МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Центр цифровых
образовательных технологий

09.04.01 Информатика и вычислительная техника

# Шумоподавление на изображениях
Вариант 2

ЛАБОРАТОРНАЯ РАБОТА № 4

по дисциплине:
**Машинное обучение**

**Исполнитель:**

студент группы 8ВМ42 Текере Ричард

**Руководитель:**

доцент ОИТ, ИШИТР Друки А.А.

Томск - 2025

## ВВЕДЕНИЕ

Автокодировщик (autoencoder) – это разновидность нейронной сети, которая обучается воспроизводить на выходе свои же входные данные. Такая сеть состоит из двух основных частей: кодировщика, сжимающего входные данные в скрытое представление (снижение размерности), и декодировщика, восстанавливающего исходные данные из этого сжатого представления. В процессе обучения автокодировщик находит внутренние особенности (признаки) данных и учится эффективно кодировать информацию.

Особым типом автокодировщиков является denoising автокодировщик – автокодировщик для удаления шума. Его обучают на зашумленных входных данных, используя при этом исходные незашумленные данные в качестве целевых значений. Цель такого обучения – заставить модель восстанавливать чистые данные из искаженных, тем самым подавляя шум. В данной работе рассматривается использование автокодировщика для задачи удаления шума на примере изображений рукописных цифр из набора данных MNIST. Набор MNIST содержит 60000 обучающих и 10000 тестовых изображений размером 28×28 пикселей в оттенках серого, каждое изображение соответствует одной рукописной цифре от 0 до 9. Автокодировщик будет обучен восстанавливать исходные изображения цифр из их зашумленных версий, демонстрируя тем самым способности нейронных сетей выявлять существенные характеристики изображений и удалять случайный шум.

### Цель работы

Приобрести навыки использования автокодировщиков для шумоподавления в изображениях.Задачи работы:

1. Загрузить и предварительно обработать набор данных MNIST.
2. Добавить случайный шум к изображениям.
3. Построить модель автокодировщика с использованием Keras.
4. Обучить модель на зашумленных изображениях.
5. Оценить качество работы модели.

**Ход работы**

1. Загрузка и подготовка данных MNIST

Первым шагом мы загружаем набор данных MNIST и выполняем его предварительную обработку. Данные представляют собой изображения цифр размером 28×28, которые удобно нормализовать в диапазон [0,1] для эффективного обучения нейронной сети. Кроме того, для использования сверточных слоёв необходимо добавить измерение канала (так как изображения grayscale имеют один канал). Ниже приведен код загрузки данных, нормализации значений пикселей и преобразования размеров массивов:

```
from keras.datasets import mnist
import numpy as np
import matplotlib.pyplot as plt

# Загрузка набора данных MNIST
(x_train, _), (x_test, _) = mnist.load_data()

# Масштабирование пикселей к диапазону [0,1]
x_train = x_train.astype('float32') / 255.0
x_test  = x_test.astype('float32') / 255.0

# Добавление измерения канала (1) к изображениям
x_train = np.expand_dims(x_train, axis=-1)
x_test  = np.expand_dims(x_test, axis=-1)

# Визуализация нескольких исходных изображений из обучающей выборки
idx = np.random.randint(0, x_train.shape[0], size=10)  # случайные индексы
plt.figure(figsize=(10,4))
for i, index in enumerate(idx):
    plt.subplot(2, 5, i+1)
    plt.imshow(x_train[index].reshape(28, 28), cmap='gray')
    plt.axis('off')
plt.show()
```

Вывод:

