МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ЛАБОРАТОРНАЯ РАБОТА №3

по дисциплине «Искусственные нейронные сети»

Тема: «Регрессионная модель изменения цен на дома в Бостоне»

Студент гр. 7383 _____ Зуев Д.В.

Преподаватель _____ Жукова Н. А.

Санкт-Петербург

2020

Цели.

Реализовать предсказание медианной цены на дома в пригороде Бостона в середине 1970-х по таким данным, как уровень преступности, ставка местного имущественного налога и т. д.

Задачи.

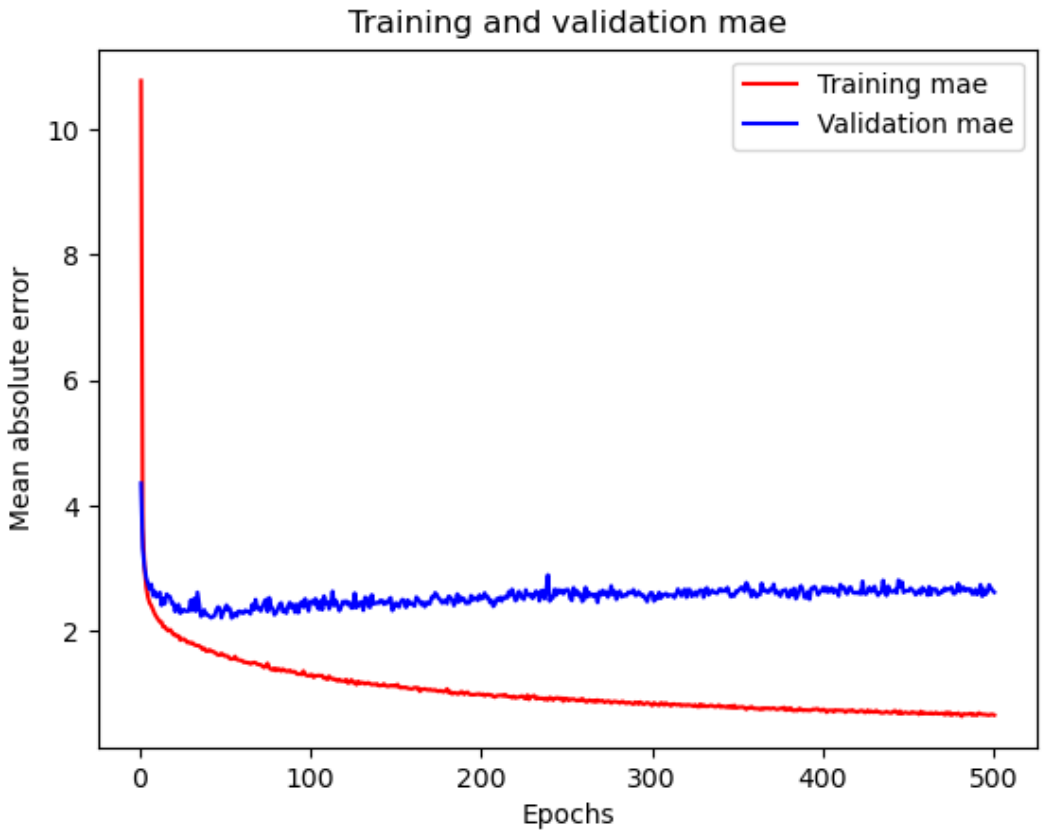
- Ознакомиться с задачей регрессии
- Изучить отличие задачи регрессии от задачи классификации
- Создать модель
- Настроить параметры обучения
- Обучить и оценить модели
- Ознакомиться с перекрестной проверкой

Ход работы.

Задача классификации определяет принадлежность объекта, описанного входными данными, к одному из заданных классов, а задача регрессии определяет значение какой-либо характеристики объекта, в зависимости от характеристик объекта, подаваемых на вход. В задаче классификации результатом будет значение из конечного множества значений, а результатом задачи регрессии может быть любое число.

1. Была создана и обучена модель искусственной нейронной сети для нахождения оптимального числа эпох (код представлен в приложении А). Количество блоков было выбрано равным 4, а количество эпох – равным 500.

Результат обучения нейронной сети представлен на графике на рис. 1.

