Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

Кафедра «Математическое моделирование» (ФН-12)

ОТЧЁТ

по проектно-технологической практике

на предприятии Федеральное государственное бюджетное учреждение науки Институт проблем управления им. В.А.Трапезникова Российской академии наук

Выполнил:
Дорохов М. А.
ФН12-61Б

Руководитель от предприятия:
Макаренко А.В.

_____ / _____

(подпись, дата) (Макаренко А.В.)

Москва
2024

Содержание

1 Постановка задачи

Целью данной проетно-технологической практики является исследование способов оценки различимости классов, алгоритмов их классификации в условиях работы с недостающими данными. Задачи:

1. Исследовать различимость классов в предложенном наборе.
2. Построить нейронную сеть на линейных слоях, классифицирующую объекты.
3. Проверить работу построенной модели на зашумленных данных.

Работа выполняется с помощью языка программирования Python, в среде Visual Studio Code.

2 Ход работы

2.1 Получение набора данных

Для выполнения задания был предложен набор данных MNIST - объёмная база данных образцов рукописного написания цифр. База данных MNIST содержит 60000 изображений для обучения и 10000 изображений для тестирования. Необходимые для исследования данные находится в файлах **train-images-idx3-ubyte.gz**, **train-labels-idx1-ubyte.gz** (обучающий набор), **t10k-images-idx3-ubyte.gz**, **t10k-labels-idx1-ubyte.gz** (тестовый набор).

Набор был открыт и загружен в NumPy массивы с использованием следующего кода:

```
with open('train-labels.idx1-ubyte', 'rb') as f:
    train_labels_data = f.read()

train_labels = np.frombuffer(train_labels_data, dtype = 'uint8', offset = 8)

train_images = []
with open('train-images.idx3-ubyte', 'rb') as f:
    f.read(4 * 4)
    for _ in range (len(train_labels)):
        image = f.read(28 * 28)
        image_np = np.frombuffer(image, dtype = 'uint8') / 255
        train_images.append(image_np)
train_images = np.array(train_images)
```

С помощью функции plot_image, в которой используются методы библиотеки Matplotlib, можно рассмотреть конкретный объект из выборки:

```
def plot_image(pixels):
    plt.imshow(pixels.reshape((28, 28)), cmap = 'gray')
    plt.show()
```

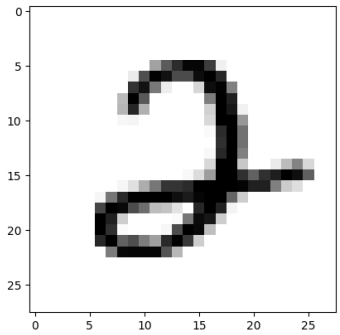

Рис. 1: Пример объекта класса

2.2 Исследование набора данных

Исследование набора MNIST было начато с визуализации объёма каждого класса. Всего в выборке 10 классов – цифры от 0 до 9. Была построена диаграмма количества объектов по всем классам:

```
def bar_count(train_labels):
    numbers, counts = np.unique(train_labels, return_counts=True)
    fig = plt.figure(figsize = (4, 4))
    ax = fig.add_subplot()
    ax.set_xlabel("Цифры")
    ax.set_ylabel("Кол-во")
    ax.set_xticks(numbers)
    ax.bar(numbers,counts)
    fig.savefig(fname = 'labelcount.svg', format = "svg")
```

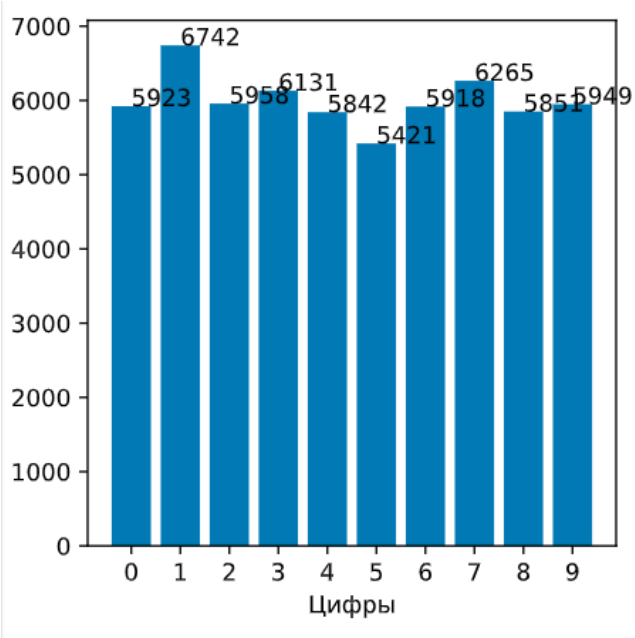

Рис. 2: Столбчатая диаграмма объёма каждого класса

Из столбчатой диаграммы на рис. 2 видно, что больше всего объектов класса 1 (6742 объекта), а меньше всего - объектов класса 5 (5421 объект). Для распределения этих 60000 объектов по классам был применён метод *unique* библиотеки *NumPy*, который принимает массив и возвращает список уникальных значений этого массива, а при истинном параметре *return_ counts* возвращает и их количество в нём.

Были построены диаграммы, показывающие среднюю "закрашиваемость"цифр по пикселям от 0 до 784-го. Объекты в этой работе рассматривались как векторы размерностью 784, учитывая одномерную пространственную зависимость.

