

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

ОТЧЕТ
по лабораторной работе №4
"Распознавание рукописных символов"
по дисциплине «Искусственные нейронные сети»

Студентка гр. 8382

Ивлева О.А.

Преподаватель

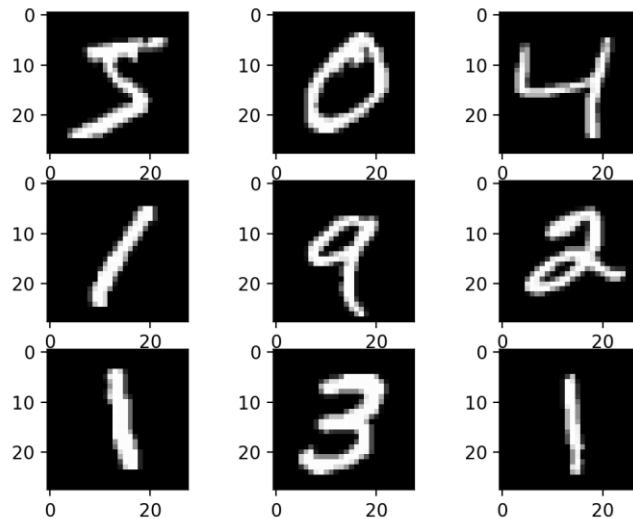
Жангиров Т.Р.

Санкт-Петербург

2021

Цель.

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



Задание.

- 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

Выполнение работы.

Работа выполнялась на базе операционной системы Windows 10 в среде разработки PyCharm.

Реализация загрузки пользовательского изображения.

Изображение загружается как объект Image, предоставляемый библиотекой PIL. Изображение загружается в черно-белом формате. Применяется метод `resize`, который изменяет разрешение на 28×28 .

```
def load_image(path):  
    image = Image.open(path)  
    image = image.resize((28, 28))  
    image = np.dot(np.asarray(image), np.array([1 / 3, 1 / 3, 1 / 3]))  
    image /= 255  
    image = 1 - image  
    image = image.reshape((1, 28, 28))  
    return image
```

Создание модели, параметры обучения

Была создана модель искусственной нейронной сети с помощью Keras и настроены параметры обучения.

Для оценки модели ИНС и исследования влияния различных оптимизаторов, а также их параметров были построены графики ошибок и точности.

Исследование влияния различных оптимизаторов:

1. Оптимизатор SGD

Стохастический градиентный спуск. Менялись следующие параметры: скорость обучения и момент.

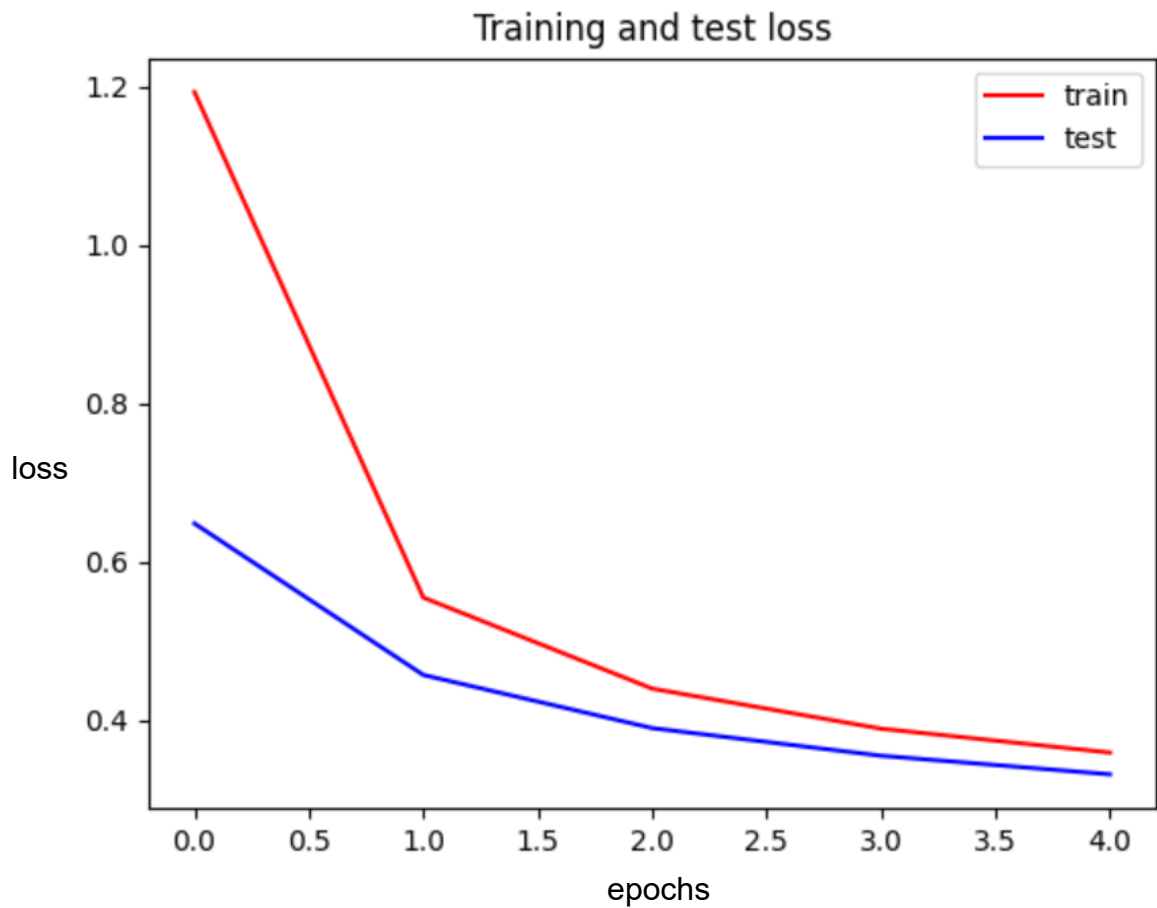


Рисунок 1 – График ошибок модели модели с оптимизатором SGD, с параметрами $\text{learning_rate} = 0.01$, $\text{momentum} = 0.0$

