МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

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

Кафедра Вычислительной техники

Машинное обучение.

Лабораторная работа №3

Практика 5-6

|  |  |
| --- | --- |
| Факультет: АВТФ  Группа: АПИМ-24  Выполнил:  Разуваев В. В. | Проверил:  Гаврилов А.В. |

**Содержание**

[Цель работы 3](#_Toc184590788)

[Задание 3](#_Toc184590789)

[Ход работы 3](#_Toc184590790)

[Задача 1 3](#_Toc184590791)

[Задача 2 5](#_Toc184590792)

[Заключение 10](#_Toc184590793)

# Цель работы

Оптимизация нейронных сетей в библиотеке Keras. Процесс решения задач с применением нейронных сетей в библиотеке Keras.

# Задание

1. В зависимости от варианта сгенерировать датасет и сохранить его в формате csv. Построить модель, которая будет содержать в себе автокодировщик и регрессионную модель. Обучить модель и разбить обученную модель на 3: Модель кодирования данных (Входные данные -> Закодированные данные), модель декодирования данных (Закодированные данные -> Декодированные данные), и регрессионную модель (Входные данные -> Результат регрессии). В качестве результата представить исходный код, сгенерированные данные в формате csv, кодированные и декодированные данные в формате csv, результат регрессии в формате csv (что должно быть и что выдает модель), и сами 3 модели в формате h5.

2. Построить сверточную нейронную сеть, которая будет классифицировать черно-белые изображения с простыми геометрическими фигурами на них.

Вариант 4: Классификация изображений с крестом или с линией (может быть горизонтальной или вертикальной).

# Ход работы

## Задача 1

Освоить основные методы представление данных и получить базовые навыки работы с библиотекой NumPy.

Вариант 4

Нормальное распределение (N(μ,σ2));

- μ: среднее значение распределения;

- : дисперсия (или стандартное отклонение​).

*X* — выборка из нормального распределения *N(0, 10),*

*e* — шум из нормального распределения *N(0, 0.3)*;

Цель регрессии: *Y=sin(X)⋅X+e*.

*Листинг 1 – Генерация датасета и сохранение его в формате csv:*

import numpy as np

import pandas as pd

# 1. Генерация данных

# Генерируем выборку X из нормального распределения N(0, 10)

# и шум e из нормального распределения N(0, 0.3)

np.random.seed(42) # Фиксируем сид для воспроизводимости

X = np.random.normal(loc=0, scale=10, size=1000).reshape(-1, 1) # Входные данные

e = np.random.normal(loc=0, scale=0.3, size=1000).reshape(-1, 1) # Шум

Y = np.sin(X) \* X + e # Целевая переменная

# Сохраняем данные в CSV

data = pd.DataFrame({'X': X.flatten(), 'Y': Y.flatten()})

data.to\_csv("generated\_data.csv", index=False, sep=';')

Рисунок 1 – Выборка из нормального распределения

Рисунок 2 – Значение переменной регрессии

**Создание модели**

Модель состоит из трёх частей:

* Автокодировщик для кодирования и декодирования данных.
* Регрессионная модель, использующая кодированные данные.

*Листинг 2 – Полный код:*

import numpy as np

import pandas as pd

import tensorflow as tf

from keras.models import Model

from keras.layers import Input, Dense

from keras.optimizers import Adam

from keras.losses import MeanSquaredError

from keras.models import load\_model

# 1. Генерация данных

# Генерируем выборку X из нормального распределения N(0, 10)

# и шум e из нормального распределения N(0, 0.3)

np.random.seed(42) # Фиксируем сид для воспроизводимости

X = np.random.normal(loc=0, scale=10, size=1000).reshape(-1, 1) # Входные данные

e = np.random.normal(loc=0, scale=0.3, size=1000).reshape(-1, 1) # Шум

Y = np.sin(X) \* X + e # Целевая переменная

# Сохраняем данные в CSV

data = pd.DataFrame({'X': X.flatten(), 'Y': Y.flatten()})

data.to\_csv("generated\_data.csv", index=False, sep=';')

# 2. Определение модели

# Входной слой

input\_layer = Input(shape=(1,), name="Input")

