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

ЛАБОРАТОРНАЯ РАБОТА №4

по дисциплине «Искусственные нейронные сети»

Тема: «Распознавание рукописных символов»

Студент гр. 7381 _____ Дорох С.В.

Преподаватель _____ Жукова Н. А.

Санкт-Петербург

2020

Цели.

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

Задачи.

- Ознакомиться с представлением графических данных
- Ознакомиться с простейшим способом передачи графических данных нейронной сети
- Создать модель
- Настроить параметры обучения
- 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.

Для решения данной задачи была выбрана архитектура модели, состоящая из 2 слоев:

```
Dense(512, activation='relu', input_shape=(28 * 28,))
Dense(10, activation='softmax')
```

Исходный размерность датасета была изменена с (60000, 28, 28) на (60000, 28*28).

Для возможности загрузки пользовательских изображений была написана функция:

```
def upload_image(filepath):
    img = Image.open(fp=filepath)
    img = np.asarray(img)
    img = img.resize((28, 28))
```

```
    k = np.array([[[0.2989, 0.587, 0.114]]])
    img = np.sum(img * k, axis=2).reshape((1, 28 * 28)) / 255.0
    return img
```

Эта функция загружает изображение, затем сжимает или растягивает его под формат изображений (28, 28), переводит его из 3-х канального изображения в одноканальное и переводит в одномерный массив размера 28*28.

В ходе работы были сравнены следующие оптимизаторы:

SGD - стохастический градиентный спуск;

Adam - алгоритм для градиентной оптимизации стохастических целевых функций первого порядка, основанный на адаптивных оценках моментов более низкого порядка;

RMSProp – алгоритм сочетает в себе идею использования только знака градиента с идеей адаптации размера шага.

Ниже можно увидеть графики функций точности и потерь вышеперечисленных оптимизаторов с различными параметрами:

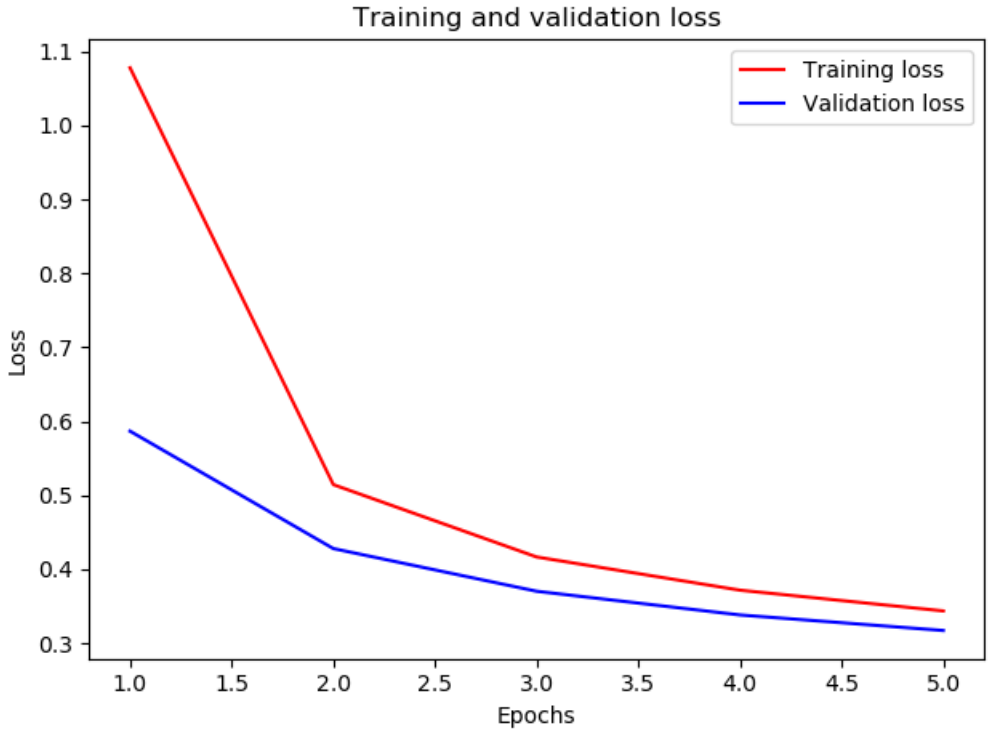

Рисунок 1 – график функции потерь модели с оптимизатором SGD с дефолтными параметрами

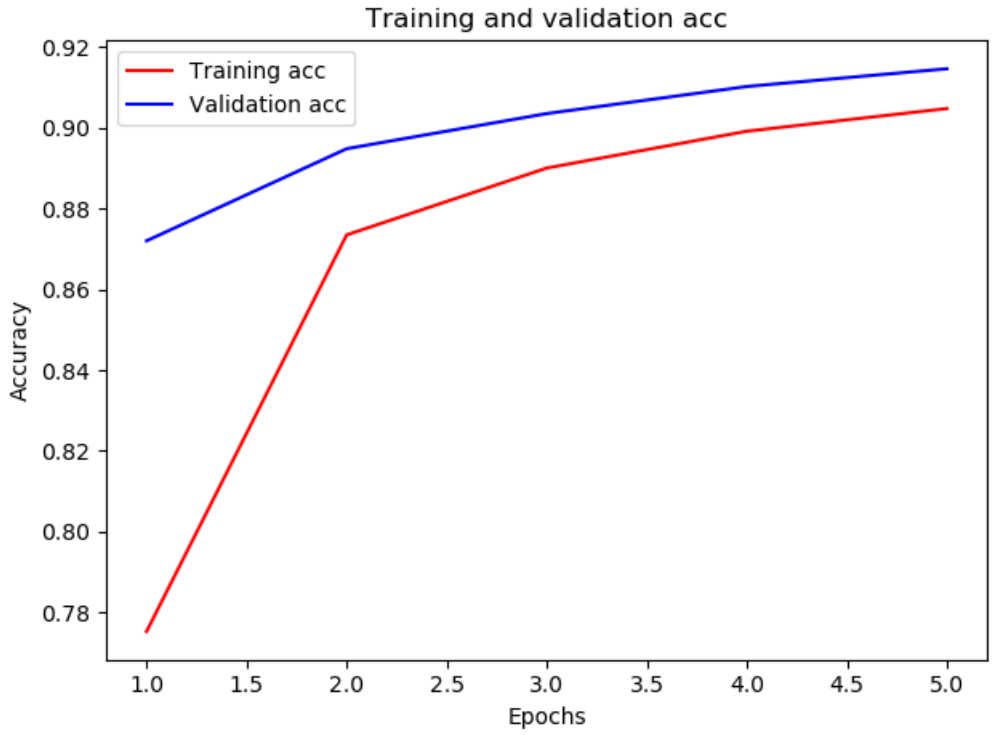

Рисунок 2 – график функции точности модели с оптимизатором SGD с дефолтными параметрами

