Санкт-Петербургский государственный морской технический университет

Отчёт

по практической работе №6

“ Обучение нейросети классификации рукописных цифр”

Выполнил: Студент 1 курса

Факультета Цифровых

Промышленных Технологий

группы 20221 Хохлов Д.Р.

Преподаватель:

Кафедра киберфизических систем

Кайнова Татьяна Денисовна

Санкт-Петербург

2024

Оглавление

[Введение 2](#_Toc186432430)

[Практическая часть 3](#_Toc186432431)

[1.1 Постановка задачи 3](#_Toc186432432)

[1.2 Действия для выполнения задачи 3](#_Toc186432433)

[1.3 Работа с датасетом 3](#_Toc186432434)

[1.4 Изменение структуры нейросети 4](#_Toc186432435)

[1.5 Построение графика 5](#_Toc186432436)

[1.6 Увеличение датасета с помощью аугментации 5](#_Toc186432437)

[1.7 Проверка нейросети после обучения 5](#_Toc186432438)

[1.8 Результаты обучения нейросети 6](#_Toc186432439)

[Заключение 7](#_Toc186432440)

[Приложение А 8](#_Toc186432441)

# Введение

Цель работы: разработка модели классификации рукописных цифр из набора данных MNIST. Используемая архитектура представляет собой полносвязную нейронную сеть с одним скрытым слоем. В процессе выполнения работы оценивается влияние модификаций сети, изменения гиперпараметров и увеличения обучающей выборки на точность модели.

# Практическая часть

## 1.1 Постановка задачи

Задача данной лабораторной работы в реализации классификатора-нейросети, которая умеет различать рукописные цифры из датасета MNIST. Так же требуется реализовать возможность увеличить датасет с помощью аугментации данных.

Классификатор будет основываться на нейросети из Практической работы №5, где была реализована нейросеть с похожей задачей.

## 1.2 Действия для выполнения задачи

Для выполнения задачи необходимо:

1. Реализовать работу с датасетом
2. Изменить структуру нейросети
3. Реализовать построение графика ошибок во время обучения
4. Реализовать создание новых данных на основе существующих(аугментацию) и добавление их в датасет
5. Реализовать проверку нейросети после обучения

## 1.3 Работа с датасетом

Для задачи используется датасет MNIST (Рисунок 1). Данный датасет включает в себя данные картинок 28x28 с рукописными цифрами и подписи к ним, от 0 до 9. Данные картинок являются одномерным массивом, содержащий цвет каждого пикселя