# Кодировщик: несколько скрытых слоев для уменьшения размерности

encoded = Dense(64, activation='relu', name="Encoder\_Layer1")(input\_layer)

encoded = Dense(32, activation='relu', name="Encoder\_Layer2")(encoded)

# Закодированные данные

bottleneck = Dense(16, activation='relu', name="Bottleneck")(encoded)

# Декодировщик: восстанавливаем исходные данные

decoded = Dense(32, activation='relu', name="Decoder\_Layer1")(bottleneck)

decoded = Dense(64, activation='relu', name="Decoder\_Layer2")(decoded)

decoded\_output = Dense(1, activation='linear', name="Decoded\_Output")(decoded)

# Регрессионная часть: предсказание целевой переменной

regression = Dense(64, activation='relu', name="Regression\_Layer1")(bottleneck)

regression = Dense(32, activation='relu', name="Regression\_Layer2")(regression)

regression\_output = Dense(1, activation='linear', name="Regression\_Output")(regression)

# Полная модель: на вход подаются X, на выходе закодированные данные, декодированные данные и регрессия

full\_model = Model(inputs=input\_layer, outputs=[decoded\_output, regression\_output], name="Autoencoder\_Regression")

# 3. Компиляция модели

# Оптимизатор Adam, функция потерь MSE, метрика MSE

full\_model.compile(optimizer=Adam(learning\_rate=0.001),

loss=[MeanSquaredError(), MeanSquaredError()], # Функции потерь для декодирования и регрессии

metrics=[[MeanSquaredError()], [MeanSquaredError()]]) # Метрики для каждого выхода

# 4. Обучение модели

# Обучаем модель на данных, разбиваем на 80% обучающей и 20% валидационной выборки

history = full\_model.fit(X, [X, Y], epochs=100, batch\_size=32, validation\_split=0.2)

# 5. Сохранение модели

full\_model.save("full\_model.h5") # Сохраняем полную модель в формате h5

# 6. Загрузка модели

# Загружаем модель для проверки

loaded\_model = load\_model("full\_model.h5", custom\_objects={'MeanSquaredError': tf.keras.losses.MeanSquaredError})

# 7. Разделение на части

# Модель кодирования (Входные данные -> Закодированные данные)

encoder\_model = Model(inputs=input\_layer, outputs=bottleneck, name="Encoder")

encoder\_model.save("encoder\_model.h5") # Сохраняем модель кодирования в формате h5

# Модель декодирования (Закодированные данные -> Декодированные данные)

encoded\_input = Input(shape=(16,), name="Encoded\_Input") # Вход закодированных данных

decoder\_layer = full\_model.get\_layer("Decoder\_Layer1")(encoded\_input)

decoder\_layer = full\_model.get\_layer("Decoder\_Layer2")(decoder\_layer)

decoder\_output = full\_model.get\_layer("Decoded\_Output")(decoder\_layer)

decoder\_model = Model(inputs=encoded\_input, outputs=decoder\_output, name="Decoder")

decoder\_model.save("decoder\_model.h5") # Сохраняем модель декодирования в формате h5

# Регрессионная модель (Входные данные -> Результат регрессии)

regression\_output = full\_model.get\_layer("Regression\_Layer1")(bottleneck)

regression\_output = full\_model.get\_layer("Regression\_Layer2")(regression\_output)

regression\_output = full\_model.get\_layer("Regression\_Output")(regression\_output)

regression\_model = Model(inputs=input\_layer, outputs=regression\_output, name="Regression")

regression\_model.save("regression\_model.h5") # Сохраняем регрессионную модель в формате h5

# 8. Генерация и сохранение выходов

# Генерируем закодированные данные

encoded\_data = encoder\_model.predict(X)

pd.DataFrame(encoded\_data).to\_csv("encoded\_data.csv", index=False, sep=';')

# Генерируем декодированные данные

decoded\_data = decoder\_model.predict(encoded\_data)

pd.DataFrame(decoded\_data).to\_csv("decoded\_data.csv", index=False, sep=';')

# Генерируем результаты регрессии

regression\_results = regression\_model.predict(X)

results = pd.DataFrame({'True\_Y': Y.flatten(), 'Predicted\_Y': regression\_results.flatten()})

results.to\_csv("regression\_results.csv", index=False, sep=';')

print("Код завершён успешно. Данные и модели сохранены.")

