|  |  |
| --- | --- |
| Picture 1 | МИНОБРНАУКИ РОССИИ  федеральное государственное бюджетное образовательное учреждение  высшего образования  «Санкт-Петербургский государственный морской технический университет» (СПбГМТУ) |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Факультет цифровых промышленных технологий

Направление подготовки 09.03.01

"**Интеллектуальные технологии киберфизических систем**"

Лабораторная работа №7

Вариант №15

Студент 2 курса группы 20221

Очного отделения

Руденко Вячеслав Сергеевич

Проверил:

Преподаватель CПбГМТУ

Кайнова Татьяна Денисовна

Санкт-Петербург

2025

СОДЕРЖАНИЕ

[ОБЩАЯ ЧАСТЬ 3](#_Toc191285460)

[1.1 Цель работы 3](#_Toc191285461)

[1.2 Формулировка задачи 3](#_Toc191285462)

[ХОД РАБОТЫ 4](#_Toc191285463)

[ЗАКЛЮЧЕНИЕ 13](#_Toc191285464)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 14](#_Toc191285465)

[Приложение А 15](#_Toc191285466)

# ОБЩАЯ ЧАСТЬ

## Цель работы

В данной лабораторной работе необходимо изучить применение фреймворков глубокого обучения (TensorFlow/Keras) и реализовать нейронную сеть для классификации рукописных цифр из набора данных MNIST. В ходе работы будут использованы продвинутые методы обучения, такие как нормализация, стандартизация и инициализация весов.

## Формулировка задачи

В соответствии с вариантом 15 требуется:

* Реализовать модель с тремя скрытыми слоями.
* Использовать функцию активации tanh.
* Применить инициализацию Глорота.
* Добавить пакетную нормализацию перед функцией активации.
* Выполнить стандартизацию входных данных.

# ХОД РАБОТЫ

Для начала импортируем необходимые библиотеки и настраиваем окружение.

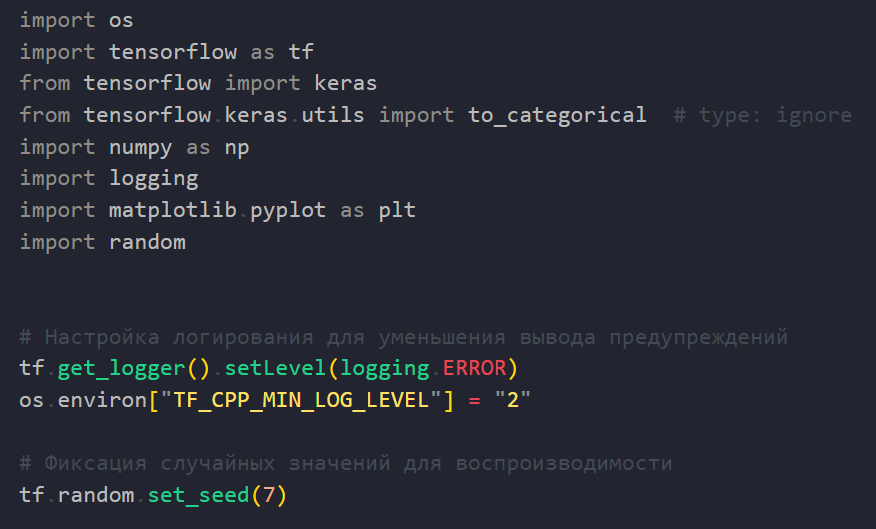


Рисунок 1 – Импорт библиотек и настройка окружения

На рисунке 1 изображен импорт необходимых библиотек, включая TensorFlow, Keras, NumPy и Matplotlib. Для уменьшения количества выводимых предупреждений устанавливается уровень логирования ERROR. Также задается фиксированное начальное состояние генератора случайных чисел, чтобы обеспечить воспроизводимость результатов при повторных запусках модели.

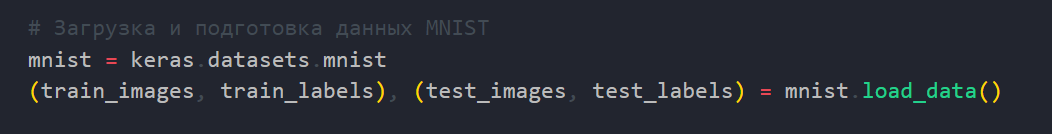


Рисунок 2 – Загрузка данных

Набор данных MNIST представляет собой стандартный набор изображений рукописных цифр размером 28x28 пикселей. На рисунке 2 показано, как данные загружаются в виде массивов: train\_images и train\_labels содержат обучающую выборку, а test\_images и test\_labels — тестовую. Данный этап необходим для подготовки входных данных, которые будут использоваться для тренировки модели.

