Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**ЛАБОРАТОРНАЯ РАБОТА №6**

**Дисциплина: Алгоритмы цифровой обработки мультимедиа**

Работу выполнила: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Д. Н. Баева

Направление подготовки: 02.03.02 Фундаментальная информатика и информационные технологии

Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_А. А. Крамаренко

**Тема работы:** Введение в нейронные сети.

**Ход работы:**

Задание 1 (самостоятельно). Построить многослойный персептрон средствами библиотеки Keras языка Python (или соответствующей библиотеки на другом языке), позволяющий распознавать цифры. Для обучения и тестирования использовать базу MNIST http://yann.lecun.com/exdb/mnist/. Возможно использовать любую функции активации, любую структуру сети и любой алгоритм обучения.

Задание 2 (самостоятельно). Исследовать нейронную сеть, построенную в задании 1. Подобрать количество эпох обучения, необходимое для выбранного вами предела корректной работы на тестовой базе. Сравнить несколько значений количества эпох по параметрам скорость обучения, скорость работы сети, процент корректной работы на тестовой базе.

Задание 3 (самостоятельно). Построить сверточную нейронную сеть,

решающую ту же задачу классификации. Рассмотреть несколько архитектур, подобрать оптимальные параметры. Сравнить результаты с предыдущей архитектурой.

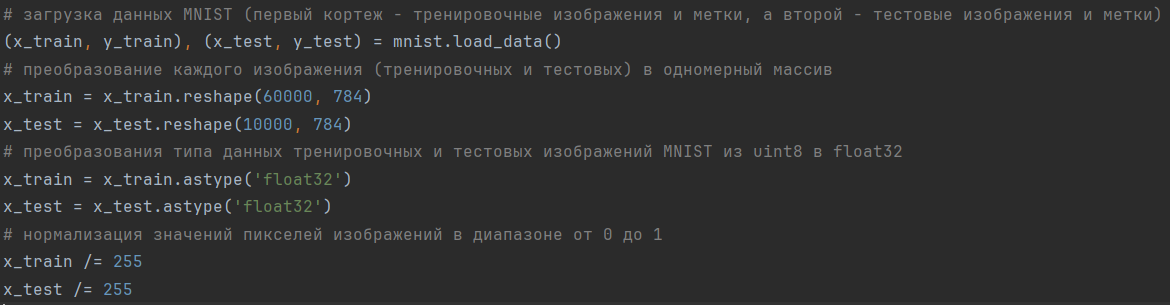
В лабораторной работе необходимо было решить задачу классификации изображений с помощью нейронной сети.

В первом задании необходимо было разработать многослойный персептрон, позволяющий распознавать цифры. Определим что это такое. Многослойный перцептрон — это один из видов нейронных сетей, который используется для классификации данных. Он состоит из входного слоя, скрытого слоя и выходного слоя. Каждый слой содержит нейроны, которые обрабатывают данные и передают их на следующий слой. Веса между нейронами определяются в процессе обучения сети. Многослойный перцептрон является частным случай персептрона Розенблатта, в котором один алгоритм обратного распространения ошибки обучает все слои. Название по историческим причинам не отражает особенности данного вида персептрона, то есть не связано с тем, что в нём имеется несколько слоёв (так как несколько слоёв было и у персептрона Розенблатта).

Для разработки использовался язык программирования Python и библиотека Кeras (согласно заданию), данные брались из базы MNIST.

В первую очередь происходит загрузка данных из базы MNIST с помощью метода mnist.load\_data(). Он загружает данные, которые содержат изображения рукописных цифр и соответствующие им метки. Возвращаются два кортежа: (x\_train, y\_train) и (x\_test, y\_test). Первый кортеж содержит тренировочные изображения и метки, а второй - тестовые изображения и метки. Изображения представлены в виде трехмерных массивов NumPy, а метки - в виде одномерных массивов NumPy.

