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

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Искусственные нейронные сети»

Тема: Распознавание рукописных символов

Студент гр.8382 _____ Терехов А.Е.

Преподаватель _____ Жангриров Т.Р.

Санкт-Петербург

2021

Цель работы

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

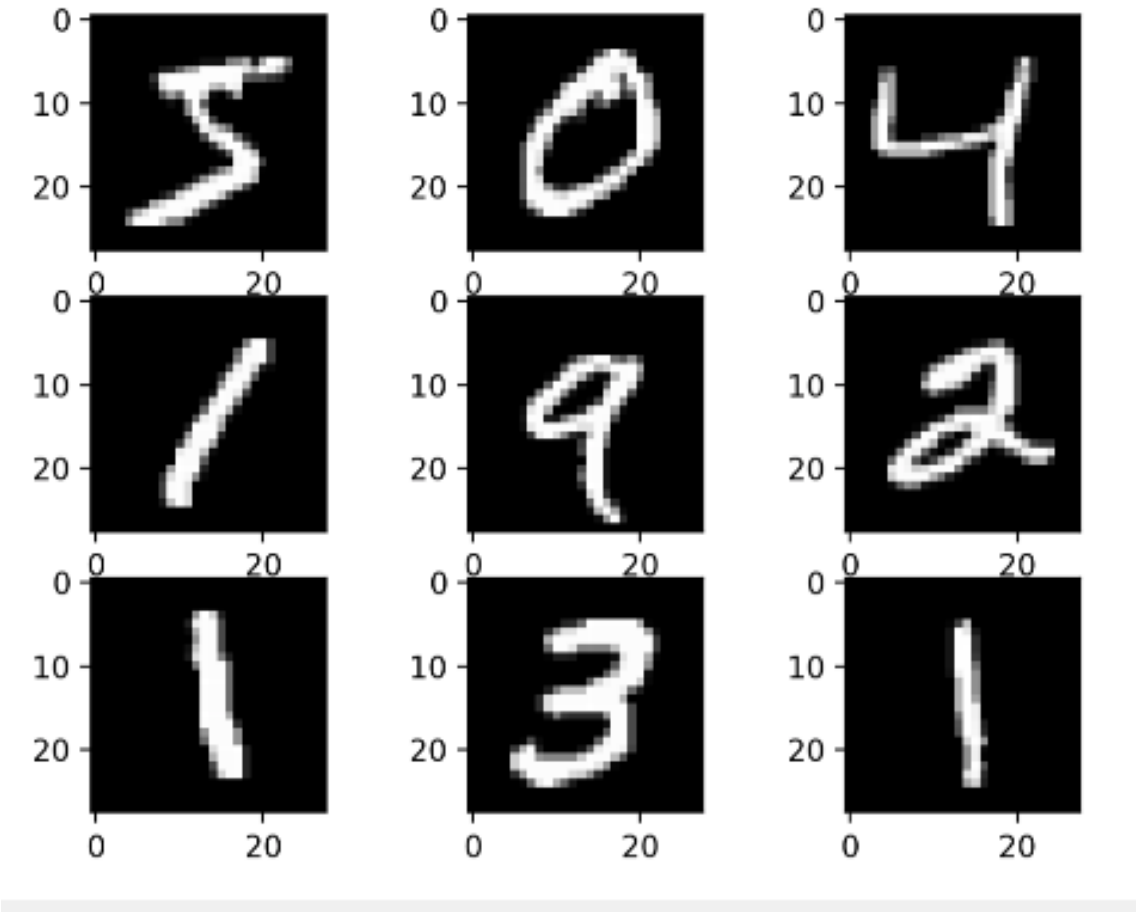


Рис. 1 – Примеры изображений

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 пользователи 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.