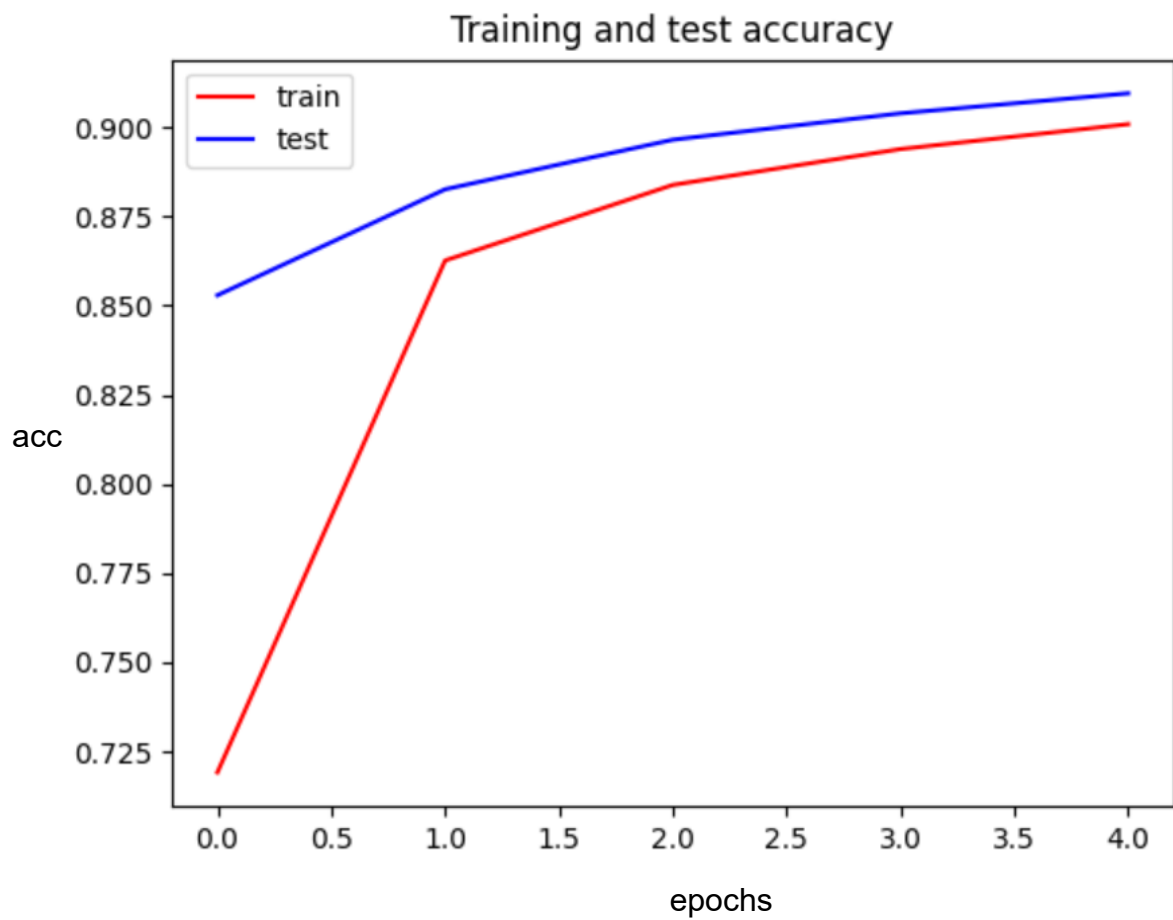


Рисунок 2 – График точности модели модели с оптимизатором SGD, с параметрами $\text{learning_rate} = 0.01$, $\text{momentum} = 0.0$

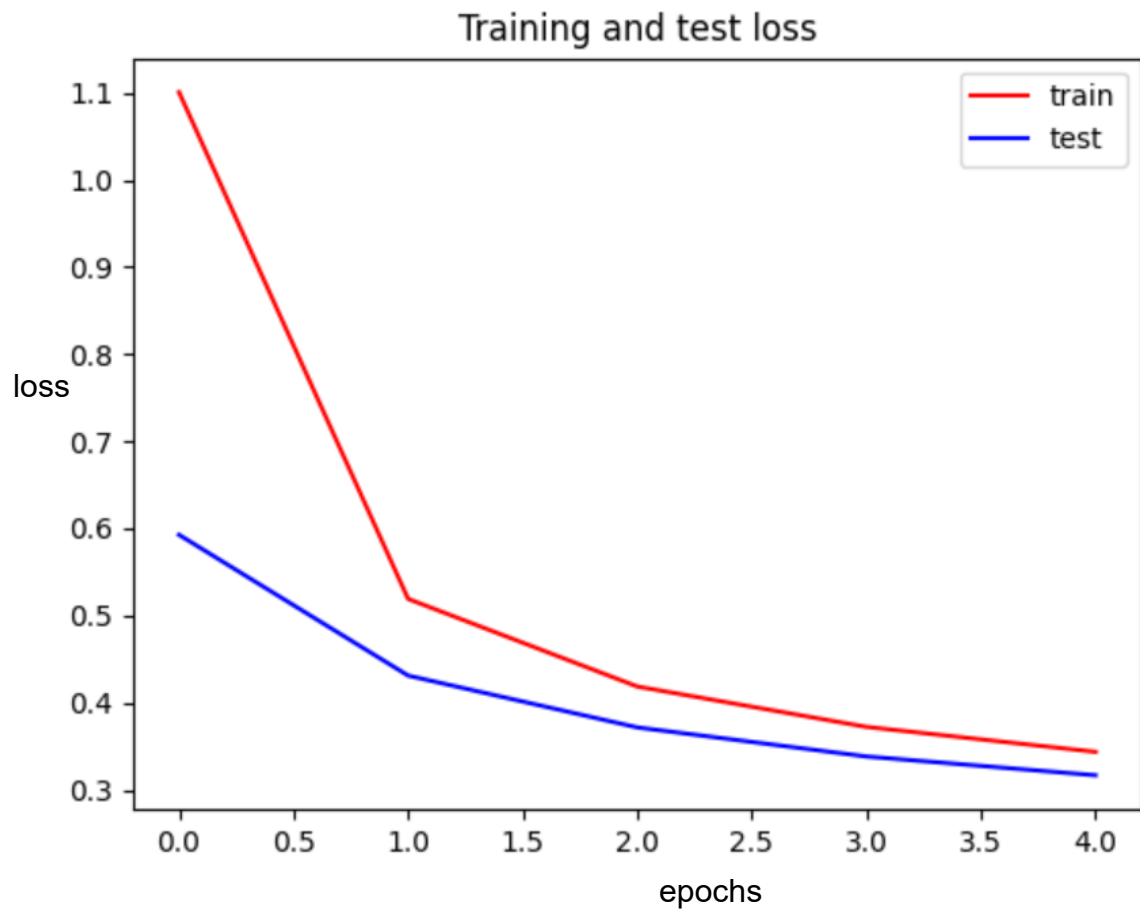


Рисунок 3 – График ошибок модели модели с оптимизатором SGD, с параметрами $\text{learning_rate} = 0.001$, $\text{momentum} = 0.1$

