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 выполнил: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Кислица Д.А.

Направление подготовки: 02.03.02 Фундаментальная информатика и информационные технологии

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

**Цель 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.*

1. Устанавливаем размер батча, количество эпох обучения, размер входных изображений и их глубину, количество нейронов, количество классов, а также количество тестовых и обучающих примеров. (Рисунок 1)

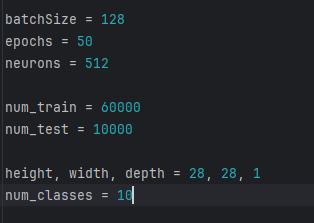


Рисунок 1 – Переменные.

2. Загружаем данные MNIST, преобразуем их в тип ‘float32’ для выполнения действий с плавающей точкой, нормализуем, преобразует метки классов в бинарные матрицы, чтобы подготовить данные для многоклассовой классификации. (Рисунок 2)

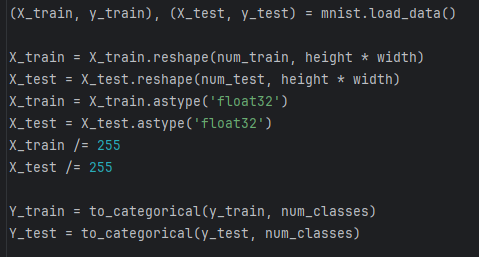


Рисунок 2 – Подготовка датасета.

3. Создаем последовательную модель Keras, которая позволяет добавлять слои последовательно при помощи функции keras.Sequential(). Добавляем два полносвязных слоя с 512 нейронами и функцией активации ReLU. Добавляем выходной слой с 10 нейронами (по количеству классов) и функцией активации softmax для многоклассовой классификации. Создаем экземпляр оптимизатора Adam с заданной скоростью обучения (0.001). Компилируем модель с использованием оптимизатора Adam и функции потерь categorical\_crossentropy. Указываем метрику точности для оценки производительности модели. Обучаем модель на обучающих данных за заданное количество эпох и с указанным размером батча. Оцениваем производительность модели на тестовых данных и выводим результаты оценки. (Рисунок 3)

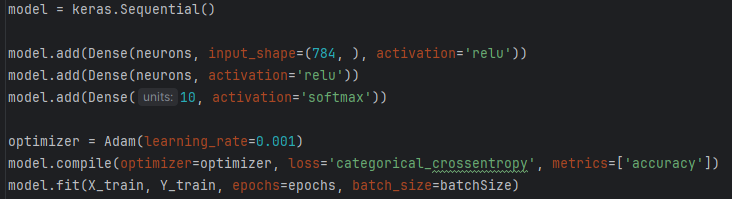


Рисунок 3 – Обучение модели

*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.*

1. Проведём обучение нейронной сети на 5, 15, 20, 30 и 50 эпохах и внесём данные в сводную таблицу. (Рисунок 4)

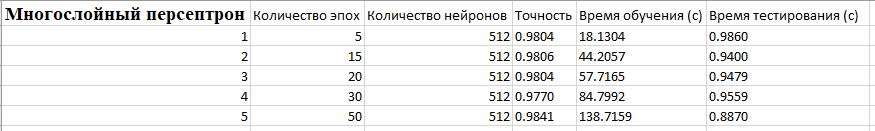


Рисунок 4 – Сводная таблица для многослойного персептрона.

Значительные изменения в точности нейронной сети были замечены при превышении порога в 30 эпох, при этом точность на 30 эпохах понизилась по сравнению с результатами предыдущих попыток на 0.3%. Однако уже на 50 эпохах точность повысилась в сравнении со всеми предыдущими попытками.

При этом время работы на тестовой базе было максимальным при 5 эпохах и составляло 0.860с, на 15 уменьшилось на 0.46с и снова начало расти вплоть до 30 эпох. На 50 эпохах работа на тестовой базе стала быстрее на 0.0691с по сравнению с 30 эпохами и составило 0.887с.