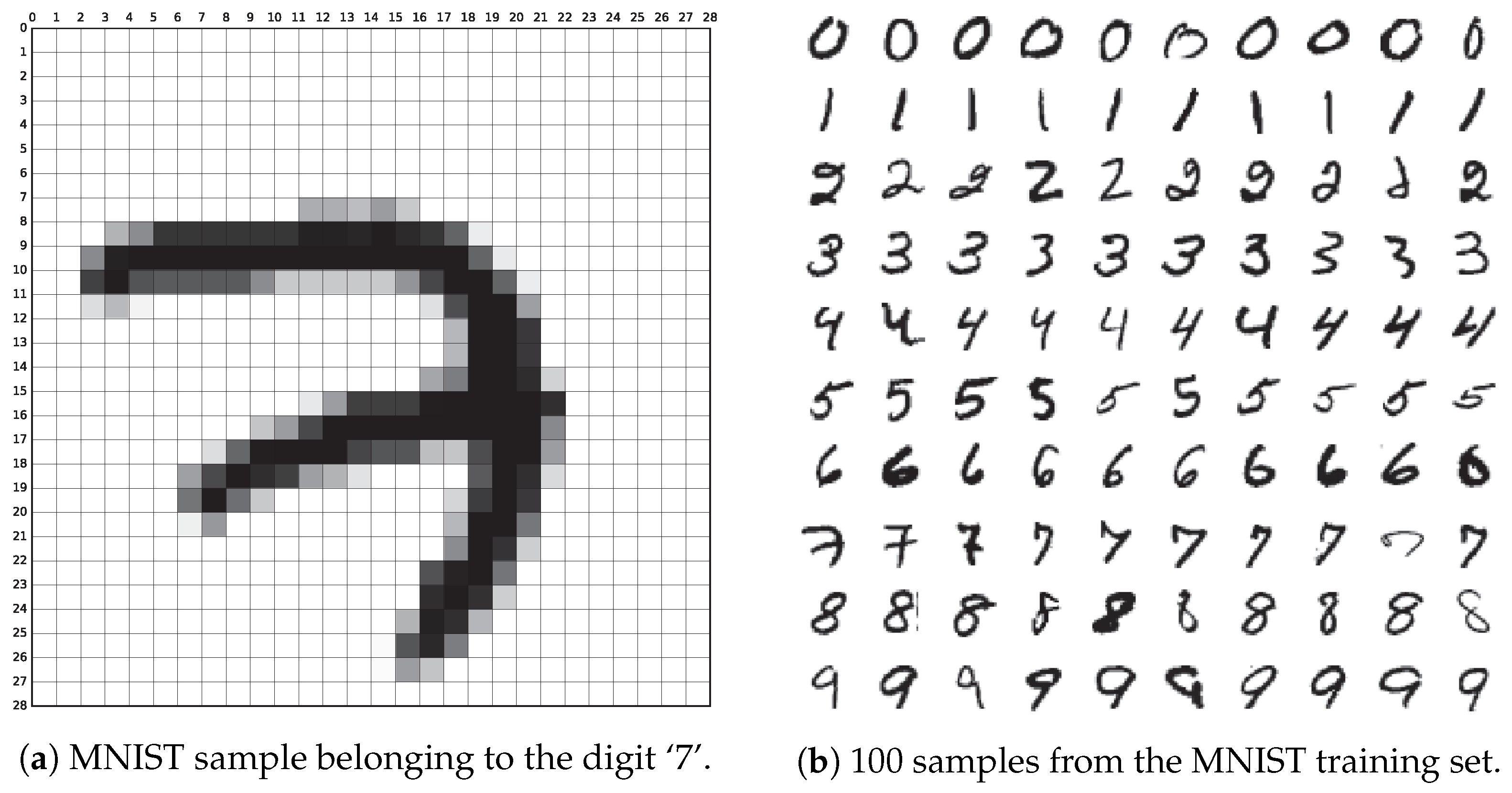
.

Рисунок 1 – Пример данных из датасета MNIST

Загрузка датасета MNIST происходит с портала с открытыми датасетами – OpenML. Для загрузки с данного сервиса используется библиотека «Scikit-Learn» для языка программирования «Python»

Каждый элемент подписи следует переформировать из строчного типа в тип вектора, состоящего из 10 значений 0 или 1, где 1 будет стоять под элементом подписи из датасета, а 0 – остальные цифры (Рисунок 2).

Рисунок 2 – пример переформирования подписи в вектор

## 1.4 Изменение структуры нейросети

Нейросеть принимает на вход 784 значения, каждое значение представляет из себя цвет пикселя из взятого изображения в датасете.

Нейронная сеть состоит из 2-х скрытых слоёв. Первый скрытый слой нейросети состоит из 128 нейронов, а второй скрытый слой состоит из 64 нейронов.

Для поставленной задачи используется структура из множественного выхода. Выход нейросети будет состоять из 10 нейронов, каждый из которого будет представлять значение от 0 до 9.