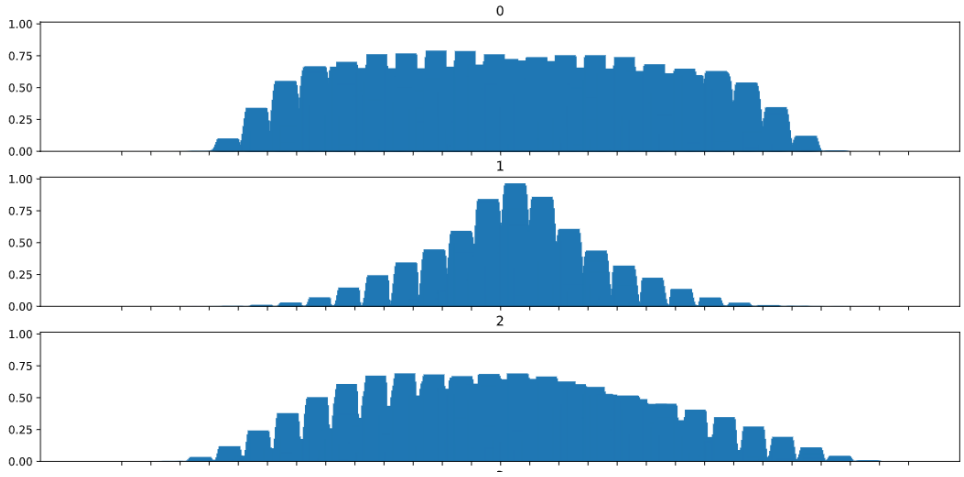

Рис. 3: Средние значения классов 0, 1 и 2

```
def pixel_density(train_images, train_labels):
    images_overall = np.zeros((10, 784))
    for num in np.unique(train_labels):
        images_overall[num] = np.mean(train_images[train_labels == num], axis = 0)
    return images_overall


def plot_density(images_overall, name):
    x = np.arange(0, 784, 1)
    fig, axes = plt.subplots(10, figsize = (15, 25), sharey=True, sharex = True)
    fig.suptitle("Закрашиваемость клеток по цифрам")
    fig.supylabel("Закрашиваемость")

    for num, image in enumerate(images_overall):
        axes[num].set_xticks([k * 28 for k in range(1, 29)])
        axes[num].set_title(str(num))
        axes[num].bar(x, image, width = 20)
    fig.savefig(fname = name, format = "svg")

plot_image(train_images[12])
bar_count(train_labels)
```

Функции *pixel_ density* и *plot_ density*, суммируют значения пикселей для каждого класса и возвращают массив из 784 средних значений по этим пикселям. По нему строится нужная диаграмма.

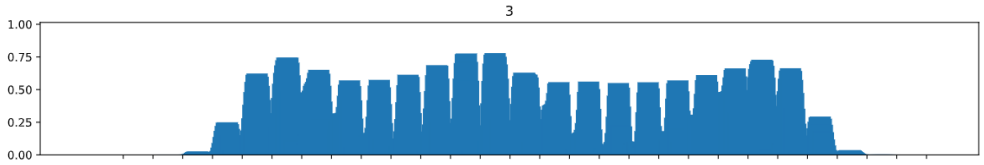

Рис. 4: Средние значения класса 3

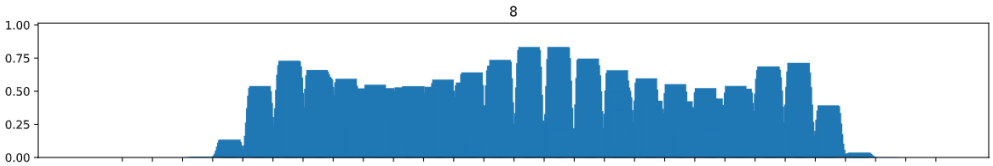

Рис. 5: Средние значения класса 8

На рис. 4 и рис. 5 приведены гистограммы закрашиваемости для изображений, переведенных в одномерный массив построчно. Построим такие же гистограммы, только теперь изображения переводятся в массив по столбцам.

```
train_images_F = np.reshape(train_images, (60000, 28, 28))
train_images_F = np.reshape(train_images_F, (60000, 784), order = 'F')
```

Метод *reshape* изменяет размерность n-мерного *NumPy* массива, а параметр *order = 'F'* указывает на обход изображения по столбцам. По полученным диаграммам видно, что, к примеру, класс 2 имеет отличный от других вид закрашивания, а классы 3 и 8 имеют довольно похожи по средним значениям.

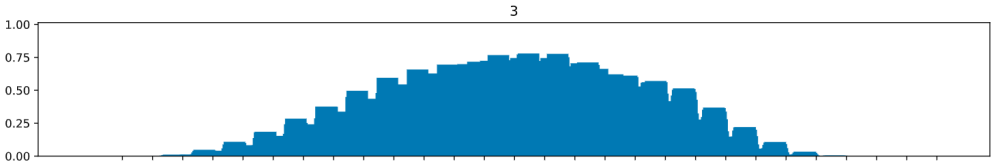

Рис. 6: Средние значения класса 3 для изображения "по столбцам"

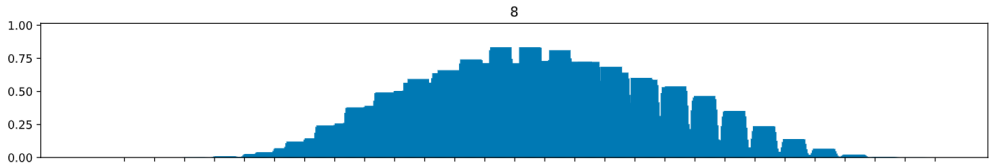

Рис. 7: Средние значения класса 8 для изображения "по столбцам"

Очередным шагом в исследовании было вычисление евклидова и косинусного расстояния между средними векторами для каждого класса, а также средних расстояний между **всеми** векторами классов. Для этого были подключены библиотеки *scipy* и *itertools*.

Косинусное расстояние между векторами a и b, задаётся формулой

$$d_{cos}(a,b) = 1 - \frac{a \cdot b}{\|a\|_2 \|b\|_2} \quad a,\ b \in \mathbb{R}^n$$

а евклидово расстояние задаётся формулой

$$d(a,b) = \left(\sum_{i=0}^{n} (a_i - b_i)^2 \right)^{\frac{1}{2}} \quad a,\ b \in \mathbb{R}^n$$

```
def distance_of_means(images_overall):
    distance_overall_cosine = distance.cdist(images_overall, images_overall,
    "cosine")
    distance_overall_eucl = distance.cdist(images_overall, images_overall,
    "euclidean")
    return distance_overall_cosine, distance_overall_eucl

def plot_distance(distance, name, titl, size):
    fig, axes = plt.subplots(1, 2, figsize = size)
    fig.suptitle(titl)
    names = ["Косинусовое расстояние", "L2-расстояние"]
    for num, d in enumerate(distance):
        axes[num].set(xlabel="Номер класса", ylabel="Номер класса")
        axes[num].set_title(names[num])
        sns.heatmap(d, ax = axes[num], annot=True)
    fig.savefig(fname = name, format = "svg")

def distance_all(train_images1, train_images2, ident):
    print(f"вычисляю...{ident[0]}_{ident[1]}")
    dist_cos = distance.cdist(train_images1, train_images2, "cosine")
    dist_eucl = distance.cdist(train_images1, train_images2, "euclidean")

    with h5py.File(f"dist_{ident[0]}_{ident[1]}", 'w') as f:
            f.create_dataset('dist_cos', data=dist_cos)
            f.create_dataset('dist_eucl', data=dist_eucl)
```