Далее необходимо выполнить предобработку изображений в правильный формат. В данном случае метод reshape используется для изменения формы тренировочных изображений. Изначально, каждое изображение представлено в виде двумерного массива размером 28x28 пикселей. Однако, для обучения многослойного персептрона, необходимо преобразовать каждое изображение в одномерный массив размером 784 (28 \* 28). Метод reshape применяется к переменной с тренировочными изображения и принимает два аргумента: новую форму массива и тип данных. В данном случае новая форма массива — это (60000, 784), что означает, что массив должен иметь 60000 строк и 784 столбца. Аналогично метод reshape применяется к переменной с тестовыми изображениями. После методом astype('float32') тренировочные и тестовые изображения переводятся из типа uint8 в float для дальнейшей нормализации. Деление x\_train /= 255 и x\_test /= 255 выполняется для нормализации значений пикселей изображений в диапазоне от 0 до 1. Изначально, значения пикселей находятся в диапазоне от 0 до 255. Нормализация помогает ускорить процесс обучения модели и улучшить ее производительность. На рисунке 1 представлен фрагмент кода, выполняющий это.

Рисунок 1 – Загрузка данных и их предобработка.

При загрузке данных из MNIST, метки представлены в виде одномерного массива NumPy, содержащего целочисленные значения от 0 до 9. Однако, для обучения многослойного персептрона, необходимо преобразовать их в категории. Это делается с помощью функции to\_categorical, принимающей два аргумента: сами метки и количество классов. Количество классов равно 10, так как в базе данных MNIST содержится 10 классов цифр от 0 до 9. Функция to\_categorical преобразует каждую метку в категорию размером 10, где все элементы равны нулю, кроме элемента с индексом, соответствующим значению метки, который равен единице. Например, если значение метки равно 3, то соответствующая категория будет иметь вид [0, 0, 0, 1, 0, 0, 0, 0, 0, 0]. Функция применяется к тренировочным и тестовым изображениям соответственно (рисунок 2).

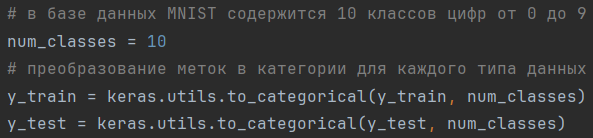


Рисунок 2 – Преобразование меток.

Основная построение нейронной сети происходит в следующем фрагменте кода. Новая модель многослойного персептрона создаётся с помощью метода model = Sequential(). Затем добавляются два слоя с 512 нейронами и одинаковой функцией активации ReLU с помощью метода model.add(). После добавления каждого из вышеперечисленных слоёв добавляется слой Dropout с коэффициентом отсева 0,2 (model.add(Dropout(0.2)). Это метод регуляризации, который случайно отклоняет (обнуляет) некоторых нейронов во время обучения для предотвращения переобучения сети. Выходной слой многослойного персептрона имеет другую функцию активации softmax. Функция активации ReLU (Rectified Linear Unit) — это нелинейная функция активации, которая используется в нейронных сетях для добавления нелинейности в выходные данные. Она определяется как f(x) = max(0, x), где x - входной сигнал. Функция работает следующим образом: если входной сигнал больше нуля, то возвращается его же значение, если же входной сигнал меньше или равен нулю, то возвращается ноль. Функция активации softmax преобразует выходные значения нейронов в вероятности, которые суммируются до 1. В данном случае выходные значения нейронов будут преобразованы в вероятности, которые будут использоваться для классификации входных данных на num\_classes классов. Добавление слоёв отображено на рисунке 3.

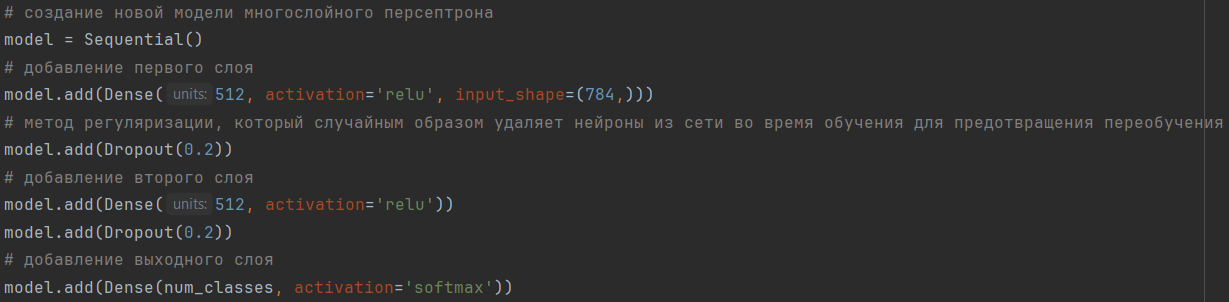


Рисунок 3 – Создание персептрона и добавление двух скрытых слоёв и одного выходного.

