**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 _____ Лосев М.Л.

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

**Выполнение работы.**

1. В задаче классификации требуется определить принадлежность объекта к одному из классов по набору признаков. В задаче регрессии требуется оценить значение некоторой неизвестной величины, которая некоторым образом зависит от одной или нескольких известных величин. Таким образом, эти задачи различаются по смыслу искомой величины: для классификации это наиболее вероятная категория (ее номер, целое число), а для регресии это величина, которая в общем случае может быть любым числом.
2. Чтобы изучить влияние количества эпох на результат обучения модели, была создана и обучена с применением перекрестной проверки модель (код представлен в приложении А). При количестве блоков 4 по графику (представлен на рис.1) видно, что уменьшение MAE для проверочных данных примерно после 20 эпохи становится очень медленным, а после 60 эпохи прекращается.

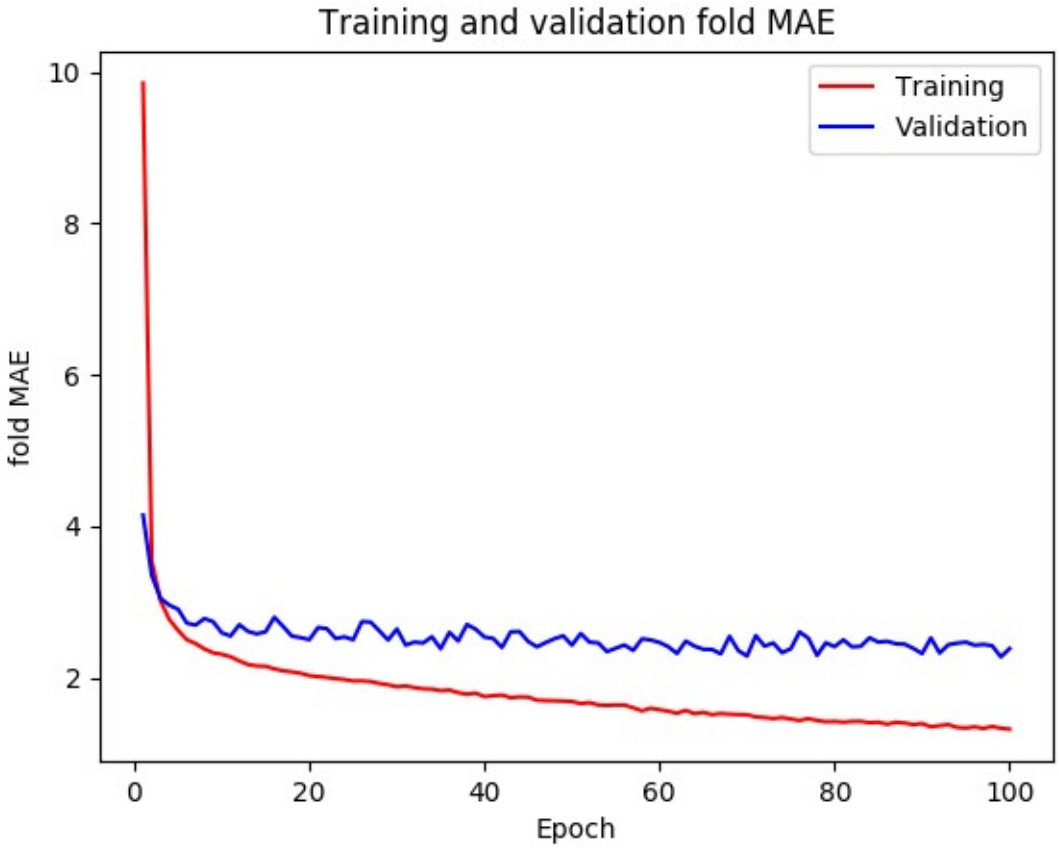


*Рисунок 1 – график среднего значения MAE по 4 блокам*

3. Переобучение начинается примерно после 60 эпохи (см. рис.1), потомоу что MAE прекращает уменьшаться.
4. Целесообразно брать количества блоков не очень маленькими, чтобы доля проверочных данных была не слишком велика (имея n блоков, получаем долю проверочных данных 1/n). Сравним результаты при K = 4, 5, 6. Графики представлены на рисунках со 2 по 17. Графики усредненной ошибки по всем блокам для этих значений K почти одинаковы, но для K=4 ошибка, кажется, убывает немного быстрее, поэтому 4 блока оптимальнее, чем 5 или 6.
5. Графики для моделей (при разных блоках) и усредненные графики представлены ниже.