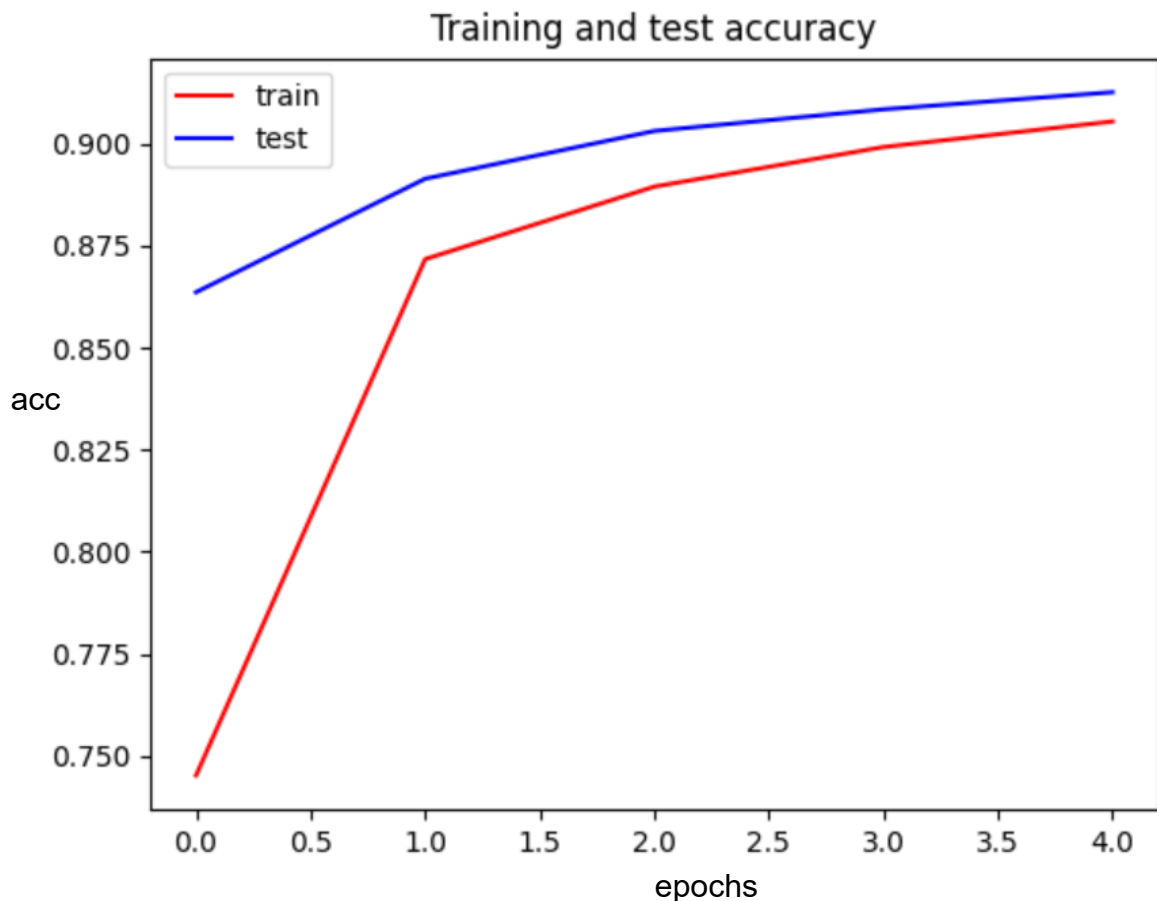


Рисунок 4 – График точности модели модели с оптимизатором SGD, с параметрами `learning_rate = 0.001`, `momentum = 0.1`

Максимальная точность модели с данным оптимизатором при различных параметрах - test_acc: 0.9125999808311462

2. Оптимизатор RMSprop

Оптимизатор делит скорость обучения для веса на скользящее среднее значение последних градиентов этого веса. Менялись следующие параметры: скорость обучения.

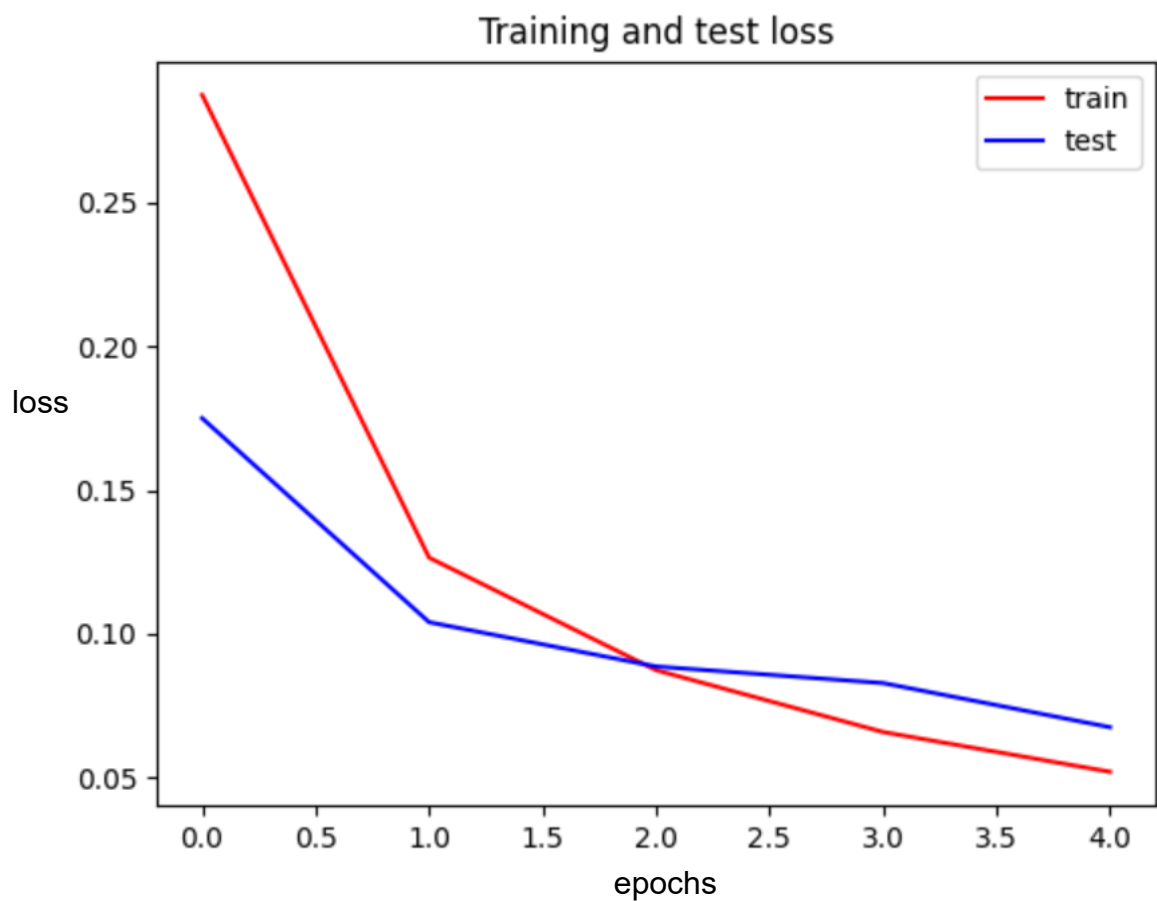


Рисунок 5 – График ошибок модели модели с оптимизатором RMSprop, с параметрами $\text{learning_rate} = 0.001$

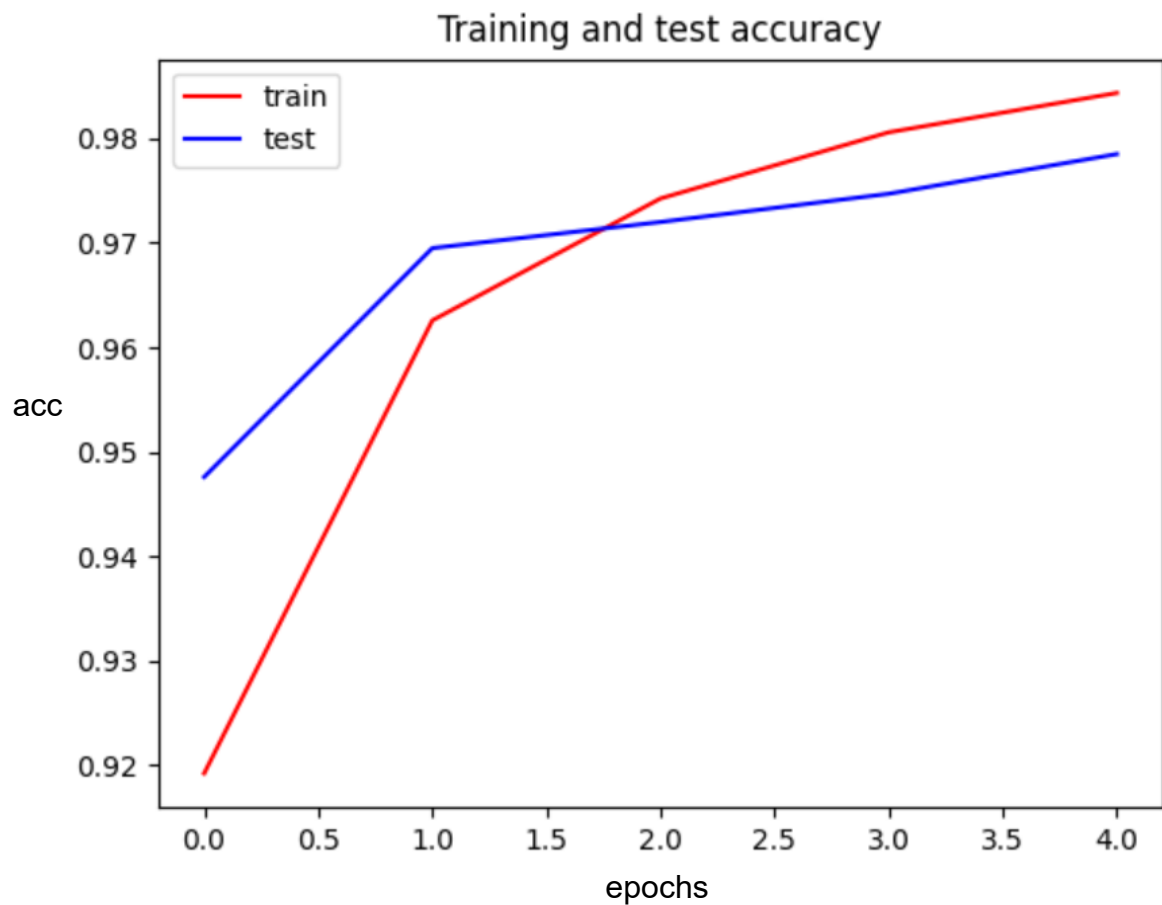


Рисунок 6 – График точности модели модели с оптимизатором RMSprop, с параметрами $\text{learning_rate} = 0.001$