Перед обучением нейронной сети предварительно задаются параметры. Выбирается функция потерь (loss), оптимизатор (optimizer) и метрики (metrics). Функция потерь определяет, как модель оценивает свою точность во время обучения, оптимизатор – как модель обновляет свои веса на основе градиентов, вычисленных во время обратного распространения ошибки, а метрики – как модель оценивает свою производительность на основе тестовых данных. Это параметры функции компиляции model.compile(). Для разрабатываемого персептрона была выбрана функция потерь categorical\_crossentropy (категориальная кросс-энтропию), оптимизатор – adam и метрика – accuracy (точность). Метод представлен на рисунке 4.

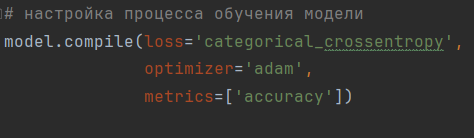


Рисунок 4 – Параметры обучения персептрона.

Метод model.fit() в библиотеке Keras используется для обучения модели на тренировочных данных. Он принимает несколько аргументов, включая тренировочные данные, количество эпох (один полный проход через все обучающие данные во время обучения модели), размер пакета (количество образцов данных, которые будут переданы модели за один раз во время обучения), функцию потерь и метрики. Для обучения многослойного персептрона на тренировочных данных x\_train и y\_train количество эпох было установлено на 15, размер пакета – 128. Помимо этого с помощью аргумента verbose = 1 в консоль выводится информация о потерях, метриках и прогресс-бар для каждой эпохи обучения. Соответствующие функции представлены на рисунке 5.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 5 – Обучение многослойного персептрона.

В конце программы конфигурация нейронной сети сохраняется в отдельную модель с помощью функции model.save(). Далее в консоль выводится информация об оценке потерь и точности модели (с помощью функции model.evaluate()), подсчитывается процент корректной работы и время работы персептрона. Последнее вычисляется с помощью одноимённой функции из модуля time сохранением в двух переменных времени начала и конца работы нейронной сети соответственно и вычислением их разницы (рисунок 6).

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 6 – Отображение информации о потерях, точности и времени.

Для удобства анализа полученной информации об потерях, точности и времени, а также просмотра изображений, которые персептрон обрабатывает, была выполнена визуализация. Для этого использовался инструмент визуализации Tensorboard (входит в состав библиотеки Tesnorflow), который позволяет отслеживать процесс обучения модели и анализировать ее результаты. Функция tensorboard = TensorBoard(log\_dir='/LW\_6/logs\_1', histogram\_freq=0, write\_graph=True, write\_images=False) создает объект TensorBoard. Параметры означают следующее: log\_dir это путь к директории, в которой будут храниться файлы журнала TensorBoard, histogram\_freq это частота записи гистограммы в файл, write\_graph это флаг, указывающий, должен ли быть записан граф модели в файл журнала, а write\_images это флаг, указывающий, должны ли быть записаны изображения в файл журнала. Метод add\_images объекта writer, добавляет изображения в файл журнала. Записываются первые 20 изображений из набора данных MNIST. При обучении нейронной сети объект TensorBoard передавался в качестве обратного вызова (объекты, которые могут выполнять действия на различных этапах обучения (например, в начале или конце эпохи, до или после каждого пакета и т. д.). Соответствующие функции представлены на рисунке 7.

**Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание**Рисунок 7 – Визуализация работы персептрона.

На рисунках 8 и 9 представлена информация о работе многослойного персептрона на 15 эпохах, соответствующие потери и точности работы модели, и изображение фрагмента набора данных MNIST.

Изображение выглядит как текст, снимок экрана, Шрифт, меню

Автоматически созданное описание

Рисунок 8 – Информация об эпохах обучения, потерях, точности модели и затраченном времени работы.