Расстояния подсчитаны с помощью метода *cdist* библиотеки *scipy*, и были записаны в Hdf5 файл. Для ускорения вычислений использовался метод *starmap* библиотеки *multiprocessing*:

```
    images_classed = list(train_images[train_labels == i]
        for i in np.unique(train_labels))

    items = combinations_with_replacement(np.unique(train_labels), 2)

    slices_to_process = []
    for item in items:
        slices_to_process.append((train_images[train_labels==item[0]],
            train_images[train_labels==item[1]], item))

    start = time.time()
    with Pool() as pool:
       pool.starmap(distance_all, slices_to_process)
```

Метод *combinations_with_replacement* библиотеки *itertools* принимает массив и возвращает всевозможные комбинации его значений. Второй параметр равен 2, поэтому возвращались пары значений. Это было сделано, чтобы разбить метки классов попарно и дать их на вход методу *distance_all* для подсчета всевозможных расстояний. Все вычисления расстояний проводились параллельно.

В итоге получена матрица расстояний между средними векторами, где столбцы и строки — метки классов, а цвет ячейки отражает величину значения в ячейке (с выводом самих значений). Для средних расстояний между классами было построено 50 гистограмм в виде квадратной картинки (расстояния между 0 и 0, 0 и 1, 0 и 2, ...). На каждой гистограмме построены 25-й и 75-й квантили, среднее и медиана.

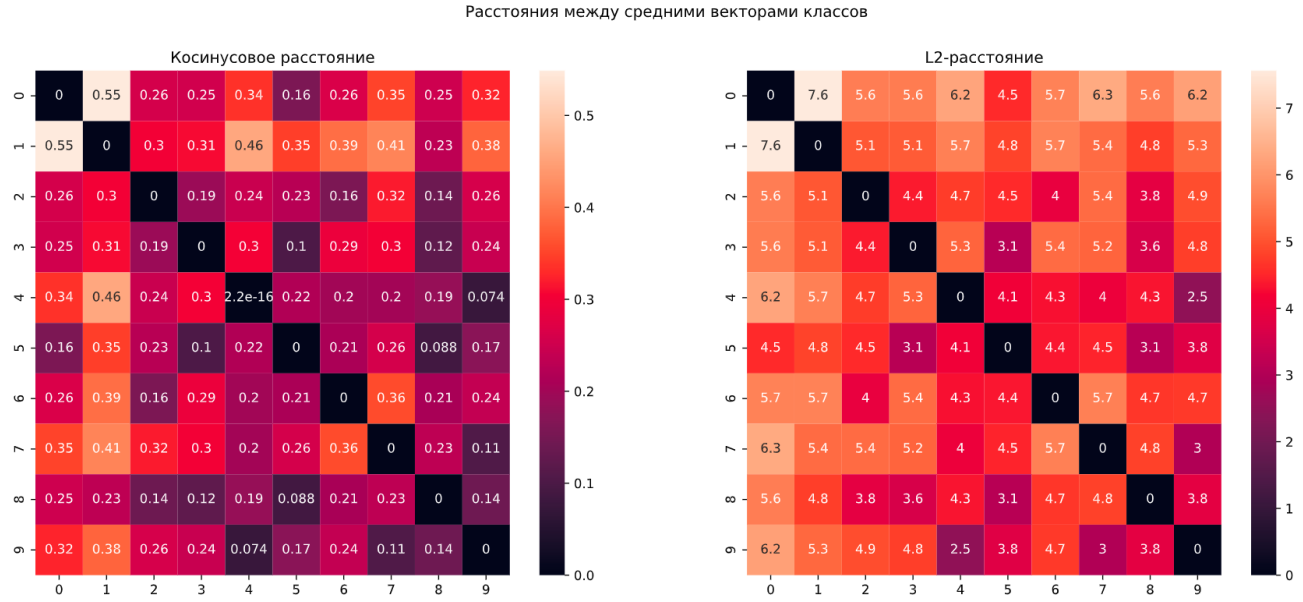

Рис. 8: Матрицы расстояний между средними векторами

По построенным матрицам на рис. 8 видно, что самыми различимыми являются классы 0 и 1 (их косинусное расстояние равно 0.55, а самыми трудно различимыми являются классы 4 и 9. Это можно объяснить разными способами написания цифры 4. В способе написания этой цифры "треугольником"она довольно похожа на цифру 9, отсюда большое сходство этих классов в среднем.

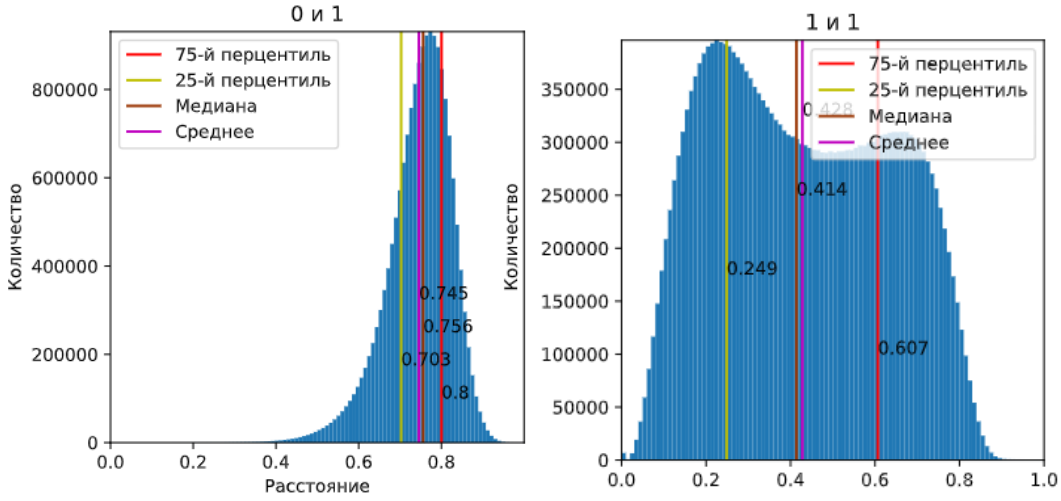

Рис. 9: Средние косинусные расстояния между классами 0 и 1, 1 и 1

Рассмотрим гистограммы средних расстояний между классами. Выберем пару примечательных из них – расстояния между 0 и 1, и расстояния между 1 и 1. На первой гистограмме можно заметить, что её значения резко возрастают и резко убывают, причем значение расстояния довольно велико. Это происходит по причине сильной различности в написании цифр 0 и 1. На правой гистограмме можно отметить две точки максимума, которые появились за счет множества способов писать цифру 1. Можно заметить, как на гистограмме 1 и 1, в точке 0 ненулевое значение. Из этого можно сделать вывод, что в MNIST не все объекты классов уникальны, т.е. там находятся идентичные.

