МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

КАФЕДРА ИИТ

Моделирование алгоритмов обучения и функционирования искусственных нейронных сетей на базе фреймфорка TensorFlow

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ
ПО ДИСЦИПЛИНЕ «КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ»

КР.ПО4.190333-03 81 00

Листов 24

Руководитель	Ю. В. Савицкий
Выполнил	П. И. Галанин
Консультант по ЕСПД	Ю. В. Савицкий

2021

Содержание

					КР.ПО4.190333-03 81 00		
Изм	*Лист*	*№ докум.*	*Подп.*	*Дата*			
Разраб.		*Галанин*			*Моделирование алгоритмов обучения и функционирования искусственных нейронных сетей на базе фреймфорка TensorFlow Пояснительная записка*	*Лит.*	*Лист*	*Листов*
Пров.		*Савицкий*				*К*	*2*	*24*
							БрГТУ	
Н. контр.		*Савицкий*						
Утв.								

ВВЕДЕНИЕ

Слово «интеллект» (от. лат. intellectus - ум, рассудок, разум) означает способность к мышлению. Ещё при развитии вычислительной техники была проблема создания систем интеллектуальной деястельности - искусственного интелекта. Это проблема поднималась из-за того, что многие задачи не могут быть решены точными алгоритмическими методами. Также эта сеть должна была не только уметь функциорировать в неловкой ситуации, но и уметь обучаться.

Исскуственная нейронная сеть может выполнять разные задачи, например, прогнозирование, распознавание образов.

В задаче прогнозирования мы можем дать, например, дать 7 образцов градусов Цельсий и ожидаемый 7 образцов градусов Фаренгейт, которые выводятся по формуле:

$$F = C \cdot 1.8 + 32$$

Прогав эти образы, например, 50 раз по сети, мы настроим веса (найдем эти числа 1.8 и 32). Теперь дав новый образец в виде градуса Цельсий, мы получим наш результат в градусах Фаренгейт, которые нейронная сеть хорошо определяет (подав 100, получим, например, $211.74736 \approx 212$).

$$100 \cdot 1.8 + 32 = 212$$

То есть научили сеть прогнозировать.

В задаче распознования мы можем разделять образы на классы. Например, в качестве образов мы можем использовать картинки кошек и собак в количестве 25 тысяч. Нейронная сеть прогнав даже 3 эпохи сможет распознавать на картинке кошку или собаку.

В ходе данного курсового проекта необходимо смоделировать нейронную сеть, которая может распознавать не 2 класса, а 10 классов (10 цифр): нуль, один, два, три, четыре, пять, шесть, семь, восемь, девять.

1 Постановка задачи

1.1 Условие

В ходе курсового проекта требуется выполнить следующие пункты:

1) выполнить создание и обучение моделей нейронных сетей на базе вышеуказанного фреймворка, задача: распознование рупописных символов на основе БД MNIST;
2) исследовать производительность, точность обучения и распознавания рукописных цифр на базе полученных модулей;
3) задачи 1 и 2 выполнить для различных конфигураций нейронных сетей;
4) выполнить сравнительный анализ полученных моделей.

1.2 Входная информация по задаче

База данных MNIST (сокращение от «Modified National Institute of Standards and Technology») - объёмная база данных образцов рукописного написания цифр. База данных является стандартом, предложенным Национальным институтом стандартов и технологий США с целью калибрации и сопоставления методов распознавания изображений с помощью машинного обучения в первую очередь на основе нейронных сетей.

В ходе курсового проекта будет использоваться база данных MNIST, которая содержит черно-белые изображения рукописных символов, которые будем подавать на нейронную сеть.

1.3 Выходная информация по задаче

Выходной информацией будет индекс максимального значения выходного нейрона сети. Например, на картинке цифра 0, тогда сеть должа выдать 0. Например, на картинке цифра 1, тогда сеть должа выдать 1. Например, на картинке цифра 8, тогда сеть должа выдать 8. Например, на картинке цифра 9, тогда сеть должа выдать 9. Но сеть не всегда может правильно угадывать цифру. Для продолжения обучения мы должны видеть, хотя бы результат. Поэтому нам так важна цифра.

2 Анализ и описание используемого фреймворка

2.1 TensorFlow

2.1.1 История

TensorFlow - открытая программная библиотека для машинного обучения, разработанная компанией Google для решения задач построения и тренировки нейронной сети с целью автоматического нахождения и классификации образов, достигая качества человеческого восприятия. Применяется как для исследований, так и для разработки собственных продуктов Google. Основной API для работы с библиотекой реализован для Python, также существуют реализации для R, C Sharp, C++, Haskell, Java, Go и Swift.

Является продолжением закрытого проекта DistBelief. Изначально TensorFlow была разработана командой Google Brain для внутреннего использования в Google, в 2015 году система была переведена в свободный доступ с открытой лицензией Apache 2.0.

2.1.2 Лицензия

TensorFlow доступна по лицензии Apache License 2.0. Это дает нам возможность редактировать сам фреймворк. В ходе курсового проекта нам не понадобится редактировать фреймворк.

2.1.3 Операционная система

TensorFlow доступен на разных операционных системах: Windows, Android, iOS, Linux, MacOS. В ходе курсового проекта была выбрана операционная система Linux Debian 10 Xfce.

2.1.4 Язык

Чтобы писать на TensorFlow мы можем использовать выскоуровневые языки такие как C++, Python, JavaScript. В ходе курсового проекта был выбран язык Python.