Время обучения постепенно возрастало по мере увеличения количества эпох.

*Задание 3 (самостоятельно). Построить сверточную нейронную сеть,*

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

1. Создаем последовательную модель Keras, которая позволяет добавлять слои последовательно при помощи функции keras.Sequential(). Добавляем свёрточный слой с 32 фильтрами размером 3x3 и функцией активации ReLU. (Рисунок 5)



Рисунок 5 – Свёрточная нейронная сеть.

2. Проведём обучение нейронной сети на 5, 15, 20, 30 и 50 эпохах и внесём данные в сводную таблицу. (Рисунок 6)

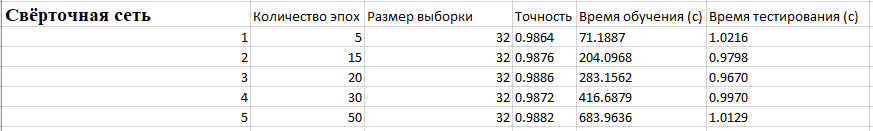


Рисунок 6 – Сводная таблица для свёрточной нейронной сети.

При обучении свёрточной нейронной сети мы затрачиваем больше времени чем для прошлой модели. Так же выросло и время, которое модель проводит за работой с тестовыми данными. Однако точность на любом количестве эпох у данной модели превышает точность на том же количестве эпох у предыдущей модели.

Ответы на контрольные вопросы.

1. *Опишите принцип обучения нейронной сети с учителем.*

Принцип обучения нейронной сети с учителем заключается в **использовании предварительно размеченных данных.** Эти данные включают в себя входные значения и соответствующие им целевые (ответы). Нейросеть обучается предсказывать последние на основе входных данных.

**Процесс обучения проходит через следующие этапы**:

А) **Подготовка данных**. Подготавливается набор входных данных, для которых известны ожидаемые результаты.

Б) **Процесс обучения**. Нейронная сеть просматривает входные данные и делает предсказания на основе своих текущих параметров (весов). Затем эти предсказания сравниваются с ожидаемыми результатами.

В) **Обратное распространение ошибки** **(backpropagation).** Если предсказание нейросети не совпадает с ожидаемым результатом, то происходит процесс, называемый обратным распространением ошибки. В ходе него вычисляется, насколько каждый вес внутри нейросети способствовал общей ошибке. Затем эти веса корректируются в направлении, которое уменьшит ошибку.

Г) **Итерации**. Повторяются множество раз с использованием новой системы входных данных из обучающего набора. С течением времени веса нейросети подстраиваются таким образом, чтобы ошибка между предсказаниями и ожидаемыми результатами были минимизирована.

Д) **Тестирование и оценка**. Обученные нейронные сети тестируют на новых данных, которые ранее не использовались.

2. *Опишите структуру нейрона.*

**Входы**: Нейрон принимает входные данные (сигналы) от других нейронов или внешних источников. Каждый вход может иметь свой вес, который определяет его влияние на выход нейрона.

**Весовые коэффициенты**: Каждому входному сигналу присваивается вес, который может изменяться в процессе обучения. Вес определяет важность данного входа для активации нейрона.

**Сумматор**: Нейрон суммирует взвешенные входные сигналы, что можно представить как линейную комбинацию входов.

**Функция активации**: После суммирования применяется функция активации, которая определяет, будет ли нейрон активирован (выдаст ли сигнал дальше). Это может быть, например, сигмоидная функция, ReLU или гиперболический тангенс.

**Выход**: Результат работы нейрона передается другим нейронам или на выход модели. Выходной сигнал может быть использован как вход для следующих слоев нейронов.

3. *Структура персептрона*

**Персептрон** — это простейшая форма искусственного нейрона, предложенная Фрэнком Розенблаттом в 1957 году. Он служил вдохновением для развития более сложных нейронных сетей. Персептрон принимает несколько входов, применяет к ним взвешенные коэффициенты (веса) и суммирует результаты. Затем, к полученной сумме применяется функция активации, которая определяет, будет ли нейрон активирован и передаст ли сигнал на выход.