Вообще говоря, для оценки сходства классов в случае набора данных MNIST целесообразно использовать именно **косинусную** меру, поскольку она более пригодна как оценочная мера для сходства, к тому же у неё есть полезное свойство: она позволяет работать с разре-

женными данными. Евклидова мера же является мерой геометрического расстояния между объектами.

К примеру, рассмотрим гистограммы для классов 1 и 4. Гистограмма косинусного расстояния (слева) показвыает, что классы 1 и 4 довольно схожи между собой (много расстояний, примерно равных 0.9). По гистогрмаме евклидова расстояния это увидеть нельзя, поскольку это мера не оценивает сходство между объектами.

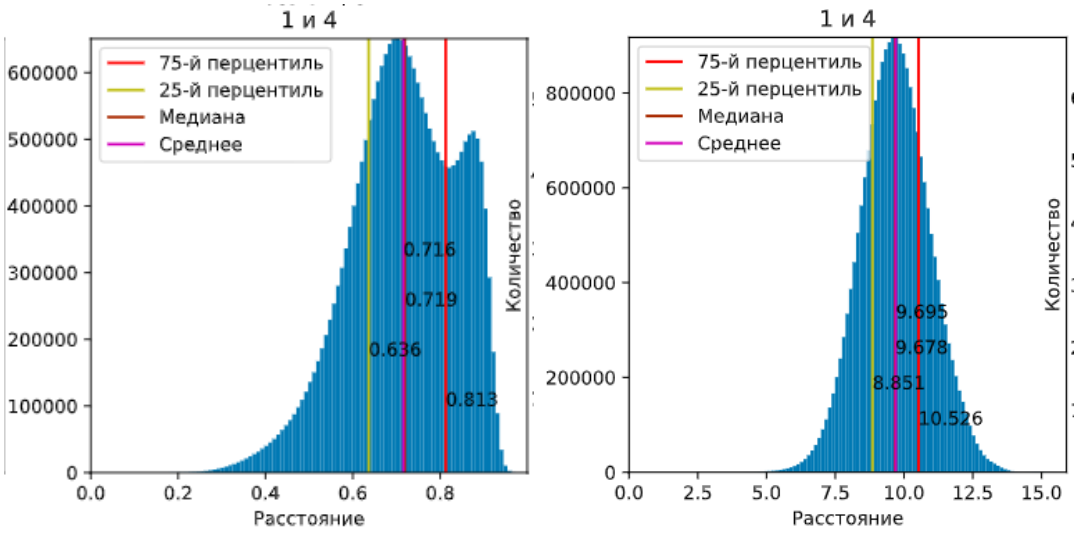

Рис. 10: Средние расстояния между классами 1 и 4 (слева - косинусное, справа - евклидово)

2.3 Построение модели для классификации

Нейронная сеть —- это сложная дифференцируемая функция, задающая отображение из исходного признакового пространства в пространство ответов, все параметры которой могут настраиваться одновременно и взаимосвязанно.

Линейный слой – линейное преобразование над входящими данными (его обучаемые параметры — это матрица $W \in \mathbb{R}^{d \times k}$, и вектор $b \in \mathbb{R}^k$): $x \mapsto Wx + b,\ x \in \mathbb{R}^d$

Функция активации — нелинейное преобразование, поэлементно применяющееся к пришедшим на вход данным. Благодаря функциям активации нейронные сети способны порождать более информативные признаковые описания, преобразуя данные нелинейным образом.

Построение нейронной сети на линеных слоях будем выполнять с помощью библиотек *Tensorflow* и *Keras*:

```
def train_model(train_images, model_name):
model = keras.Sequential([
    Dense(128, activation='relu', input_shape=(784, )),
    Dropout(0.2, input_shape=(784, )),
    Dense(64, activation='relu', input_shape=(784, )),
    Dense(10, activation='softmax')
])
opt = keras.optimizers.Adam(learning_rate=0.001)
model.compile(optimizer=opt,
         loss='categorical_crossentropy',
         metrics=['accuracy', f1_score])
history = model.fit(train_images, target_train_cat, batch_size=32,
    epochs=EPOCHS, validation_split = 0.2)
table = np.zeros((3, EPOCHS))
vtable = np.zeros((3, EPOCHS))
table[0], table[1], table[2], vtable[0], vtable[1], vtable[2]
    = history.history['loss'], history.history['accuracy'],
            history.history['f1_score'], history.history['val_loss'],
                history.history['val_accuracy'], history.history['val_f1_score']
model.save(model_name)
return table, vtable, history.history['accuracy'][EPOCHS-1]
```

В качестве оптимизатора был выбран *Adam* со скоростью обучения 0.001.

Отслеживаемые метрики – *accuracy, f1_ score*. Метрика *accuracy* – это отношение правильно предсказанных объектов к общему их числу:

$$\text{accuracy} = \frac{\text{predicted}}{\text{total}}$$

Для описания метрики *f1_ score* введём обозначения:

- TP - истинно-положительный результат,

- FP - ложно-положительный результат,

- TN - истинно-отрицательный результат,

- FN - ложно-отрицательный результат.

Также введём *Precision* и *Recall*:

$$\text{Precision} = \frac{TP}{TP + FP}, \quad \text{Recall} = \frac{TP}{TP + FN}.$$

Значение *Precision* показывает насколько редко модель делает ложно-положительные выводы, а *Recall* - как хорошо она делает истинно-положитеьные выводы. Метрика *f1_ score* - гармоническое среднее вышеуказанных двух:

$$\text{F1} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}.$$

Функция потерь – *categorical_ crossentropy* (категориальная кросс-энтропия), используемая в задачах многоклассовой классификации. Задаётся следующей формулой:

$$H(y, \hat{y}) = -\sum_i y_i log(\hat{y}_i),$$

где y_i - вероятность истинного значения, $\hat{y}_i$ - вероятность предсказанного значения.