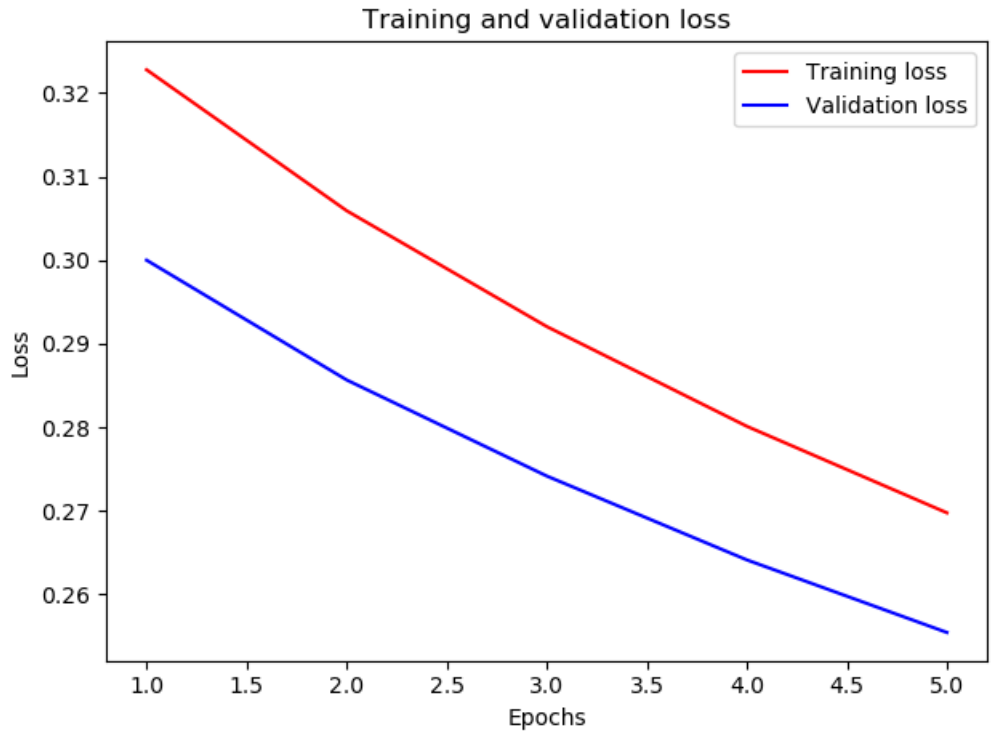

Рисунок 3 – график функции потерь модели с оптимизатором SGD с параметрами learning_rate=0.01, momentum=0.1

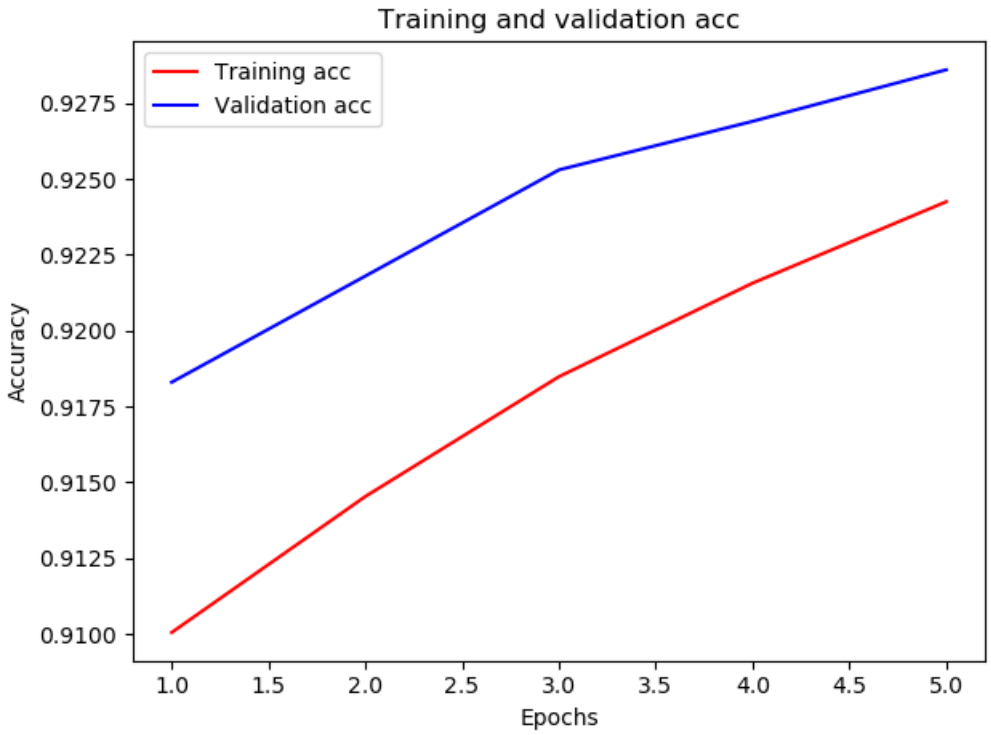

Рисунок 4 – график функции точности модели с оптимизатором SGD с параметрами learning_rate=0.01, momentum=0.1