2.1.5 Версия

Последняя версия TensorFlow имеет версию 2.5.0.

Узнать версию можем через tensorflow.__version__.

Листинг:

```
1 import tensorflow
2 print(f'TensorFlow verion : {tensorflow.__version__}')
```

С версии 2.0 TensofFlow использует GPU. То есть на Windows, Linux требуется ещё устанавливать CUDA for CPU TensorFlow. TensorFlow можно запустить в двух режимах: 1) с поддержкой графического процессора, 2) без поддержки графического процессора.

Для такого, чтобы включить поддержку графического процессора, то в Python используем библиотеку os.

Листинг:

```
1 import os
2 os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
```

2.2 Keras

2.2.1 История

Keras - открытая нейросетевая библиотека, написанная на языке Python. Она представляет собой надстройку над фреймворками Deeplearning4j, TensorFlow и Theano. Нацелена на оперативную работу с сетями глубинного обучения, при этом спроектирована так, чтобы быть компактной, модульной и расширяемой. Она была создана как часть исследовательских усилий проекта ONEIROS (англ. Open-ended Neuro-Electronic Intelligent Robot Operating System), а ее основным автором и поддерживающим является Франсуа Шолле (фр. François Chollet), инженер Google.

TensorFlow является фреймворком низкоуровневым. Один разработчик Google написал свою надстройку Keras. В итоге Google внедрила keras в сам tensorflow. В ходе курсового проекта для удобства моделирования будет использоваться надстройка Keras.

2.2.2 Лицензия

Keras доступен по лицензии MIT. То есть мы имеем полное право её модифицировать. Но в ходе курсового проекта нам этого делать не прийдется.

2.2.3 Операционная система

Keras является мультиплатформенный. Так как мы используем Linux, то библиотека для курсового проекта нам подходит.

2.2.4 Язык

Keras остается нашей библиотекой для фреймворка TensorFlow. В ходе курсового проекта выбран высоко уровневый язык Python. То есть Keras также будем использовать в Python.

2.2.5 Версия

Так как Keras мы используем как библиотеку включенную поумолчанию в TensorFlow, то версия Keras совпадает с версией TensorFlow.

Узнать версию можем через tensorflow.keras.__verison__.

Листинг:

```
1 from tensorflow import keras
2 print(f'Keras verion : {tensorflow.keras.__version__}')
```

3 Моделирование нейронных сетей и алгоритмов обучения

3.1 Моделирование нейронной сети

Для распознавания цифр спроектирована нейронная сеть состоящая из трех слоев.

На входном слое $28 * 28 = 784$ нейрона, да ещё и нейрон смещения (bias). Итого 785.

Скрытый слой только один, состоящий из 128 нейронов. Срытый слоев может быть несколько, хоть и два и три. Нейронов в срытом слое может быть разное количество, хоть и 50 нейронов.

На выходном слое у нас 10 нейронов, то есть 10 классов: ноль, один, два, три, четыре, пять, шесть, семь, восемь, девять.

Спроектированная нейронная сеть на рисунке **3.1 (стр. 10)**.

На вход будем подавать изображения с БД MNIST.

Листинг:

```
(x_train, y_train), (x_test, y_test) =
tensorflow.keras.datasets.mnist.load_data()
```

Где «x_train» - изображения цифр обучающей выборки.

Где «y_train» - вектор соответсвующих значений цифр, если на i-отом изображении нарисовано 5, то y_train[i] = 5.

Где «x_test» - изображения цифр тестовой выборки.

Где «y_test» - вектор соответствующих значений цифр для тестовой выборки, если на i-отом изображении нарисовано 5, то y_test[i] = 5.

Для подачи значений на входной слой значения нужно нормировать поделив на 255, где 255 - максимальное число, то есть при делении получим значение от нуля до единицы.

Листинг:

```
# стандартизация входных данных
x_train = x_train / 255
x_test = x_test / 255
```

Моделируем сеть библиотекой Keras.

Листинг:

```
model = tensorflow.keras.Sequential([
    tensorflow.keras.layers.Flatten(input_shape=(28, 28, 1)),
    tensorflow.keras.layers.Dense(128, activation='relu'),
    tensorflow.keras.layers.Dense(10, activation='softmax')  ])
```