Изображение выглядит как рукописный текст, текст, Шрифт, каллиграфия

Автоматически созданное описание

Рисунок 9 – Первые 20 изображений из набора данных MNIST.

В следующем задании необходимо было разработать свёрточную нейронную сеть, решающую ту же задачу классификации. Сверточная нейронная сеть (Convolutional Neural Network, CNN) — это архитектура нейронной сети, которая специально разработана для обработки изображений. Она использует сверточные слои для извлечения признаков из изображений и пулинговые слои для уменьшения размерности пространства признаков. CNN также может содержать полносвязные слои для классификации изображений. Обычно такие сети используются для задач компьютерного зрения, таких как распознавание объектов и классификация изображений.

Для реализации вновь использовался язык программирования Python, библиотека Tensorflow и аналогичный набор данных MNIST.

Как и в случае с многослойным персептроном, сперва необходимо выполнить загрузку данных из MNIST, нормализация данных и преобразование двухмерных изображений в четырехмерные. Последнее выполняется с использованием уже ранее упоминавшейся функции reshape(). Он изменяют форму массивов train\_images и test\_images из двумерных в четырехмерные, добавляя размерность канала. Это необходимо для использования их в сверточной нейронной сети, которая ожидает входные данные в форме (num\_samples, height, width, channels). В данном случае num\_samples — это количество образцов в датасете MNIST, height и width — это размеры изображений (28x28), а channels — это количество каналов (1 для черно-белых изображений). Фрагмент кода, отвечающий за загрузку и предобработку данных представлен на рисунке 10.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 10 – Загрузка данных MNIST и их предобработка.

Далее происходит построение сети. Новая модель свёрточной нейронной сети создаётся с помощью метода model = Sequential(). Затем добавляются первый слой с 32 фильтрами размера 3x3 и функцией активации ReLU с помощью метода model.add(). Входной формат данных для этого слоя — это тензор (28, 28, 1), где 1 — это количество каналов (в данном случае черно-белое изображение). Далее добавляется слой пуллинга с помощью метода model.add(layers.MaxPooling2D((2, 2))), который уменьшает размерность пространства признаков, уменьшая количество параметров и вычислений в сети, что предотвращает переобучение. Аналогичным образом добавляются ещё три свёрточных слоя с функцией активации ReLU и 64 фильтрами размера 3х3 и слои пуллинга. После третьего слоя слой пуллинга заменяется слоем “выравнивания” – он преобразует многомерный тензор в одномерный тензор, что позволяет передавать данные в полносвязные слои. Он используется после последнего свёрточного слоя, чтобы преобразовать выходные данные из трехмерного тензора в одномерный тензор перед передачей их в полносвязный слой. Выходной слой CNN имеет 64 фильтра с функцией активации ReLU – первый полносвязный слой, а второй полносвязный слой имеет 10 нейронов, что соответствует количеству классов в датасете MNIST. Добавление слоёв отображено на рисунке 11.