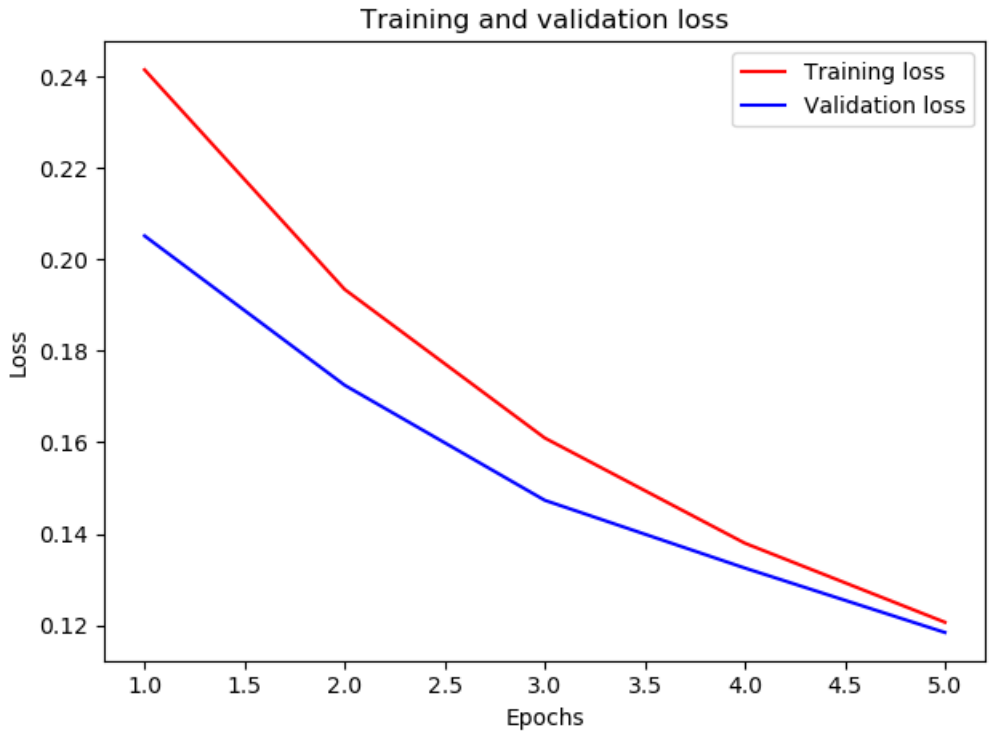

Рисунок 5 – график функции потерь модели с оптимизатором SGD с параметрами learning_rate=0.01, momentum=0.9

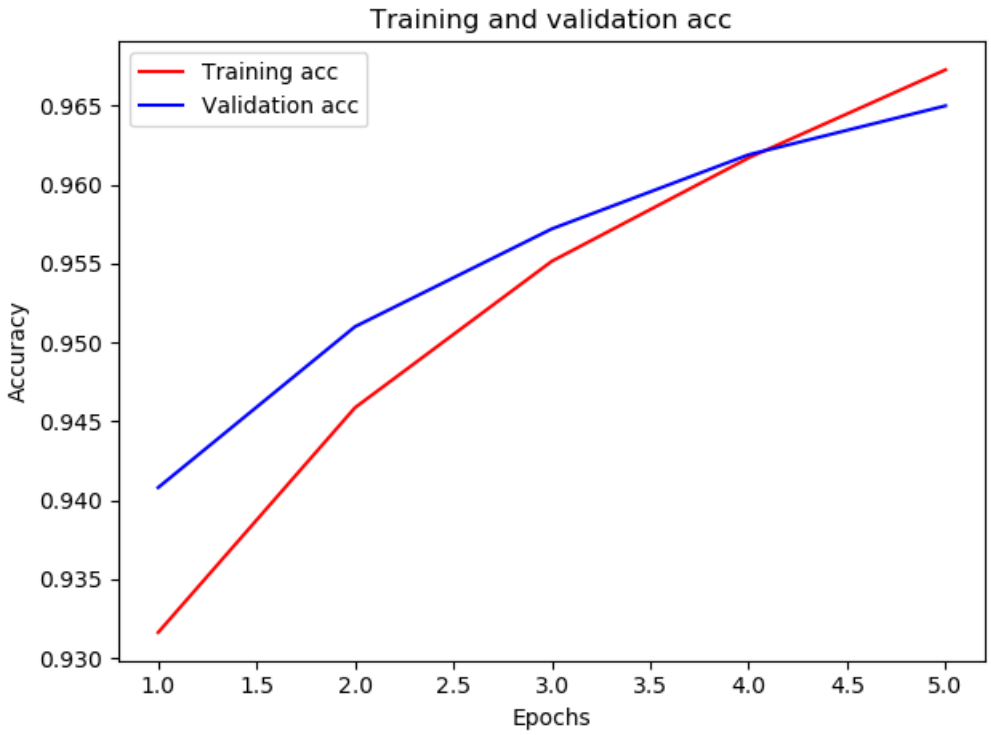

Рисунок 6 – график функции точности модели с оптимизатором SGD с параметрами learning_rate=0.01, momentum=0.9

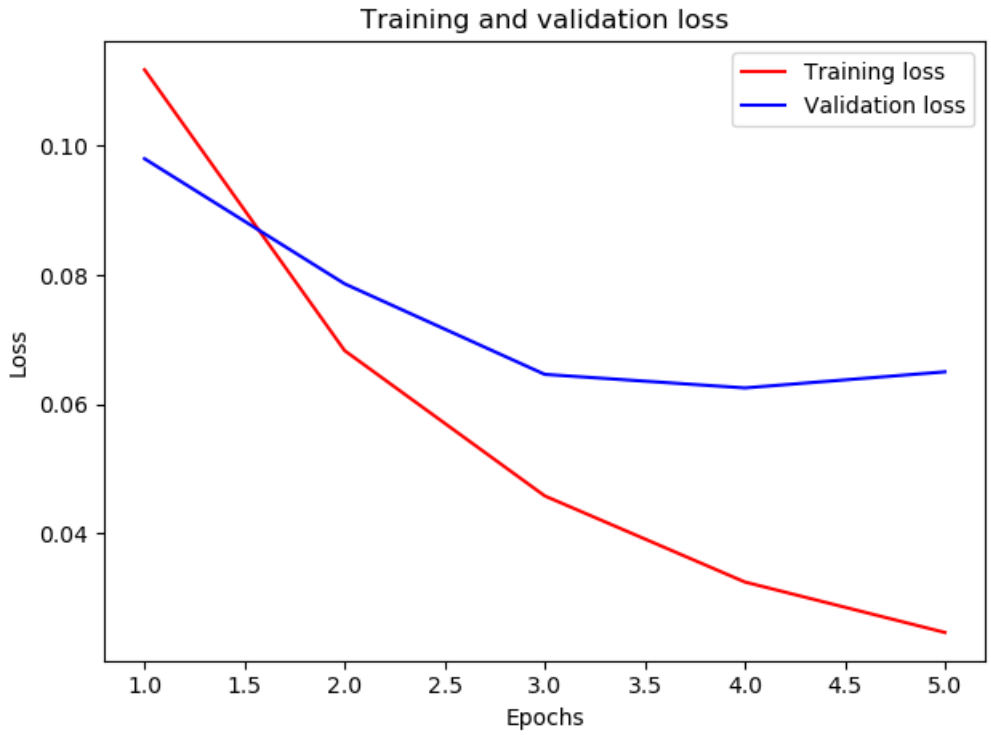

Рисунок 7 – график функции потери модели с оптимизатором Adam с дефолтными параметрами