Была построена нейронная сеть из 4 слоёв, в качестве функций активации были взяты:

- *relu* - функция активации, вообще говоря, нелинейная, как и любые её комбинации. Главная особенность - ненулевые производные в точках $x > 0$.

$$\text{relu}(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases}.$$

- *softmax* - функция активации, возвращающая вероятностную интерпретацию поступивших на вход данных. Используется в качестве последнего слоя нейронной сети.

$$\text{softmax}(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}.$$

- *linear* - линейная функция активации, в конечном варианте модели не используется. Скорее пригодна для задач регрессии, где выходное значение нейросети $y \in \mathbb{R}^m,\ m \geq 1$.

$$\text{linear}(x) = x.$$

Использован слой *Dropout*, исключающий определённый процент (в данной модели 20%) случайных нейронов на разных эпохах во время обучения нейронной сети. Это очень эффективный способ усреднения моделей внутри нейронной сети. В результате более обученные нейроны получают в сети больший вес.

Модель обучалась 15 эпох, с размером батча 32. Для валидации было выделено 20% от тестовых данных. Обучение проводится методом *fit* библиотеки *Keras*. Модель сохранена в файл **"model_0.keras"**. значения функции потерь и метрик на каждой эпохе сохранены в массив *history*, по нему построены их графики.

Модель обработала тестовые данные, и по предсказанным значениям для оценки качества работы были построены матрицы ошибок. Матрица ошибок - это матрица 2×2 вида

Значение F1 для модели во время обучения получилось равным около 0.98, после теста - 0.977, поэтому можно предположить, что модель получилась качественной. Проверить это мы можем с помощью построенных матриц ошибок.

		Predicted	
Actual		TP	FP
		FN	TN

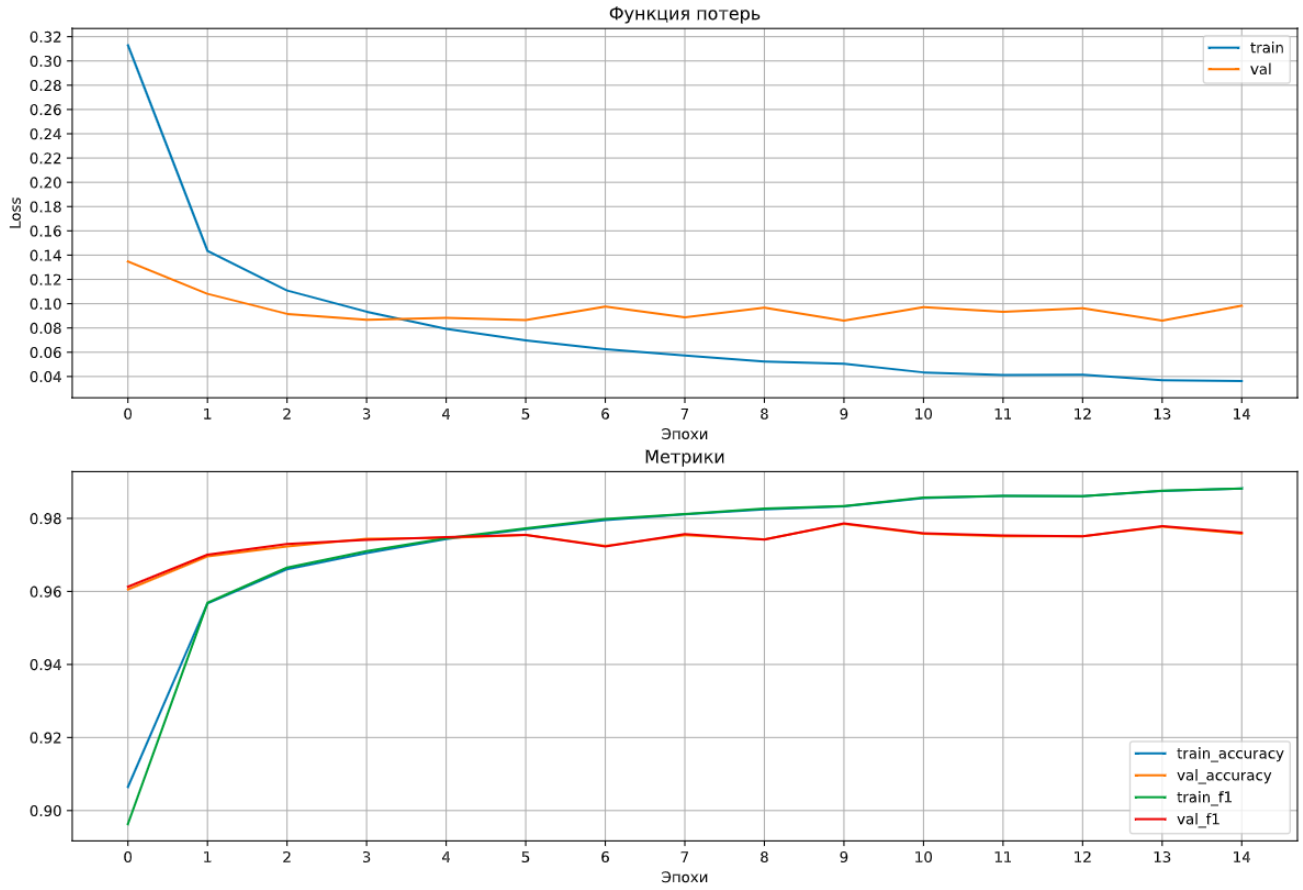

Рис. 11: График функции потерь и метрик

По матрицам ошибок на рис. 11 видно, что неправильно предсказан довольно малый процент классов. Например, в случае класса 6 TN = 23, FN = 21, TP = 8846, FP = 937, тоесть правильно предсказаны 8846+937 = 9783 объекта, неправильно – 44. Матрицы ошибок строились с помощью метода *confusion_matrix* библиотеки *scikit-learn*. Также был построен аналог для всех 10 классов.