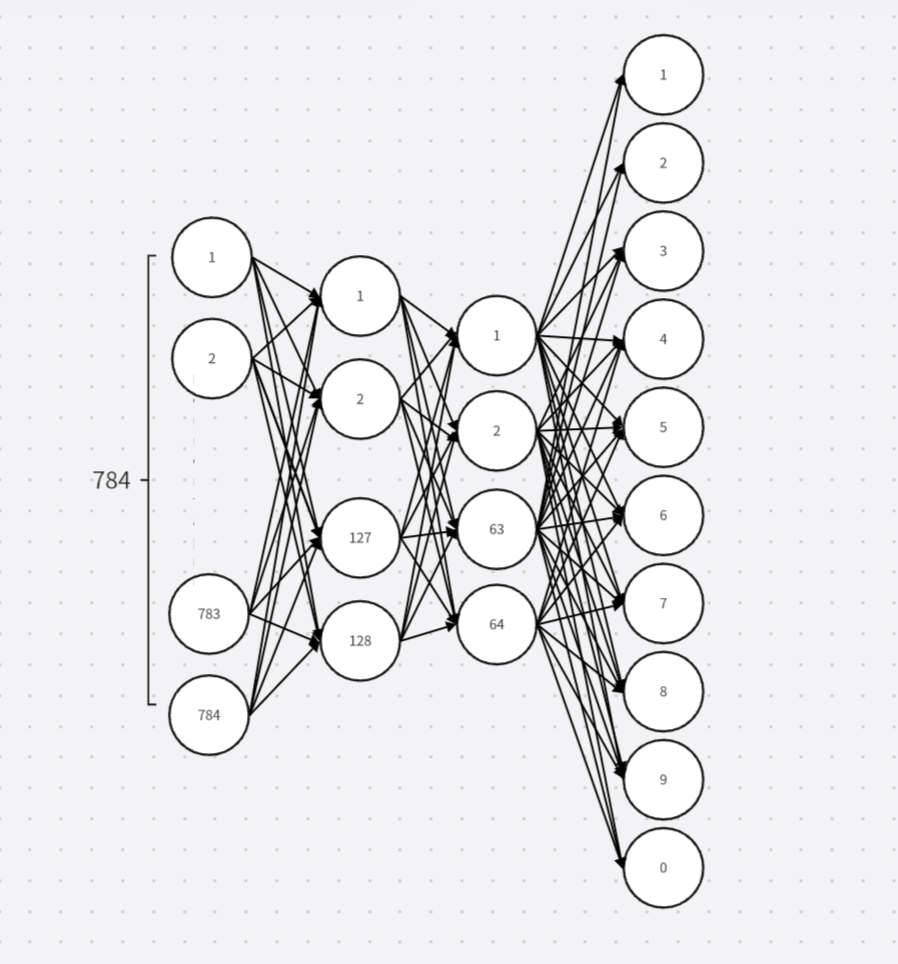


Рисунок 3 – Структура нейросети

## 1.5 Построение графика

Во время обучения нейросети на каждой эпохе вычисляется коэффициент ошибки и количество правильно определённых изображений. После обучения для визуализации можно построить график отображения изменения ошибки на каждой эпохе. Таким образом можно определить подходят ли заданные параметры, в виде learning\_rate, количество скрытых слоёв и количество нейронов в них для заданной задачи.

## 1.6 Увеличение датасета с помощью аугментации

Для того чтобы увеличить уже данный датасет используется аугментация, т.е. создание новых изображений с помощью изменения изображений из датасета.

Была реализована программа для сдвига изображений в разные стороны. Всего получилось 5 вариантов для каждого изображения из датасета. Таки образом датасет состоящий из 70000 изображений увеличился в датасет состоящий из 350000 изображений и подписей к ним.