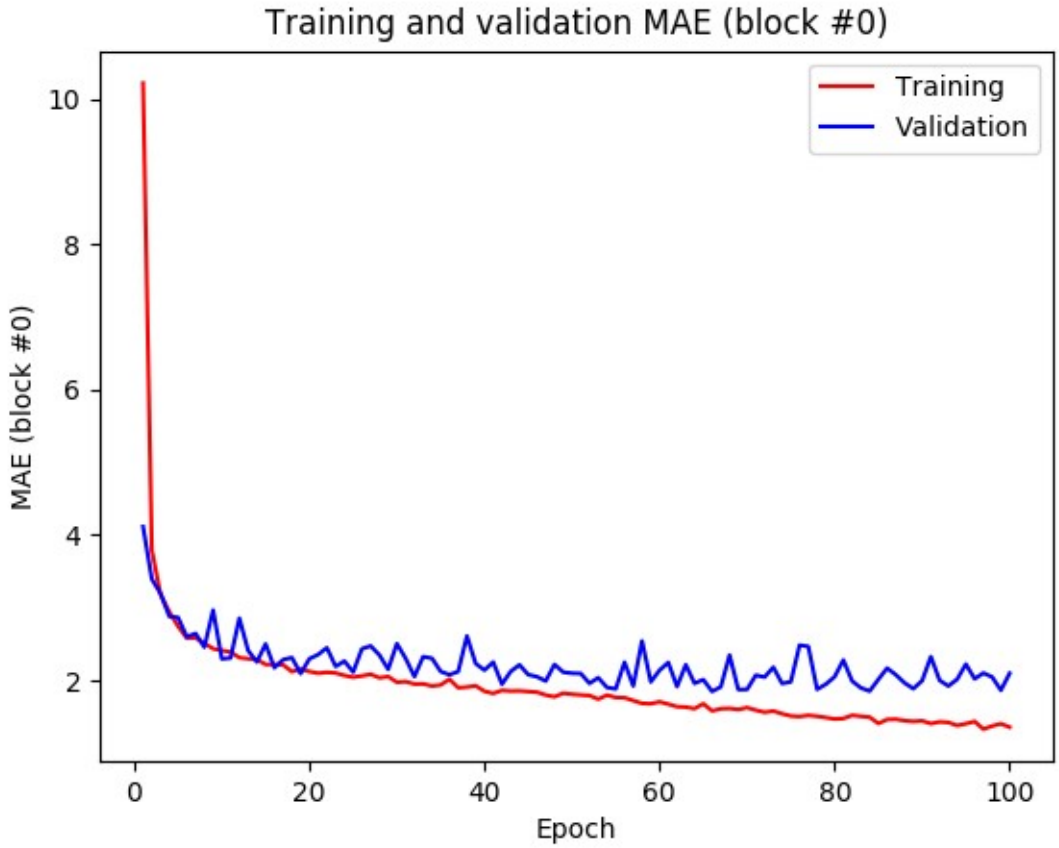


*Рисунок 2 – график значения MAE блока 0 из 4*

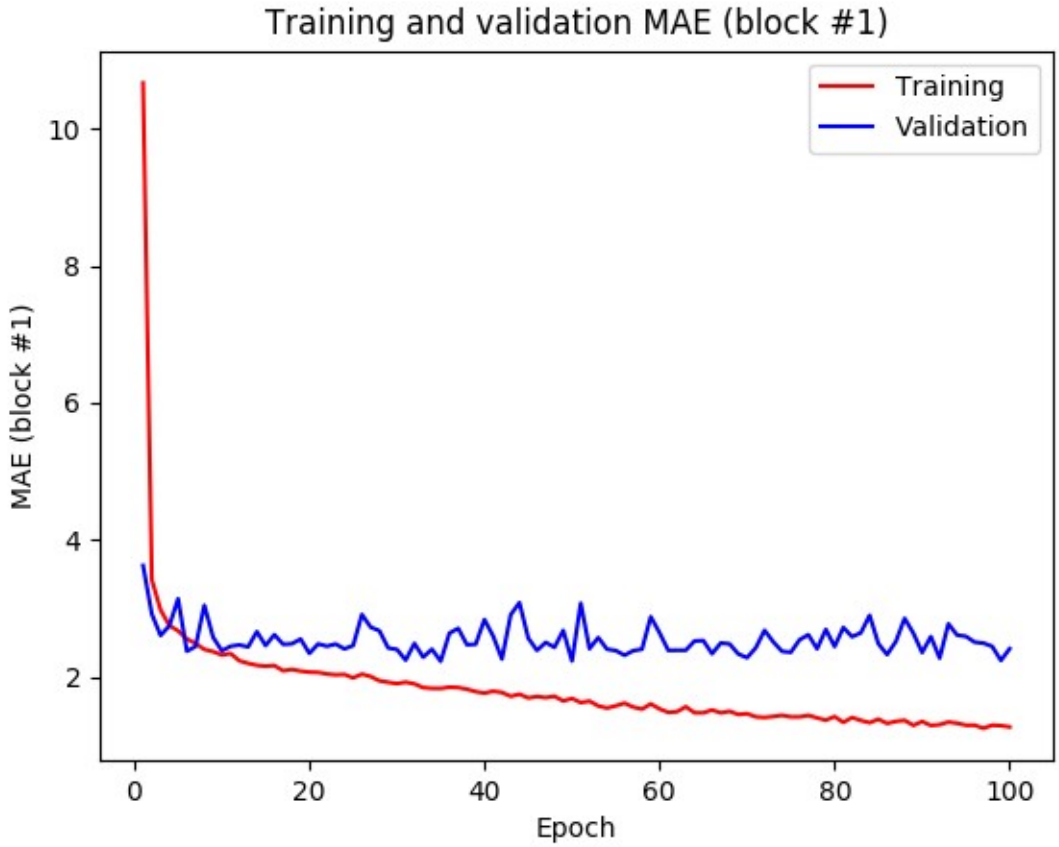