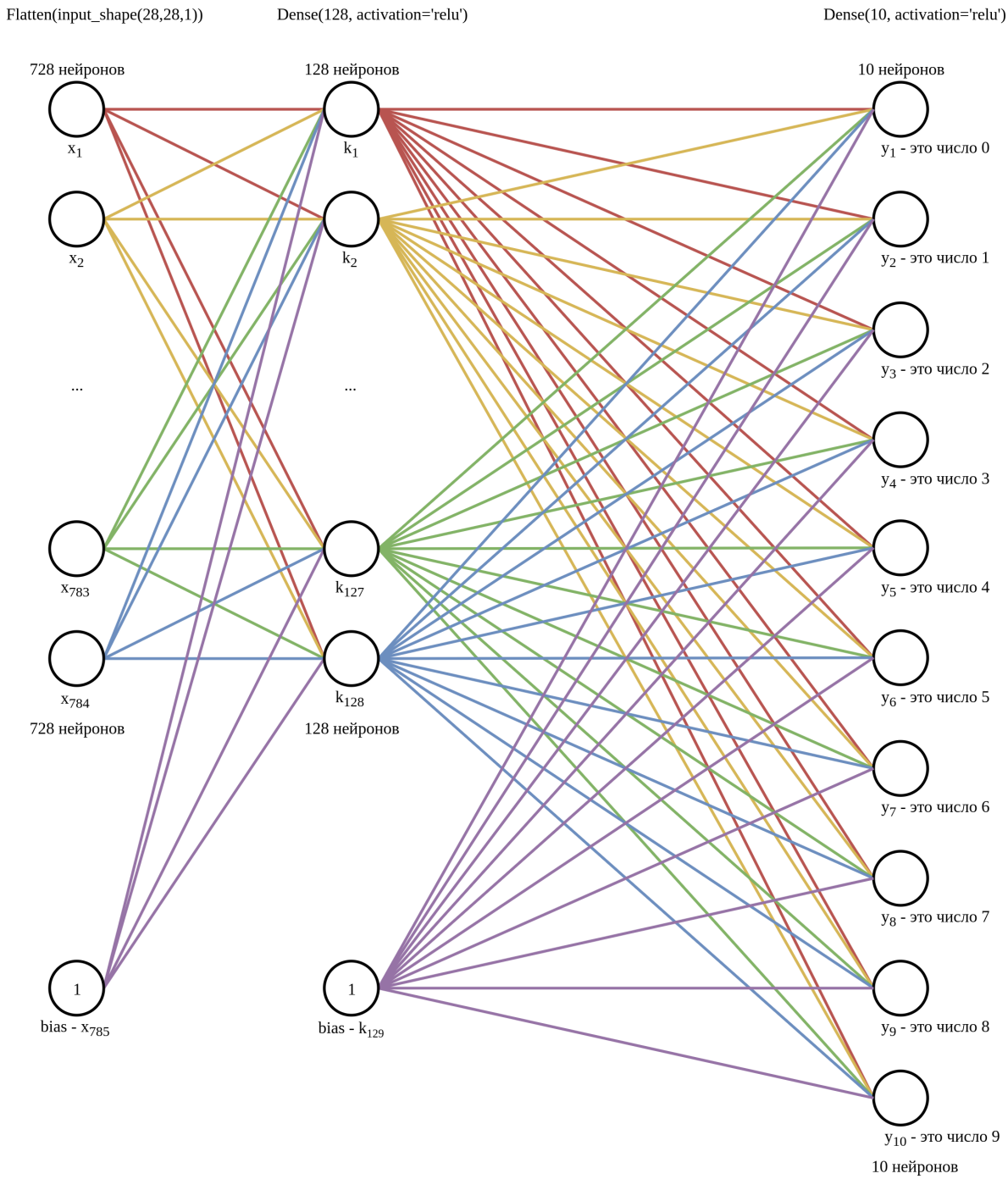

Рисунок 3.1 – Схема нейронной сети

3.2 Функция активации

В искусственных нейронных сетях функция активации нейрона определяет выходной сигнал, который определяется входным сигналом или набором входных сигналов.

В нашей сети функции активации имеют скрытый и выходной слой. Скрытый слой имеет функцию активации ReLu. Выходной слой имеет функцию активации softmax.

3.2.1 ReLu

ReLu - функция активации, которая позволяет решать не линейные задачи.

ReLu имеет формулу вида $y_i = max(0, x_i)$, для i-того нейрона.

График функции изображен на рисунке **3.2 (стр. 11)**.

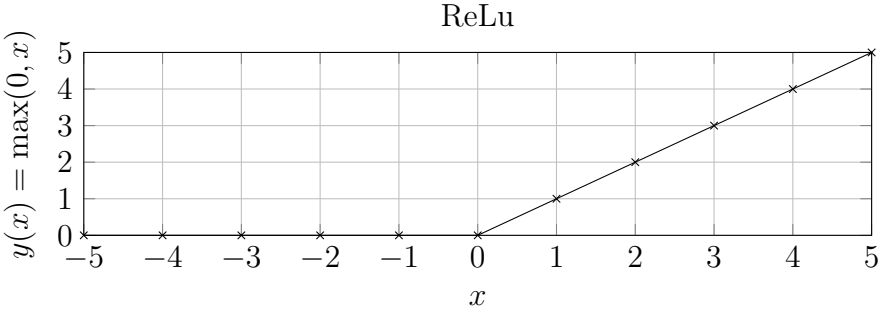

Рисунок 3.2 – ReLu

3.2.2 Softmax

Softmax - функция активации, которая вычисляет вероятность для каждого возможного выходного класса.

Softmax имеет формулу вида $y_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_i}}$, для i-того нейрона.

4 Разработка и описание программных модулей

4.1 Jupyter Notebook

При написании кода приходится при каждом запуске скрипта Python, снова начитатся обучать нейронная сеть. Чтобы такого небыло можно поблочно выполнять, перевыполнять Python код в Jupyter Notebook. Скриншот Jupyter Notebook на рисунке **4.1 (стр. 12)**.