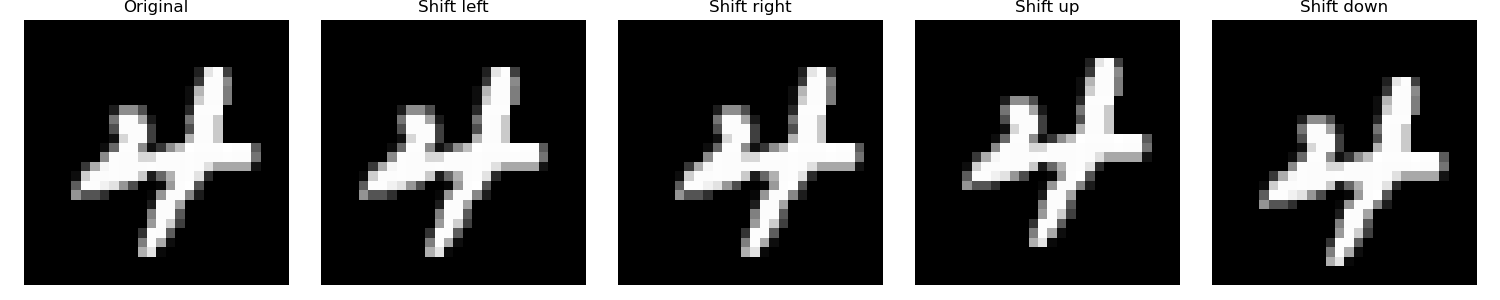


Рисунок 4 – Пример аугментации

## 1.7 Проверка нейросети после обучения

После обучения нейросети осуществляется проверка нейросети на изображениях, которых нет в датасете. Для проверки была реализована небольшая функция загрузки изображений и конвертация из RGB в Unit RGB. Таким образом можно загружать свои изображения 28х28 пикселей.

## 1.8 Результаты обучения нейросети

После 10 эпох обучения нейросеть выдала следующие результаты (Рисунок 5):

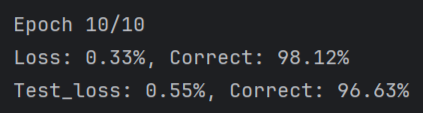


Рисунок 5 – Результаты обучения

Был получен график изменения ошибки во время изменения эпох (Рисунок 6)

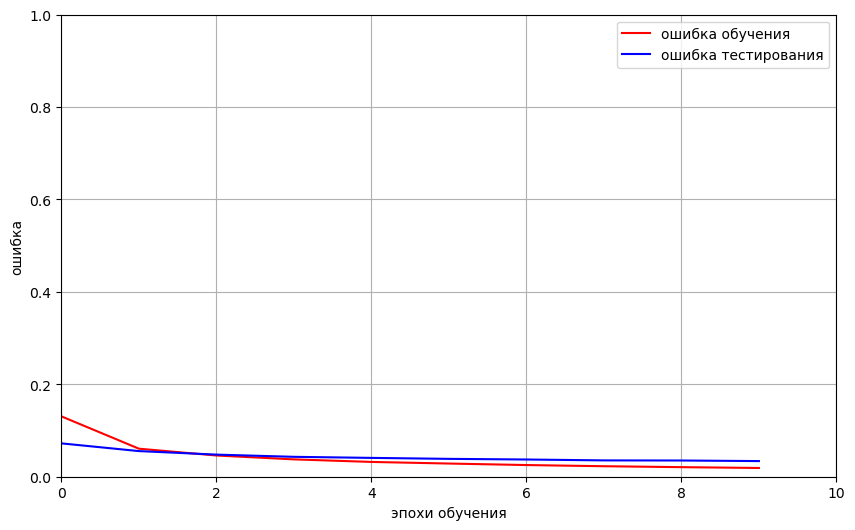


Рисунок 6 – График изменения ошибки во время обучения

Как видно на графике, тренировочная ошибка и тестовая ошибка уменьшается во время обучения. Тренировочная ошибка уменьшается быстрее чем тестовая, т.к. тренировочных данных больше и нейросеть по ним постоянно проходится, запоминая каждую деталь.

Была произведена проверка нейросети на изображениях, нарисованных в программе для рисования “Paint”, загруженных и обработанных для определения на них цифр нейросетью (Рисунок 7).