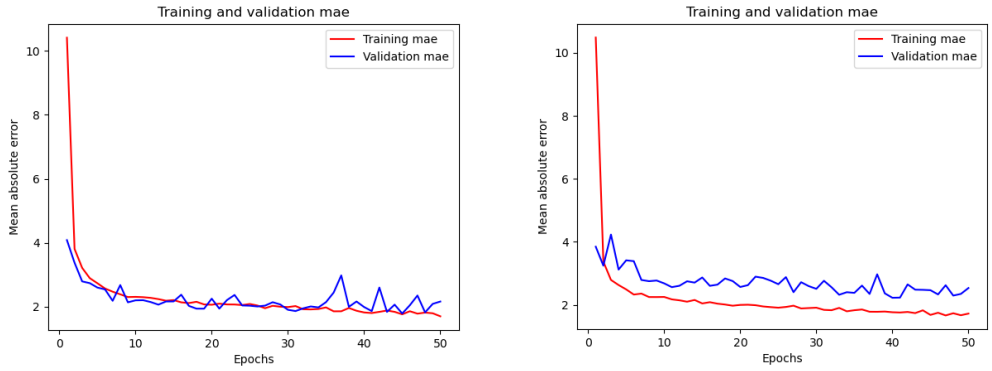


1

Рисунок 1 - Нахождение оптимального числа эпох

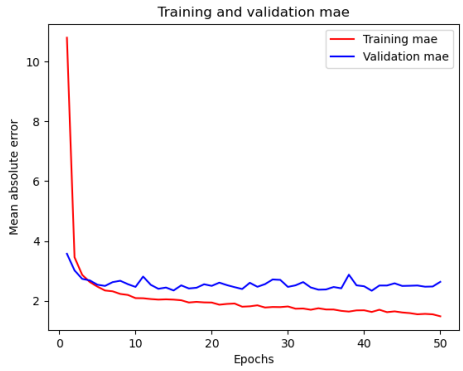
По рис. 1 видно, что ошибка на проверочных данных уменьшается до 40-50 эпох обучения, после она либо не меняется, либо становится больше при уменьшении ошибки на тестовых данных. Это говорит о переобучении модели, поэтому оптимальным значением числа эпох будет 50.

2. Было проведено тестирование обучения модели на изменяющемся числе блоков, на которые делятся данные. Значения числа блоков были взятыми 4, 6 и 8. Промежуточные результаты для 4 блоков представлены на рис. 2.

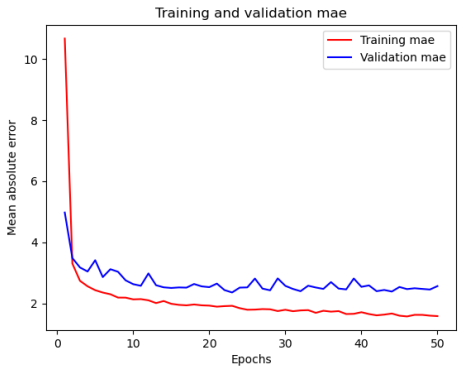


а)

б)



в)



г)

Рисунок 2 - Ошибка для блока а) 1, б) 2, в) 3, г) 4

График среднего значения ошибки для 4 блоков представлен на рис. 3.