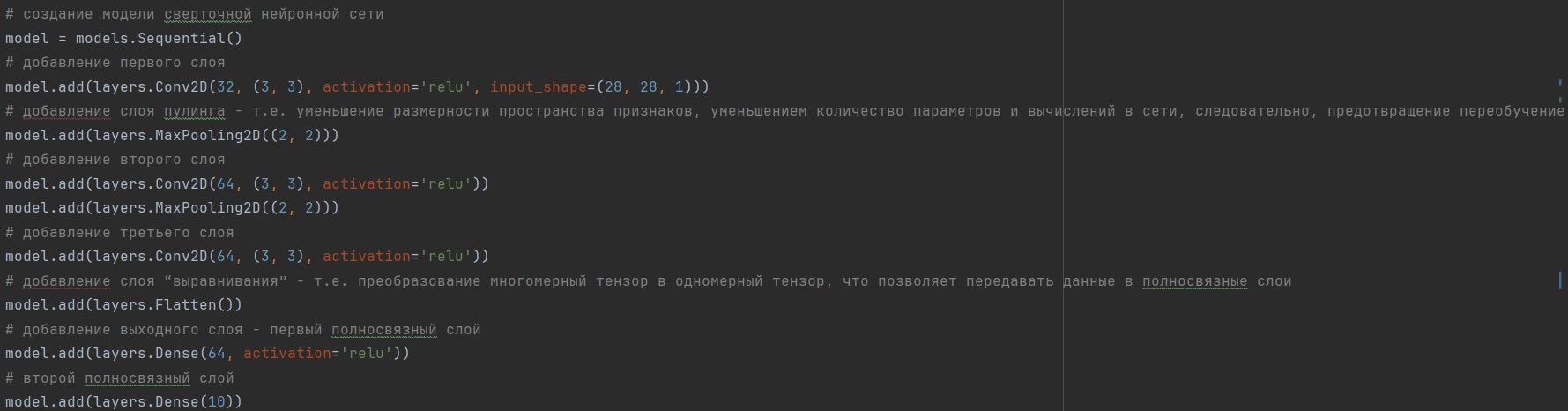


Рисунок 11 – Создание CNN и добавление слоёв.

Настройка параметров обучения и функция обучения сети аналогична соответствующим методам в персептроне с отличием лишь в количестве параметров. Соответствующие методы представлены на рисунке 12.

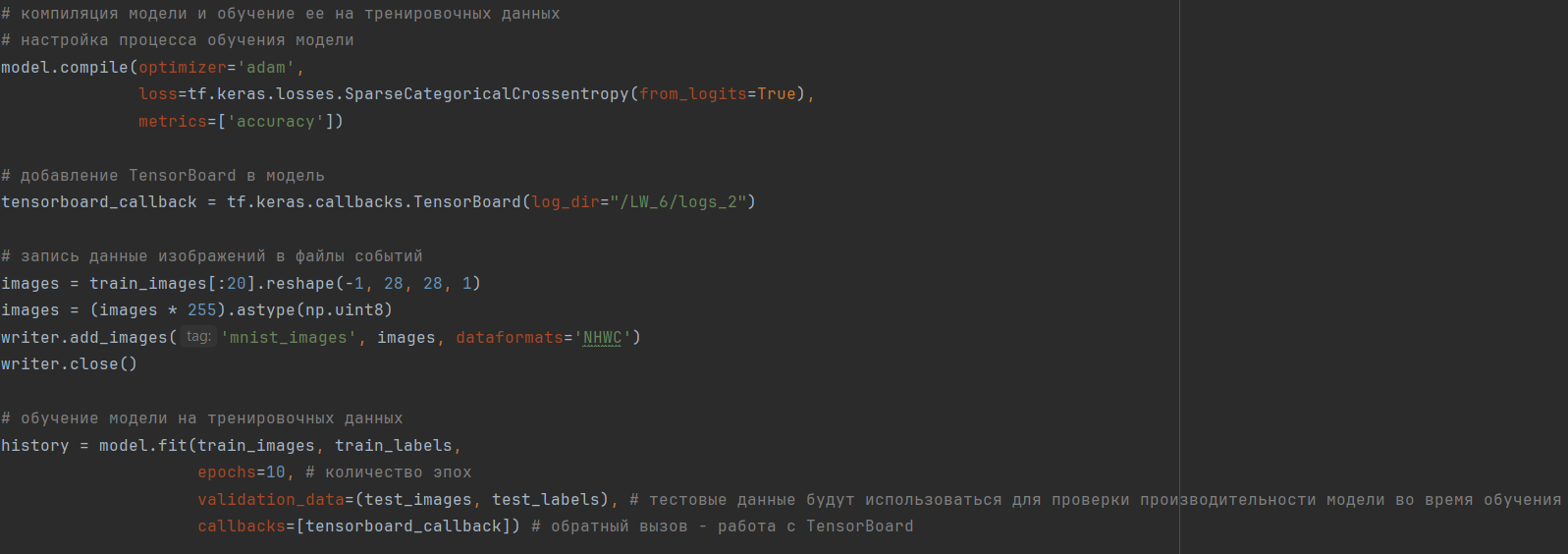


Рисунок 12 – Обучение CNN.

В конце программы тем же самым образом, что и в предыдущей сети сохраняется модель с конфигурацией сети (в той же самой папке) и вызываются методы по отображению информации об оценке потерь и точности модели, а также затраченном времени. Конец программы отображён на рисунке 13.