Основные теоретические положения

Набор данных MNIST уже входит в состав Keras в форме набора из четырех массивов Numpy.

```
import tensorflow as tf

mnist = tf.keras.datasets.mnist
(train_images, train_labels),(test_images, test_labels) = mnist.
   load_data()
```

Здесь `train_images`и `train_labels`— это тренировочный набор, то есть данные, необходимые для обучения. После обучения модель будет проверяться тестовым (или контрольным) набором, `test_images`и `test_labels`. К хранятся в массивах Numpy, а метки — в массиве цифр от 0 до 9. Изображения и метки находятся в прямом соответствии, один к одному.

Для проверки корректности загрузки достаточно сравнить тестовое изоб-

ражение с его меткой.

```
import matplotlib.pyplot as plt

plt.imshow(train_images[0],cmap=plt.cm.binary)
plt.show()
print(train_labels[0])
```

Исходные изображения представлены в виде массивов чисел в интервале [0, 255]. Перед обучением их необходимо преобразовать так, чтобы все значения оказались в интервале [0, 1].

```
train_images = train_images / 255.0
test_images = test_images / 255.0
```

Также необходимо закодировать метки категорий. В данном случае прямое кодирование меток заключается в конструировании вектора с нулевыми элементами со значением 1 в элементе, индекс которого соответствует индексу метки.

```
from keras.utils import to_categorical

train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

Теперь можно задать базовую архитектуру сети.

```
from tensorflow.keras.layers import Dense, Activation, Flatten
from tensorflow.keras.models import Sequential

model = Sequential()
model.add(Flatten())
model.add(Dense(256, activation’=’relu))
model.add(Dense(10, activation’=’softmax))
```

Чтобы подготовить сеть к обучению, нужно настроить еще три параметра для этапа компиляции:

1. функцию потерь, которая определяет, как сеть должна оценивать качество своей работы на обучающих данных и, соответственно, как корректировать ее в правильном направлении;
2. оптимизатор — механизм, с помощью которого сеть будет обновлять себя, опираясь на наблюдаемые данные и функцию потерь;
3. метрики для мониторинга на этапах обучения и тестирования — здесь нас будет интересовать только точность (доля правильно классифицированных изображений).

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

Теперь можно начинать обучение сети, для чего в случае использования библиотеки Keras достаточно вызвать метод fit сети — он пытается адаптировать (fit) модель под обучающие данные.

```
model.fit(train_images, train_labels, epochs=5, batch_size=128)
```

В процессе обучения отображаются две величины: потери сети на обучающих данных и точность сети на обучающих данных.

Теперь проверим, как модель распознает контрольный набор:

```
test_loss, test_acc = model.evaluate(test_images, test_labels)
print('test_acc:', test_acc)
```

Ход работы

В ходе данной работы были рассмотрены архитектуры представленные в таблице 1. Эти модели были скомпилированы с оптимизатором adam и обучены в течение пяти эпох и при размере серии 128.

Таблица 1 – Скрытые слои сетей

№ сети	Функция активации	Количество нейронов
1	relu	256
2	relu	256
	relu	64
3	relu	128
4	relu	64

При обучении данных моделей были получены следующие результаты. В таблице 2 представлены результаты достигнутые моделями на пятой эпохе обучения.

Таблица 2 – Сравнение архитектур

№ сети	Потери	Точность	Потери при валидации	Точность при валидации
1	0.036998	0.989800	0.085666	0.974400
2	0.028307	0.991778	0.080704	0.975000
3	0.057612	0.983667	0.104857	0.969667
4	0.088719	0.974311	0.132334	0.960600

Учитывая полученные результаты, было решено в дальнейшем использовать первую сеть, так как добавление слоя не дало никаких преимуществ, и данная сеть достигла наилучших результатов среди сетей с одним слоем.

После того как была определена подходящая архитектура, была проведена серия запусков с разными оптимизаторами. А именно, использовались: Adam, Nadam, RMSProp, SGD.

Функция обновления в оптимизаторе Adam имеет вид:

$$m_w^{(t+1)} \leftarrow \beta_1 m_w^{(t)} + (1 - \beta_1)\nabla_w L^{(t)}$$

$$v_w^{(t+1)} \leftarrow \beta_2 v_w^{(t)} + (1-\beta_2)(\nabla_w L^{(t)})^2$$

$$\widehat{m_w} = \frac{m_w^{(t+1)}}{1-\beta_1^{t+1}}$$