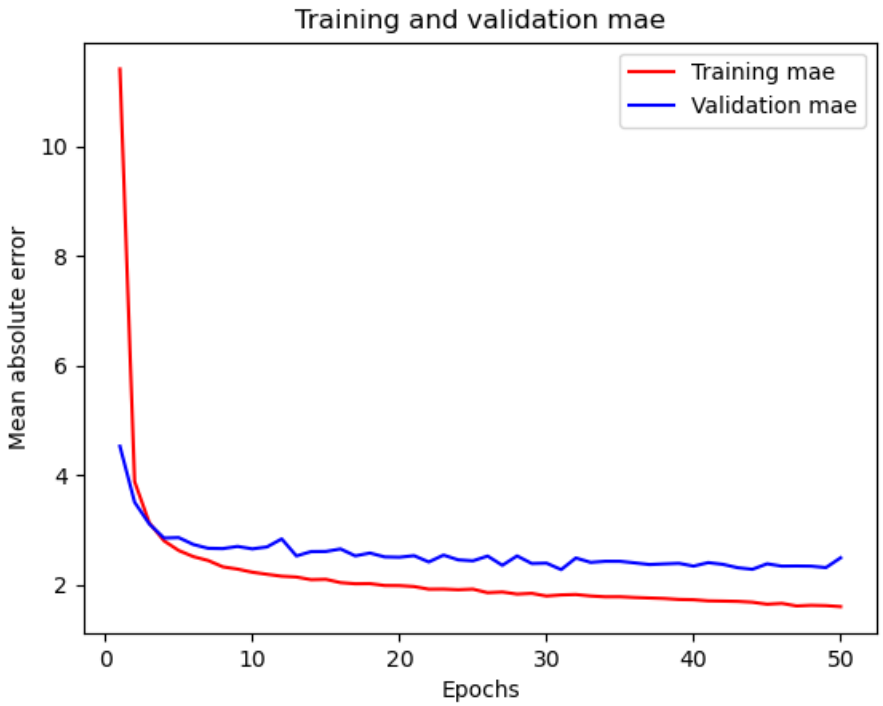


Рисунок 3 - Средняя ошибка для 4 блоков

Промежуточные результаты для 6 блоков представлены на рис. 4.

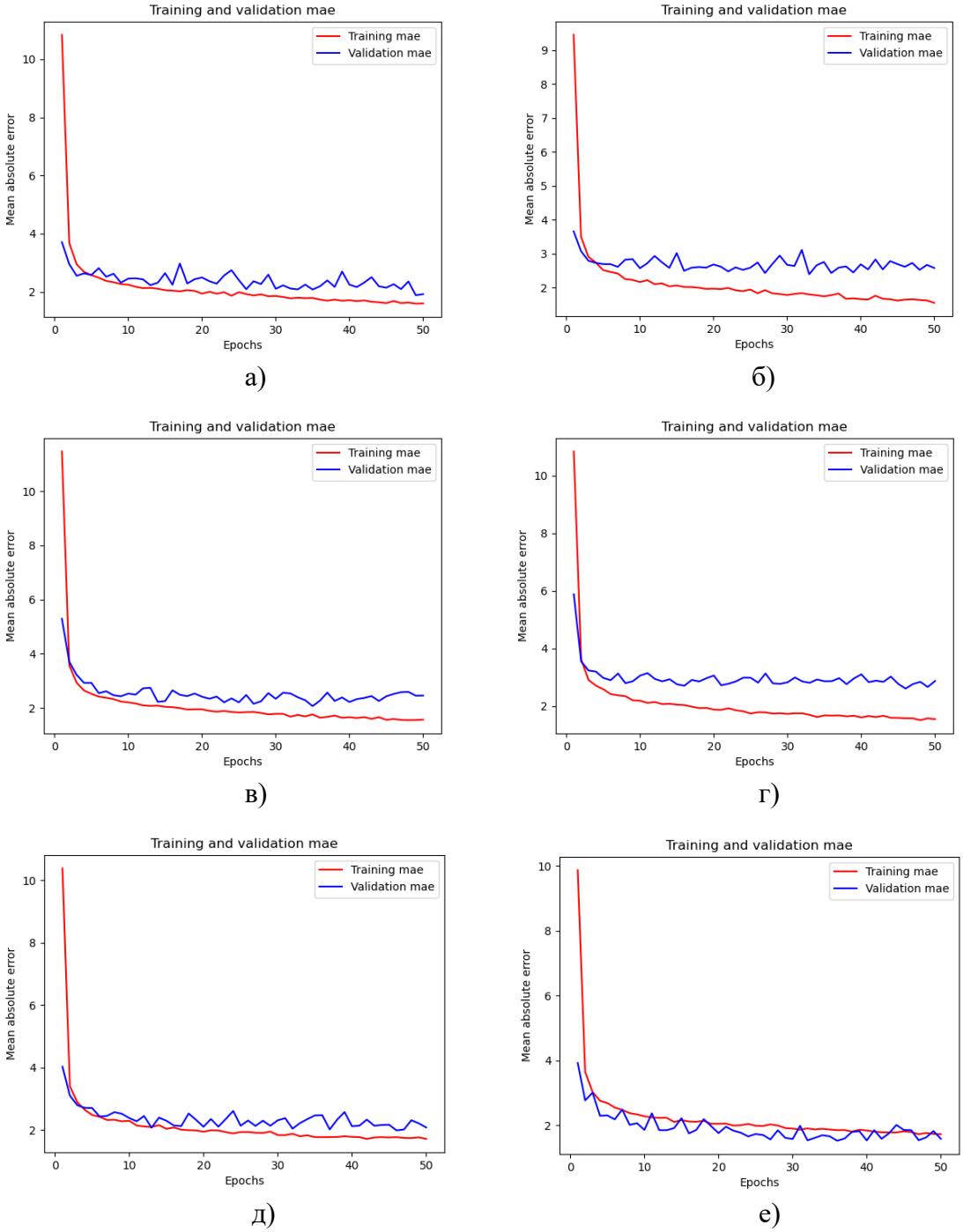


Рисунок 4 - Ошибка для блока а) 1, б) 2, в) 3, г) 4, д) 5, е) 6

График среднего значения ошибки для 4 блоков представлен на рис. 5.