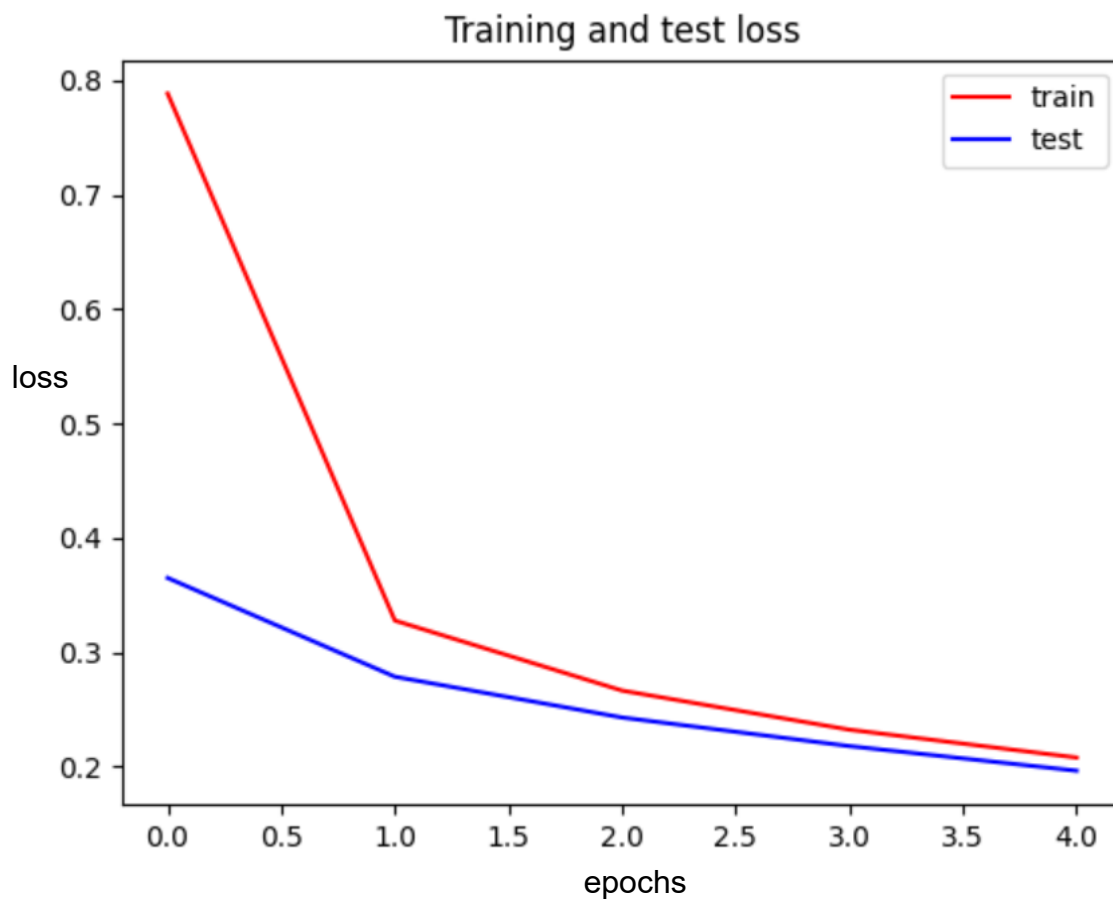


Рисунок 7 – График ошибок модели модели с оптимизатором RMSprop, с параметрами $\text{learning_rate} = 0.0001$

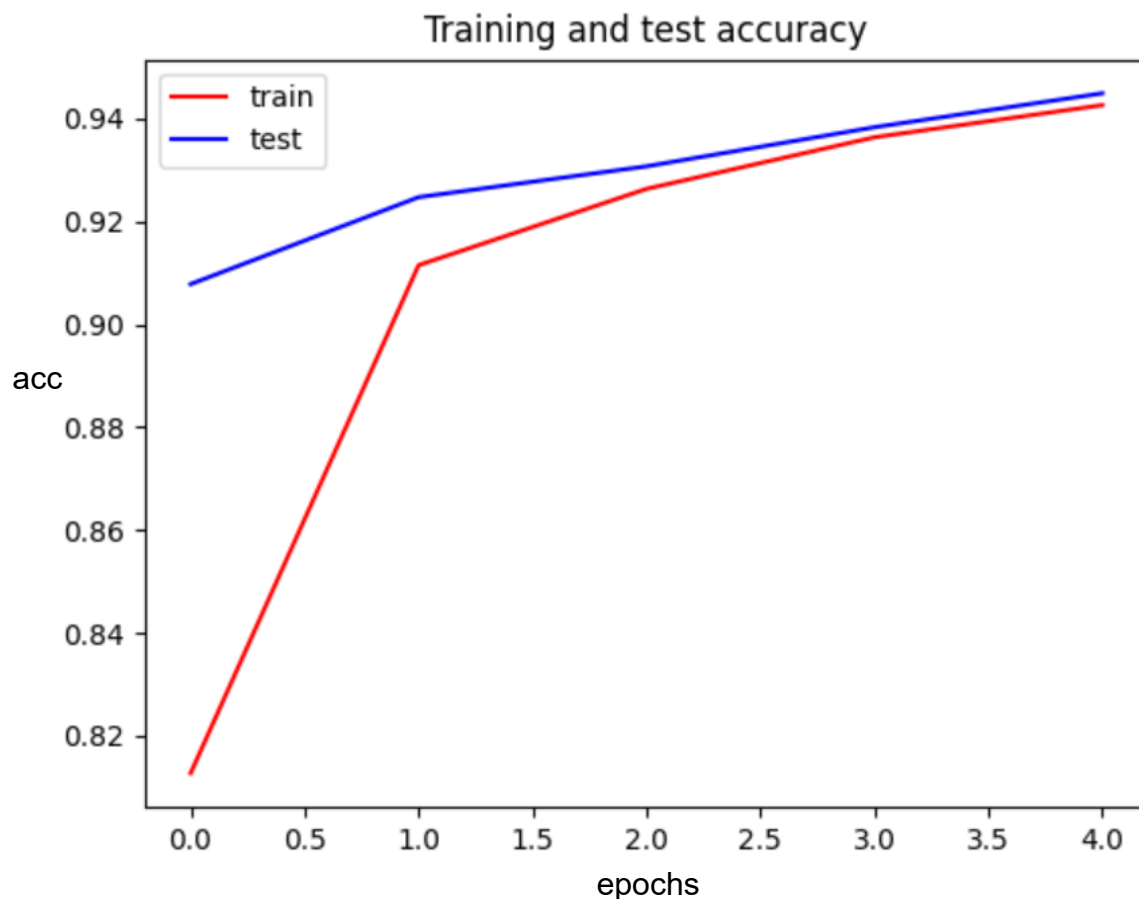


Рисунок 8 – График точности модели модели с оптимизатором RMSprop, с параметрами $\text{learning_rate} = 0.0001$

Максимальная точность модели с данным оптимизатором при различных параметрах = 0.9785000085830688

3. Оптимизатор Adagrad

Для этого оптимизатора скорость обучения параметра (веса) зависит от частоты его обновления: чем чаще обновляется параметр, тем меньше скорость его обучения. Менялись следующие параметры: скорость обучения.