$$\widehat{v_w} = \frac{v_w^{(t+1)}}{1-\beta_2^{t+1}}$$

$$w^{(t+1)} \leftarrow w^{(t)} - \eta \frac{\widehat{m_w}}{\sqrt{\widehat{v_w}} + \epsilon}$$

где ϵ является малым значением, чтобы избежать деления на 0, а β_1 и β_2 – коэффициенты забывания для градиентов и вторых моментов градиентов соответственно.

Функция обновления в оптимизаторе NAdam имеет очень похожа на функцию, используемую в оптимизаторе Adam:

$$w^{(t+1)} \leftarrow w^{(t)} - \eta \frac{\overline{m_w}}{\sqrt{\widehat{v_w}} + \epsilon}$$

где $\overline{m_w} = \beta_1 \widehat{m_w} + (1-\beta_1)\nabla_w L^{(t)}$.

Формула обновления в оптимизаторе RMSProp имеет вид:

$$w^{(t+1)} \leftarrow w^{(t)} - \frac{\eta}{\sqrt{v^{(t)} + \epsilon}} \nabla Q_i(w)$$

где $v^{(t)} = \rho v^{(t-1)} + (1-\rho)(\nabla Q_i(w))^2$

При использовании метода SGD – стохастического градиентного спуска функция выглядит так:

$$w^{(t+1)} \leftarrow w^{(t)} - \eta \nabla Q_i(w)$$

где $Q(w) = \frac{1}{n}\sum_i = 1^n Q_i(w)$ – целевая функция. $Q_i(w)$ – значение функции потерь на i-ом примере.

Полученные результаты представлены в таблице 3.

Таблица 3 – Сравнение влияния оптимизаторов и их параметров

Оптимизатор	Параметры	Потери	Точность	Потери при валидации	Точность при валидации
Adam	$default$	0.036998	0.989800	0.085666	0.974400
	$\eta = 0.01$	0.084622	0.972800	0.164272	0.958067
	$\beta_1 = 0.1$	0.050857	0.985489	0.093166	0.972533
	$\beta_2 = 0.8$	0.051275	0.984733	0.088172	0.972867