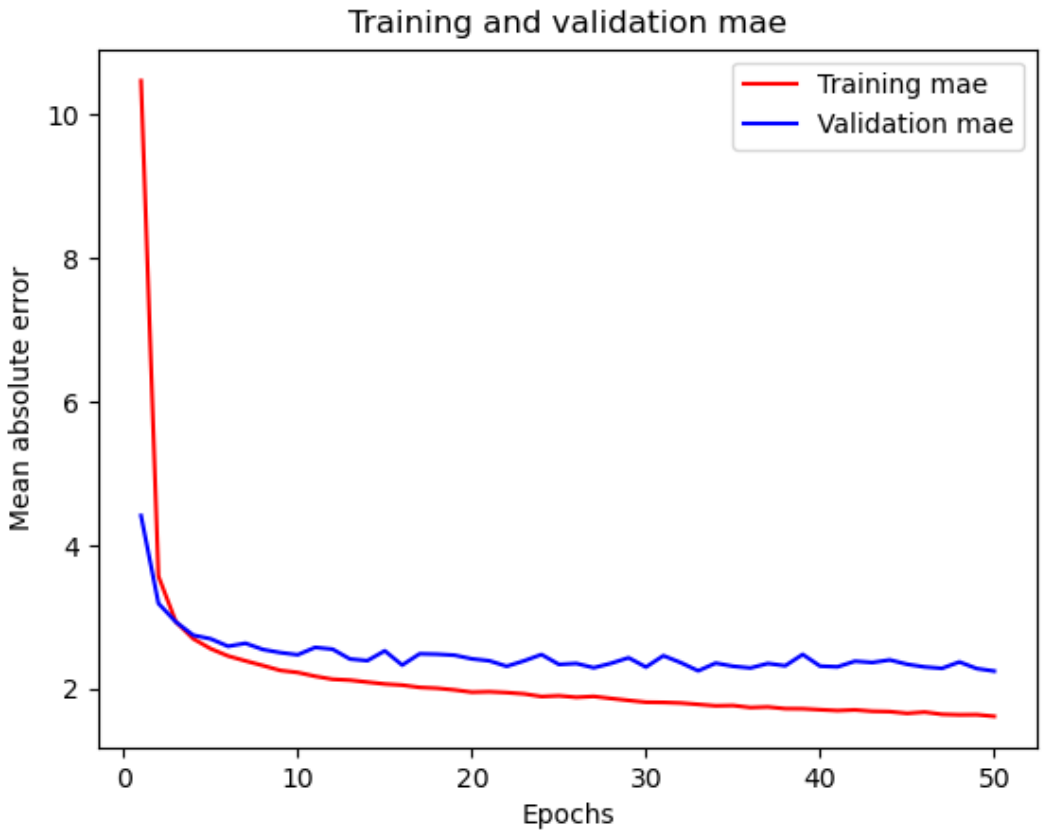
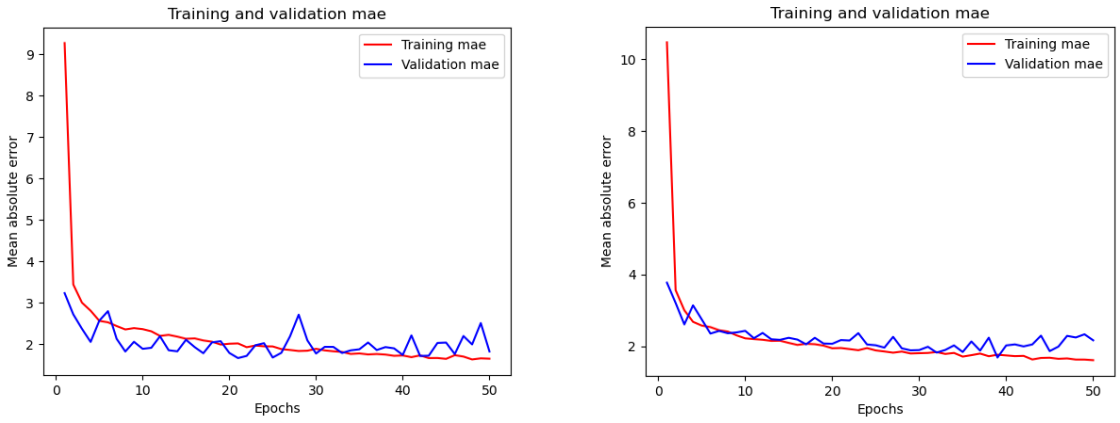
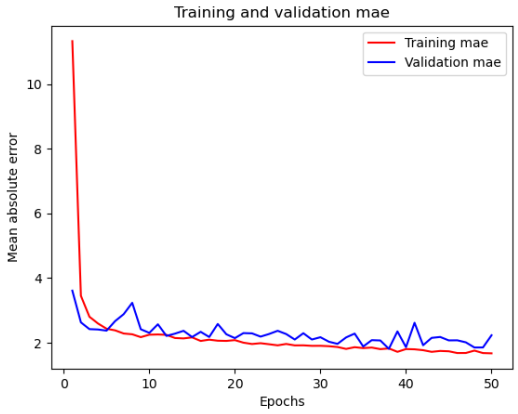