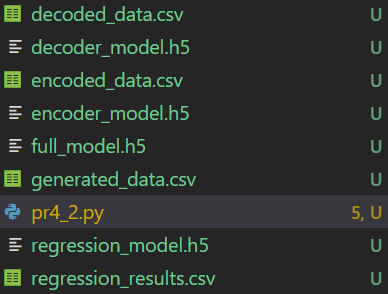


Рисунок 3 – Созданные файлы

Generated\_data.csv: Исходные данные.

encoded\_data.csv: Закодированные данные.

decoded\_data.csv: Декодированные данные.

regression\_results.csv: Результаты регрессии.

encoder.h5, decoder.h5, regression\_model.h5: Модели.

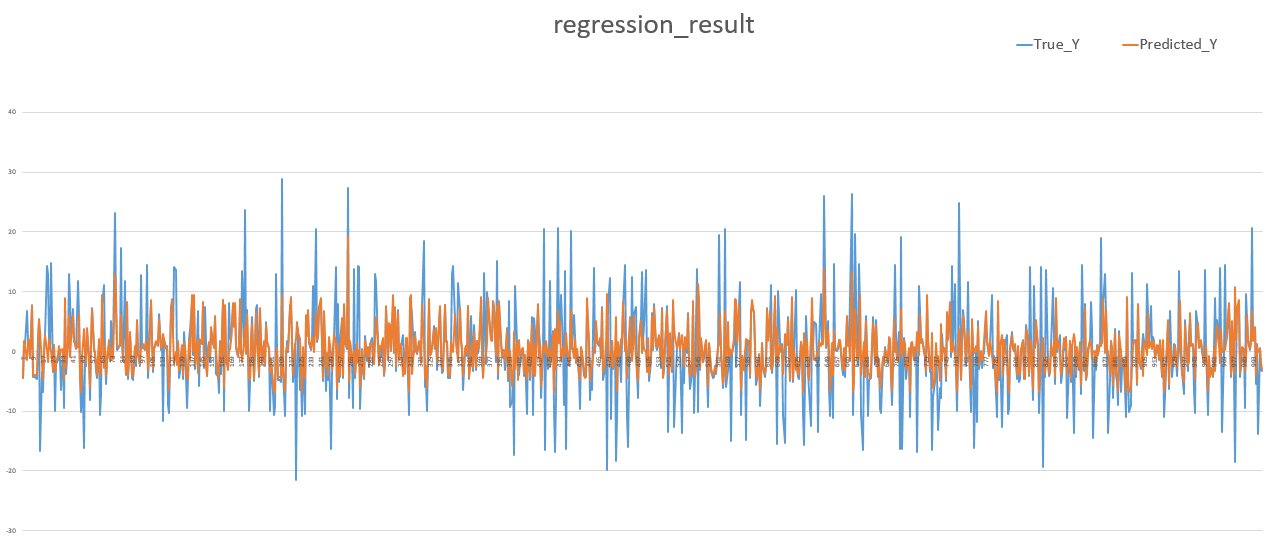
**

Рисунок 4 – Содержимое regression\_results.csv

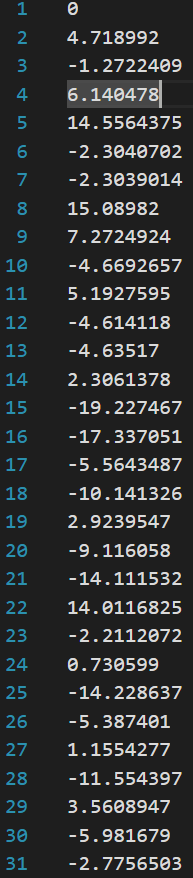
 

Рисунок 5 – Содержимое decoded\_data.csv и encoded\_data.csv

Наблюдается корреляция между предсказанными результатами регрессии и истинными результатами регрессии.

Структура кода:

Генерация данных: X∼N(0,10), e∼N(0,0.3), Y=sin (X)⋅X+e.