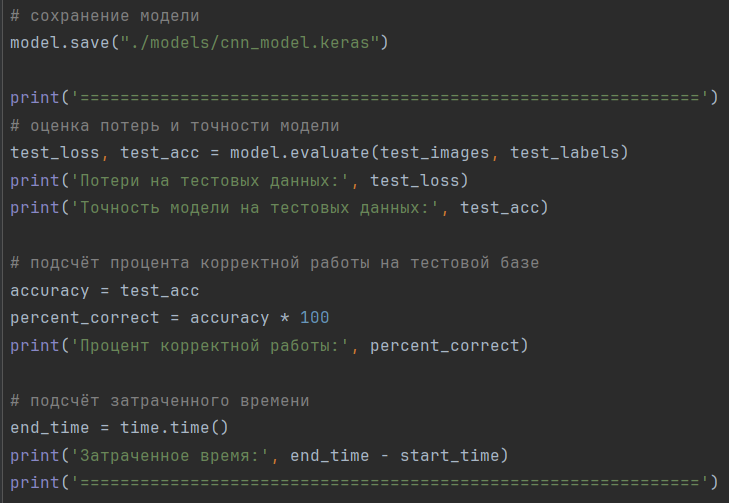


Рисунок 13 – Информация о работе CNN.

На рисунках 14 и 15 отображена информация о работе свёрточной нейронной сети на 10 эпохах, соответствующие потери и точности работы модели, и изображение фрагмента набора данных MNIST.

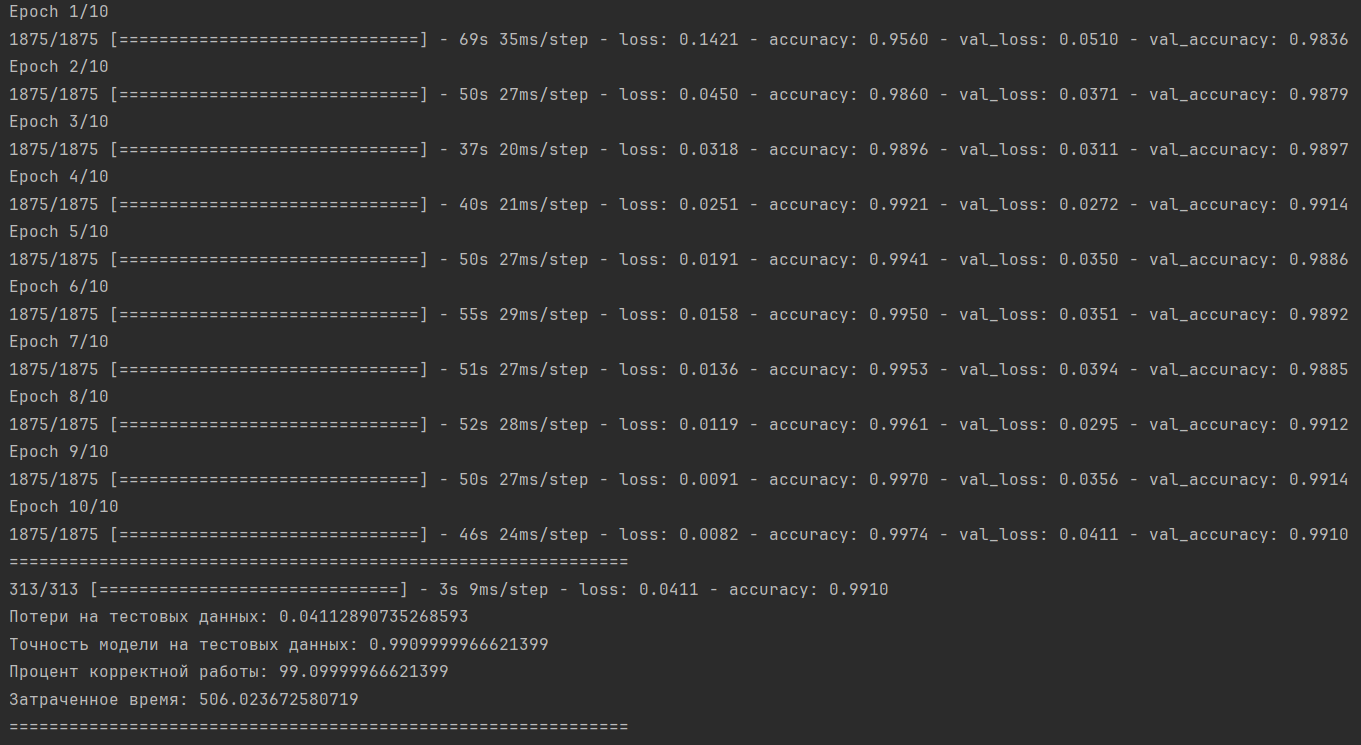


Рисунок 14 – Информация об эпохах обучения, потерях, точности модели и затраченном времени работы.