Рисунок 5 - Средняя ошибка для 6 блоков

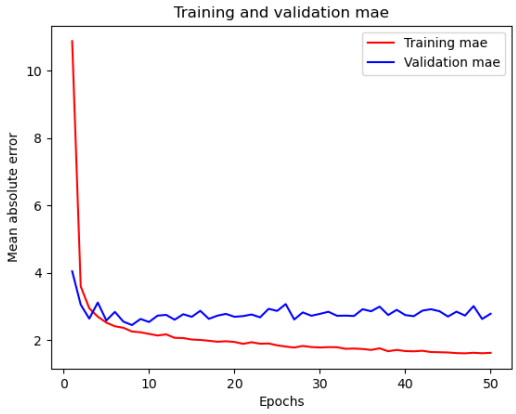
Промежуточные результаты для 8 блоков представлены на рис. 6.



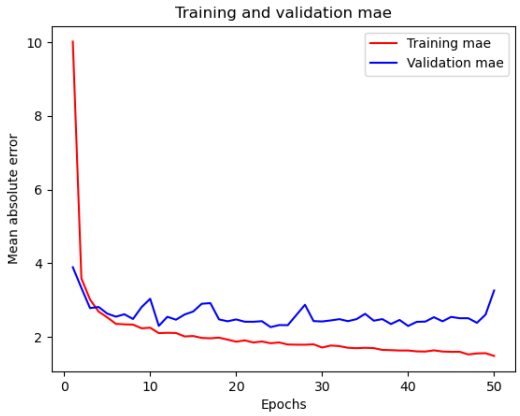
а) б)



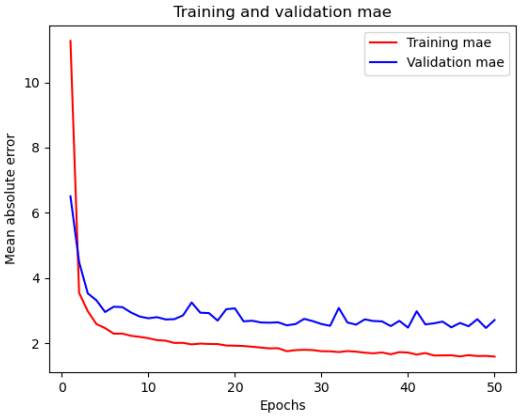
в)



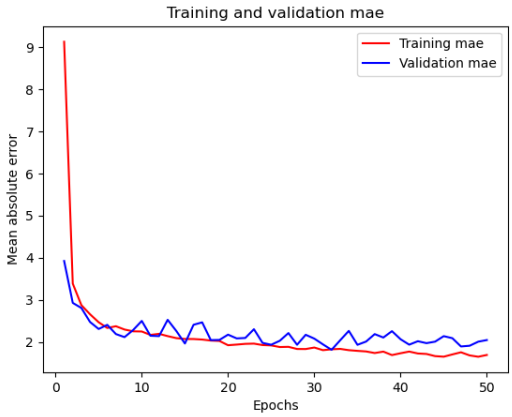
г)