Построение модели:

- Кодировщик: уменьшает размерность входных данных.

- Декодировщик: восстанавливает данные из закодированных.

- Регрессор: предсказывает целевую переменную Y.

Обучение модели и разбивает её на три части:

- Кодировщик.

- Декодировщик.

- Регрессионную модель.

Сохранение результатов:

- Закодированные данные.

- Декодированные данные.

- Результаты регрессии.

- Сохранённые модели.

## Задача 2

Структура программы:

Генерация данных: Используется функция gen\_data, которая создает изображения и их метки.

Нормализация данных: Нормализация уменьшает влияние масштаба пикселей и улучшает обучение.

One-hot кодировка меток: Преобразование меток в формат, понятный модели.

Архитектура модели: Используются сверточные слои для извлечения признаков, пулинг для уменьшения размерности, и полносвязные слои для классификации.

Визуализация: Построение графика точности (Accuracy), показывающего, как изменялась точность модели на тренировочных и валидационных данных в процессе обучения, а также графика функции потерь (Loss), показывающий изменение ошибки модели (функции потерь) на тренировочных и валидационных данных.

Сохранение модели: Сохранение модели в формате .keras для дальнейшего использования.

*Листинг 2 – Полный код для задачи 2.*

import numpy as np # Для работы с массивами

from tensorflow.keras.utils import to\_categorical # Для преобразования меток в one-hot кодировку

from sklearn.model\_selection import train\_test\_split # Для разделения данных на тренировочную и тестовую выборки

from var4 import gen\_data # Импорт функции для генерации данных

# Генерация данных

data, labels = gen\_data(size=1000, img\_size=50) # Генерация 1000 изображений размером 50x50 пикселей

# Преобразование строковых меток в числовые

class\_map = {'Cross': 0, 'Line': 1} # Определяем соответствие между строковыми и числовыми метками

numeric\_labels = np.vectorize(class\_map.get)(labels.flatten()) # Преобразуем метки в числовой формат

# Нормализация данных

data = data / 255.0 # Приводим значения пикселей к диапазону [0, 1]

data = data[..., np.newaxis] # Добавляем измерение для канала (1 для черно-белых изображений)

# Проверяем распределение классов

unique, counts = np.unique(numeric\_labels, return\_counts=True)

print(f"Распределение классов: {dict(zip(unique, counts))}")

# Разделение данных на обучающую и тестовую выборки

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

data, numeric\_labels, test\_size=0.2, random\_state=42

)

# Преобразование меток в формат one-hot encoding

y\_train = to\_categorical(y\_train, num\_classes=2)

y\_test = to\_categorical(y\_test, num\_classes=2)

# Импорт необходимых классов для построения модели

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

from tensorflow.keras.optimizers import Adam

# Создание модели сверточной нейронной сети

model = Sequential([

Conv2D(32, (3, 3), activation='relu', input\_shape=(50, 50, 1)), # Сверточный слой с 32 фильтрами

MaxPooling2D((2, 2)), # Слой подвыборки (пулинга) для уменьшения размерности

Dropout(0.25), # Слой регуляризации для предотвращения переобучения

Conv2D(64, (3, 3), activation='relu'), # Еще один сверточный слой с 64 фильтрами

MaxPooling2D((2, 2)), # Еще один слой подвыборки

Dropout(0.25), # Регуляризация

Conv2D(128, (3, 3), activation='relu'), # Углубляем сеть (128 фильтров)

MaxPooling2D((2, 2)), # Подвыборка

Dropout(0.25),

Flatten(), # Преобразуем данные в одномерный массив для Dense-слоев

Dense(256, activation='relu'), # Полносвязный слой с 256 нейронами

Dropout(0.5), # Регуляризация

Dense(2, activation='softmax') # Выходной слой с 2 классами

])

# Компиляция модели

model.compile(

optimizer=Adam(learning\_rate=0.0001), # Используем оптимизатор Adam с пониженной скоростью обучения

loss='categorical\_crossentropy', # Функция потерь для задачи классификации

metrics=['accuracy'] # Метрика для отслеживания точности

)