*Рисунок 3 – график значения MAE блока 1 из 4*

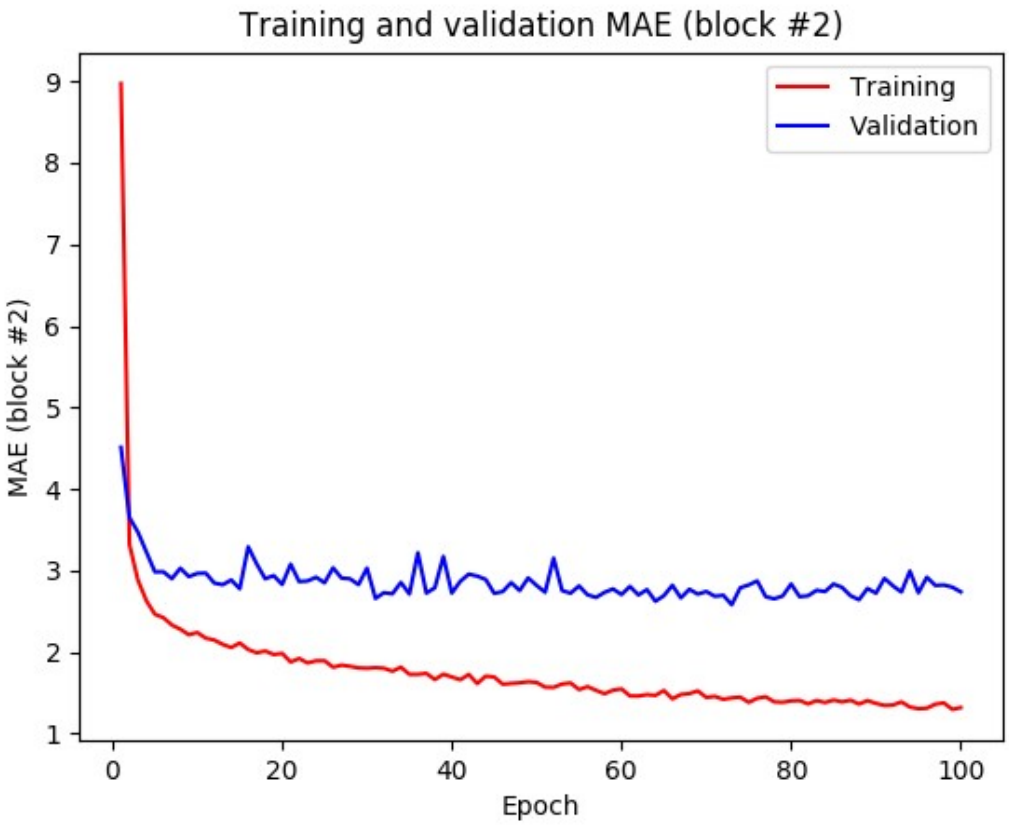