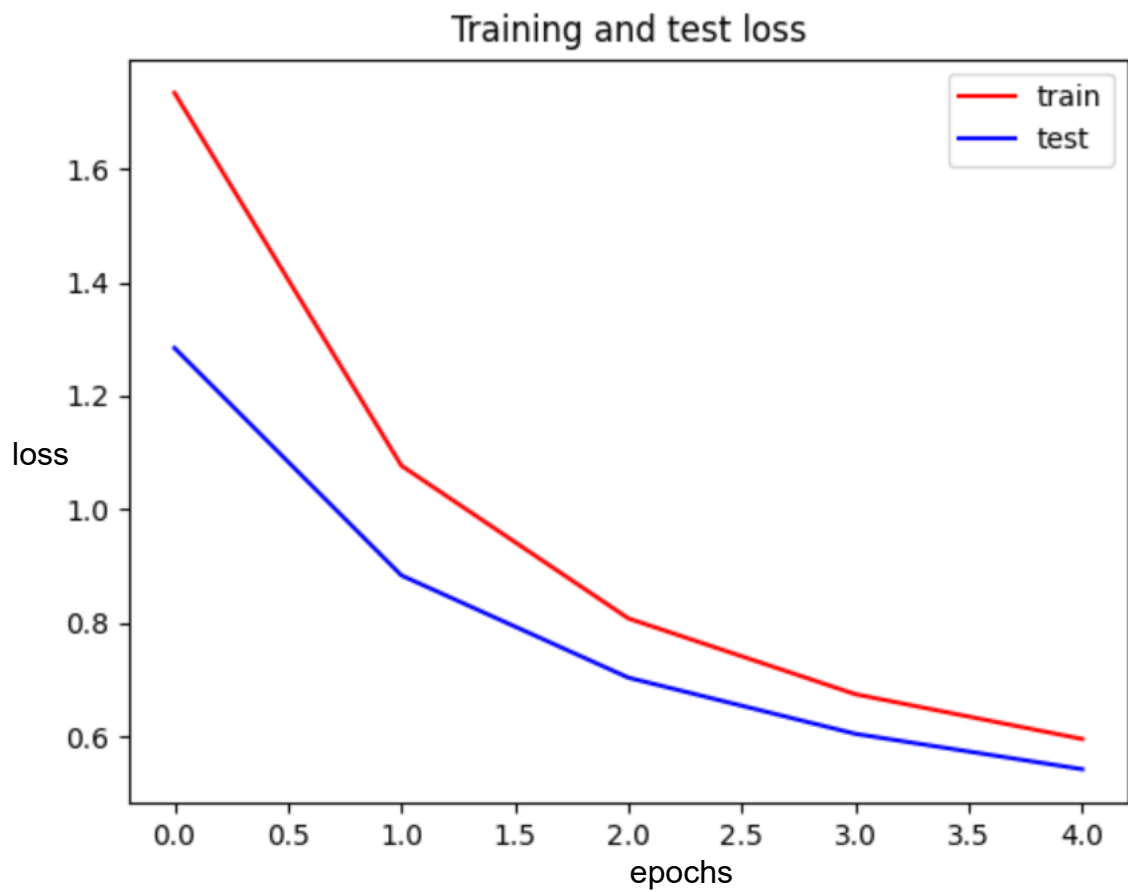


Рисунок 9 – График ошибок модели модели с оптимизатором Adagrad, с параметрами $\text{learning_rate} = 0.001$

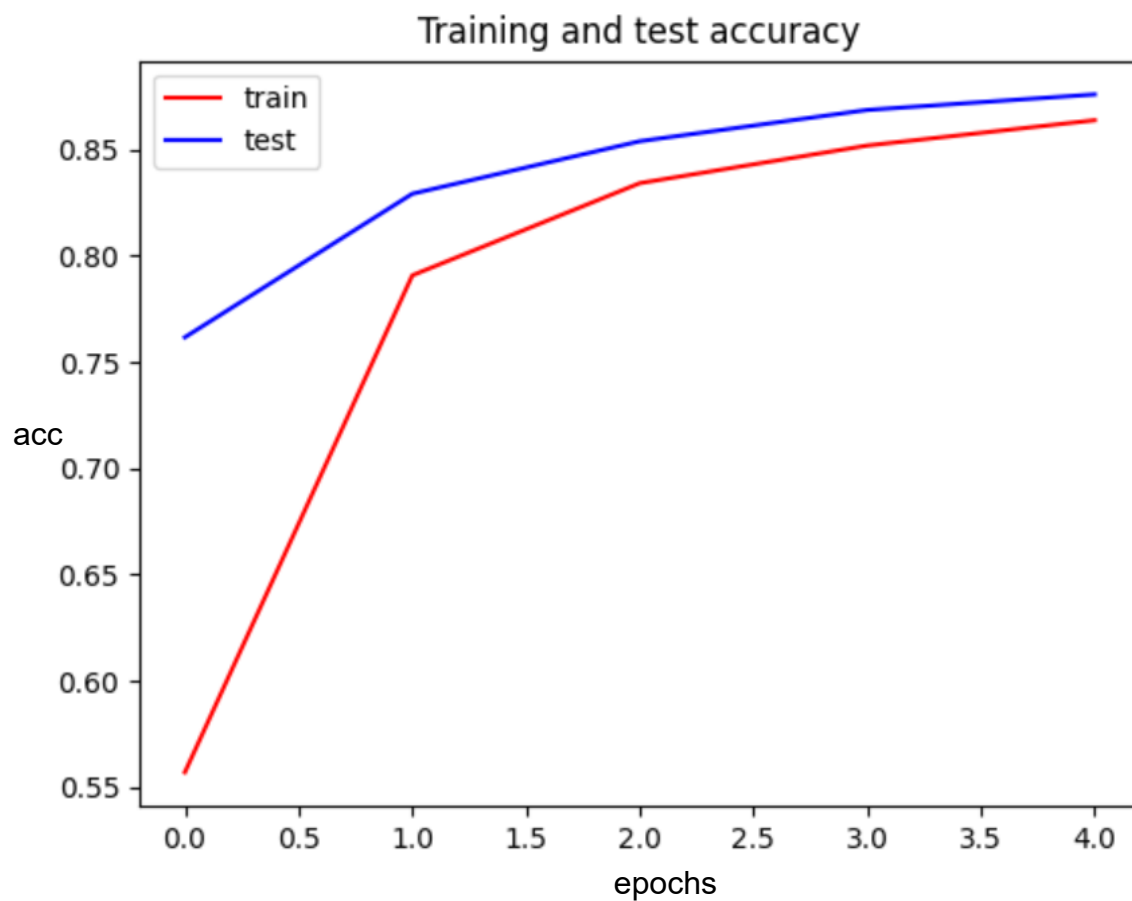


Рисунок 10 – График точности модели модели с оптимизатором Adagrad, с параметрами $\text{learning_rate} = 0.001$