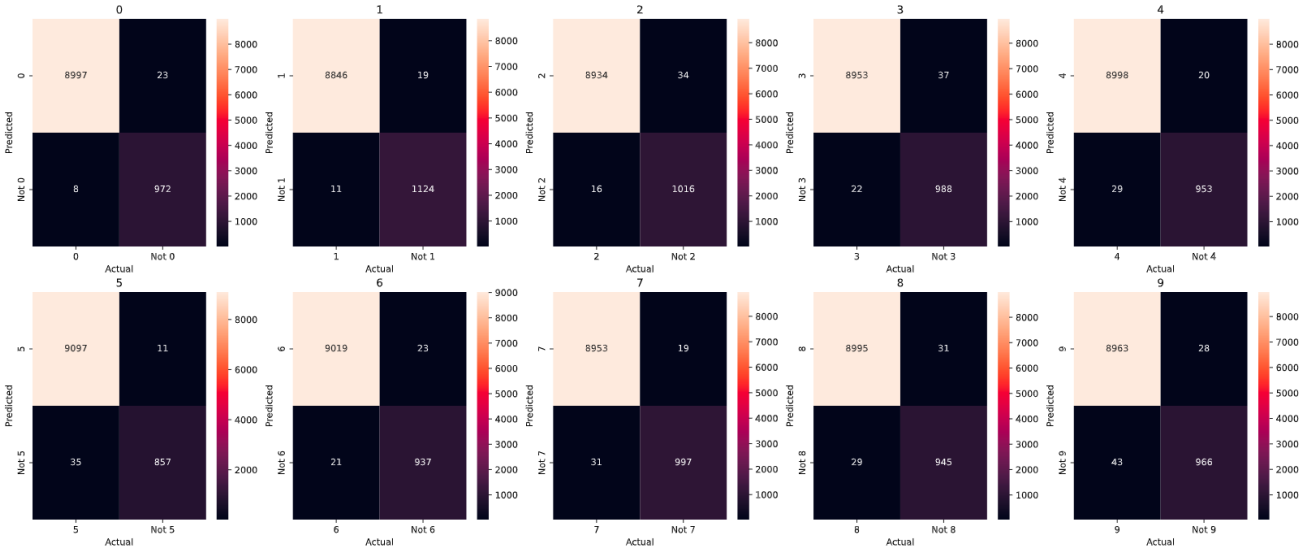

Рис. 12: Матрицы ошибок на тестовых данных

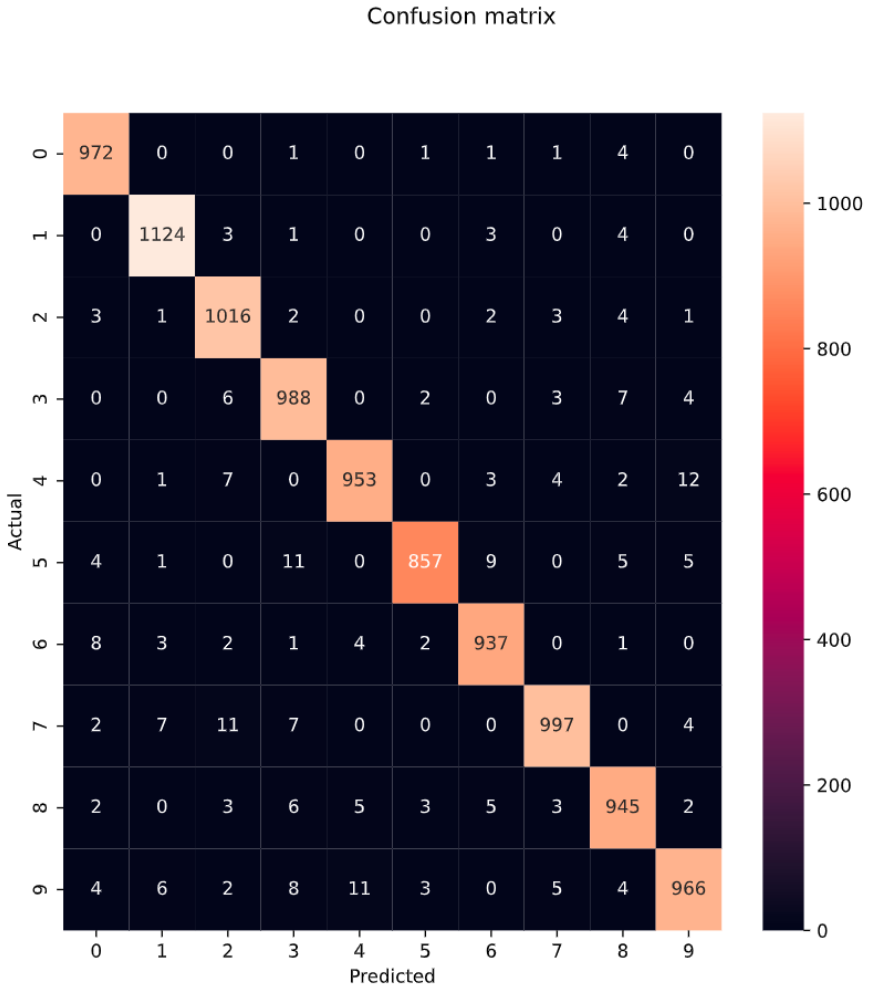

Рис. 13: Матрица ошибок по всем классам на тестовых данных

2.4 Проверка модели на зашумлённых данных

Следующий шаг - проверка реализованной модели на данных с шумами. Шумы добавлялись следующим образом: выбирался процент шума (5%, 10%, ..., 25%), затем с помощью библиотеки *random* случайно выбирался соответствующий процент пикселей и полностью закрашивались (присваивалось значение 1).

```
def create_dataset():
with h5py.File('datasets_and_model_metrics.hdf5', 'w') as f:
    f.create_dataset("noise_0/datasets/_train", data = train_images)
    f.create_dataset("noise_0/datasets/_test", data = test_images)
tables =  np.zeros((6, 3, EPOCHS))
vtables =  np.zeros((6, 3, EPOCHS))
indexes = np.arange(784)

t = 0
tables[t], vtables[t], _ = train_model(train_images, f"model_{0}.keras")
for s in [0.05, 0.1, 0.15, 0.2, 0.25]:
    t += 1
    te_img = test_images
    tr_img = train_images
    for img in tr_img.reshape(60000, 784):
        random.shuffle(indexes)
        for i in indexes[:int(s*784)]:
            img[i] = 1

    for img in te_img.reshape(10000, 784):
        random.shuffle(indexes)
        for i in indexes[:int(s*784)]:
            img[i] = 1

    with h5py.File('datasets_and_model_metrics.hdf5', 'a') as f:
        f.create_dataset(f"noise_{int(s*100)}/datasets/_train", data = tr_img)
        f.create_dataset(f"noise_{int(s*100)}/datasets/_test", data = te_img)
    tables[t], vtables[t], _ = train_model(tr_img, f"model_{int(s*100)}.keras")
```

Полученные наборы данных сохранялись в файл **"datasets_and_model_metrics.hdf5"**. На каждом из наборов (оригинальном и зашумлённых) была обучена реализованная модель нейросети, результаты обучения сохранены в этот же файл. Каждая модель была протестирована на тестовых данных с зашумленностью 5%, 10%, ..., 25%. По результатам была построена таблица 6×6 со значениями *accuracy*. Для новых моделей так же, как и для оригинальной, были построены графики фукнции потерь и метрик, а также матрицы ошибок.