# Обучение модели

history = model.fit(

X\_train, y\_train, # Обучающие данные

validation\_split=0.2, # Используем 20% обучающих данных для проверки

epochs=50, # Количество эпох

batch\_size=32, # Размер батча

verbose=1 # Показ прогресса обучения

)

# Оценка модели на тестовой выборке

test\_loss, test\_accuracy = model.evaluate(X\_test, y\_test, verbose=0)

print(f"Точность на тестовой выборке: {test\_accuracy \* 100:.2f}%")

# Визуализация точности

import matplotlib.pyplot as plt

plt.plot(history.history['accuracy'], label='Training Accuracy') # График точности на обучении

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy') # График точности на валидации

plt.title('Accuracy') # Заголовок

plt.xlabel('Epochs') # Ось X

plt.ylabel('Accuracy') # Ось Y

plt.legend() # Легенда

plt.show()

# Визуализация функции потерь

plt.plot(history.history['loss'], label='Training Loss') # График потерь на обучении

plt.plot(history.history['val\_loss'], label='Validation Loss') # График потерь на валидации

plt.title('Loss') # Заголовок

plt.xlabel('Epochs') # Ось X

plt.ylabel('Loss') # Ось Y

plt.legend() # Легенда

plt.show()

# Сохранение модели

model.save('cross\_or\_line\_classifier.keras') # Сохраняем модель в формате Keras

В первой попытке количество эпох для обучения равнялось 20. Точность модели составила 71%.

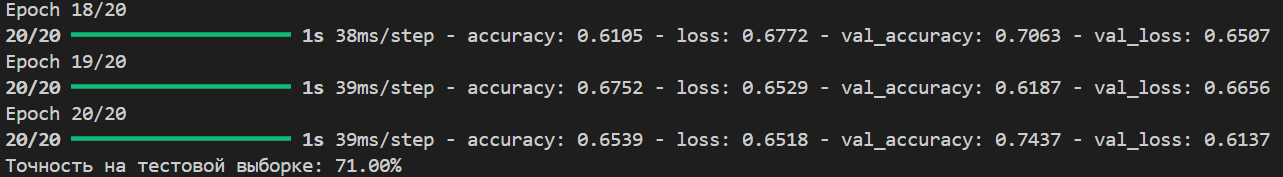


Рисунок 6 – Результат работы нейронной сети (количество эпох 20)

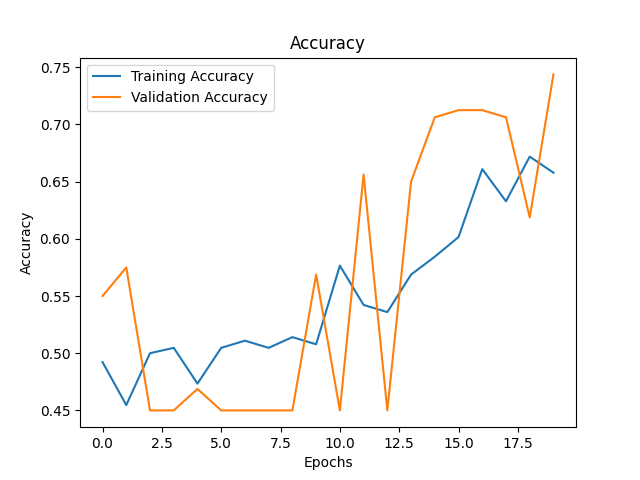


Рисунок 7 – График точности (количество эпох 20)

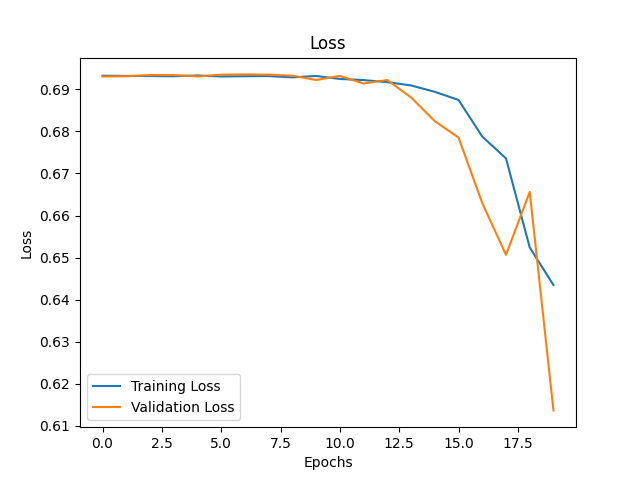


Рисунок 8 – График функции потерь (количество эпох 20)

При увеличении количества эпох до 50, точность модели достигла 99.5%.