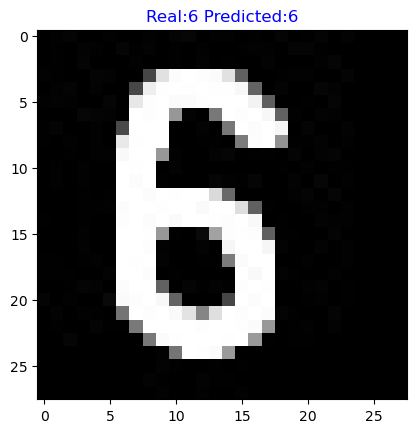
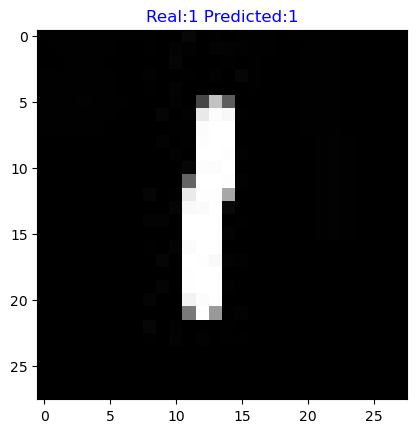
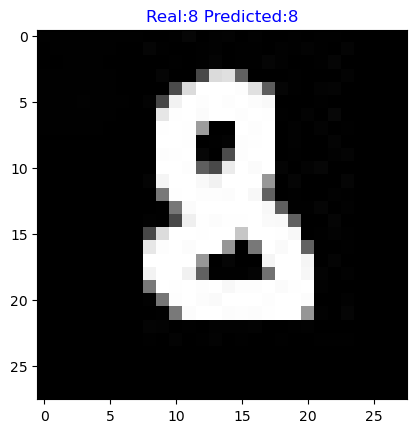
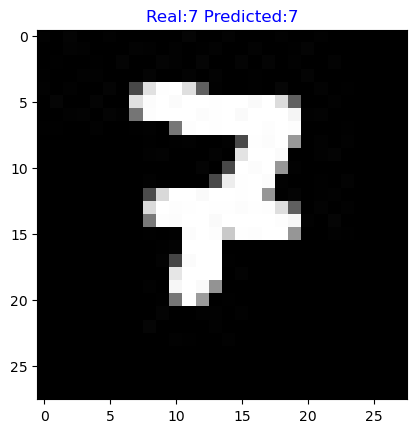
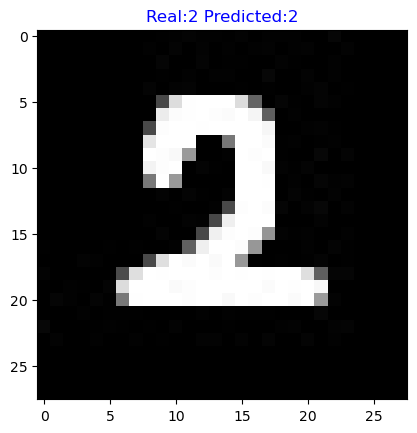
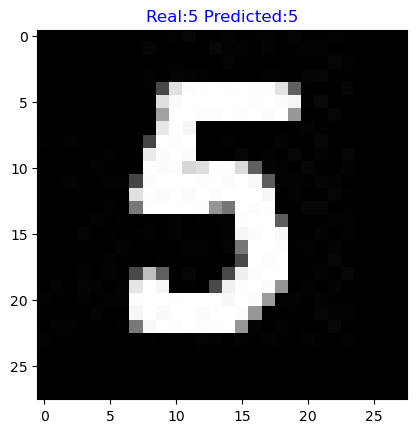
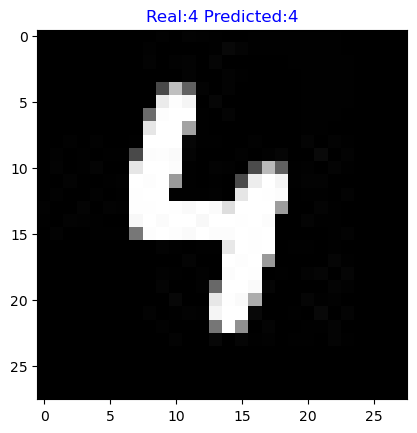
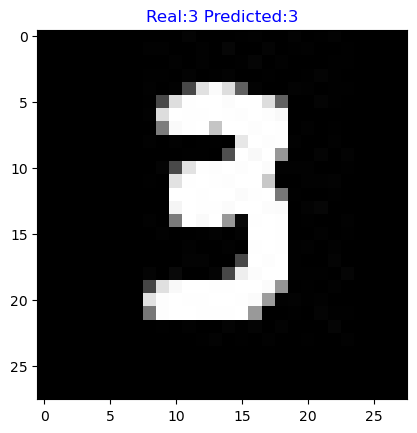
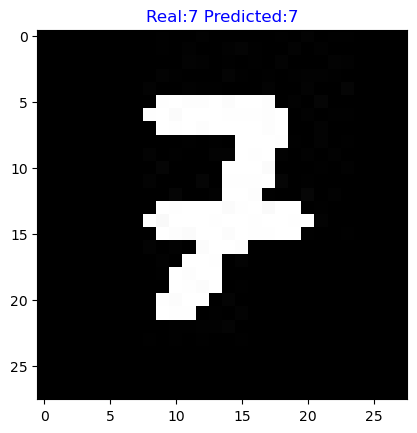
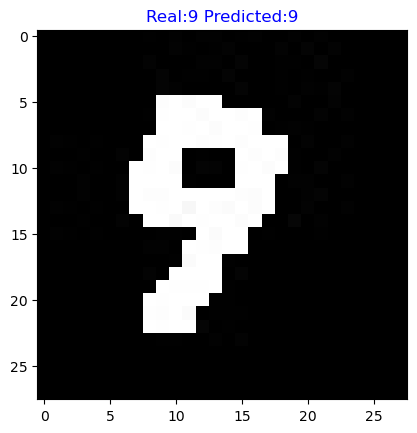