Nadam	$default$	0.044134	0.987800	0.089970	0.971400
	$\eta = 0.01$	0.099190	0.970400	0.159917	0.958467
	$\beta_1 = 0.1$	0.056188	0.983533	0.101688	0.970400
	$\beta_2 = 0.8$	0.052527	0.984622	0.088471	0.972800
RMSprop	$default$	0.058964	0.982689	0.090658	0.972800
	$\eta = 0.01$	0.077923	0.982200	0.228166	0.965200
	$\rho = 0.5$	0.076265	0.977889	0.107974	0.970333
SGD	$default$	0.388786	0.895000	0.377826	0.896800
	$\eta = 0.0001$	2.174480	0.232978	2.150582	0.266200

При использовании различных оптимизаторов как правило наилучшие результаты достигались при стандартных значениях. Из этого следует, что предустановленные значения действительно подходят для решения большинства задач, и без веских причин их менять не имеет смысла.

После всех проведенных исследований было решено в дальнейшей работе использовать оптимизатор Adam со стандартными параметрами.

Было написано простое приложения с использованием PyQt5, с возможностью порисовать. Для того чтобы воспользоваться им необходимо лишь наличие библиотеки PyQt5, и передать аргументом командной строки ключ `gui`. Все варианты обучения сохранялись в формате `.h5`, что позволило избежать обучения каждый раз при запуске приложения. Интерфейс представлен на рисунке 2.

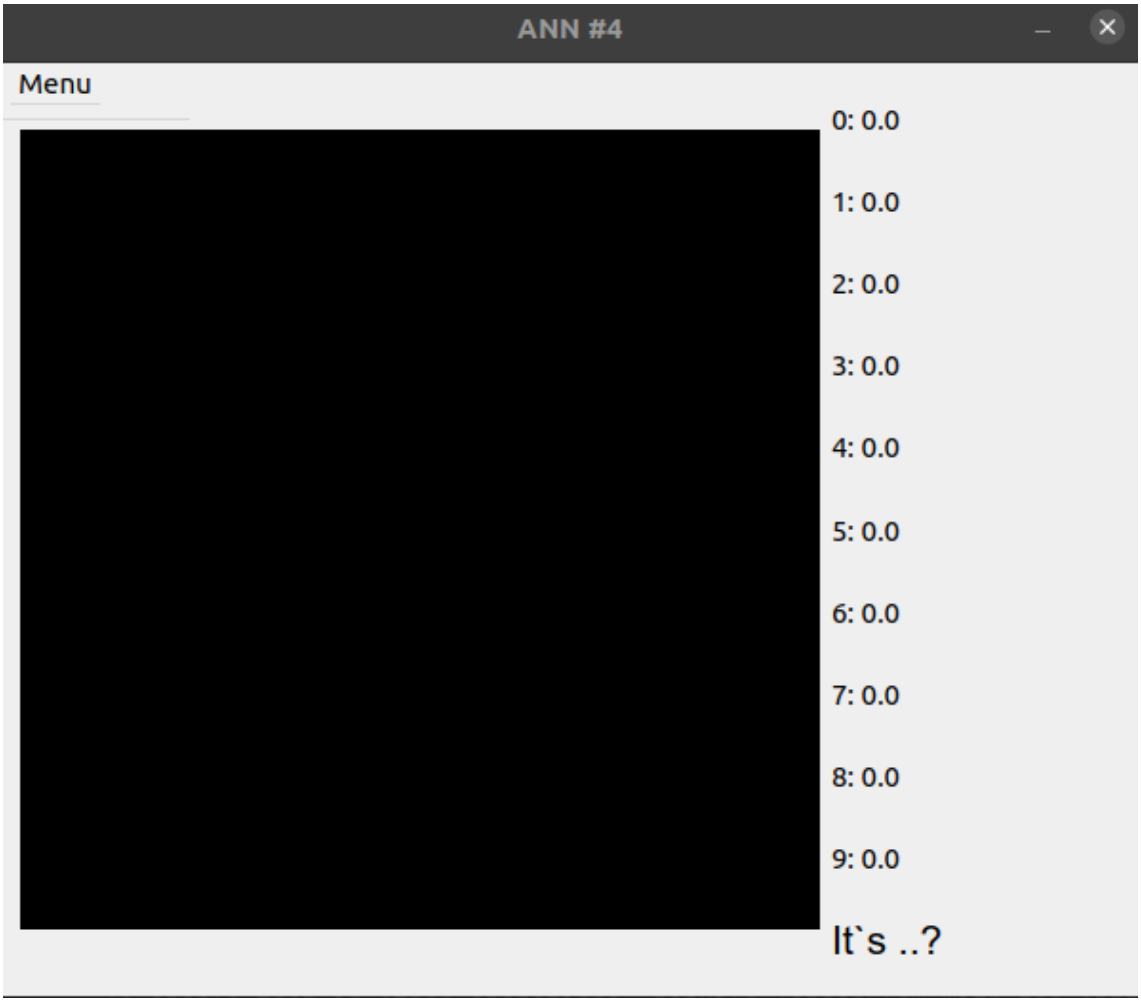


Рис. 2 – Интерфейс приложения

При рисовании приложение выводит свое предсказание, а также все значения, полученные сетью при вызове метода `predict(X,y)`. Пример работы программы представлен на рисунке 3.

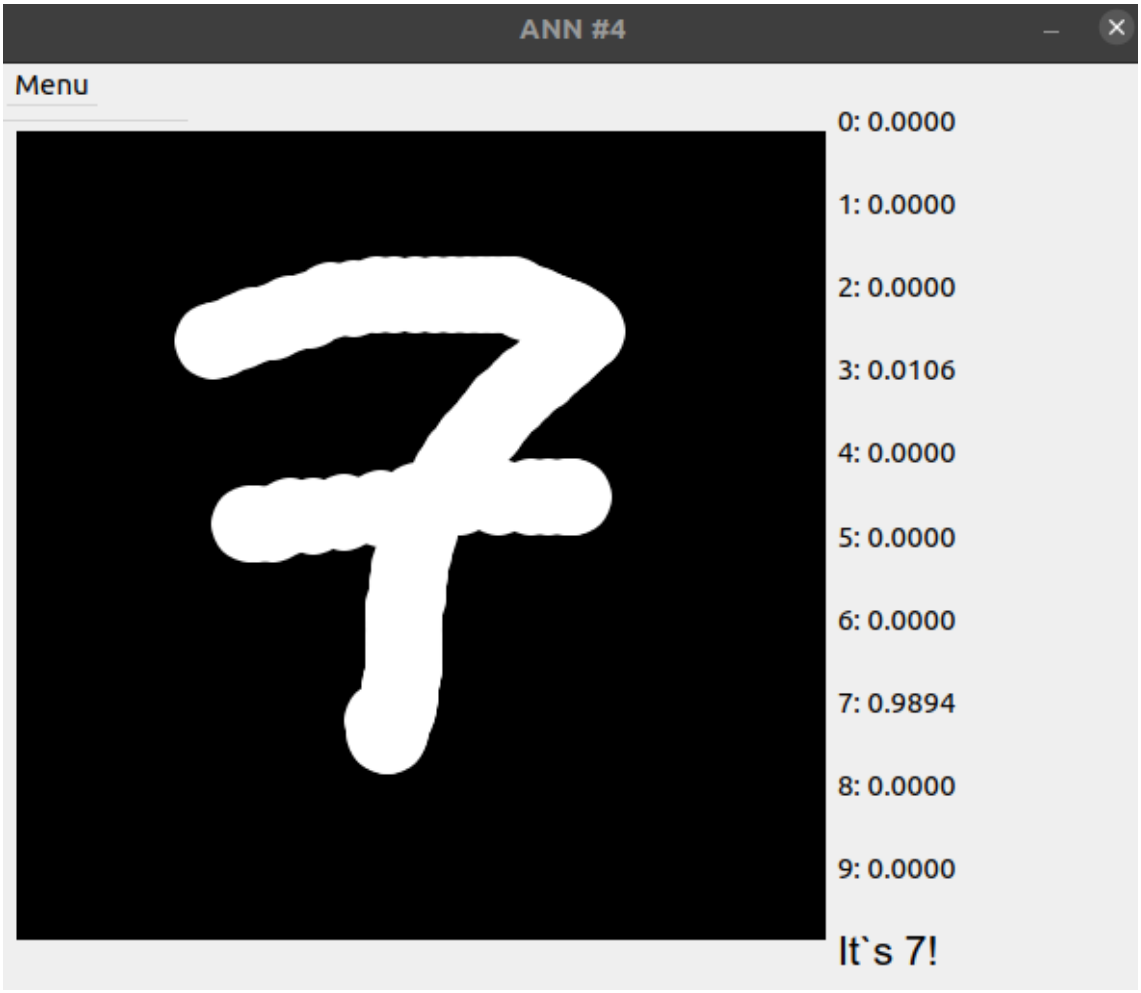