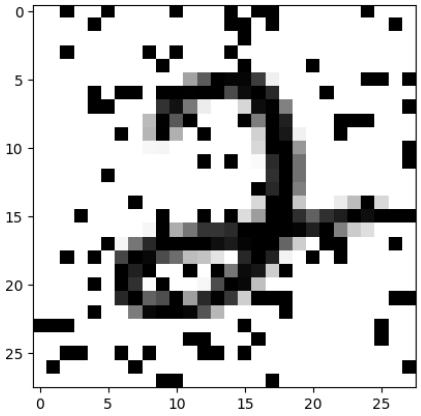

Рис. 14: Зашумленный объект класса (10% шума)

По построенной матрице ошибок для данных с шумами видно, что модель стала чаще ошибаться в определении класса.

На таблице *table_ accuracy*, как видно на рис. 20, изображены значения метрики *F1* для моделей, обученных и протестированных на данных с разными шумами. По горизонтали - варианты зашумления (от меньшего к большему), на которых модель тестировалась. По вертикали - варианты, на которых она обучалась. Так, модель, обученная на чистых данных, довольно неплохо срабатывает на зашумленных, однако, модель, обученная на зашумленных данных, плохо срабатывает на всех вариантах зашумления. Таблица строилась с помощью функции *create_ accuracy_ table*.

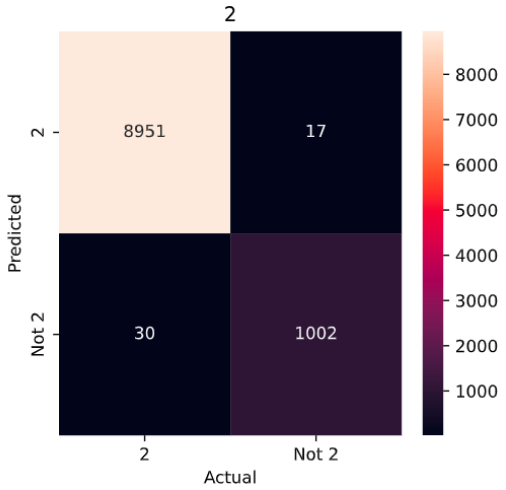

Рис. 15: Матрица ошибок класса 2 (5% шума)

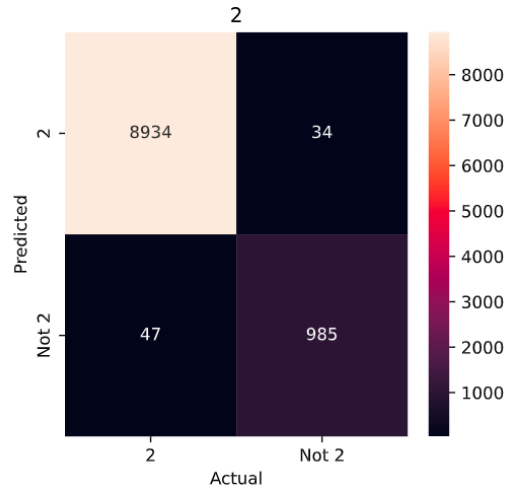

Рис. 16: Матрица ошибок класса 2 (10% шума)

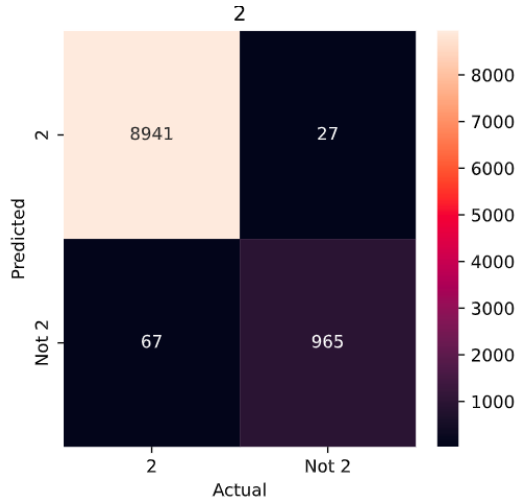

Рис. 17: Матрица ошибок класса 2 (15% шума)

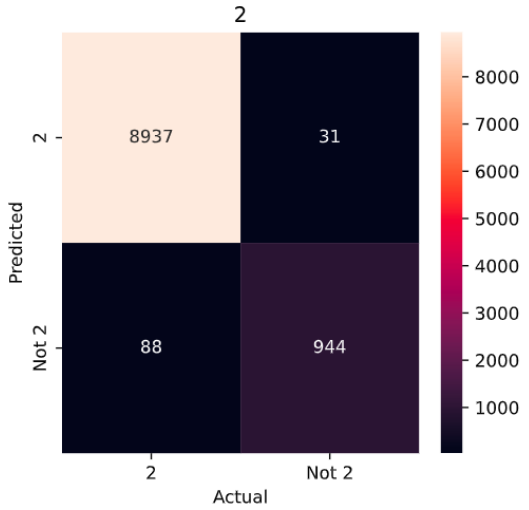

Рис. 18: Матрица ошибок класса 2 (20% шума)

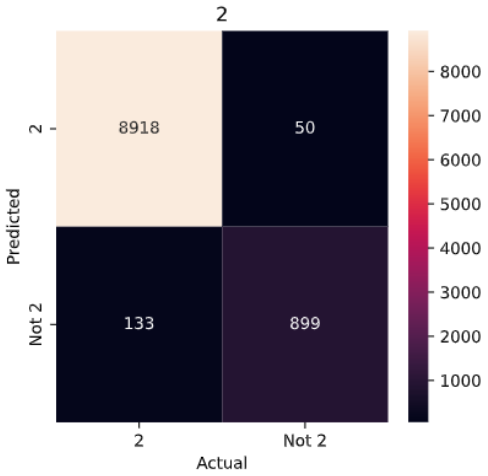

Рис. 19: Матрица ошибок класса 2 (25% шума)