Рисунок 7 – Проверка обученной нейросети

Как видно на Рисунке 7, нейросеть справляется со своей задачей определять рукописные цифры.

# Заключение

В ходе выполнения лабораторной работы была разработана модель классификации рукописных цифр. Последовательное улучшение гиперпараметров и увеличение обучающей выборки позволили значительно повысить точность модели.

Работа продемонстрировала важность тщательной настройки гиперпараметров и обработки данных для повышения производительности моделей машинного обучения.

# Приложение А

Листинг программы

Листинг - Импортирование библиотек

#!pip install numpy  
#!pip install prettytable  
#!pip install scikit-learn  
#%%  
import random  
from abc import ABC, abstractmethod  
  
import numpy as np  
  
from sklearn.datasets import fetch\_openml  
  
import os  
  
from tqdm import tqdm  
  
import matplotlib.pyplot as plt  
%matplotlib inline

Листинг – класс Dataset, для работы с датасетом

class Dataset:  
 def \_\_shift\_image(self,image, direction):  
 img\_2d = image.reshape(28, 28)  
 shifted = np.zeros\_like(img\_2d)  
   
 if direction == 'left':  
 shifted[:, :-1] = img\_2d[:, 1:]  
 elif direction == 'right':  
 shifted[:, 1:] = img\_2d[:, :-1]  
 elif direction == 'up':  
 shifted[:-1, :] = img\_2d[1:, :]  
 elif direction == 'down':  
 shifted[1:, :] = img\_2d[:-1, :]  
   
 return shifted.reshape(784)  
   
 def load\_data\_openml(self,title,version):  
 dataset = fetch\_openml(title, version=version, as\_frame=False)  
 self.dataset\_images = dataset["data"].astype('float32') / 255  
 self.dataset\_labels = dataset["target"].reshape(-1, 1).astype('int32')  
 def augmentation(self):  
   
 directions = ['left', 'right', 'up', 'down']  
 n\_samples = self.dataset\_images.shape[0]  
   
 # Создаем массивы для расширенного набора (оригинал + 4 сдвинутых версии)  
 x\_augmented = np.zeros((n\_samples \* 5, 784))  
 y\_augmented = np.zeros((n\_samples \* 5, 1))  
   
 for i in range(n\_samples):  
 # Добавляем оригинальное изображение  
 x\_augmented[i \* 5] = self.dataset\_images[i]  
 y\_augmented[i \* 5] = self.dataset\_labels[i]  
   