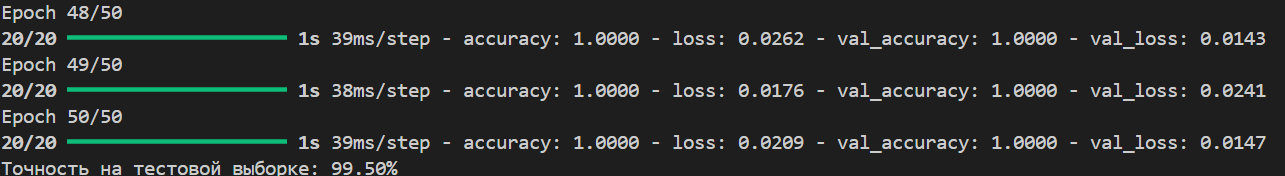


Рисунок 9 – Результат работы нейронной сети (количество эпох 50)

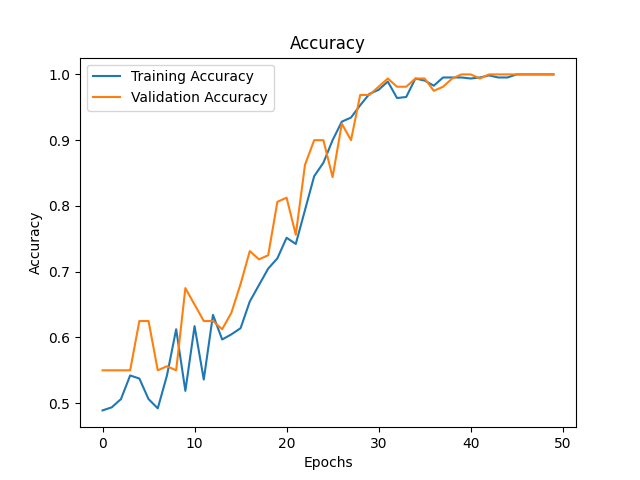


Рисунок 10 – График точности (количество эпох 50)

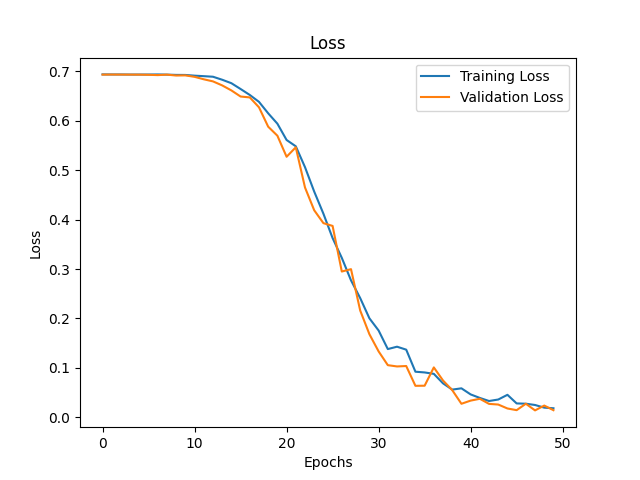


Рисунок 11 – График функции потерь (количество эпох 50)

Параллельное снижение Loss показывает, что модель корректно подстраивает свои веса и успешно обучается на тренировочных данных. Уменьшение ошибки на валидационных данных указывает, что модель применяет свои знания к новым данным. Близость значений тренировочной и валидационной ошибки говорит о том, что модель пока не начинает "запоминать" тренировочные данные (переобучения не происходит).

# Заключение

В ходе выполнения лабораторной работы была реализована модель, содержащая в себе автокодировщик и регрессионную модель, которая также осуществляет сохранение сгенерированного датасета, результатов кодировки/декодировки, результатов регрессии. Также была построена сверточная нейронная сеть, осуществляющая классификацию изображений с крестом или с линией.