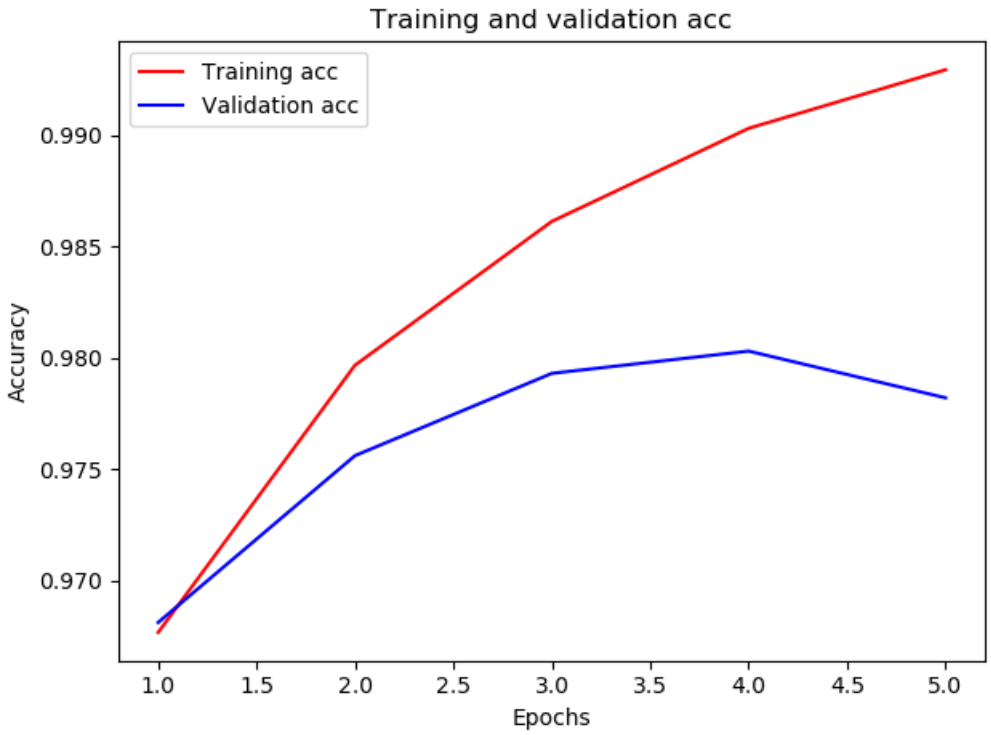

Рисунок 8 – график функции точности модели с оптимизатором Adam с дефолтными параметрами

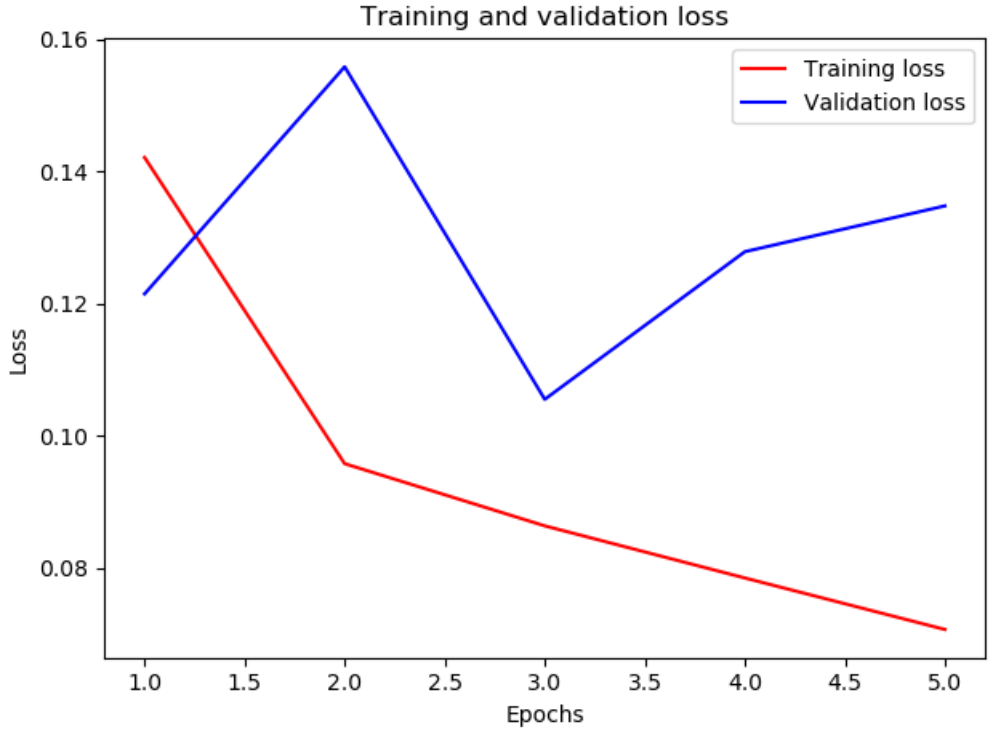

Рисунок 9 – график функции потери модели с оптимизатором Adam с параметрами learning_rate=0.01