**Многослойный персептрон (MLP)** — это тип искусственной нейронной сети, который состоит из нескольких слоев нейронов и способен решать сложные задачи, такие как классификация и регрессия. Основная структура многослойного персептрона включает три основных типа слоев:

**Структура многослойного персептрона**

**Входной слой:**

Это первый слой сети, который принимает входные данные. Каждый нейрон в этом слое соответствует одному входному признаку. Входной слой не выполняет никаких вычислений, а просто передает данные на следующий слой.

**Скрытые слои:**

Многослойный персептрон содержит один или несколько скрытых слоев, которые находятся между входным и выходным слоями. Эти слои выполняют основные вычисления и обрабатывают входные данные.

Каждый нейрон в скрытых слоях принимает взвешенные суммы входных сигналов, применяет функцию активации (например, ReLU, сигмоид или гиперболический тангенс) и передает результат на следующий слой.

Скрытые слои могут быть как однослойными, так и многослойными, что позволяет модели обучаться более сложным паттернам.

**Выходной слой:**

Это последний слой сети, который выдает результаты работы модели.

Количество нейронов в выходном слое соответствует количеству классов в задаче классификации или количеству целевых переменных в задаче регрессии.

В выходном слое часто используется функция активации softmax для многоклассовой классификации или линейная функция для регрессии.

4. *Смысл применения оптимизационных методов в задачах обучения нейронной сети.*

**Смысл применения оптимизационных методов в задачах обучения нейронной сети** заключается в **поиске наилучшего набора параметров** (весов и смещений) **модели**, чтобы при решении конкретной задачи модель выдавала наилучшие результаты.

Оптимизаторы помогают улучшить модель, настраивая её параметры для минимизации значения функции потерь. Чем меньше её значение, тем ближе предсказание модели к реальным значениям.

5. *Что такое аугментация?*

Аугментация данных — это метод увеличения объема обучающего набора данных путем применения различных трансформаций к исходным данным. Это может включать вращение изображений, изменение их размера, добавление шума или изменение яркости. Аугментация помогает улучшить обобщающую способность модели и предотвратить переобучение, так как модель становится более устойчивой к изменениям в данных.

6. *Принцип пакетного и последовательного обучения*

**Пакетное обучение (batch learning)** подразумевает использование фиксированного количества примеров (батча) для обновления весов модели за одну итерацию. Это позволяет более эффективно использовать вычислительные ресурсы и ускорять процесс обучения.

**Последовательное обучение (online learning)** происходит по одному примеру за раз, что позволяет модели адаптироваться к новым данным в реальном времени.

**batch\_size** — это размер батча, то есть количество примеров, обрабатываемых одновременно во время одной итерации обучения. Выбор размера батча может влиять на скорость обучения и качество модели

7. *Покажите известные Вам функции активации.*

1) Сигмоидная функция

2) Гиперболический тангенс

3) ReLU

4) Leaky ReLU

5)Softmax

**Листинг программы**

**perceptron.py**

import keras  
from keras.\_tf\_keras.keras.datasets import mnist  
from keras.\_tf\_keras.keras.layers import Dense  
from keras.\_tf\_keras.keras.utils import to\_categorical  
from keras.\_tf\_keras.keras.optimizers import Adam  
import time  
  
batchSize = 128  
epochs = 50  
neurons = 512  
  
num\_train = 60000  
num\_test = 10000  
  
height, width, depth = 28, 28, 1  
num\_classes = 10  
  
(X\_train, y\_train), (X\_test, y\_test) = mnist.load\_data()  
  
X\_train = X\_train.reshape(num\_train, height \* width)  
X\_test = X\_test.reshape(num\_test, height \* width)  
X\_train = X\_train.astype('float32')  
X\_test = X\_test.astype('float32')  
X\_train /= 255  
X\_test /= 255  
  