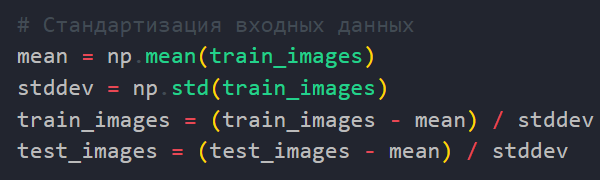


Рисунок 3 – Стандартизация данных

На рисунке 3 изображен код, который производит стандартизацию данных. В первую очередь вычисляется среднее значение (mean) и стандартное отклонение (stddev) пикселей изображений. Затем все изображения нормализуются путём вычитания среднего и деления на стандартное отклонение. Этот процесс приводит данные к распределению с нулевым средним и единичной дисперсией, что помогает ускорить обучение модели и избежать проблем с насыщением нейронов.

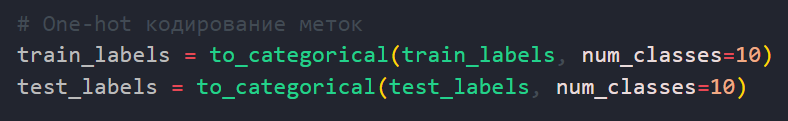


Рисунок 4 – One-hot кодирование меток

Как происходит One-hot кодирование изображено на рисунке 4. One-hot кодирование используется для преобразования числовых меток классов в бинарные вектора. В данном случае цифры от 0 до 9 преобразуются в массивы из 10 элементов, где значение 1 стоит на позиции, соответствующей номеру класса, а остальные элементы равны 0.



Рисунок 5 – Определение модели с тремя скрытыми слоями и функцией активации tanh

Код, изображенный на рисунке 5, задает структуру модели. Входной слой представляет собой слой Flatten, который преобразует изображение размером 28x28 в одномерный массив из 784 элементов, что необходимо для передачи данных в полносвязанные слои. Далее идут три скрытых слоя, содержащие соответственно 64, 32 и 16 нейронов, каждый из которых использует функцию активации tanh.

Перед каждым скрытым слоем применяется пакетная нормализация (BatchNormalization), что позволяет уменьшить внутренние ковариационные сдвиги и ускорить обучение сети. Пакетная нормализация стандартизирует входные данные каждого слоя, снижая зависимость от масштабов входных признаков и улучшая стабильность процесса обучения.

Выходной слой состоит из 10 нейронов с сигмоидной функцией активации. Данный слой отвечает за определение вероятности принадлежности входного изображения к одному из 10 классов. Использование sigmoid позволяет получить выходные значения в диапазоне от 0 до 1, что помогает интерпретировать предсказания модели как вероятности для каждого класса.

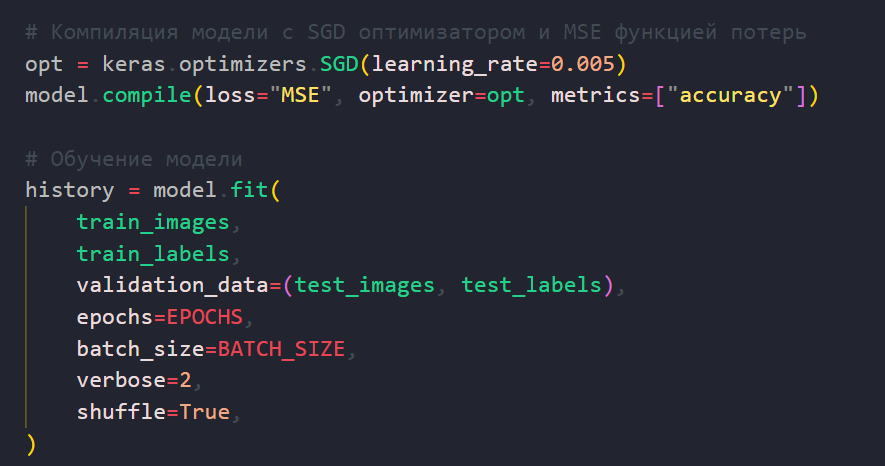


Рисунок 6 – Компиляция и обучение модели

На рисунке 6 изображена компиляция и запуск обучения модели. В качестве оптимизатора применяется стохастический градиентный спуск (SGD) с малым шагом обучения 0.005. Этот метод основан на итеративном обновлении весов нейросети на основе вычисленных градиентов функции потерь, что делает его простым и эффективным для больших объемов данных. Использование малой скорости обучения позволяет более плавно адаптировать веса модели и избежать резких колебаний в процессе оптимизации.