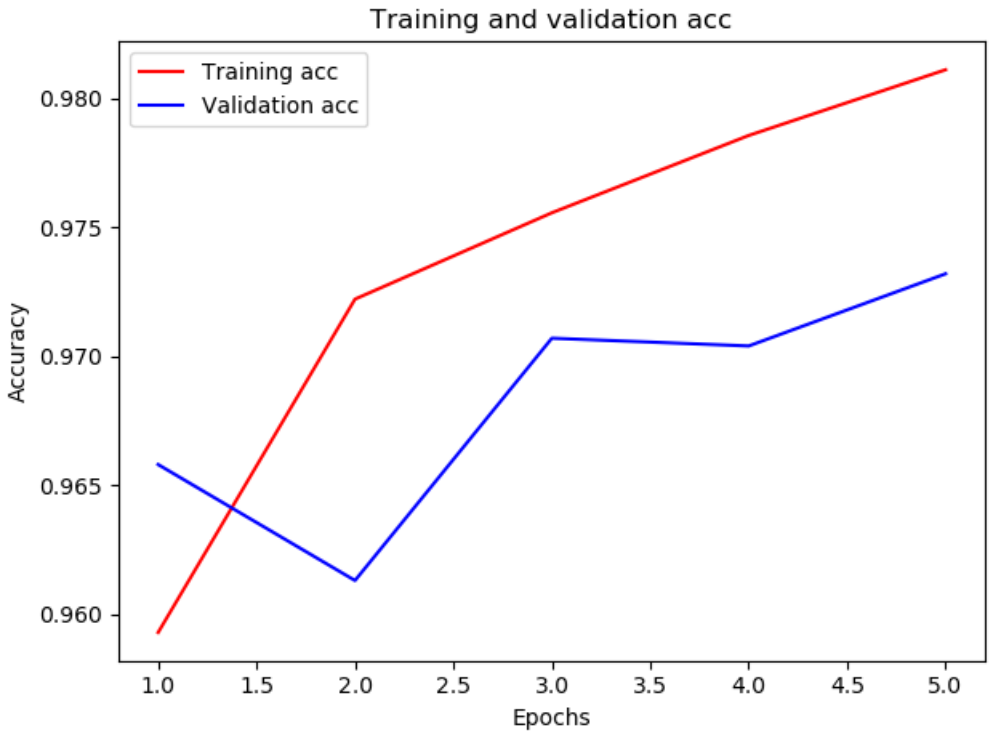

Рисунок 10 – график функции точности модели с оптимизатором Adam с параметрами learning_rate=0.01

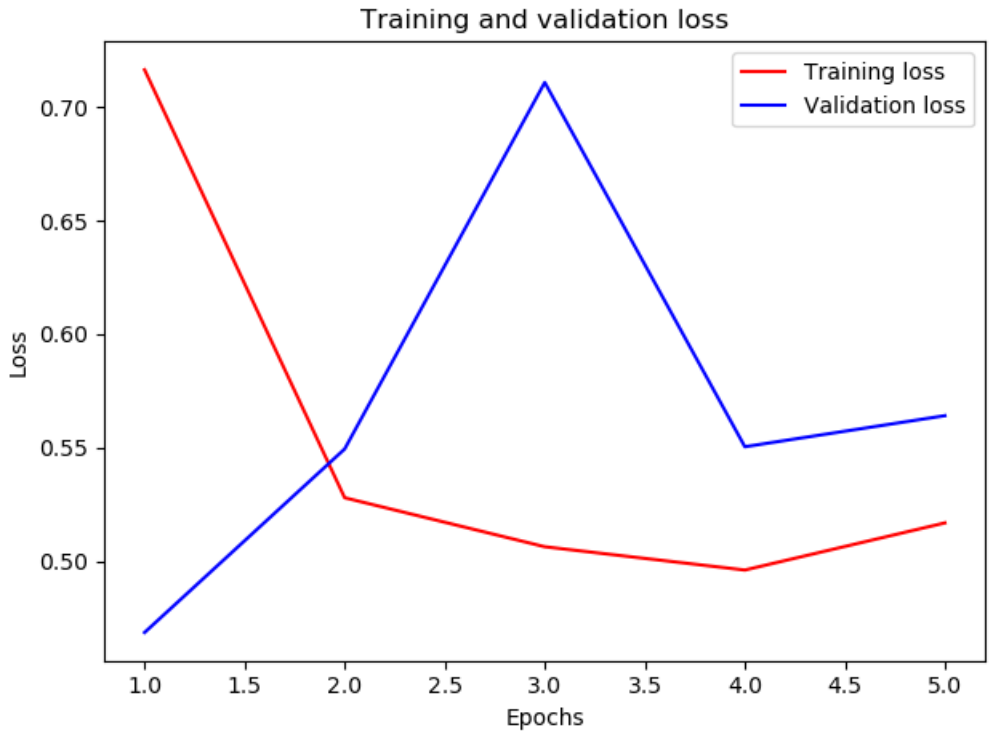

Рисунок 11 – график функции точности модели с оптимизатором Adam с параметрами learning_rate=0.1

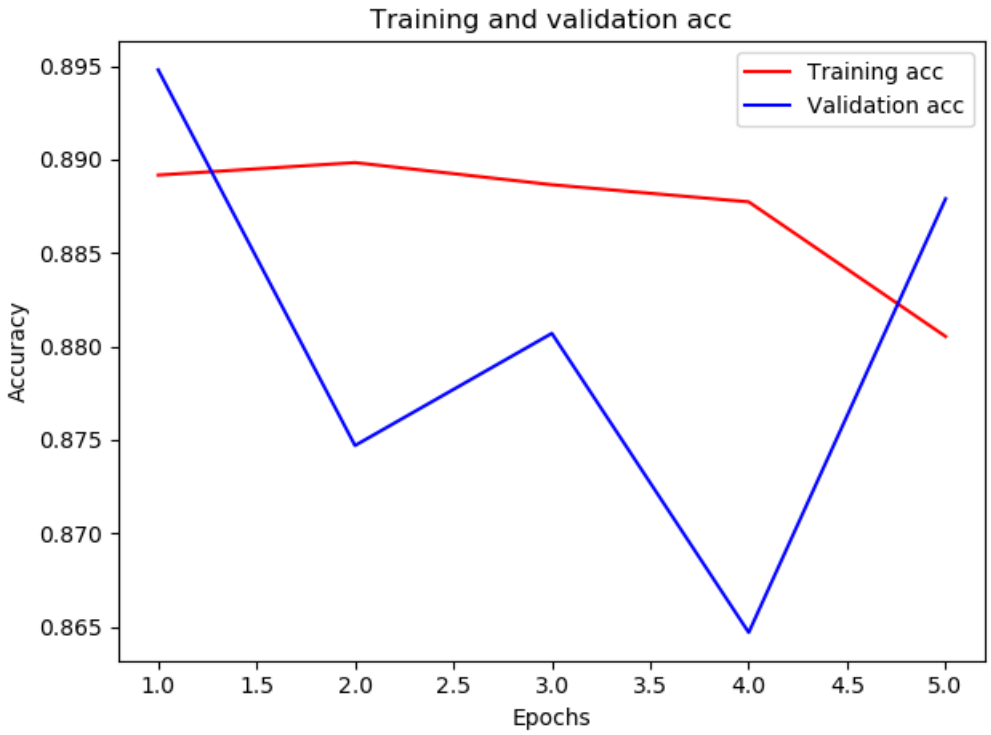

Рисунок 12 – график функции точности модели с оптимизатором Adam с параметрами learning_rate=0.1