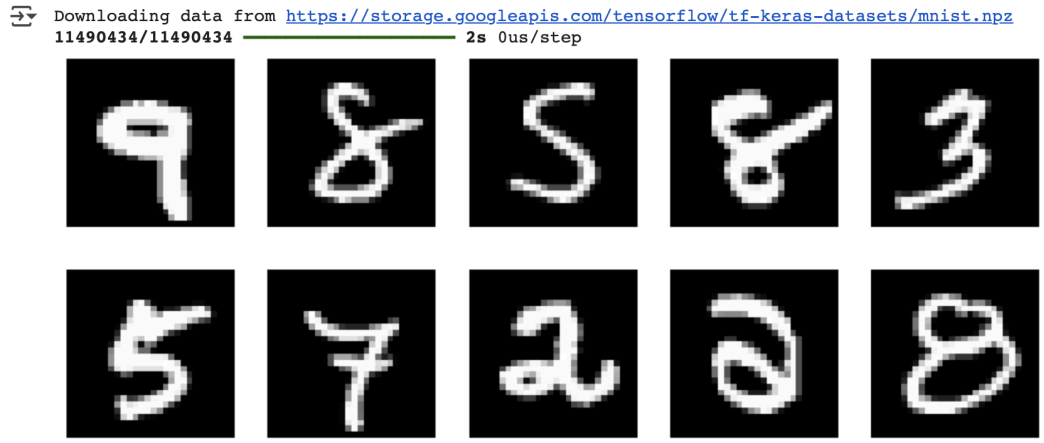



Рисунок 1 — Примеры исходных изображений из набора данных MNIST после нормализации.

2. Добавление шума к изображениям

Пусть noise_factor = 0.5 определяет интенсивность шума: при увеличении этого коэффициента изображения будут сильнее искажены. После добавления шумовых значений к пикселям полученные значения ограничиваются диапазоном [0,1], чтобы оставить корректные значения яркости. Ниже приведен код, добавляющий шум к изображениям и визуализирующий те же самые примеры цифр, но уже зашумленные:

```
# Добавление гауссовского шума к изображениям
noise_factor = 0.5
x_train_noisy = x_train + noise_factor * np.random.normal(loc=0.0, scale=1.0,
size=x_train.shape)
x_test_noisy  = x_test  + noise_factor * np.random.normal(loc=0.0, scale=1.0,
size=x_test.shape)

# Обрезка значений, выходящих за границы [0,1]
x_train_noisy = np.clip(x_train_noisy, 0.0, 1.0)
x_test_noisy  = np.clip(x_test_noisy, 0.0, 1.0)

# Визуализация зашумленных изображений (те же примеры, что и выше)
plt.figure(figsize=(10,4))
for i, index in enumerate(idx):
    plt.subplot(2, 5, i+1)
    plt.imshow(x_train_noisy[index].reshape(28, 28), cmap='gray')
```

```
    plt.axis('off')
plt.show()
```

Рисунок 2 — Изображения чисел после добавления случайного шума (noise_factor = 0.5).

В результате добавления шума исходные изображения заметно искажены. На приведенных примерах видно, что на цифрах появились точки и помехи разной интенсивности, местами затрудняющие распознавание цифры. Тем не менее, общая структура и форма каждой цифры все еще просматривается, что важно для возможности восстановления.

3.Построение модели автокодировщика

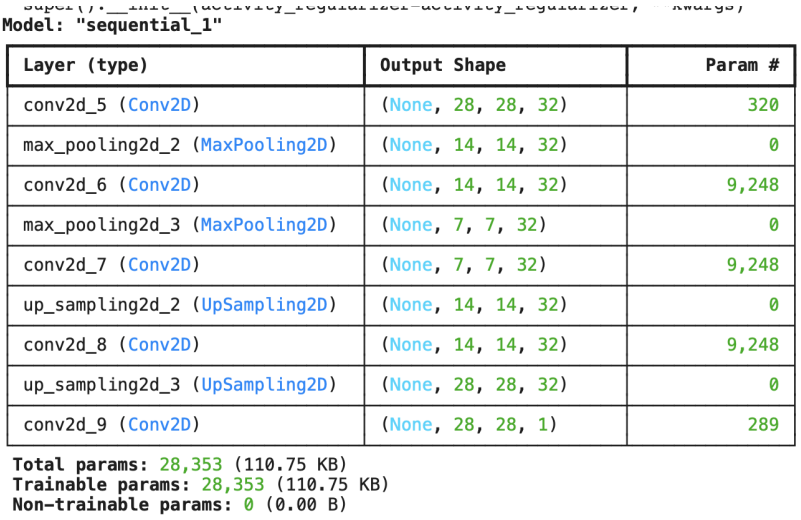
Для решения задачи шумоподавления строим автокодировщик – нейронную сеть, которая будет обучаться воспроизводить чистые изображения из зашумленных входных данных. Будем использовать сверточную архитектуру, так как сверточные слои хорошо подходят для обработки изображений, сохраняя пространственную структуру. Архитектура автокодировщика включает энкодер (сверточные и pooling-слои для понижения размерности) и декодер (слои upsampling и сверточные для восстановления исходного разрешения изображения). Ниже представлена реализация модели автокодировщика на Keras:

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, UpSampling2D

# Построение сверточного автокодировщика
model = Sequential()
# Энкодер: сжатие изображения до скрытого представления
model.add(Conv2D(32, (3,3), activation='relu', padding='same',
input_shape=(28, 28, 1)))
model.add(MaxPooling2D((2,2), padding='same'))        # => размер 14x14x32
model.add(Conv2D(32, (3,3), activation='relu', padding='same'))
model.add(MaxPooling2D((2,2), padding='same'))        # => размер 7x7x32
(скрытое представление)
# Декодер: восстановление исходного размера изображения из скрытого
представления
model.add(Conv2D(32, (3,3), activation='relu', padding='same'))
model.add(UpSampling2D((2,2)))                        # => размер 14x14x32
model.add(Conv2D(32, (3,3), activation='relu', padding='same'))
model.add(UpSampling2D((2,2)))                        # => размер 28x28x32
model.add(Conv2D(1, (3,3), activation='sigmoid', padding='same'))  # выходной
слой 28x28x1

# Компиляция модели автокодировщика
model.compile(optimizer='adam', loss=binary_crossentropy,metrics=['accuracy'])

# Вывод сводки архитектуры модели
model.summary()
```



**Model: "sequential_1"**

| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d_5 (Conv2D) | (None, 28, 28, 32) | 320 |
| max_pooling2d_2 (MaxPooling2D) | (None, 14, 14, 32) | 0 |
| conv2d_6 (Conv2D) | (None, 14, 14, 32) | 9,248 |
| max_pooling2d_3 (MaxPooling2D) | (None, 7, 7, 32) | 0 |
| conv2d_7 (Conv2D) | (None, 7, 7, 32) | 9,248 |
| up_sampling2d_2 (UpSampling2D) | (None, 14, 14, 32) | 0 |
| conv2d_8 (Conv2D) | (None, 14, 14, 32) | 9,248 |
| up_sampling2d_3 (UpSampling2D) | (None, 28, 28, 32) | 0 |
| conv2d_9 (Conv2D) | (None, 28, 28, 1) | 289 |

**Total params:** 28,353 (110.75 KB)
**Trainable params:** 28,353 (110.75 KB)
**Non-trainable params:** 0 (0.00 B)

Рисунок 3 — Сводка модели автокодировщика, показывающая слои энкодера и декодера.

Согласно сводке, построенная модель является сверточным автокодировщиком. Энкодер последовательно снижает размерность входного изображения 28×28 до скрытого представления 7×7×32 (путем двух уровней свёртки с MaxPooling). Декодер затем восстанавливает изображение первоначального размера, выполняя две операции UpSampling (обратное масштабирование) и свёртки. Выходной слой имеет размер 28×28×1 и использует сигмоиду, что соответствует диапазону пикселей [0,1]. В качестве функции потерь выбрана среднеквадратичная ошибка (mean squared error, MSE), измеряющая разницу между выходным изображением и оригиналом. Оптимизатор Adam используется для эффективного обучения сети.

5.Обучение модели на зашумленных данных

После определения архитектуры можно приступать к обучению автокодировщика. В процессе обучения модели подаются зашумленные изображения на вход и исходные "чистые" изображения в качестве целевых выходов. Таким образом, модель будет корректировать свои параметры, стремясь минимизировать среднеквадратичную ошибку реконструкции. Обучение проводим на обучающей выборке MNIST, контролируя качество на тестовой выборке (валидация) для отслеживания прогресса. Запустим обучение на 15 эпох с размером мини-выборки 128:

```
# Обучение автокодировщика на зашумленных данных
history = model.fit(x_train_noisy, x_train,
                    epochs=20,
                    batch_size=128,
                    validation_data=(x_test_noisy, x_test)
```

Ожидается, что с увеличением числа эпох ошибка будет уменьшаться, а автокодировщик – улучшать качество восстановления изображений. Чтобы наглядно проиллюстрировать динамику обучения, построим график функции потерь по эпохам:

```
# График истории обучения (функция потерь на обучении и валидации)
# Построение графиков accuracy и loss
plt.figure(figsize=(12, 5))
```