Функция потерь, применяемая в модели, — среднеквадратичная ошибка (MSE). Она вычисляет разницу между предсказанными значениями модели и реальными метками классов, возводит её в квадрат и усредняет. Данный метод помогает минимизировать отклонения предсказаний от истинных значений. Однако, MSE может быть менее предпочтительной для классификационных задач по сравнению с кросс-энтропией, так как она менее чувствительна к большим отклонениям. В данной работе применение MSE оправдано с учетом особенностей задачи.

Обучение модели выполняется в течение 20 эпох с размером мини-пакета 32, что позволяет ускорить вычисления и стабилизировать процесс градиентного спуска. Перед каждой эпохой порядок данных перемешивается, что предотвращает заучивание последовательности примеров и улучшает способность модели к обобщению.



Рисунок 7 – Визуализация результатов

Код, изображенный на рисунке 7, отвечает за визуализацию результатов обучения. Строятся два графика: первый показывает изменение функции потерь, а второй — динамику точности на обучающей и тестовой выборке. Графики позволяют проанализировать процесс обучения модели и выявить возможные проблемы, такие как переобучение.

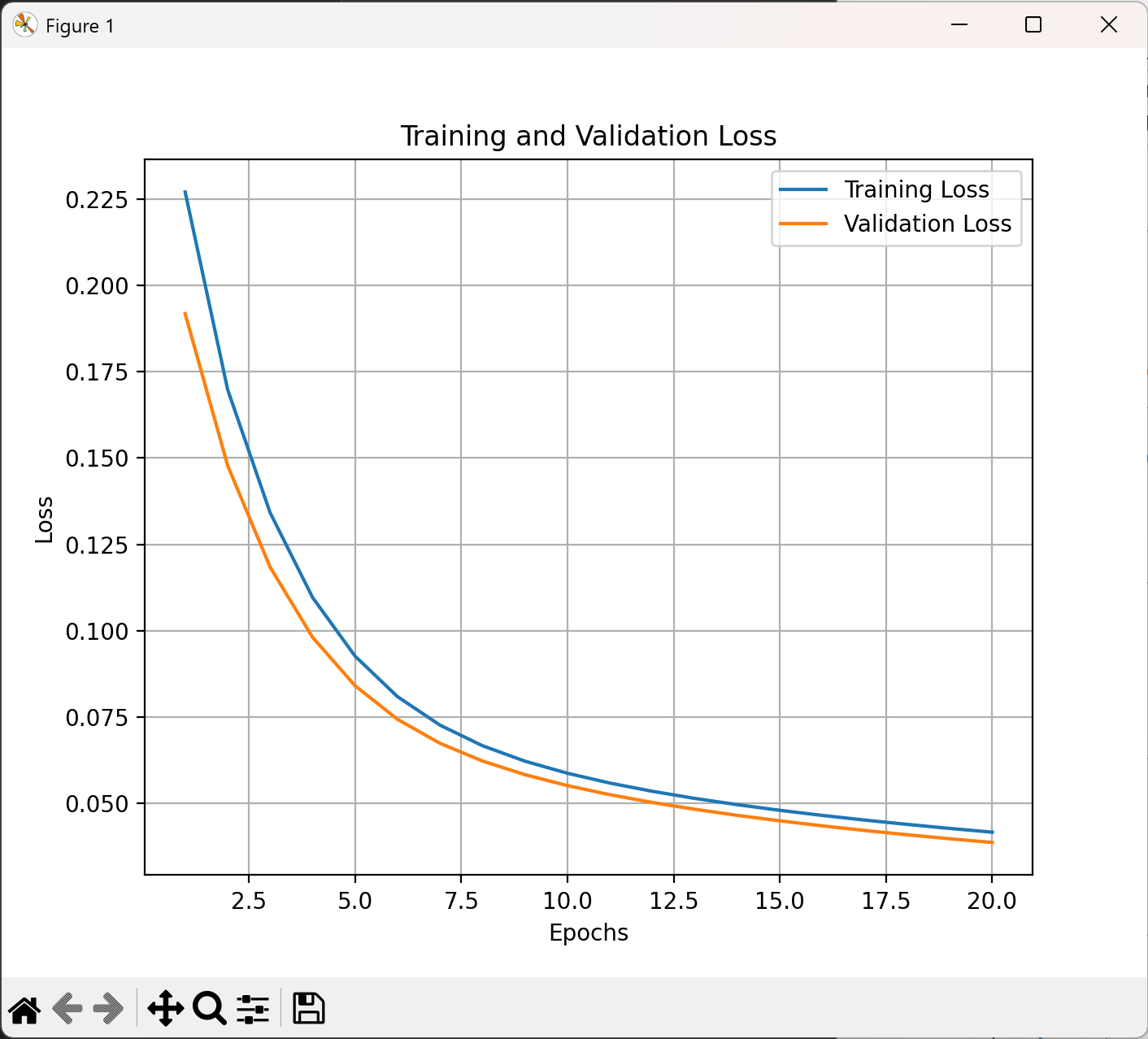


Рисунок 8 – График функции потерь (Loss)

На рисунке 8 изображении представлен график функции потерь (Loss).

Видно, что как на обучающей (синяя линия), так и на валидационной (оранжевая линия) выборках функция потерь стабильно уменьшается на протяжении 20 эпох.

В начале обучения наблюдается быстрое снижение потерь, что свидетельствует о быстрой адаптации модели к данным.

К концу обучения потери уменьшаются, но становятся практически постоянными, что говорит о достижении модели состояния сходимости.