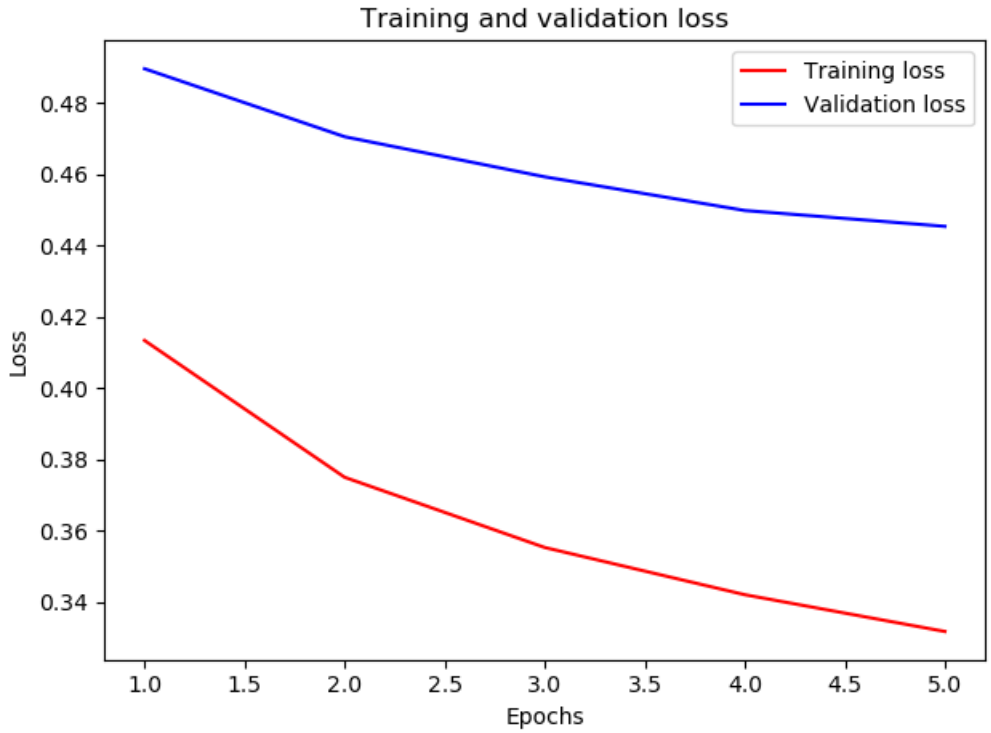

Рисунок 13 – график функции потери модели с оптимизатором RMSProp с дефолтными параметрами

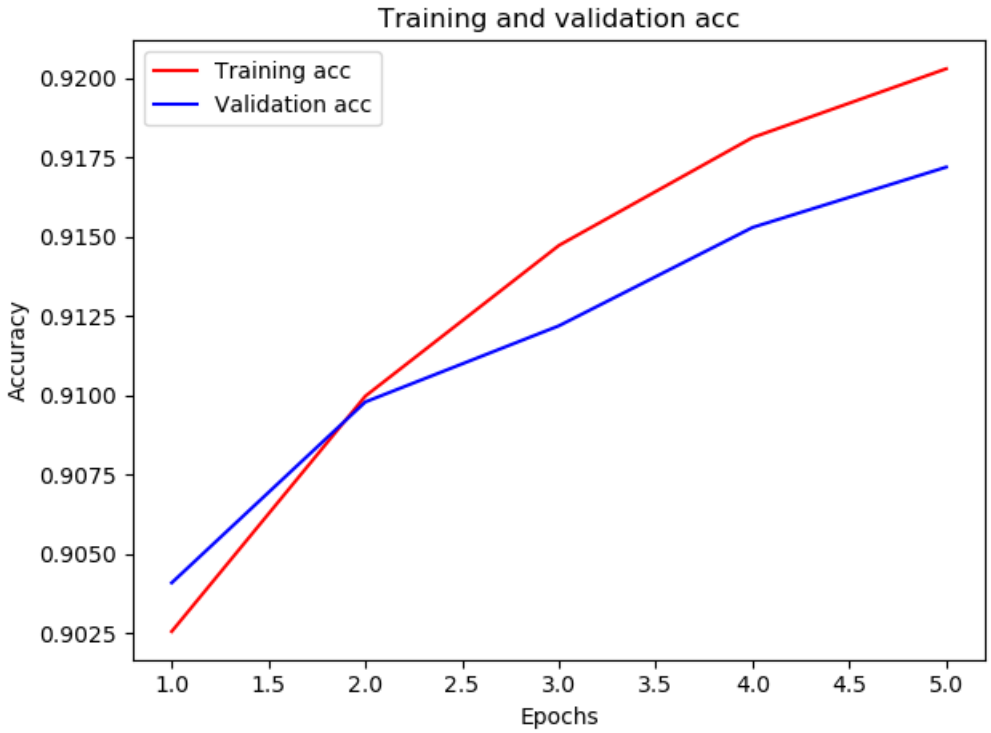

Рисунок 14 – график функции точности модели с оптимизатором RMSProp с дефолтными параметрами