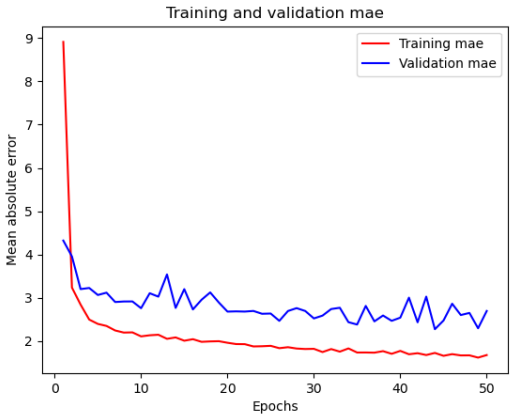
д)



е)



ж)



з)

Рисунок 6 - Ошибка для блока а) 1, б) 2, в) 3, г) 4, д) 5, е) 6, ж) 7, з) 8

График среднего значения ошибки для 4 блоков представлен на рис. 7.

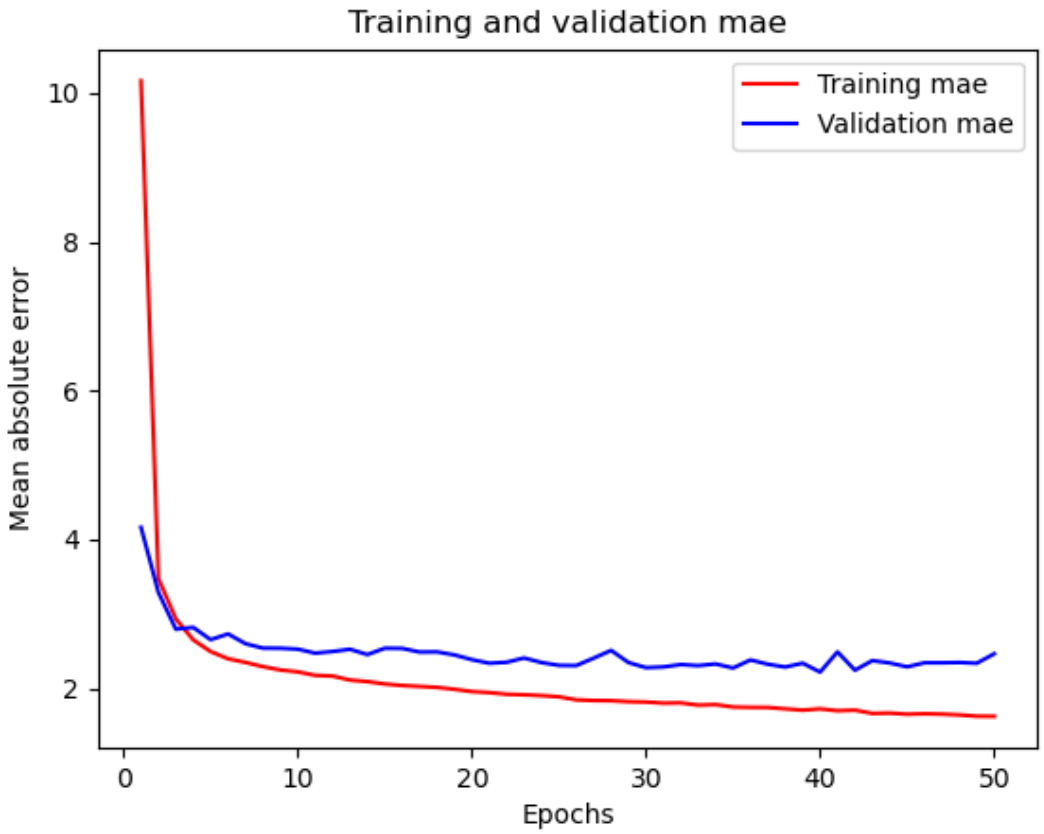


Рисунок 7 - Средняя ошибка для 8 блоков

По вышеприведенным графикам видно, что наименьшая ошибка наблюдается в модели, использующей 6 блоков.

Вывод.

В ходе выполнения данной лабораторной работы было изучено влияние количества эпох и количества блоков в перекрестной проверке на результат обучения модели искусственной нейронной сети, решающей задачу регрессии.