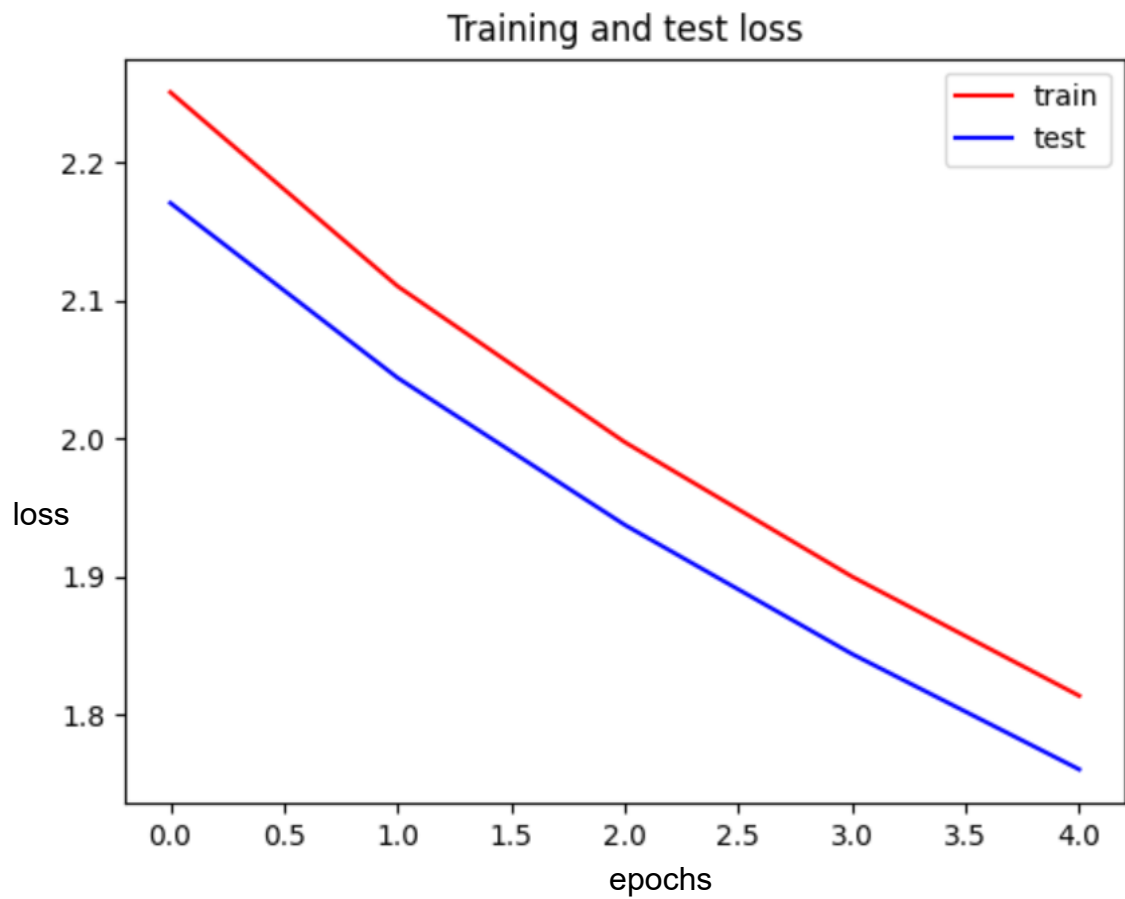


Рисунок 11 – График ошибок модели модели с оптимизатором Adagrad, с параметрами $\text{learning_rate} = 0.0001$

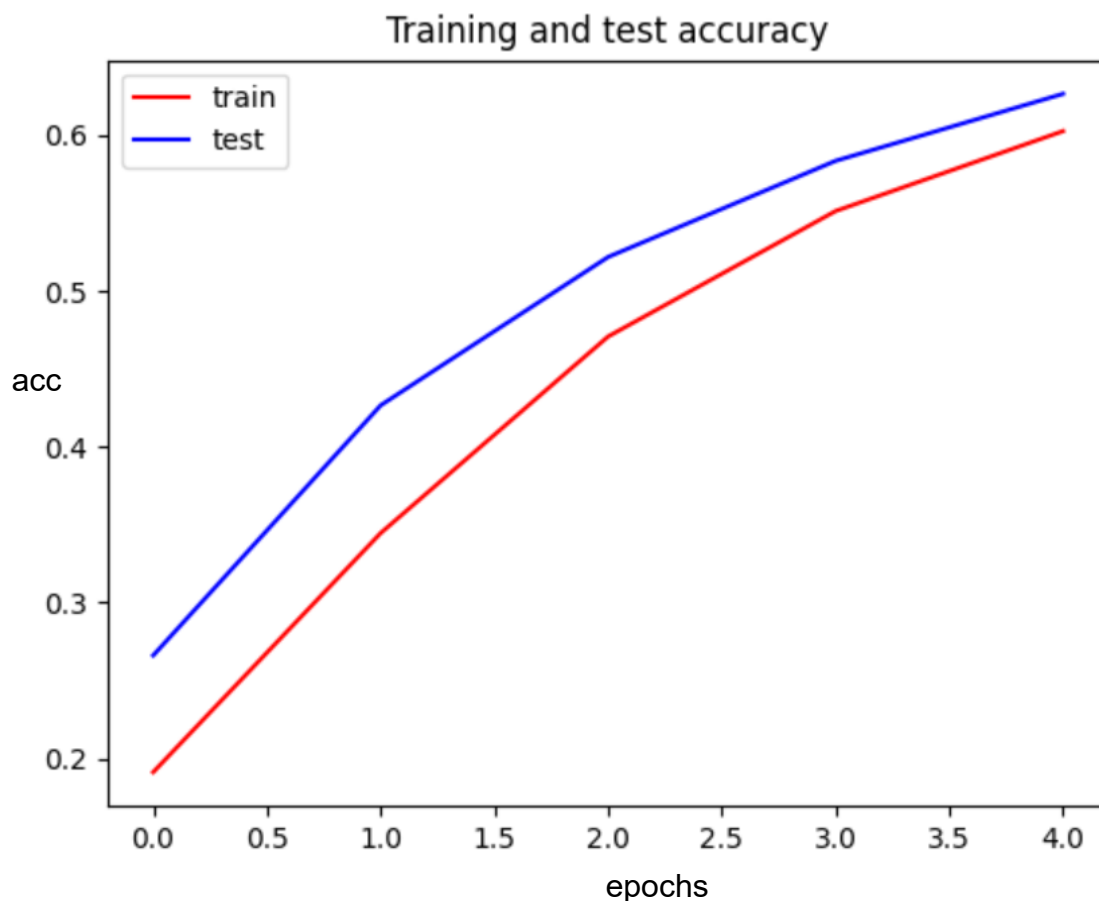


Рисунок 12 – График точности модели модели с оптимизатором Adagrad, с параметрами $\text{learning_rate} = 0.0001$

Максимальная точность модели с данным оптимизатором при различных параметрах = 0.8758000135421753

4. Оптимизатор Adam

Вариант стохастической оптимизации. Менялись следующие параметры: скорость обучения, beta_1 , beta_2 .