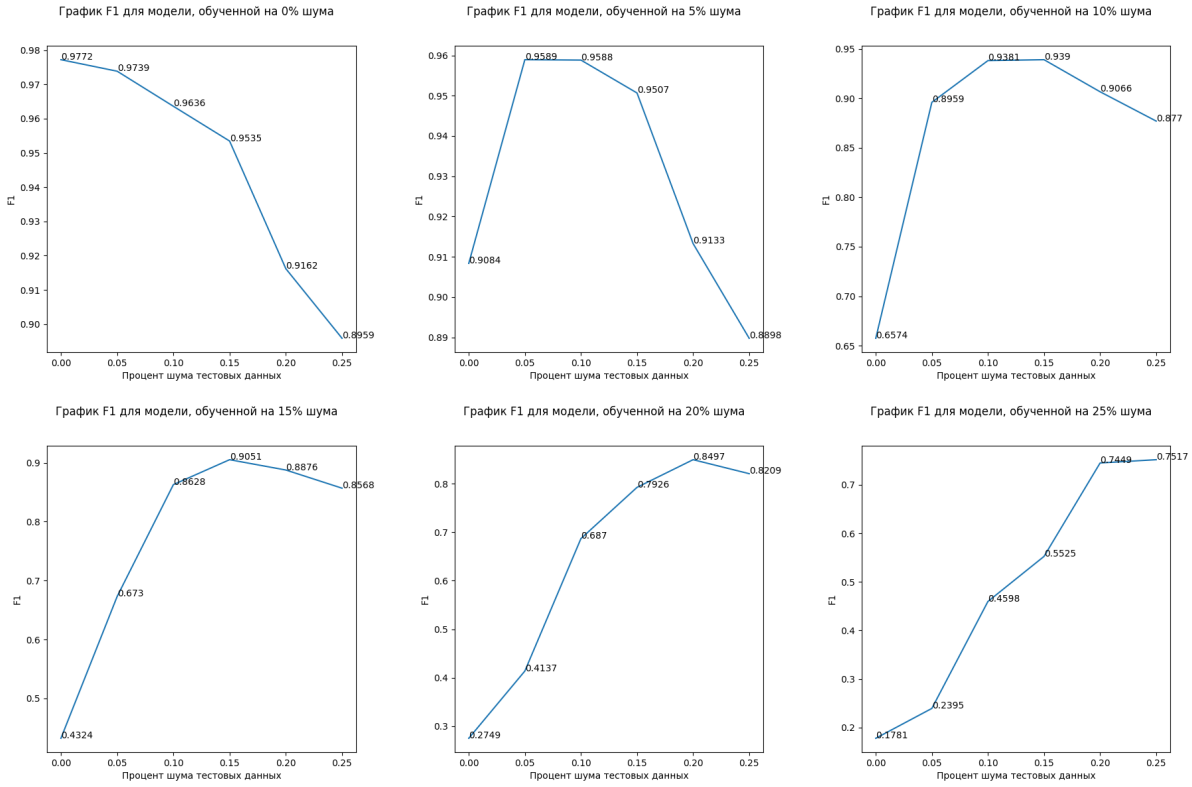

Рис. 20: График метрики F1 по результатам тестирования модели

```
def create_accuracy_table():
matrix = np.zeros((6,6))
matrix1 = np.zeros((6,6))
i = 0
for n in [0, 0.05, 0.1, 0.15, 0.2, 0.25] :
    j = 0
    model, _, _= load_model(int(n*100))
    for m in [0, 0.05, 0.1, 0.15, 0.2, 0.25]:
        with h5py.File("datasets_and_model_metrics.hdf5", 'r') as f:
            test_im = f[f"noise_{int(m*100)}/datasets/_test"]
            test_images = test_im[:]
        result = model.evaluate(test_images, target_test_cat)
        print(result)
```

```
            matrix[i][j] = result[1]
            matrix1[i][j] = result[2]
            j += 1
        i += 1
    with h5py.File("datasets_and_model_metrics.hdf5", 'a') as f:
        f.create_dataset("table_models_accuracy", data = matrix)
        f.create_dataset("table_models_f1_score", data = matrix1)
```

Процент шума	0%	5%	10%	15%	20%	25%
0%	0.977	0.973	0.963	0.953	0.916	0.895
5%	0.908	0.958	0.958	0.950	0.913	0.889
10%	0.657	0.895	0.938	0.939	0.906	0.876
15%	0.432	0.673	0.862	0.905	0.887	0.856
20%	0.274	0.413	0.686	0.792	0.849	0.820
25%	0.178	0.239	0.459	0.552	0.744	0.751

Таблица 1: Таблица метрики F1 по итогам тестирования моделей (по горизонтали - для тестовых данных, по вертикали - для обучающих.

3 Заключение

В процессе выполнения задания была разработана модель на линейных слоях для классификации набора MNIST. Был исследован предложенный набор данных, исследована различимость классов в нём. Получен опыт работы с библиотеками *NumPy, Matplotlib, Keras, Tensorflow, Scikit-learn, Scipy* для языка программирования Python. Освоен формат файлов HDF5. Получена полносвязная нейронная сеть, её работа проверена на зашумленных данных. Обучение модели на чистых данных привело к высокой её эффективности, однако обучение на зашумленных вызвало проблемы в распознавании данных с другим уровнем шума. По результатам тестирования модели, обученной на чистых данных, было получено значение $f1\_score = 0.97$. Построены диаграммы распределения косинусного и евклидова расстояний между классами набора. По ним были сделаны выводы о наличии идентичных объектов, о сходстве классов 1 и 4, 3 и 8. Изучены методы построения нейросетей и способы оценки их качества. Построены матрицы ошибок на тестовых данных для проверки работы моделей. Задача была решена полностью.

Список литературы

[1] Официальный сайт Python / URL: https://www.python.org/.

[2] Документация NumPy / URL: https://numpy.org/doc/.

[3] Официальный сайт Matplotlib / URL: https://matplotlib.org/.

[4] Официальный сайт Keras / URL: https://keras.io/.

[5] Официальный сайт Tensorflow / URL: https://www.tensorflow.org/.

[6] MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges / URL: https://yann.lecun.com/exdb/mnist/.

[7] Официальный сайт scikit-learn / URL: https://scikit-learn.org/stable/.

[8] Линейные нейронные сети / URL: https://docs.exponenta.ru/deeplearning/ug/linear-neural-networks.html

[9] Confusion matrix / URL: https://en.wikipedia.org/wiki/Confusion_matrix