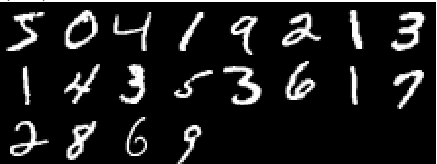


Рисунок 15 – Первые 20 изображений из набора данных MNIST.

**Листинг программы**

Файл Multilayer\_Perceptron\_MNIST.py

# импорт библиотеки Keras и необходимых соответствующих модулей  
import keras  
from keras.datasets import mnist  
from keras.models import Sequential  
from keras.layers import Dense, Dropout  
from keras.optimizers import RMSprop  
from keras.callbacks import TensorBoard  
import time  
from torch.utils.tensorboard import SummaryWriter  
import numpy as np  
  
# замер начала времени работы  
start\_time = time.time()  
writer = SummaryWriter(log\_dir='/LW\_6/logs\_1')  
  
# загрузка данных MNIST (первый кортеж - тренировочные изображения и метки, а второй - тестовые изображения и метки)  
(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()  
# преобразование каждого изображения (тренировочных и тестовых) в одномерный массив  
x\_train = x\_train.reshape(60000, 784)  
x\_test = x\_test.reshape(10000, 784)  
# преобразования типа данных тренировочных и тестовых изображений MNIST из uint8 в float32  
x\_train = x\_train.astype('float32')  
x\_test = x\_test.astype('float32')  
# нормализация значений пикселей изображений в диапазоне от 0 до 1  
x\_train /= 255  
x\_test /= 255  
  
# в базе данных MNIST содержится 10 классов цифр от 0 до 9  
num\_classes = 10  
# преобразование меток в категории для каждого типа данных  
y\_train = keras.utils.to\_categorical(y\_train, num\_classes)  
y\_test = keras.utils.to\_categorical(y\_test, num\_classes)  
  
# создание новой модели многослойного персептрона  
model = Sequential()  
# добавление первого слоя  
model.add(Dense(512, activation='relu', input\_shape=(784,)))  
# метод регуляризации, который случайным образом удаляет нейроны из сети во время обучения для предотвращения переобучения  
model.add(Dropout(0.2))  
# добавление второго слоя  
model.add(Dense(512, activation='relu'))  
model.add(Dropout(0.2))  
# добавление выходного слоя  
model.add(Dense(num\_classes, activation='softmax'))  
  
# компиляция модели и обучение ее на тренировочных данных  
# настройка процесса обучения модели  
model.compile(loss='categorical\_crossentropy',  
 optimizer='adam',  
 metrics=['accuracy'])  
  
# добавление TensorBoard в модель  
tensorboard = TensorBoard(log\_dir='/LW\_6/logs\_1', histogram\_freq=0,  
 write\_graph=True, write\_images=False)  
  
# запись данные изображений в файлы событий  
images = x\_train[:20].reshape(-1, 28, 28, 1)  
images = (images \* 255).astype(np.uint8)  
writer.add\_images('mnist\_images', images, dataformats='NHWC')  
writer.close()  
  
# обучение модели на тренировочных данных  
history = model.fit(x\_train, y\_train,  
 batch\_size=128, # размер пакета  
 epochs=15, # количество эпох  
 verbose=1, # вывод информации о потерях, метриках и прогресс-бар  
 validation\_data=(x\_test, y\_test), # тестовые данные будут использоваться для проверки производительности модели во время обучения  
 callbacks=[tensorboard]) # обратный вызов - работа с TensorBoard  
  