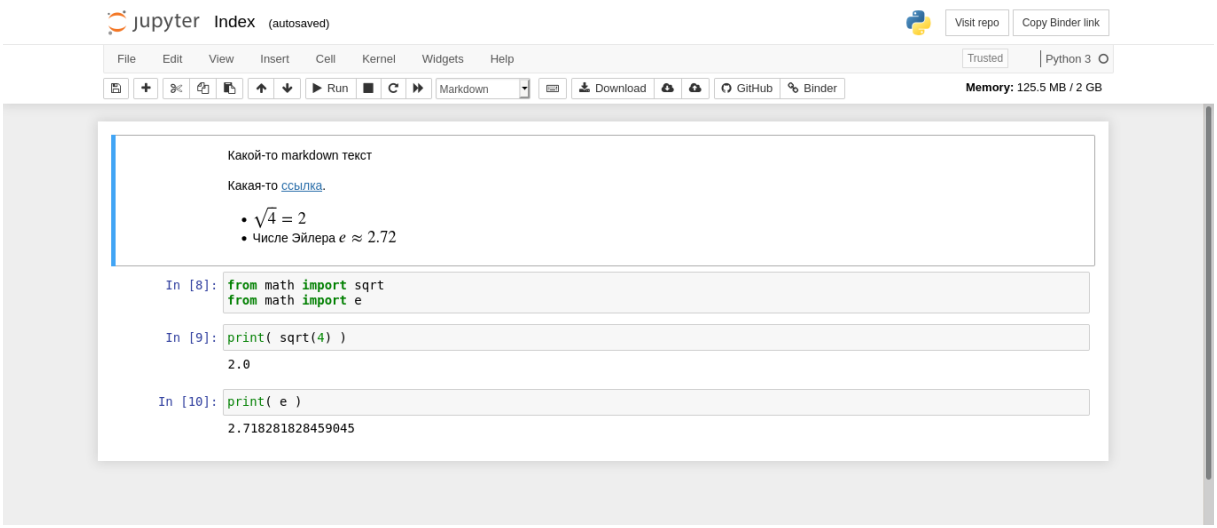

Рисунок 4.1 – Jupyter Notebook

4.2 Google Colaboratory

Jupyter Notebook - это удобно, но пользуя веб версией мы не получаем все библиотеки Python. В итоге приходится скачивать Jupyter Notebook на компьютер и ручками скачивать библиотеки, такие как numpy и tensorflow. Также для обучения нейронной сети на tensorflow нужен графический процессор. Эти две проблемы решает Google Colaboratory, который содержит различные библиотеки Python и дает доступ к другой машине, которая будет обучать нейронную сеть даже с телефона. Скриншот Google Colaboratory на рисунке **4.2 (стр. 13)**.

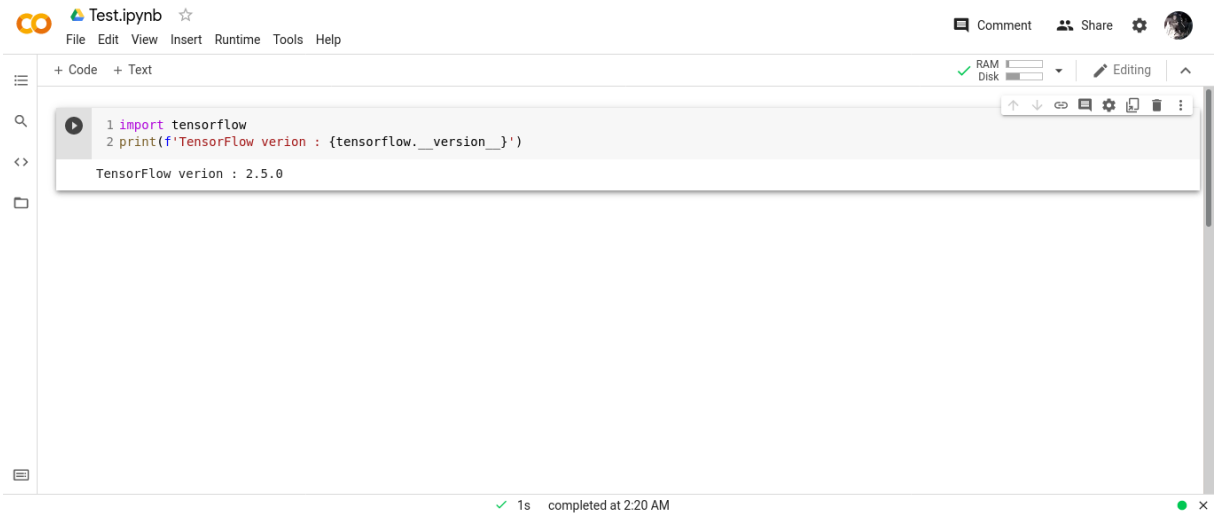

Рисунок 4.2 – Google Colaboratory

4.3 Google Colaboratory секции

4.3.1 Секция библиотек

В секции библиотек подключаем следующие библиотеки:

а) numpy - для создания массивов

б) matplotlib.pyplot - для рисования графиков и картинок

в) tensorflow - фреймворк

Листинг:

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import tensorflow
```

4.3.2 Секция загрузки БД mnist

Функция tensorflow.keras.datasets.mnist.load_data()

Входные параметры: нет.

Назначение: загружает в кортедж кортеджей картинки и результат картинок для обучающей и тестовой выборки.

Возвращаемые данные: кортедж кортеджей.

Листинг:

```
(x_train, y_train), (x_test, y_test) =
tensorflow.keras.datasets.mnist.load_data()
```

Где «x_train» - изображения цифр обучающей выборки.

Где «y_train» - вектор соответсвующих значений цифр, если на i-отом изображении нарисовано 5, то y_train[i] = 5.

Где «x_test» - изображения цифр тестовой выборки.

Где «y_test» - вектор соответствующих значений цифр для тестовой выборки, если на i-отом изображении нарисовано 5, то y_test[i] = 5.

4.3.3 Секция нормирования входных параметров

В базе данных MNIST изображения рукописных цифр черно-белые. Преобразовав изображения в массив мы имеем градацию цветов в диапазоне от единицы до 255. Максимальное число 255. Если мы разделим каждый пиксель на 255, то получим массив из значение не от 1 до 255, а со значениями от нуля до единицы, которые можем подавать на входные нейроны.

Листинг:

```
# стандартизация входных данных
x_train = x_train / 255
x_test = x_test / 255
```

4.3.4 Секция классификации выходного слоя

Функция tensorflow.keras.datasets.mnist.load_data()

Входные параметры:

а) y - массив чисел

б) num_classes - количество классов

Назначение: создаст массив размером num_classes, в который под цифрой запишет 1.
Например, если это число 0, то массив [1,0,0,0,0,0,0,0,0,0].
Например, если это число 1, то массив [0,1,0,0,0,0,0,0,0,0].
Например, если это число 2, то массив [0,0,1,0,0,0,0,0,0,0].
Например, если это число 3, то массив [0,0,0,1,0,0,0,0,0,0].
Например, если это число 4, то массив [0,0,0,0,1,0,0,0,0,0].
Например, если это число 5, то массив [0,0,0,0,0,1,0,0,0,0].
Например, если это число 6, то массив [0,0,0,0,0,0,1,0,0,0].
Например, если это число 7, то массив [0,0,0,0,0,0,0,1,0,0].
Например, если это число 8, то массив [0,0,0,0,0,0,0,0,1,0].
Например, если это число 9, то массив [0,0,0,0,0,0,0,0,0,1].
Возвращаемые данные: кортедж кортеджей.

Листинг:

```
y_train_cat = tensorflow.keras.utils.to_categorical(y_train, 10)
y_test_cat = tensorflow.keras.utils.to_categorical(y_test, 10)
```

4.3.5 Секция печати картинок обучающей выборки

Печатаем 200 картинок обучающей выборки.

Листинг:

```
# отображение первых 20*24=480 изображений из обучающей выборки
plt.figure(figsize=(10,14)) # размер в дюймах
for i in range(480):
    plt.subplot(20,24,i+1)  # картинки 20 по строке и 24 по столбцу
    plt.xticks([])          # не печатать оси по x
    plt.yticks([])          # не печатать оси по y
    plt.title(y_train[i])   # печатать в заголовке картинки цифру
```

```
        plt.imshow(                 # печатать в рамке картинку
            x_train[i],
            cmap=plt.cm.binary
        )
    plt.show()                      # печатаем картинку в окно
```

Результат на рисунке **4.3 (стр. 16)**.

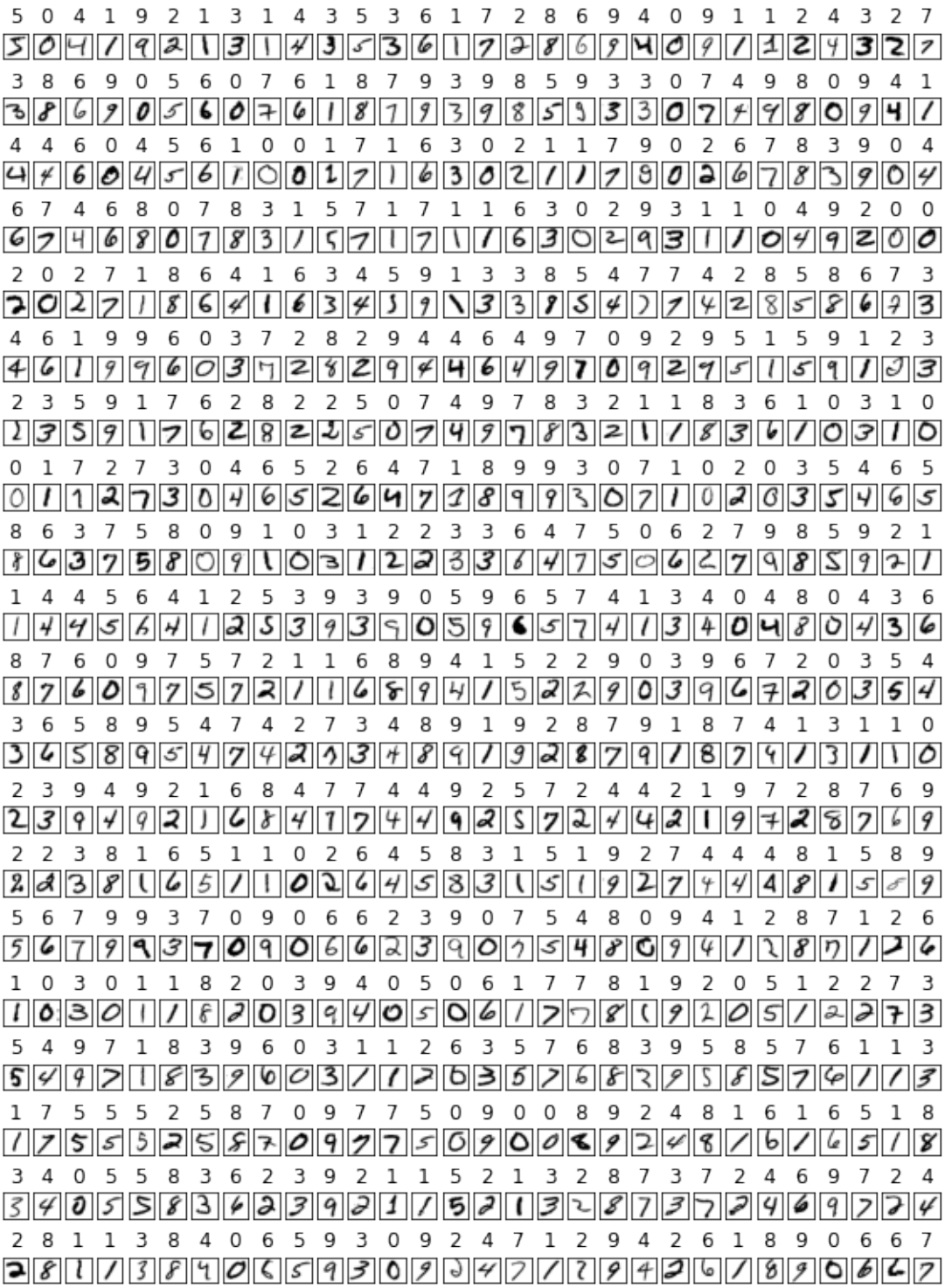

Рисунок 4.3 – Выборка цифр

4.3.6 Секция моделирования нейронной сети

Функция keras.Sequential

Входные параметры: список слоев нейронной сети.

Назначение: создает объект модель нейронной сети, у которой появляются методы.

Возвращаемые данные: object - объект модели.

Flatten - создает слой, который будет брать значения пикселей картинки построчно.

Dense - создаёт слой, который связывает нейронны предыдущего слоя с текущим.

Листинг:

```
model = tensorflow.keras.Sequential([
    tensorflow.keras.layers.Flatten(input_shape=(28, 28, 1)),
    tensorflow.keras.layers.Dense(128, activation='relu'),
    tensorflow.keras.layers.Dense(10, activation='softmax')
])
```

Метод .summary

Входные параметры: нет.

Назначение: печатает характеристики созданной модели.

Возвращаемые данные: None.

Листинг:

```
print( model.summary() )
```

4.3.7 Секция компиляции нейронной сети

Метод .compile

Входные параметры:

а) (необязательный) optimizer - тип оптимизации;

б) (необязательный) loss - функция потерь;

в) (необязательный) metrics - метрика.

Назначение: компилирует нейронную сеть.

Возвращаемые данные: None.

Листинг:

```
model.compile(
    #optimizer='adam',
    optimizer=tensorflow.keras.optimizers.Adam(0.001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

4.3.8 Секция обучения нейронной сети

Метод .fit

Входные параметры:

а) x_train - входное обучающее множество;

б) y_train_cat - требуемые значения на выходе;

в) (необязательный) batch_size - размер batch'а (после скольки изображений будет меняться веса);

г) (необязательный) epochs - количество эпох;

д) (необязательный) validation_split - разбиение обучающей выборки.

Назначение: выводит информацию при обучении на каждой эпохе.

Возвращаемые данные: object.

Листинг:

```
model.fit(
    x_train,
    y_train_cat,
    batch_size=32,
    epochs=5,
    validation_split=0.2
)
```

4.3.9 Секция оценки качества нейронной сети на тестовой выборке

Метод .evaluate

Входные параметры:

а) x_test - картинки тестовой выборки;

б) y_test_cat - ожидаемый результат тестовой выборки.

Назначение: выводит в списке значение потерь и значение показатель обучения.

Возвращаемые данные: list - список.

Листинг:

```
model.evaluate(x_test, y_test_cat)
```

Листинг: Вывод в консоль

```
[0.12078429013490677, 0.9781000018119812]
```

4.3.10 Секция вывода пердсказанного числа и картинки

Функция print_info_about_image_by_index(n)

Входные параметры: n - индекс картинки.

Назначение: выводит картинку, а в заголовке предсказанное значение.

Возвращаемые данные: нет.

Листинг:

```
def print_info_about_image_by_index(n):
    x = numpy.expand_dims( x_test[n], axis=0 )
    res = model.predict(x)
    print(f'Десять выходных значений: {res}')
    print(f'Распознанная цифра : {numpy.argmax(res)}')
    print()
    plt.imshow(x_test[n], cmap=plt.cm.binary)
    plt.show()
```

Листинг:

```
# 10 раз вызвали функцию
for i in range(0, 10):
    print_info_about_image_by_index(i)
```

Результат на рисунке **4.4 (стр. 20)**.