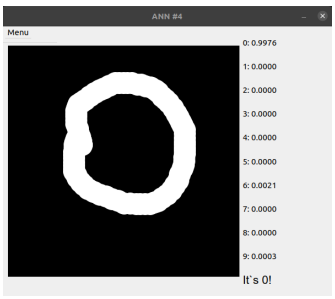


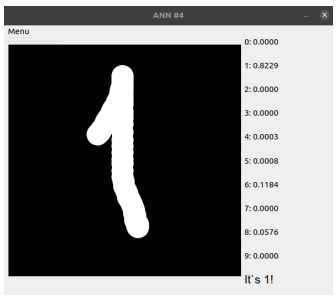
Рис. 3 – Работа программы

Программа также позволяет сохранять и загружать изображения. Данные функции доступны из меню программы расположенном сверху, а также можно использовать шорткаты.

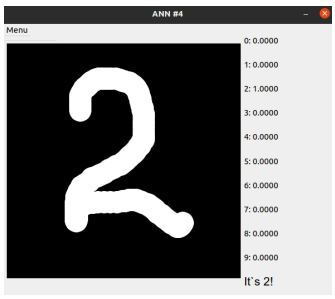
- Ctrl+O – для загрузки изображения;
- Ctrl+S – для сохранения изображения;
- Ctrl+X – для очистки рабочей области.



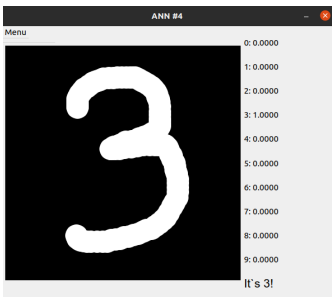
Цифра 0



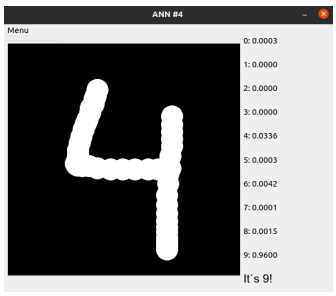
Цифра 1



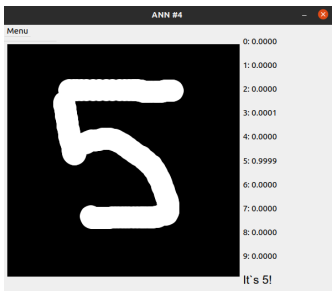
Цифра 2



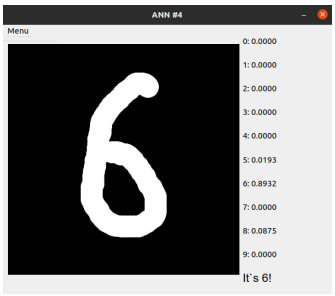
Цифра 3



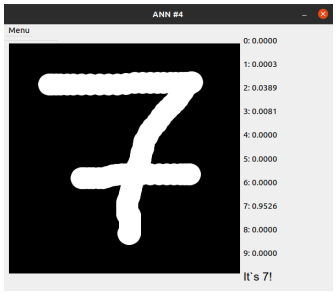
Цифра 4



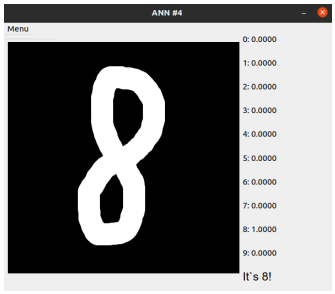
Цифра 5



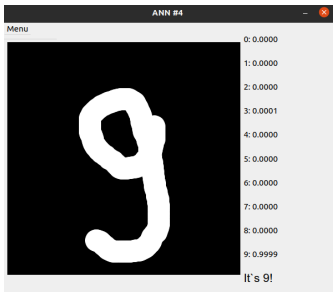
Цифра 6



Цифра 7



Цифра 8



Цифра 9

Рис. 4 – Результаты предсказаний

Ошибка была допущена только при 4, но она действительно имеет что-то общее с 9.

Вывод

В лабораторной работе была реализована классификация черно-белых изображений рукописных цифр (28x28) по 10 категориям (от 0 до 9). Было проведено сравнение влияния на обучение различных оптимизаторов и их параметров. Лучшим из рассмотренных оказался оптимизатор Adam со стандартными параметрами($\eta = 0.001, \beta_1 = 0.9, \beta_2 = 0.999$).

Было написано gui приложение для ввода рукописных цифр и получения результатов работы сети. Для приложения использовалась сеть достигшая наилучших результатов.