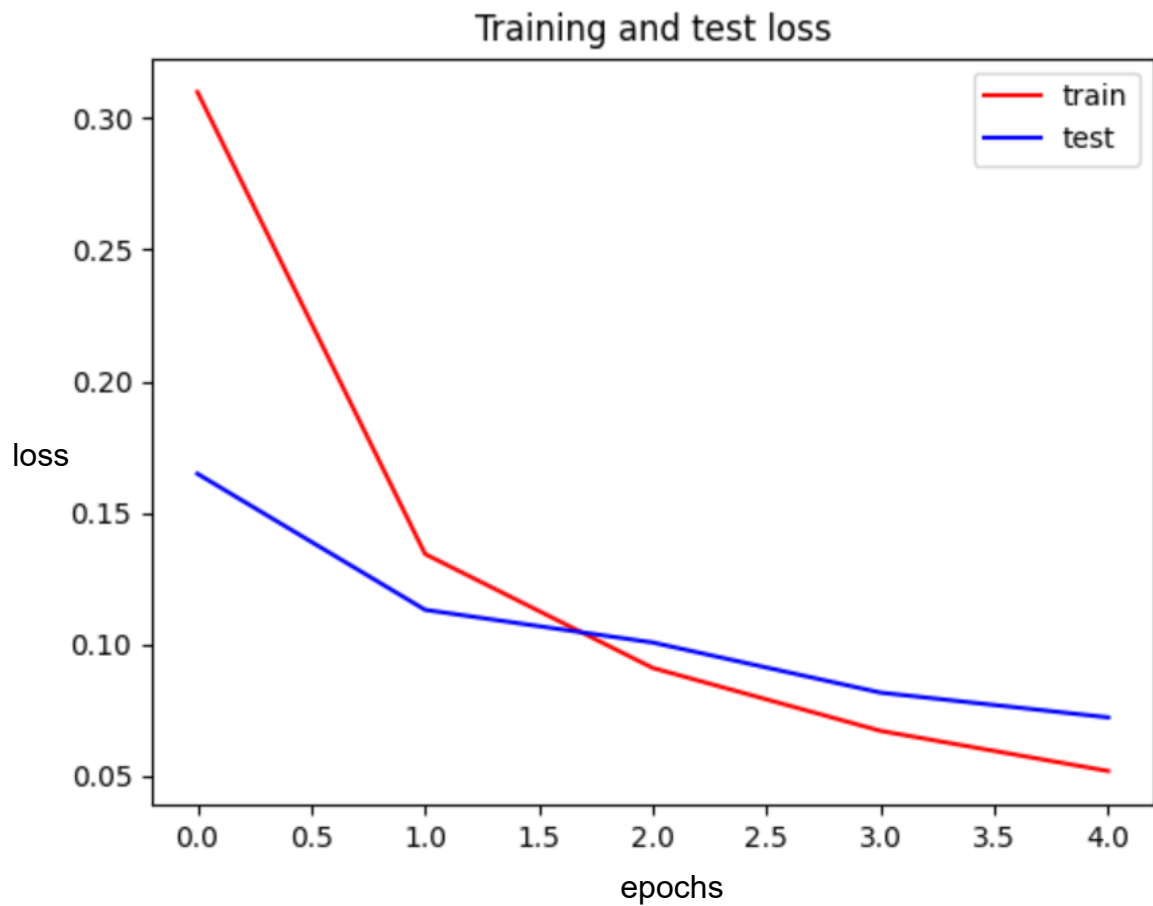


Рисунок 13 – График ошибок модели модели с оптимизатором Adam, с параметрами $\text{learning_rate}=0.001$, $\text{beta_1}=0.9$, $\text{beta_2}=0.999$

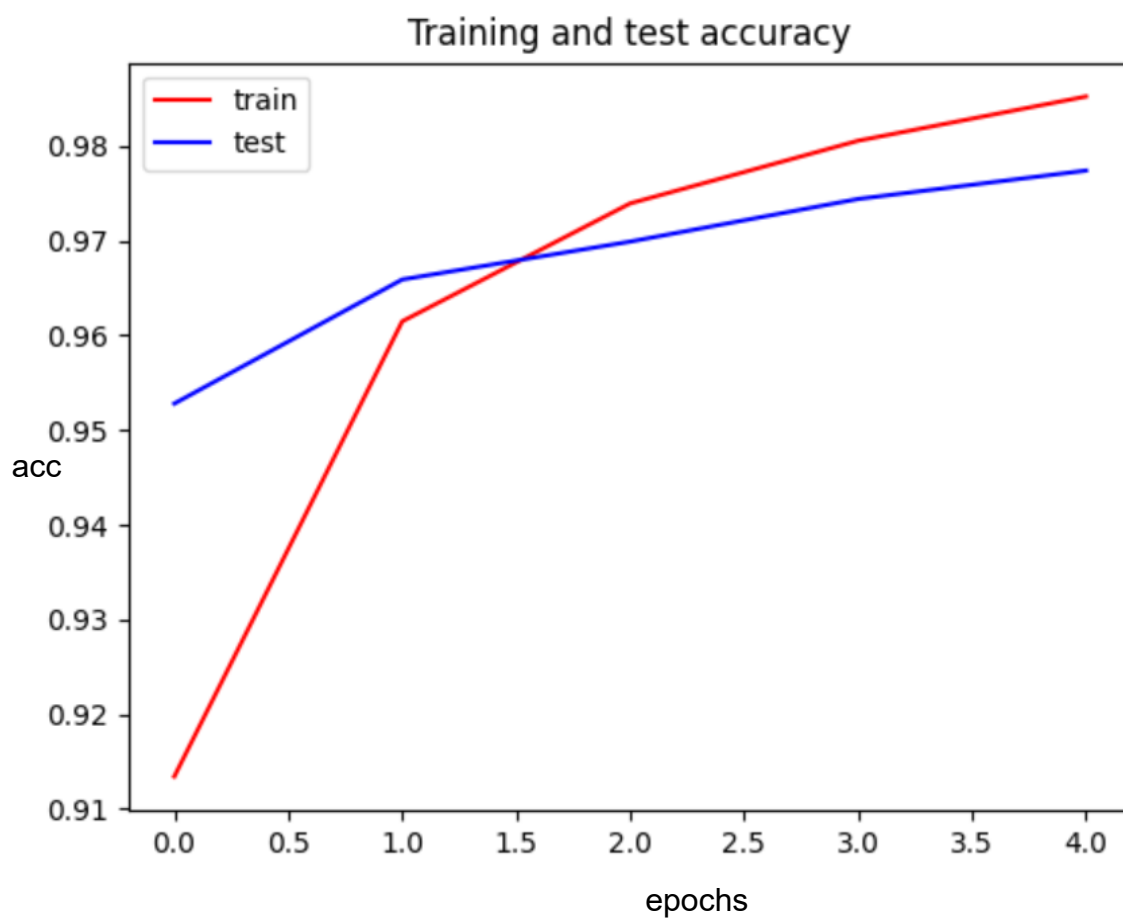


Рисунок 14 – График точности модели модели с оптимизатором Adam, с параметрами $\text{learning_rate}=0.001$, $\text{beta_1}=0.9$, $\text{beta_2}=0.999$