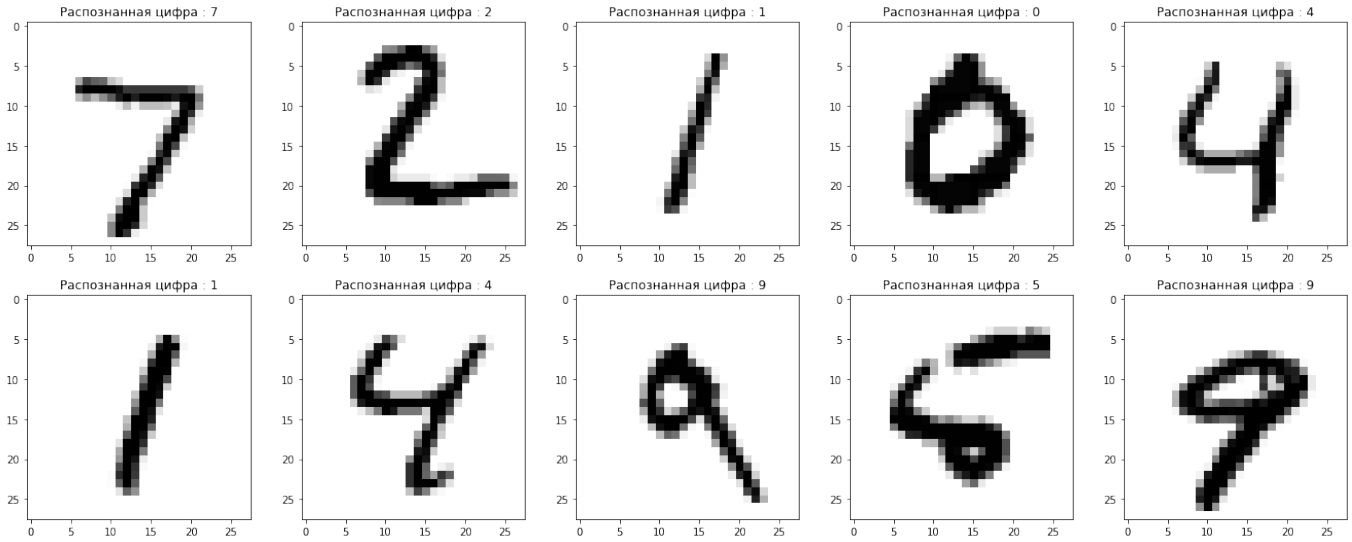

Рисунок 4.4 – Картинка и результат нейронной сети

4.3.11 Отбор неправильно распознанных изображений

При тестировании у нас есть уже какие-то ответы нейронной сети. То что изображено на картинке мы также знаем. Если то, что изображено на картинке совпадает с результатом выданным нейронной сетью, то добавляем в массив True, иначе False. Неправльно распознанные изображения у нас с меткой False. С помощью маски мы отбираем неправильные изображения.

Листинг:

```
mask = pred == y_test
x_false = x_test[~mask]
p_false = pred[~mask]
print(f'Размерность x_false : {x_false.shape}')
print(f'Размерность p_false : {p_false.shape}')
```

Листинг: Вывод в консоль

```
Размерность x_false : (243, 28, 28)
Размерность p_false : (243,)
```

В итоге сеть не распознала 243 изображения.

4.3.12 Вывод неправильных картинок

Листинг:

```
# Вывод первых 243 неверных результатов
plt.figure(figsize=(10,10)) # размер в дюймах
for i in range(243):
    plt.subplot(13,20,i+1)  # расположить картинки в 13x20
    plt.xticks([])          # не выводить оси по x
    plt.yticks([])          # не выводить оси по y
    plt.title(p_false[i])   # печать в заголовок картинки цифру
    plt.imshow(x_false[i], cmap=plt.cm.binary) # печатает в рамку
plt.show()                  # напечатать картинку
```

Не правильно распознанные картинки изображены на рисунке **4.5 (стр. 21)**.