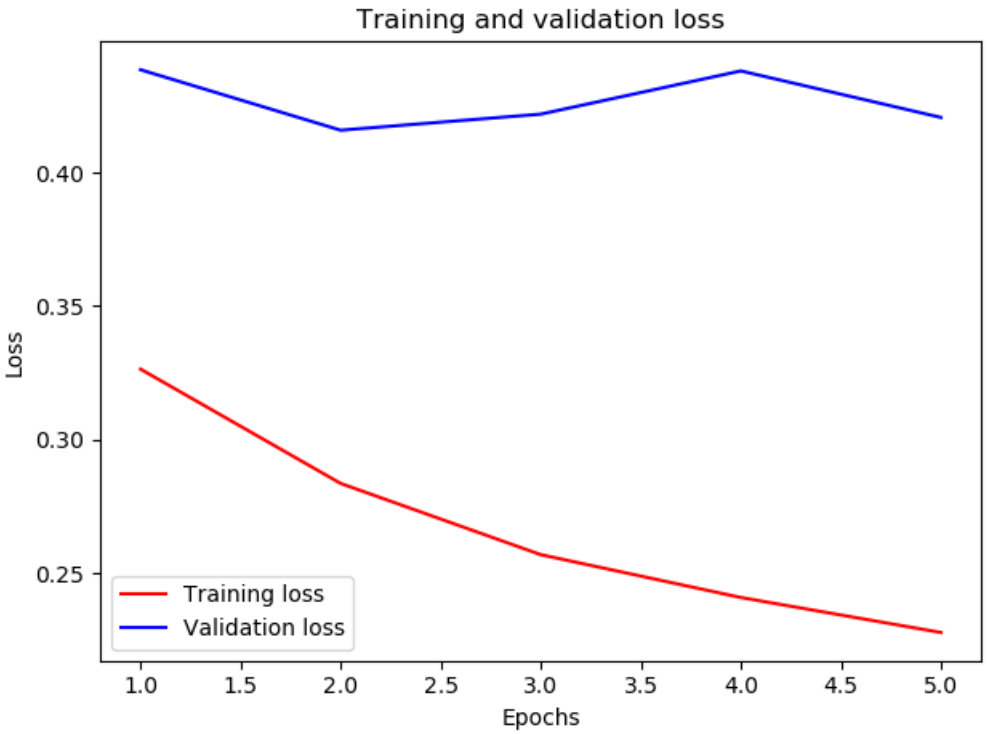

Рисунок 15 – график функции потери модели с оптимизатором RMSProp с параметрами learning_rate=0.01

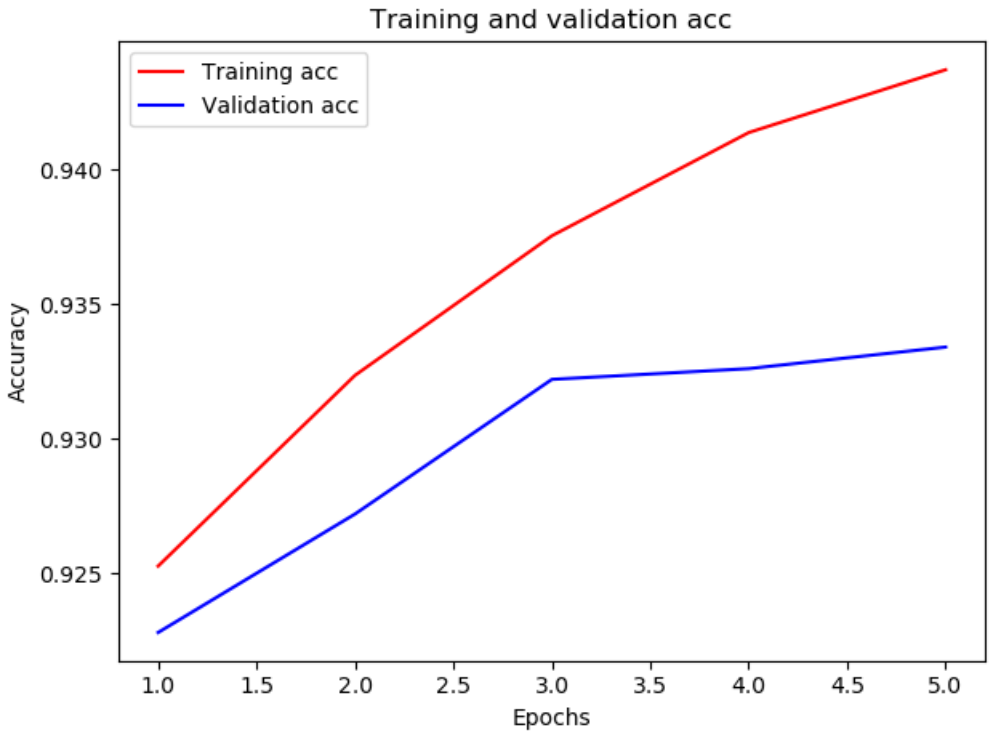

Рисунок 16 – график функции точности модели с оптимизатором RMSProp с параметрами learning_rate=0.01

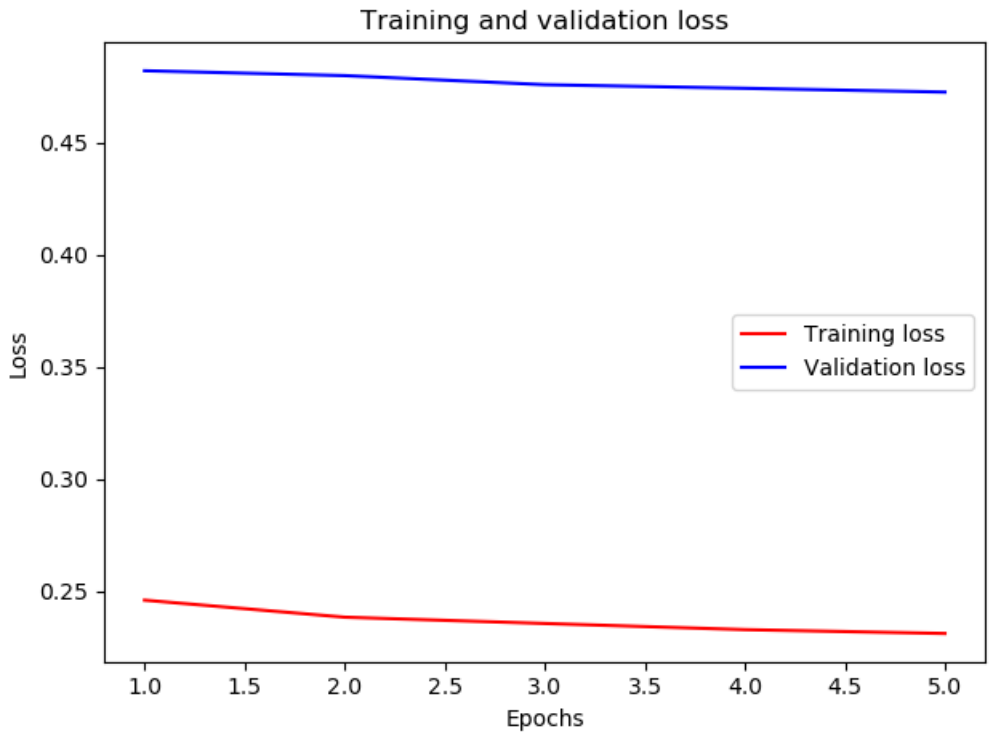

Рисунок 17 – график функции потери модели с оптимизатором RMSProp с параметрами momentum=0.1