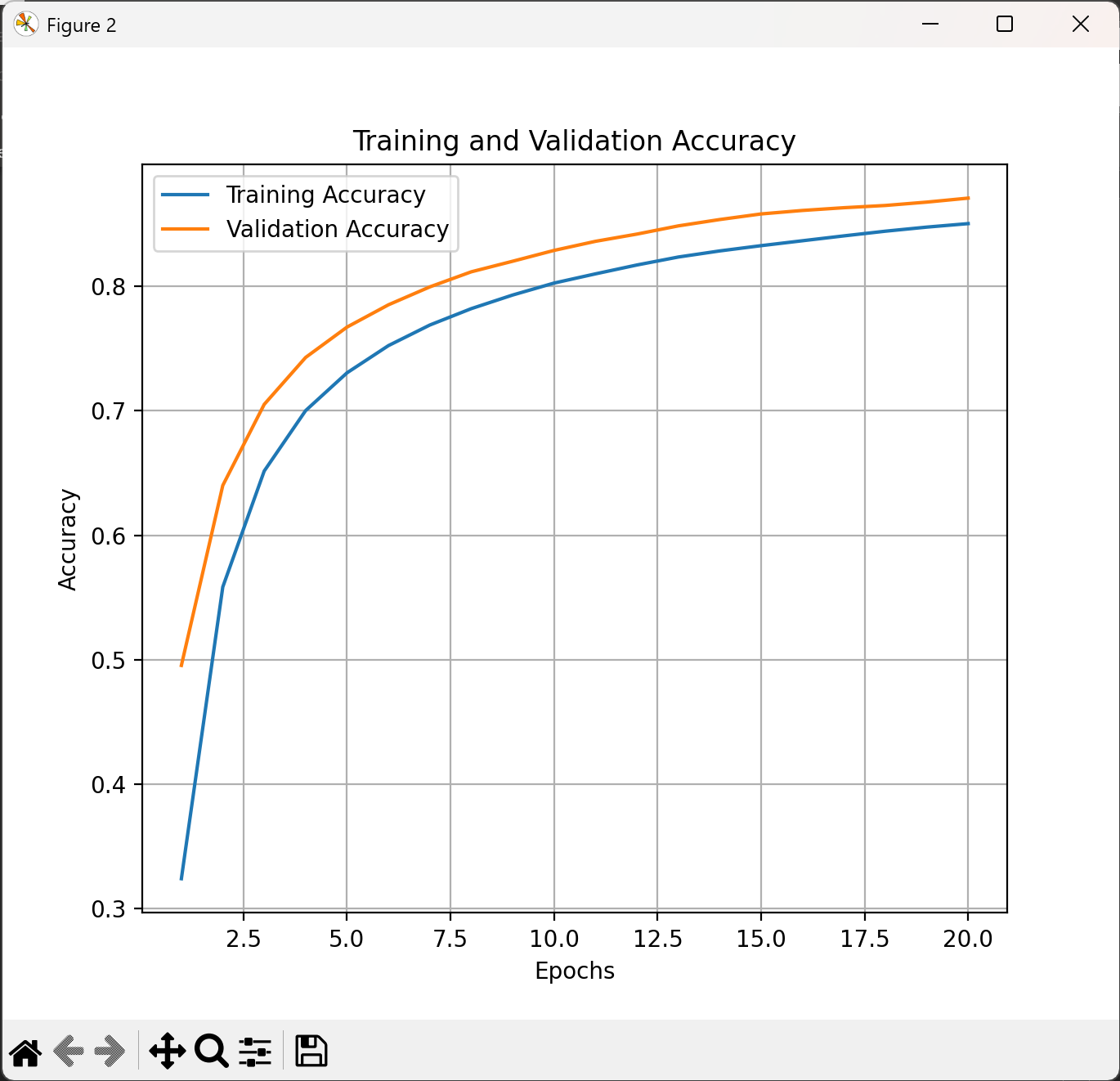


Рисунок 9 – График точности (Accuracy)

На рисунке 9 изображении представлен график функции точности (Accuracy). Точность на валидационной выборке (оранжевая линия) выше, чем на обучающей (синяя линия), что может быть связано с различием в сложности примеров или особенностями обучения.

Видно, что модель быстро набирает точность в первые несколько эпох, затем рост замедляется и выходит на плато.

Итоговая точность модели на тестовых данных превышает 80%.

