test_labels)

    # архитектура сети
    model = tensorflow.keras.models.Sequential()

    model.add(tensorflow.keras.layers.Flatten(input_shape=(28, 28)))
    model.add(tensorflow.keras.layers.Dense(256, activation='relu'))
    model.add(tensorflow.keras.layers.Dense(100, activation='relu'))
    model.add(tensorflow.keras.layers.Dense(10, activation='softmax'))

    model.compile(optimizer=tensorflow.keras.optimizers.Adam(),
loss='categorical_crossentropy', metrics=['accuracy'])

    # обучение сети
    H = model.fit(train_images, train_labels, epochs=5, batch_size=128,
validation_data=(test_images, test_labels),
                  verbose=0)

    # получение ошибки и точности в процессе обучения
    loss = H.history['loss']
    val_loss = H.history['val_loss']
    acc = H.history['acc']
```

```
    val_acc = H.history['val_acc']
    epochs = range(1, len(loss) + 1)

    # построение графика ошибки
    plt.plot(epochs, loss, 'bo', label='Training loss')
    plt.plot(epochs, val_loss, 'b', label='Validation loss')
    plt.title('Training and validation loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.show()

    # построение графика точности
    plt.clf()
    plt.plot(epochs, acc, 'bo', label='Training acc')
    plt.plot(epochs, val_acc, 'b', label='Validation acc')
    plt.title('Training and validation accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.show()

    # проверка распознавания контрольного набора
    test_loss, test_acc = model.evaluate(test_images, test_labels)
    print('test_loss:', test_loss)
    print('test_acc:', test_acc)

    return model


model = create_model()
while True:
    print('Educate (Press ENTER) / Testing (print an image file path) OR print `q`
for quit?')
    req = input()
    if req == 'q': break
    elif req == '': model = create_model()
    else: testing(model, user_image_test(req))
```