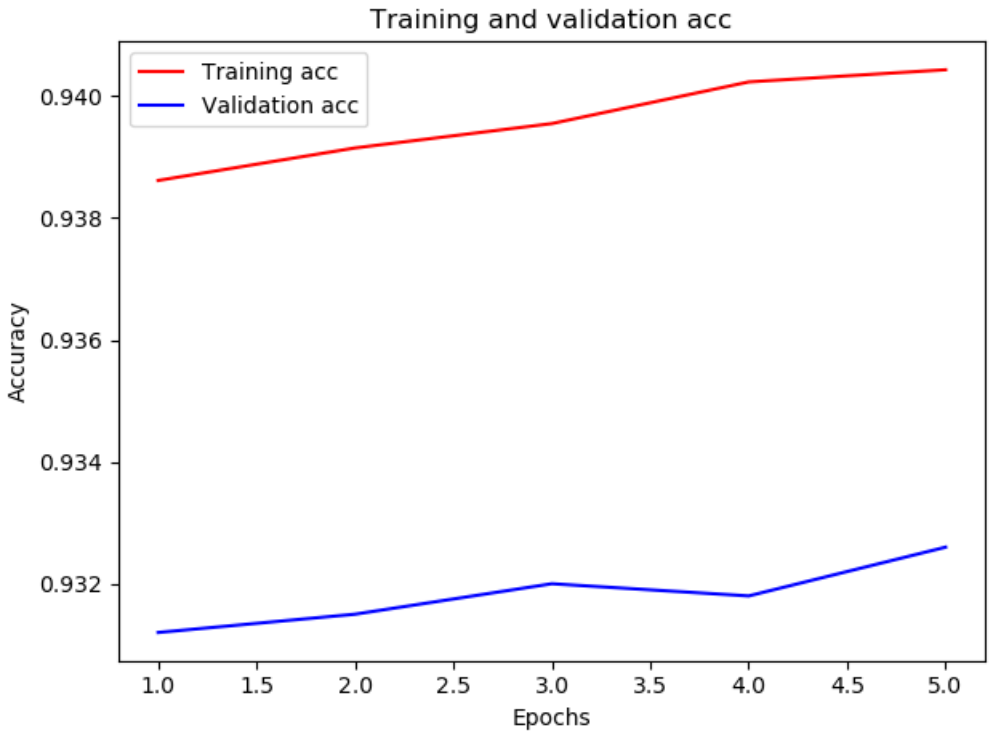

Рисунок 18 – график функции точности модели с оптимизатором RMSProp с параметрами momentum=0.1

Ниже представлены наилучшие значения потерь и точности на моделях:

SGD:

```
val_acc: 0.9143000245094299, val_loss: 0.3170919665336609
val_acc: 0.9296000003814697, val_loss: 0.2563252597212791
val_acc: 0.9638000130653381, val_loss: 0.12482133488655091
```

Adam:

```
val_acc: 0.9790999889373779, val_loss: 0.06756138030178845
val_acc: 0.97079998254776, val_loss: 0.11172418619282544
val_acc: 0.8834999799728394, val_loss: 0.5340271870613098
```

RMSProp:

```
val_acc: 0.892300009727478, val_loss: 0.5628221628189087
val_acc: 0.9298999905586243, val_loss: 0.48966799705028535
val_acc: 0.9326000213623047, val_loss: 0.4724045996785164
```

Исходя из вышеперечисленных графиков и результатов можно сделать вывод что наилучший результат показали модели, использующие в качестве оптимизаторов SGD и Adam.

.

Вывод.

В ходе выполнения данной работы была изучена задача распознавания рукописных цифр и исследовано влияние различных оптимизаторов на обучение моделей. Также была произведена работа по работе и обработке изображений.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

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

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

train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images / 255.0
test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images / 255.0



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

model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(28 * 28,)))
model.add(Dense(10, activation='softmax'))



optimizers = (SGD(),
              SGD(learning_rate=0.01, momentum=0.1),
              SGD(learning_rate=0.01, momentum=0.9),
              Adam(),
              Adam(learning_rate=0.01),
              Adam(learning_rate=0.1),
              RMSprop(),
              RMSprop(learning_rate=0.01),
              RMSprop(momentum=0.1))
```

```
    def upload_image(filepath):
        img = Image.open(fp=filepath)
        img = np.asarray(img)
        img = img.resize((28, 28))
        k = np.array([[[0.2989, 0.587, 0.114]]])
        img = np.sum(img * k, axis=2).reshape((1, 28 * 28)) / 255.0
        return img


    def explore_effect(optimizers):
        for optimizer in optimizers:
            model.compile(
                optimizer=optimizer,
                loss="categorical_crossentropy",
                metrics=["accuracy"],
            )
            history = model.fit(
                train_images,        train_labels,        epochs=5,
batch_size=128,
                validation_data=(test_images,         test_labels),
verbose=0
            )

            history_dict = history.history
            loss_values = history_dict["loss"]
            val_loss_values = history_dict["val_loss"]
            epochs = range(1, len(loss_values) + 1)
            plt.plot(epochs,   loss_values,   "r",   label="Training
loss")
            plt.plot(epochs,          val_loss_values,          "b",
label="Validation loss")
            plt.title(f"Training and validation loss")
            plt.xlabel("Epochs")
            plt.ylabel("Loss")
            plt.legend()
            plt.show()
```

```
            plt.clf()
            acc_values = history_dict["acc"]
            val_acc_values = history_dict["val_acc"]
            plt.plot(epochs, acc_values, "r", label="Training acc")
            plt.plot(epochs, val_acc_values, "b", label="Validation
acc")
            plt.title(f"Training and validation acc")
            plt.xlabel("Epochs")
            plt.ylabel("Accuracy")
            plt.legend()
            plt.show()
            print(f"val_acc:    {max(val_acc_values)},    val_loss:
{min(val_loss_values)}")

explore_effect(optimizers)
```