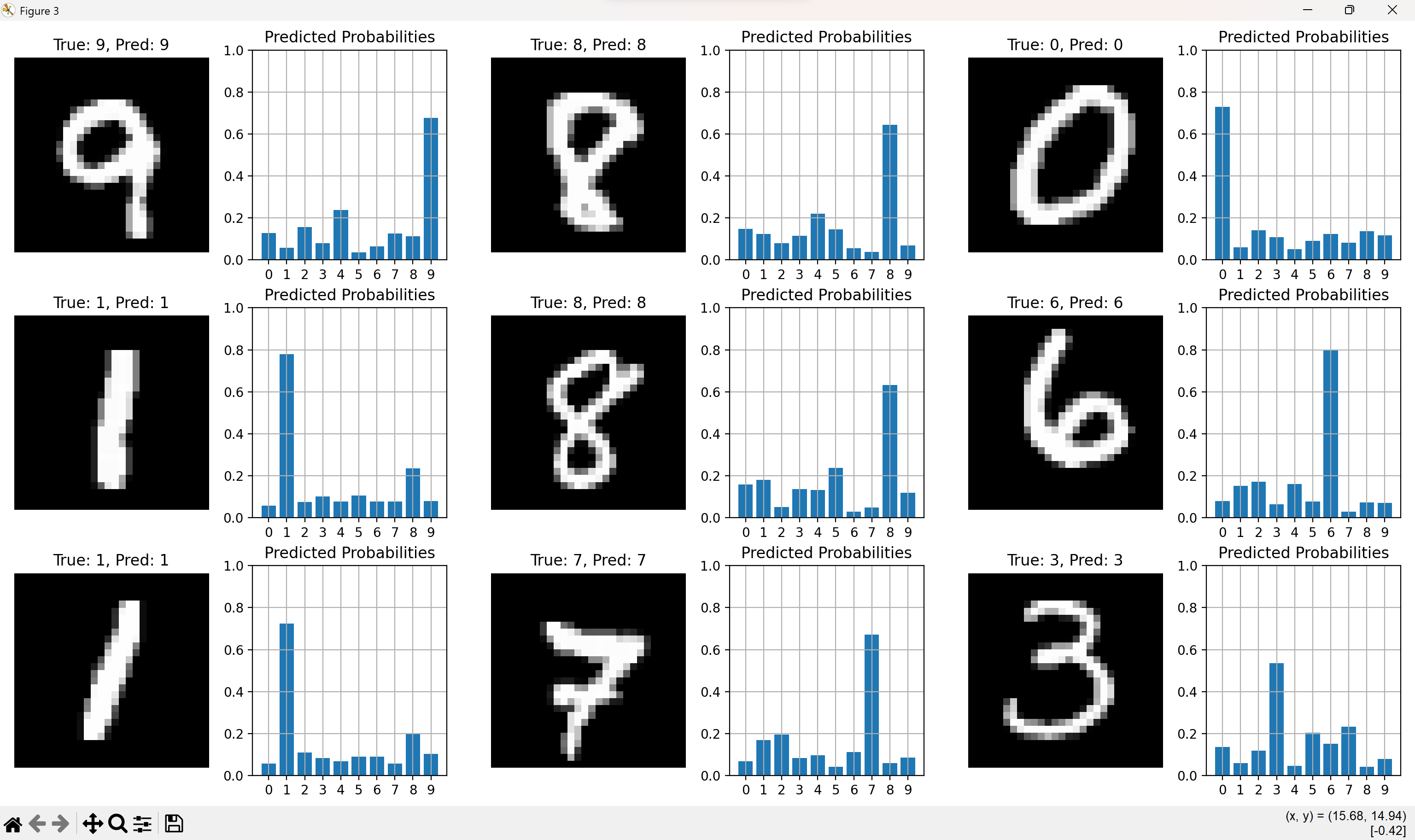


Рисунок 10 – График точности (Accuracy)

На 10 рисунке изображено девять примеров классификации цифр: cлева отображаются оригинальные изображения цифр из MNIST. Справа для каждой цифры показаны распределения вероятностей, предсказанных моделью.

Модель в большинстве случаев предсказывает правильный класс (столбец с наибольшей вероятностью соответствует предсказанию). Визуально цифры хорошо распознаются, что говорит о высокой точности классификации.

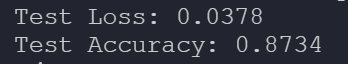


Рисунок 11 – Конечная точность (Loss и Accuracy)

На рисунке 11 изображены конечные показатели нейронной сети после обучения.

Вывод по обучению нейронной:

* Модель обучена успешно, так как графики показывают устойчивое снижение потерь и рост точности.
* Точность на тестовых данных стабильно высокая (более 80%), что подтверждает способность модели обобщать знания на новые данные.
* Потери на обучении и тестировании схожи, что говорит о хорошем качестве модели без явного переобучения.
* Модель уверенно предсказывает правильные классы, что видно по графикам распределения вероятностей.

В целом, полученная модель показывает хороший уровень классификации, а добавление улучшений (другие функции активации, дополнительные регуляризации) может повысить её точность.