 # Добавляем сдвинутые версии  
 for j, direction in enumerate(directions):  
 x\_augmented[i \* 5 + j + 1] = self.\_\_shift\_image(self.dataset\_images[i], direction)  
 y\_augmented[i \* 5 + j + 1] = self.dataset\_labels[i]  
   
 print(f"Размер оригинального набора данных: {self.dataset\_images.shape[0]} изображений")  
 print(f"Размер расширенного набора данных: {x\_augmented.shape[0]} изображений")  
 print(f"Коэффициент увеличения: {x\_augmented.shape[0] / self.dataset\_images.shape[0]:.1f}x")  
 self.dataset\_images = x\_augmented  
 self.dataset\_labels = y\_augmented  
   
 def \_\_call\_\_(self, \*args, \*\*kwargs):  
   
 return self.dataset\_images, self.dataset\_labels

Листинг – класс Activation, Layer, NN для работы с нейросетью

class Activation:  
 @staticmethod  
 def SIGMOID():  
 def activation(x):  
 return 1 / (1 + np.exp(-x))  
 def derivative(x):  
 return x \* (1 - x)  
 return activation, derivative  
   
 @staticmethod  
 def TANH():  
 def activation(x):  
 return np.tanh(x)  
 def derivative(x):  
 return 1 - x \*\* 2  
 return activation, derivative  
  
#%%  
class Layer:  
 def \_\_init\_\_(self, input\_size, output\_size, activation\_func):  
 self.activate, self.derivative = activation\_func()  
 self.weights = np.random.uniform(-0.5,0.5,size=(input\_size, output\_size))  
 self.biases = np.zeros(output\_size)   
 def \_\_call\_\_(self):  
 return {  
 'weights':self.weights,   
 'biases':self.biases  
 }  
#%%  
class NN(ABC):  
 def \_\_init\_\_(self, layers):  
 self.layers = layers  
 self.params = self.\_\_params()  
   
 def \_\_params(self):  
 params = {}  
 for i, layer in enumerate(layers):  
 params[f'layer{i}'] = {"weights":layer.weights,  
 "biases":layer.biases}  
 return params  
 def \_forward(self, x):  
 for layer in self.layers:  
 z = np.dot(x, layer.weights) + layer.biases  
 x = layer.activate(z)  
 layer.output = np.array([x])  
 return x  
   
 def \_backward(self, x, y, lr):  
 error = -(y - self.layers[-1].output)  
 for layer in reversed(self.layers):  
 delta = error \* layer.derivative(layer.output)  
 prev\_output = np.array([x]) if layer == self.layers[0] else self.layers[self.layers.index(layer) - 1].output  
   
 layer.weights -= lr \* np.dot(prev\_output.T, delta)  
 layer.biases -= lr \* np.sum(delta, axis=0)  
 error = np.dot(delta, layer.weights.T)  
   
 @abstractmethod  
 def train(self):  
 pass  
   
   
   
 def \_\_call\_\_(self, input\_data):  
 output = []  
 for i in range(input\_data.shape[0]):  
 output.append(self.\_forward(input\_data[i]))  
 return output

Листинг – класс MnistNN для создания нейросети на основе датасета MNIST

class MnistNN(NN):  
 def \_\_init\_\_(self, layers):  
 super().\_\_init\_\_(layers)  
   
 def \_\_MSE(self, output, p\_out):  
 error = 1/10 \* np.sum((output - p\_out)\*\*2, axis = 0) #MSE  
 return error  
   