Y\_train = to\_categorical(y\_train, num\_classes)  
Y\_test = to\_categorical(y\_test, num\_classes)  
  
start\_time = time.time()  
  
model = keras.Sequential()  
  
model.add(Dense(neurons, input\_shape=(784, ), activation='relu'))  
model.add(Dense(neurons, activation='relu'))  
model.add(Dense(10, activation='softmax'))  
  
optimizer = Adam(learning\_rate=0.001)  
model.compile(optimizer=optimizer, loss='categorical\_crossentropy', metrics=['accuracy'])  
model.fit(X\_train, Y\_train, epochs=epochs, batch\_size=batchSize)  
print("Выводим результат обучения...")  
  
start\_ev\_time = time.time()  
model.evaluate(X\_test, Y\_test)  
end\_ev\_time = time.time()  
  
accuracy = model.evaluate(X\_test,Y\_test)  
  
end\_time = time.time()  
interval = end\_time - start\_time  
interval\_ev = end\_ev\_time - start\_ev\_time  
print(f'Точность {accuracy[1]\*100:.2f}%')  
print(f"Время Обучения: {interval:.6f} секунд")  
print(f"Время Теста: {interval\_ev:.6f} секунд")  
  
model.save("perceptron.keras")

**cnn.py**

import keras  
from keras.api.datasets import mnist  
from keras.api.utils import to\_categorical  
import time  
  
height, width, depth = 28, 28, 1  
num\_classes = 10  
  
batch\_size = 32  
num\_epochs = 50  
hidden\_size = 512  
  
(X\_train, y\_train), (X\_test, Y\_test) = mnist.load\_data()  
  
X\_train = X\_train.reshape(X\_train.shape[0], height, width, depth)  
X\_train = X\_train.astype('float32')  
X\_train /= 255  
  
X\_test = X\_test.reshape(X\_test.shape[0], height, width, depth)  
X\_test = X\_test.astype('float32')  
X\_test /= 255  
  
Y\_train = to\_categorical(y\_train, num\_classes=num\_classes)  
Y\_test = to\_categorical(Y\_test, num\_classes=num\_classes)  
  
start\_time = time.time()  
  
model = keras.Sequential()  
  
model.add(keras.layers.Conv2D(32, kernel\_size=(3, 3), activation='relu', input\_shape=(width, height, depth)))  
model.add(keras.layers.MaxPooling2D(pool\_size=(2, 2)))  
  
# model.add(keras.layers.Conv2D(32, kernel\_size=(3, 3), activation='relu', input\_shape=(width, height, depth)))  
# model.add(keras.layers.MaxPooling2D(pool\_size=(2, 2)))  
# model.add(keras.layers.Conv2D(32, kernel\_size=(3, 3), activation='relu', input\_shape=(width, height, depth)))  
# model.add(keras.layers.MaxPooling2D(pool\_size=(2, 2)))  
  
model.add(keras.layers.Flatten())  
model.add(keras.layers.Dense(128, activation='relu'))  
model.add(keras.layers.Dropout(0.5))  
model.add(keras.layers.Dense(10, activation='softmax'))  
  
model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])  
model.fit(X\_train, Y\_train, batch\_size=batch\_size,epochs=num\_epochs,verbose=1,validation\_split=0.1)  
model.save\_weights(f'{len(model.layers)}-{num\_epochs}-{hidden\_size}.weights.h5')  
print("Выводим результат обучения...")  
  
start\_ev\_time = time.time()  
model.evaluate(X\_test, Y\_test)  
end\_ev\_time = time.time()  
  
accuracy = model.evaluate(X\_test,Y\_test)  
  
end\_time = time.time()  
interval = end\_time - start\_time  
interval\_ev = end\_ev\_time - start\_ev\_time  
print(f'Точность {accuracy[1]\*100:.2f}%')  
print(f"Время Обучения: {interval:.6f} секунд")  
print(f"Время Теста: {interval\_ev:.6f} секунд")