# ЗАКЛЮЧЕНИЕ

В данной лабораторной работе было изучено применение фреймворков глубокого обучения (TensorFlow/Keras) и реализована нейронная сеть для классификации рукописных цифр из набора данных MNIST. В ходе работы были использованы продвинутые методы обучения, такие как нормализация, стандартизация и инициализация весов.  
СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. ГОСТ Р 7.0.97-2016. Национальный стандарт Российской Федерации. Система стандартов по информации, библиотечному и издательскому делу. Организационно-распорядительная документация. Требования к оформлению документов: утвержден и введен в действие Приказом Федерального агентства по техническому регулированию и метрологии от 14.05.2018 N 244-ст: Дата введения 2018-07-01. - URL: <https://docs.cntd.ru/document/1200159234> (дата обращения: 02.10.2024). - Текст: электронный.

# Приложение А

Листинг кода

Листинг 1 - index.py

import os

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras.utils import to\_categorical # type: ignore

import numpy as np

import logging

import matplotlib.pyplot as plt

import random

# Настройка логирования для уменьшения вывода предупреждений

tf.get\_logger().setLevel(logging.ERROR)

os.environ["TF\_CPP\_MIN\_LOG\_LEVEL"] = "2"

# Фиксация случайных значений для воспроизводимости

tf.random.set\_seed(7)

# Глобальные параметры обучения

EPOCHS = 20 # Количество эпох обучения (добавлено мной)

BATCH\_SIZE = 32 # Размер батча для обучения (добавлено мной)

# Загрузка и подготовка данных MNIST

mnist = keras.datasets.mnist

(train\_images, train\_labels), (test\_images, test\_labels) = mnist.load\_data()