 def train(self, images, labels, epochs, learning\_rate=0.01):  
 self.epochs = epochs  
 self.chart\_y\_train = []  
 self.chart\_y\_test = []  
 index\_list = list(range(210000))  
 for i in range(epochs):  
 random.shuffle(index\_list)  
 train\_loss, train\_correct, test\_loss, test\_correct = 0.0, 0, 0.0, 0  
 p\_bar = tqdm(index\_list, total=210000)  
 p\_bar.set\_description(f"Epoch {i+1}/{epochs} Status:train")  
 for j in p\_bar:   
   
 p\_out = self.\_forward(images[j])  
   
 train\_loss += np.sum(self.\_\_MSE(labels[j],p\_out))  
 train\_correct += int(labels[j].argmax() == p\_out.argmax())  
   
 self.\_backward(images[j], labels[j], learning\_rate)  
   
 p\_bar = tqdm(zip(images[210000:350000], labels[210000:350000]), total = 140000)  
 p\_bar.set\_description(f"Epoch {i+1}/{epochs} Status:test")  
 for image, label in p\_bar:  
 p\_out = self.\_forward(image)  
 test\_loss += np.sum(self.\_\_MSE(label,p\_out))  
 test\_correct += int(label.argmax() == p\_out.argmax())  
   
 self.chart\_y\_train.append(1.0-train\_correct/210000)  
 self.chart\_y\_test.append(1.0-test\_correct/140000)  
 print(f"Epoch {i+1}/{epochs}\n"  
 f"Loss: {round((train\_loss/210000.0) \* 100, 2)}%, Correct: {round((train\_correct/210000.0) \* 100, 2)}%\n"  
 f"Test\_loss: {round((test\_loss/140000.0) \* 100, 2)}%, Correct: {round((test\_correct/140000.0) \* 100, 2)}%")  
   
   
 def plot\_learning(self):  
 epochs = [i for i in range(self.epochs)]  
 plt.figure(figsize=(10, 6))  
 plt.plot(epochs, self.chart\_y\_train, 'r-', label='ошибка обучения')  
 plt.plot(epochs, self.chart\_y\_test, 'b-', label='ошибка тестирования')  
 plt.axis([0, len(epochs), 0.0, 1.0])  
 plt.xlabel('эпохи обучения')  
 plt.ylabel('ошибка')  
 plt.legend()  
 plt.grid(True)  
 plt.show()

Листинг – создание экземпляра нейросети, обучение, вывод графика обучения

#~~Структура нейросети~~#  
layers = (  
 Layer(784, 128, Activation.TANH),  
 Layer(128,64,Activation.TANH),  
 Layer(64, 10, Activation.SIGMOID),  
)  
#%%  
model = MnistNN(layers) #создание объекта нейросети  
print(model.params['layer0']['weights']) #вывод параметров  
#%%  
LEARNING\_RATE = 0.01  
EPOCH = 10  
#%%  
#~~Обучение нейросети~~#  
model.train(train\_images,train\_labels, EPOCH, learning\_rate=LEARNING\_RATE)   
#%%  
model.plot\_learning()

Листинг – загрузка своих изображений и их обработка

def RGB2URGB(img):  
 gray = lambda rgb: np.dot(rgb[..., :3], [0.299, 0.587, 0.114])  
 return (1 - (gray(img).astype("float32") / 255))  
#%%  
PATH = "C:/Users/madro/Documents/GitHub/Khokhlov\_Dimitrii\_20121\_23/AI/P.6/data/test/"  
#%%  
def show(images, labels):  
 for img, label in zip(images, labels):  
 plt.imshow(np.array([img]).reshape(28,28), cmap = 'grey')  
 predict = model(np.array([img]))[0].argmax()  
 color = "red" if predict != label[0] else "blue"  
 plt.title(f"Real:{label[0]} Predicted:{predict}", color=color)  
 plt.show()  
#%%  
  
#%%  
images=[]  
for i in range(1, len(os.listdir(PATH)) + 1):  
 img = plt.imread(f"{PATH}test{i}.jpg",format = "jpeg")  
 images.append(RGB2URGB(img).reshape(784))  
labels = np.array([9,9,7,9,3,4,5,2,7,8,1,6]).reshape(-1,1)  
show(images, labels)