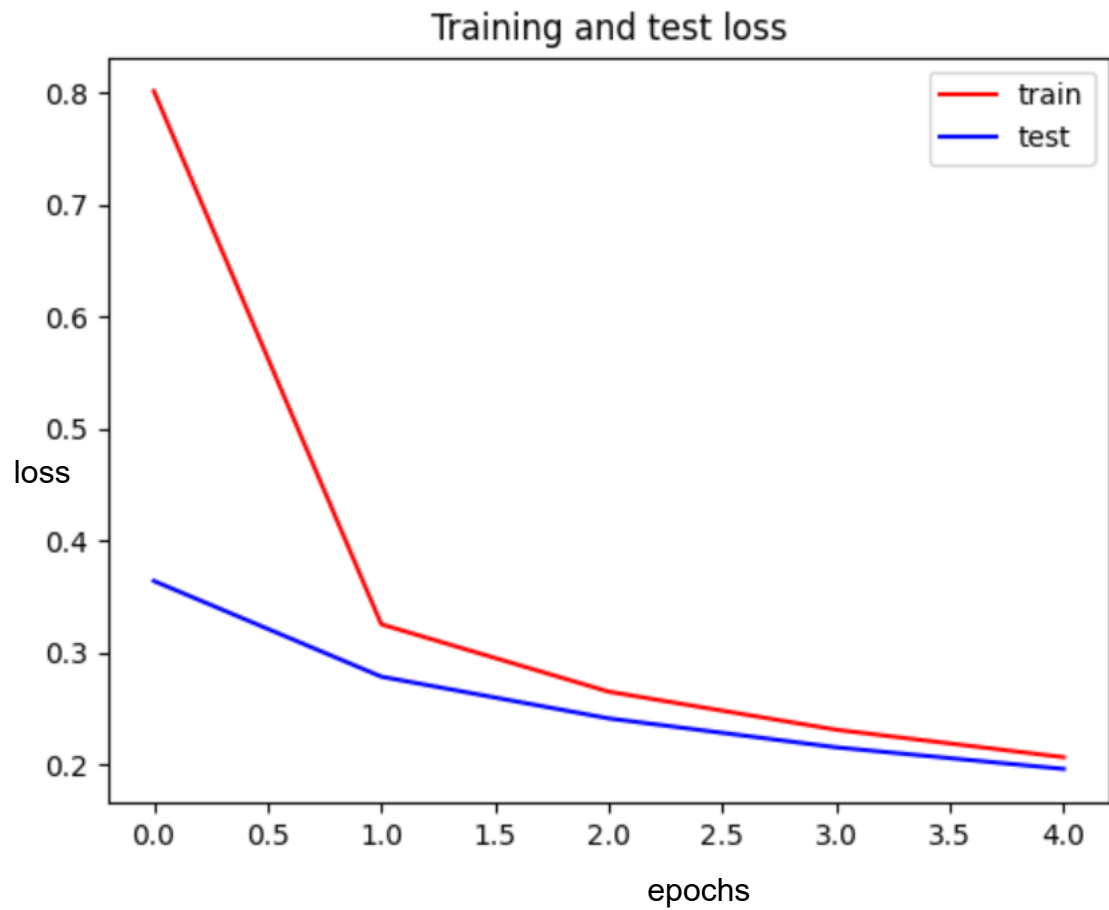


Рисунок 15 – График ошибок модели модели с оптимизатором Adam, с параметрами $\text{learning_rate}=0.0001$, $\text{beta_1}=0.8$, $\text{beta_2}=0.9$

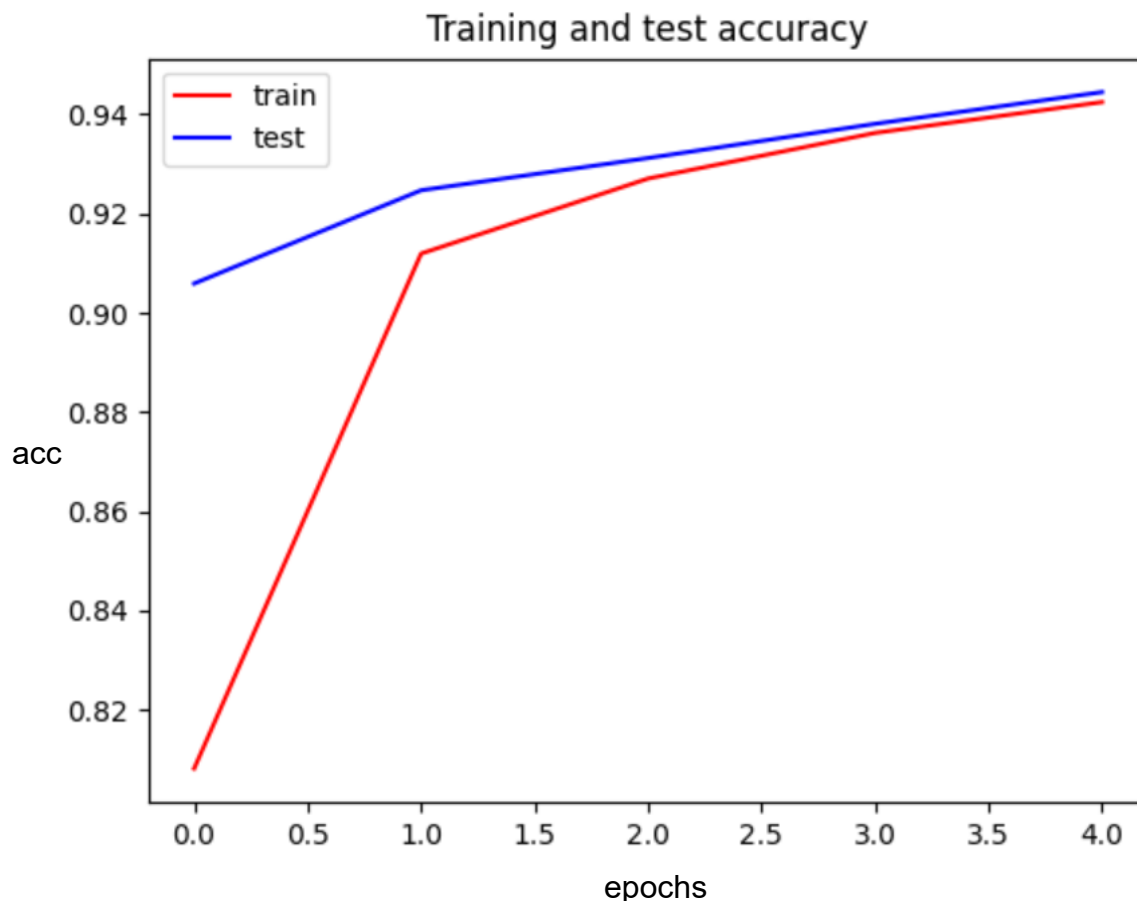


Рисунок 16 – График ошибок модели модели с оптимизатором Adam, с параметрами $\text{learning_rate}=0.0001$, $\text{beta_1}=0.8$, $\text{beta_2}=0.9$

Максимальная точность модели с данным оптимизатором при различных параметрах = 0.978400004863739

Исходя из полученных результатов можно сделать вывод о том, что для данной задачи лучше всего использовать оптимизатор Adam с параметрами $\text{learning_rate}=0.001$, $\text{beta_1}=0.9$, $\text{beta_2}=0.999$, так как была достигнута максимальная точность модели 0.978400004863739. Для этого оптимизатора с такими параметрами было загружено изображение цифры 6 и подано на вход ИНС – она верно распознала цифру.

Выводы.

В ходе выполнения лабораторной работы было изучено решение задачи классификации небольших черно-белых изображений. Была создана и обучена

модель, которая способна распознавать цифры на изображении. Также было проведено сравнение обучения модели при разных оптимизаторах и их параметрах.

ПРИЛОЖЕНИЕ А

Исходный код программы. Файл lr4.py

```
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
from PIL import Image
from keras.utils import to_categorical
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import *

def build_model(optimizer):
    model = Sequential()
    model.add(Flatten())
    model.add(Dense(256, activation='relu'))
    model.add(Dense(10, activation='softmax'))
    model.compile(optimizer=optimizer, loss='categorical_crossentropy',
metrics=['accuracy'])
    return model

def load_image(path):
    image = Image.open(path)
    image = image.resize((28, 28))
    image = np.dot(np.asarray(image), np.array([1 / 3, 1 / 3, 1 / 3]))
    image /= 255
    image = 1 - image
    image = image.reshape((1, 28, 28))
    return image

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

train_images = train_images / 255.0
test_images = test_images / 255.0

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

model = build_model(Adam(learning_rate=0.0001, beta_1=0.8, beta_2=0.999))
# model = build_model(Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999))
history = model.fit(train_images, train_labels, epochs=5, batch_size=128,
validation_data=(test_images, test_labels))
```

```
plt.title('Training and test loss')
plt.plot(history.history['loss'], 'r', label='train')
plt.plot(history.history['val_loss'], 'b', label='test')
plt.legend()
plt.show()
plt.clf()
plt.title('Training and test accuracy')
plt.plot(history.history['accuracy'], 'r', label='train')
plt.plot(history.history['val_accuracy'], 'b', label='test')
plt.legend()
plt.show()
plt.clf()

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

image = load_image('6.png')
res = model.predict(image)
print(np.argmax(res))
```