# сохранение модели  
model.save("./models/multilayer\_perceptron.keras")  
  
print('==============================================================')  
# оценка потерь и точности модели  
score = model.evaluate(x\_test, y\_test, verbose=0)  
print('Потери на тестовых данных:', score[0])  
print('Точность модели на тестовых данных:', score[1])  
  
# подсчёт процента корректной работы на тестовой базе  
accuracy = score[1]  
percent\_correct = accuracy \* 100  
print('Процент корректной работы:', percent\_correct)  
  
# подсчёт затраченного времени  
# замер конца времени работы  
end\_time = time.time()  
print('Затраченное время:', end\_time - start\_time)  
print('==============================================================')

Файл Multilayer\_Perceptron\_MNIST.py

import tensorflow as tf  
from tensorflow.keras import datasets, layers, models  
import time  
from torch.utils.tensorboard import SummaryWriter  
import numpy as np  
  
# замер начала времени работы  
start\_time = time.time()  
writer = SummaryWriter(log\_dir='/LW\_6/logs\_2')  
  
# загрузка данных MNIST (первый кортеж - тренировочные изображения и метки, а второй - тестовые изображения и метки)  
(train\_images, train\_labels), (test\_images, test\_labels) = datasets.mnist.load\_data()  
# нормализация данных  
train\_images = train\_images / 255.0  
test\_images = test\_images / 255.0  
# изменение форму массивов из двумерных в четырехмерные, добавляя размерность канала  
train\_images = train\_images.reshape(train\_images.shape[0], 28, 28, 1)  
test\_images = test\_images.reshape(test\_images.shape[0], 28, 28, 1)  
  
# создание модели сверточной нейронной сети  
model = models.Sequential()  
# добавление первого слоя  
model.add(layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(28, 28, 1)))  
# добавление слоя пулинга - т.е. уменьшение размерности пространства признаков, уменьшением количество параметров и вычислений в сети, следовательно, предотвращение переобучение.  
model.add(layers.MaxPooling2D((2, 2)))  
# добавление второго слоя  
model.add(layers.Conv2D(64, (3, 3), activation='relu'))  
model.add(layers.MaxPooling2D((2, 2)))  
# добавление третьего слоя  
model.add(layers.Conv2D(64, (3, 3), activation='relu'))  
# добавление слоя “выравнивания” - т.е. преобразование многомерный тензор в одномерный тензор, что позволяет передавать данные в полносвязные слои  
model.add(layers.Flatten())  
# добавление выходного слоя - первый полносвязный слой  
model.add(layers.Dense(64, activation='relu'))  
# второй полносвязный слой  
model.add(layers.Dense(10))  
  
# компиляция модели и обучение ее на тренировочных данных  
# настройка процесса обучения модели  
model.compile(optimizer='adam',  
 loss=tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True),  
 metrics=['accuracy'])  
  
# добавление TensorBoard в модель  
tensorboard\_callback = tf.keras.callbacks.TensorBoard(log\_dir="/LW\_6/logs\_2")  
  
# запись данные изображений в файлы событий  
images = train\_images[:20].reshape(-1, 28, 28, 1)  
images = (images \* 255).astype(np.uint8)  
writer.add\_images('mnist\_images', images, dataformats='NHWC')  
writer.close()  
  
# обучение модели на тренировочных данных  
history = model.fit(train\_images, train\_labels,  
 epochs=10, # количество эпох  
 validation\_data=(test\_images, test\_labels), # тестовые данные будут использоваться для проверки производительности модели во время обучения  
 callbacks=[tensorboard\_callback]) # обратный вызов - работа с TensorBoard  
  
# сохранение модели  
model.save("./models/cnn\_model.keras")  
  
print('==============================================================')  
# оценка потерь и точности модели  
test\_loss, test\_acc = model.evaluate(test\_images, test\_labels)  
print('Потери на тестовых данных:', test\_loss)  
print('Точность модели на тестовых данных:', test\_acc)  
  
# подсчёт процента корректной работы на тестовой базе  
accuracy = test\_acc  
percent\_correct = accuracy \* 100  
print('Процент корректной работы:', percent\_correct)  
  
# подсчёт затраченного времени  
end\_time = time.time()  
print('Затраченное время:', end\_time - start\_time)  
print('==============================================================')