ПРИЛОЖЕНИЕ А

```
import numpy as np
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.datasets import boston_housing
import matplotlib.pyplot as plt

(train_data, train_targets), (test_data, test_targets) =
boston_housing.load_data()

print(train_data.shape)
print(test_data.shape)

print(test_targets)

mean = train_data.mean(axis=0)
train_data -= mean
std = train_data.std(axis=0)
train_data /= std

test_data -= mean
test_data /= std


def draw_test(arg, label, loss, val_loss, acc, val_acc):
    # if loss != 0:
    plt.plot(arg, loss, 'r', label='Training loss')
    plt.plot(arg, val_loss, 'b', label='Validation loss')
    plt.title('Training and validation loss')
    plt.xlabel(label)
    plt.ylabel('Loss')
    plt.legend()
    plt.show()
    plt.clf()
    # if acc != 0:
    plt.plot(arg, acc, 'r', label='Training mae')
    plt.plot(arg, val_acc, 'b', label='Validation mae')
    plt.title('Training and validation mae')
    plt.xlabel(label)
    plt.ylabel('Mean absolute error')
    plt.legend()
    plt.show()


def test_epochs():
    num_epochs = 500
    k = 4
    num_val_samples = len(train_data) // k
    loss_array = []
    val_loss_array = []
    mae_array = []
```

```
    val_mae_array = []
    for i in range(k):
        val_data = train_data[i * num_val_samples: (i + 1)
* num_val_samples]
        val_targets = train_targets[i * num_val_samples:
(i + 1) * num_val_samples]
        partial_train_data = np.concatenate([train_data[:i
* num_val_samples], train_data[(i + 1) *
num_val_samples:]],
                                            axis=0)
        partial_train_targets = np.concatenate(
            [train_targets[:i * num_val_samples],
train_targets[(i + 1) * num_val_samples:]], axis=0)
        model = build_model()
        H = model.fit(partial_train_data,
partial_train_targets, epochs=num_epochs, batch_size=1,
                      validation_data=(val_data,
val_targets), verbose=0)
        mae_array.append(H.history['mae'])
        val_mae_array.append(H.history['val_mae'])
        loss_array.append(H.history['loss'])
        val_loss_array.append(H.history['val_loss'])
    draw_test(range(1, num_epochs + 1), 'Epochs',
np.mean(loss_array, axis=1), np.mean(val_loss_array,
axis=1),
              np.mean(mae_array, axis=0),
np.mean(val_mae_array, axis=0))
    print(np.mean(val_mae_array))


def test_k():
    num_epochs = 50
    for k in range(6, 9, 2):
        build_k_block_model(k, num_epochs)


def build_k_block_model(k, num_epochs):
    num_val_samples = len(train_data) // k
    loss_array = []
    val_loss_array = []
    mae_array = []
    val_mae_array = []
    for i in range(k):
        val_data = train_data[i * num_val_samples: (i + 1)
* num_val_samples]
        val_targets = train_targets[i * num_val_samples:
(i + 1) * num_val_samples]
        partial_train_data = np.concatenate([train_data[:i
* num_val_samples], train_data[(i + 1) *
num_val_samples:]],
                                            axis=0)
        partial_train_targets = np.concatenate(
```

```
            [train_targets[:i * num_val_samples],
train_targets[(i + 1) * num_val_samples:]], axis=0)
        model = build_model()
        H = model.fit(partial_train_data,
partial_train_targets, epochs=num_epochs, batch_size=1,
                      validation_data=(val_data,
val_targets), verbose=0)
        mae_array.append(H.history['mae'])
        val_mae_array.append(H.history['val_mae'])
        loss_array.append(H.history['loss'])
        val_loss_array.append(H.history['val_loss'])
        draw_test(range(1, num_epochs + 1), 'Epochs',
H.history['loss'], H.history['val_loss'],
H.history['mae'],
                  H.history['val_mae'])
    draw_test(range(1, num_epochs + 1), 'Epochs',
np.mean(loss_array, axis=0), np.mean(val_loss_array,
axis=0),
              np.mean(mae_array, axis=0),
np.mean(val_mae_array, axis=0))
    print(np.mean(val_mae_array))


def build_model():
    model = Sequential()
    model.add(Dense(64, activation='relu',
input_shape=(train_data.shape[1],)))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(1))
    model.compile(optimizer='rmsprop', loss='mse',
metrics=['mae'])
    return model


test_epochs()
test_k()
```