```
# Accuracy
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# Loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss (Binary Crossentropy)')
plt.legend()

plt.tight_layout()
plt.show()
```

Вывод:

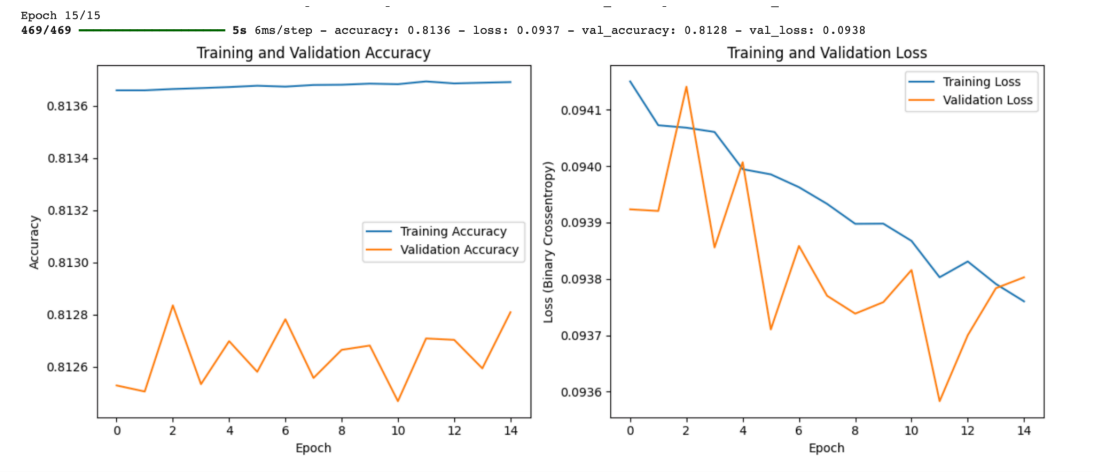



Рисунок 4 — Визуализация для обучения модели.

## 8 Визуализация результатов восстановления

Наконец, выполним визуальную проверку результатов, сравнив исходные, зашумленные и восстановленные изображения. Для этого пропустим изображения из тестового набора через обученный

автокодировщик и получим реконструированные (очищенные) версии. Отобразим несколько случайных примеров в виде колонок: в каждой колонке наверху будет исходное изображение цифры, по середине – та же цифра с добавленным шумом, а внизу – восстановленное изображение, полученное от автокодировщика. Такой формат наглядно демонстрирует эффект работы модели:

```
# Реконструирование изображений из зашумленных входов с помощью обученного автокодировщика
decoded_imgs = model.predict(x_test_noisy)

# Визуализация оригинальных, зашумленных и восстановленных изображений
n = 10  # количество примеров для отображения
plt.figure(figsize=(15, 5))
for i in range(n):
    # Исходное изображение (верхний ряд)
    ax = plt.subplot(3, n, i+1)
    plt.imshow(x_test[i].reshape(28, 28), cmap='gray')
    plt.axis('off')
    # Зашумленное изображение (средний ряд)
    ax = plt.subplot(3, n, i+1+n)
    plt.imshow(x_test_noisy[i].reshape(28, 28), cmap='gray')
    plt.axis('off')
    # Восстановленное изображение (нижний ряд)
    ax = plt.subplot(3, n, i+1+2*n)
    plt.imshow(decoded_imgs[i].reshape(28, 28), cmap='gray')
    plt.axis('off')
plt.show()
```

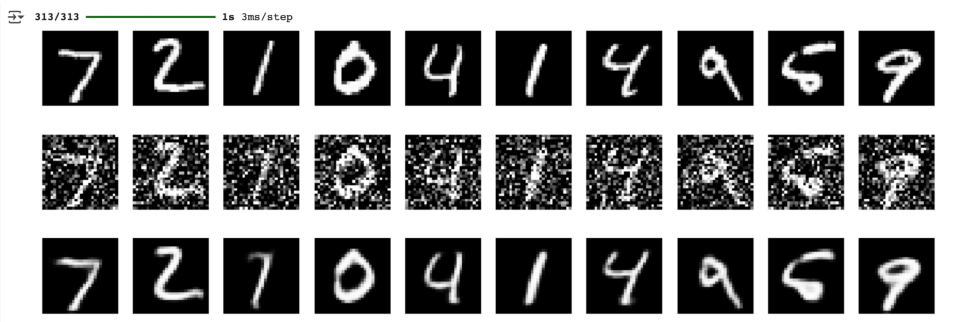



Рисунок 5 — Примеры работы автокодировщика на тестовых данных

На рисунке 5 представлены несколько примеров цифр до и после обработки автокодировщиком. Видно, что модель успешно удалила значительную часть шума: восстановленные изображения (нижний ряд) выглядят гораздо чище по сравнению с зашумленными версиями (средний ряд) и очень близки к оригиналам (верхний ряд). Даже в тех местах, где шум сильно искажал исходное изображение, автокодировщик сумел восстановить основные контуры цифры.

**Заключение**

В ходе лабораторной работы были приобретены практические навыки разработки и обучения автокодировщиков для задачи шумоподавления в изображениях. Рассмотренный сверточный автокодировщик продемонстрировал способность эффективно удалять случайный шум из изображений, восстанавливая важную информацию и улучшая качество визуального восприятия данных.

## Приложение А

[Google _Colab_Link]