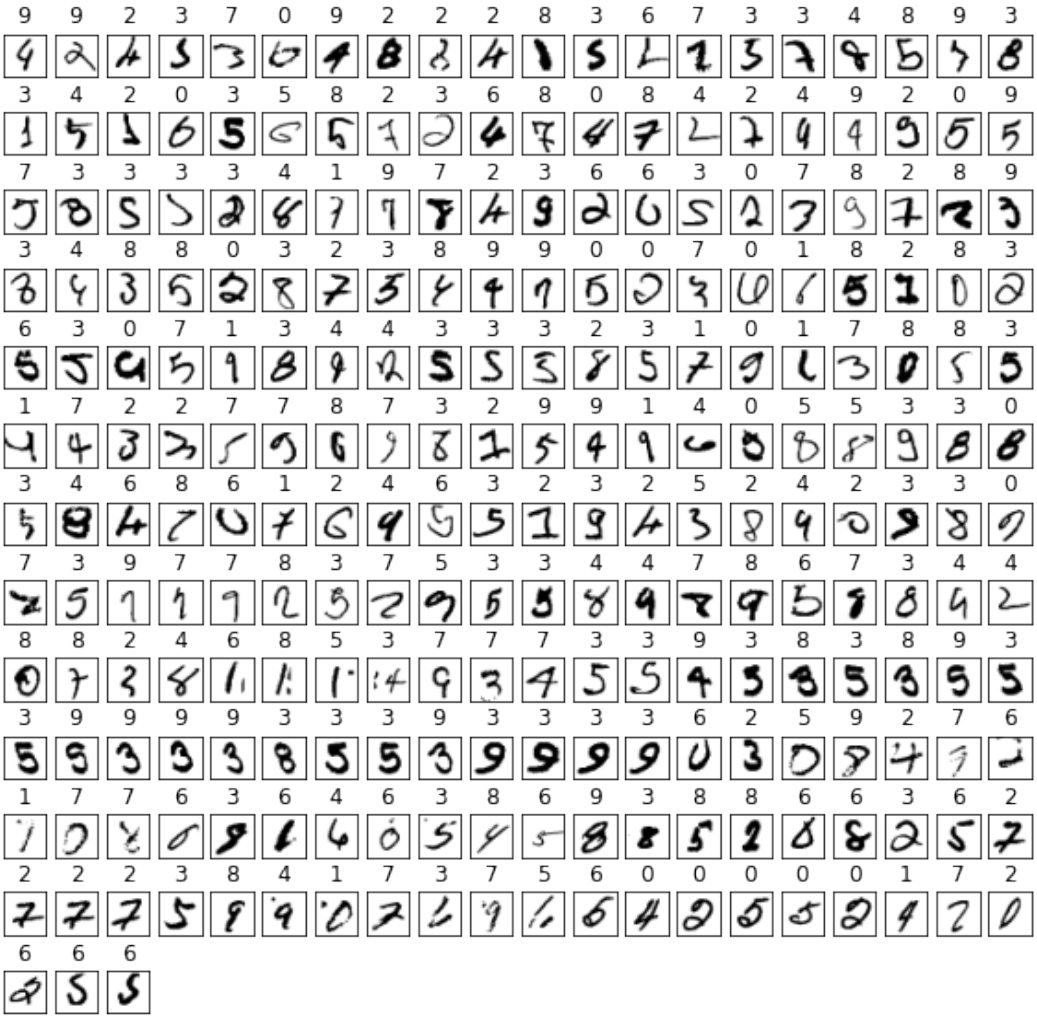

Рисунок 4.5 – Не правльно распознанные цифры

5 Тестирование, сравнительный анализ и результаты исследования

5.1 Подбор шага для оптимизатора Adam

Ожидаемый результат: подборка шага для наилутшей точности распознавания рукописной цифры.

Описнание:

При компилировании модели можно выбрать оптимизатор. В ходе курсовой работы выбран оптимизатор Adam.

Листинг:

```
1  model.compile(
2      optimizer='adam',
3      ...
4  )
```

Оптимизатор Adam по умолчанию имеет шаг 0.001. Этот шаг можно поменять на свой, чем и будем заниматься, при проведении тестов.

Листинг:

```
1  model.compile(
2      optimizer=tensorflow.keras.optimizers.Adam(0.001),
3      ...
4  )
```

Машина тестирования:

Таблица 5.1 – Конфигурация машины

OS	Windows 10 Home
Processor	AMD Ryzen 3 3250U with Radeon Graphics 2.60 GHz
Installed memory (RAM)	4 GB
System type	64-bit OS, x64-based processor

Полученный результат:

Лутшая точность при шаге 0.001. Этот шаг также является default шагом, что и видно по таблице **5.2 (стр. 23)**, так как $0.9736 \approx 0.9733$.

Таблица 5.2 – Точность распознавания в зависимости от шага для алгоритма Adam

Шаг adam	Точность при эксперименте №										Сред. знач.
	1	2	3	4	5	6	7	8	9	10	
default	0.9755	0.971	0.9753	0.9725	0.9726	0.9717	0.9737	0.9742	0.9732	0.976	0.9736
0.1	0.4418	0.6628	0.6077	0.5419	0.5823	0.5548	0.5499	0.5149	0.5213	0.6136	0.5591
0.01	0.9536	0.9593	0.9531	0.9571	0.9639	0.9519	0.9561	0.9613	0.9619	0.9538	0.9572
0.005	0.9673	0.9689	0.9685	0.9671	0.9681	0.9729	0.9626	0.9646	0.9648	0.9678	0.9673
0.001	0.9733	0.9732	0.9757	0.9741	0.9731	0.9714	0.973	0.972	0.9735	0.9732	0.9733
0.0005	0.9682	0.9693	0.9712	0.9701	0.9702	0.971	0.9706	0.9724	0.9693	0.9682	0.9701
0.0001	0.9487	0.9486	0.9492	0.9489	0.9487	0.9488	0.9499	0.9463	0.9484	0.9488	0.9486
1E-05	0.8905	0.8938	0.8929	0.8968	0.8905	0.8904	0.8923	0.8934	0.8932	0.8942	0.8928
1E-06	0.5722	0.5579	0.6092	0.5609	0.5961	0.5732	0.6217	0.5152	0.6123	0.5733	0.5792

6 Заключение

Нейронные сети, технологии прошлого века, меняют работу целых отраслей. Одни возмодности нейронной сети поражают, а другие заставляют задуматься в их пользу как специалистов.

В результате курсового проекта была разработана нейронная сеть, которая распознает рукописные цифры базы данных MNIST, написанная на tensorflow, которая использует надстройку Keras.

Архитектура сети представлена на листинге.

Листинг:

```
model = tensorflow.keras.Sequential([
    tensorflow.keras.layers.Flatten(input_shape=(28, 28, 1)),
    tensorflow.keras.layers.Dense(128, activation='relu'),
    tensorflow.keras.layers.Dense(10, activation='softmax')
])

model.compile(
    optimizer=tensorflow.keras.optimizers.Adam(0.001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

Список использованных источников

1. Как я начал изучать нейросети и python
   https://www.youtube.com/watch?v=PHdw0Uk_Lc4
2. Keras - установка и первое знакомство | #7 нейросети на Python
   https://www.youtube.com/watch?v=BQg9OZdzLLE
3. Keras - обучение сети распознаванию рукописных цифр | #8 нейросети на Python
   https://www.youtube.com/watch?v=oCXh_GFMmOE
4. Как нейронная сеть распознает цифры | #9 нейросети на Python
   https://www.youtube.com/watch?v=S3cViFMiYZ4