# Стандартизация входных данных (добавлено мной)

mean = np.mean(train\_images)

stddev = np.std(train\_images)

train\_images = (train\_images - mean) / stddev

test\_images = (test\_images - mean) / stddev

# One-hot кодирование меток

train\_labels = to\_categorical(train\_labels, num\_classes=10)

test\_labels = to\_categorical(test\_labels, num\_classes=10)

# Инициализация весов (Glorot, так как N = 15 нечётное) (добавлено мной)

initializer = keras.initializers.GlorotUniform()

# Определение модели с тремя скрытыми слоями и функцией активации tanh (добавлено мной)

model = keras.Sequential(

[

keras.layers.Flatten(

input\_shape=(28, 28)

), # Преобразование 28x28 вектор в одномерный массив

keras.layers.Dense(

64, kernel\_initializer=initializer, bias\_initializer="zeros"

),

keras.layers.BatchNormalization(), # Пакетная нормализация перед активацией (добавлено мной)

keras.layers.Activation(

"tanh"

), # Используем tanh, так как N нечётное (добавлено мной)

keras.layers.Dense(

32, kernel\_initializer=initializer, bias\_initializer="zeros"

),

keras.layers.BatchNormalization(),

keras.layers.Activation("tanh"),

keras.layers.Dense(

16, kernel\_initializer=initializer, bias\_initializer="zeros"

),

keras.layers.BatchNormalization(),

keras.layers.Activation("tanh"),

keras.layers.Dense(

10,

activation="sigmoid", # На выходе сигмоида для вероятностей классов

kernel\_initializer=initializer,

bias\_initializer="zeros",

),

]

)

# Компиляция модели с SGD оптимизатором и MSE функцией потерь (изменено мной)

opt = keras.optimizers.SGD(

learning\_rate=0.005

) # Уменьшил скорость обучения до 0.005 (добавлено мной)

model.compile(loss="MSE", optimizer=opt, metrics=["accuracy"])

# Обучение модели

history = model.fit(

train\_images,

train\_labels,

validation\_data=(test\_images, test\_labels),

epochs=EPOCHS,

batch\_size=BATCH\_SIZE,

verbose=2,

shuffle=True,

)

def plot\_training\_progress(history):

epochs\_range = range(1, len(history.history["loss"]) + 1)

# График функции потерь

plt.figure(figsize=(10, 5))

plt.plot(epochs\_range, history.history["loss"], label="Training Loss")

plt.plot(epochs\_range, history.history["val\_loss"], label="Validation Loss")

plt.xlabel("Epochs")

plt.ylabel("Loss")

plt.title("Training and Validation Loss")

plt.legend()

plt.grid(True)

# График точности

plt.figure(figsize=(10, 5))

plt.plot(epochs\_range, history.history["accuracy"], label="Training Accuracy")

plt.plot(epochs\_range, history.history["val\_accuracy"], label="Validation Accuracy")

plt.xlabel("Epochs")

plt.ylabel("Accuracy")

plt.title("Training and Validation Accuracy")

plt.legend()

plt.grid(True)

def plot\_predictions(model, test\_images, test\_labels):

indices = random.sample(

range(len(test\_images)), 9

) # Выбираем 9 случайных изображений (добавлено мной)

images = test\_images[indices]

labels = np.argmax(test\_labels[indices], axis=1) # Истинные метки

predictions = model.predict(images) # Предсказания модели

predicted\_labels = np.argmax(predictions, axis=1) # Предсказанные метки

fig, axes = plt.subplots(3, 6, figsize=(15, 10))

for i in range(9):

# Отображение изображения цифры

axes[i // 3, i % 3 \* 2].imshow(images[i], cmap="gray")

axes[i // 3, i % 3 \* 2].set\_title(

f"True: {labels[i]}, Pred: {predicted\_labels[i]}"

)

axes[i // 3, i % 3 \* 2].axis("off")

# Отображение вероятностей предсказания

axes[i // 3, i % 3 \* 2 + 1].bar(range(10), predictions[i])

axes[i // 3, i % 3 \* 2 + 1].set\_xticks(range(10))

axes[i // 3, i % 3 \* 2 + 1].set\_ylim(0, 1)

axes[i // 3, i % 3 \* 2 + 1].set\_title("Predicted Probabilities")

axes[i // 3, i % 3 \* 2 + 1].grid(True)

plt.tight\_layout()

plt.show()

plot\_training\_progress(history)

plot\_predictions(model, test\_images, test\_labels)