```
# 1. Import libraries and load dataset
from keras.datasets import mnist
import numpy as np
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, UpSampling2D

# Загрузка набора данных MNIST
(x_train, _), (x_test, _) = mnist.load_data()

# Масштабирование пикселей в диапазон [0,1]
x_train = x_train.astype('float32') / 255.0
x_test  = x_test.astype('float32') / 255.0

# Добавление размерности канала
x_train = np.expand_dims(x_train, axis=-1)
x_test  = np.expand_dims(x_test, axis=-1)

# Визуализация исходных изображений
idx = np.random.randint(0, x_train.shape[0], size=10)
plt.figure(figsize=(10,4))
for i, index in enumerate(idx):
    plt.subplot(2, 5, i+1)
    plt.imshow(x_train[index].reshape(28, 28), cmap='gray')
    plt.axis('off')
plt.show()

# 2. Добавление гауссовского шума
noise_factor = 0.5
x_train_noisy = x_train + noise_factor * np.random.normal(loc=0.0, scale=1.0,
size=x_train.shape)
x_test_noisy  = x_test  + noise_factor * np.random.normal(loc=0.0, scale=1.0,
size=x_test.shape)

x_train_noisy = np.clip(x_train_noisy, 0.0, 1.0)
x_test_noisy  = np.clip(x_test_noisy, 0.0, 1.0)

# Визуализация зашумленных изображений
plt.figure(figsize=(10,4))
for i, index in enumerate(idx):
    plt.subplot(2, 5, i+1)
    plt.imshow(x_train_noisy[index].reshape(28, 28), cmap='gray')
    plt.axis('off')
plt.show()

# 3. Построение модели автокодировщика
model = Sequential()
```

```
# Энкодер
model.add(Conv2D(32, (3,3), activation='relu', padding='same', input_shape=(28, 28, 1)))
model.add(MaxPooling2D((2,2), padding='same'))  # 14x14x32
model.add(Conv2D(32, (3,3), activation='relu', padding='same'))
model.add(MaxPooling2D((2,2), padding='same'))  # 7x7x32

# Декодер
model.add(Conv2D(32, (3,3), activation='relu', padding='same'))
model.add(UpSampling2D((2,2)))  # 14x14x32
model.add(Conv2D(32, (3,3), activation='relu', padding='same'))
model.add(UpSampling2D((2,2)))  # 28x28x32
model.add(Conv2D(1, (3,3), activation='sigmoid', padding='same'))  # 28x28x1

# Компиляция модели
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Вывод архитектуры модели
model.summary()

# 4. Обучение модели
history = model.fit(x_train_noisy, x_train,
                    epochs=15,
                    batch_size=128,
                    validation_data=(x_test_noisy, x_test))

# Построение графиков точности и функции потерь
plt.figure(figsize=(12, 5))

# Accuracy
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# Loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss (Binary Crossentropy)')
plt.legend()

plt.tight_layout()
plt.show()

# 5. Оценка модели
```

```
test_loss = model.evaluate(x_test_noisy, x_test, verbose=0)
print("Test loss (Binary Crossentropy):", test_loss)

# 6. Реконструкция изображений
decoded_imgs = model.predict(x_test_noisy)

# Визуализация оригинальных, зашумленных и восстановленных изображений
n = 10
plt.figure(figsize=(15, 5))
for i in range(n):
    # Оригинал
    ax = plt.subplot(3, n, i+1)
    plt.imshow(x_test[i].reshape(28, 28), cmap='gray')
    plt.axis('off')

    # Зашумленное
    ax = plt.subplot(3, n, i+1+n)
    plt.imshow(x_test_noisy[i].reshape(28, 28), cmap='gray')
    plt.axis('off')

    # Восстановленное
    ax = plt.subplot(3, n, i+1+2*n)
    plt.imshow(decoded_imgs[i].reshape(28, 28), cmap='gray')
    plt.axis('off')

plt.show()
```