*Рисунок 4 – график значения MAE блока 2 из 4*

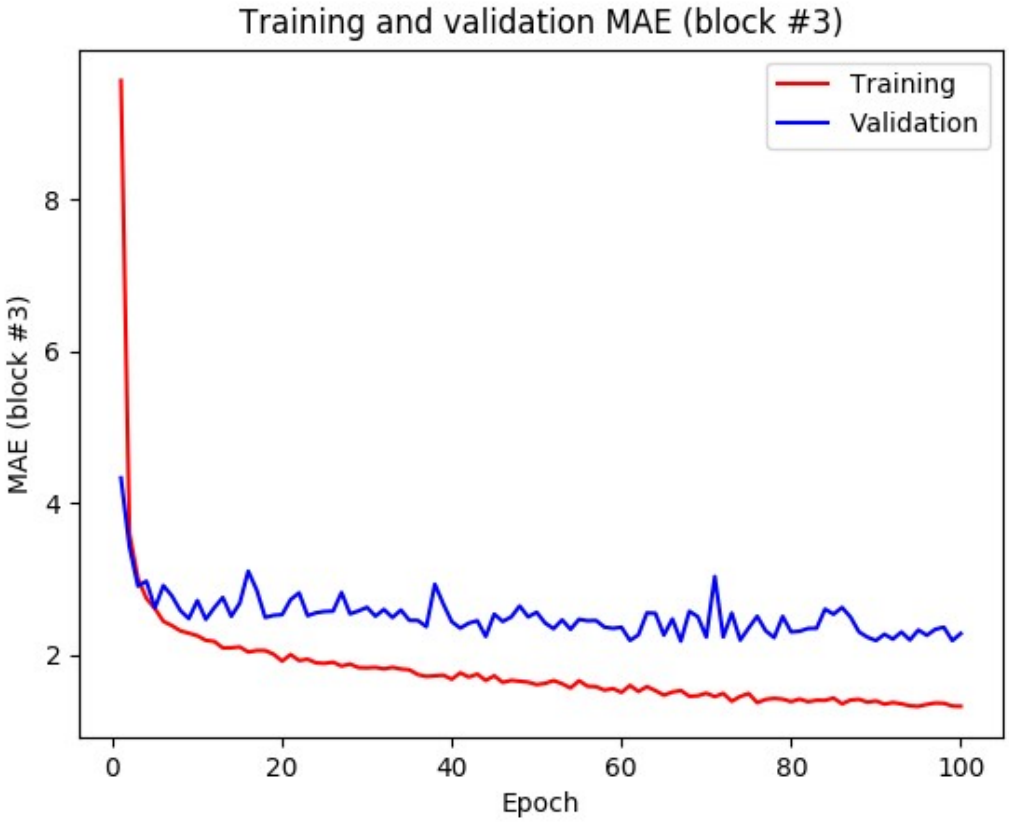


*Рисунок 5 – график значения MAE блока 3 из 4*

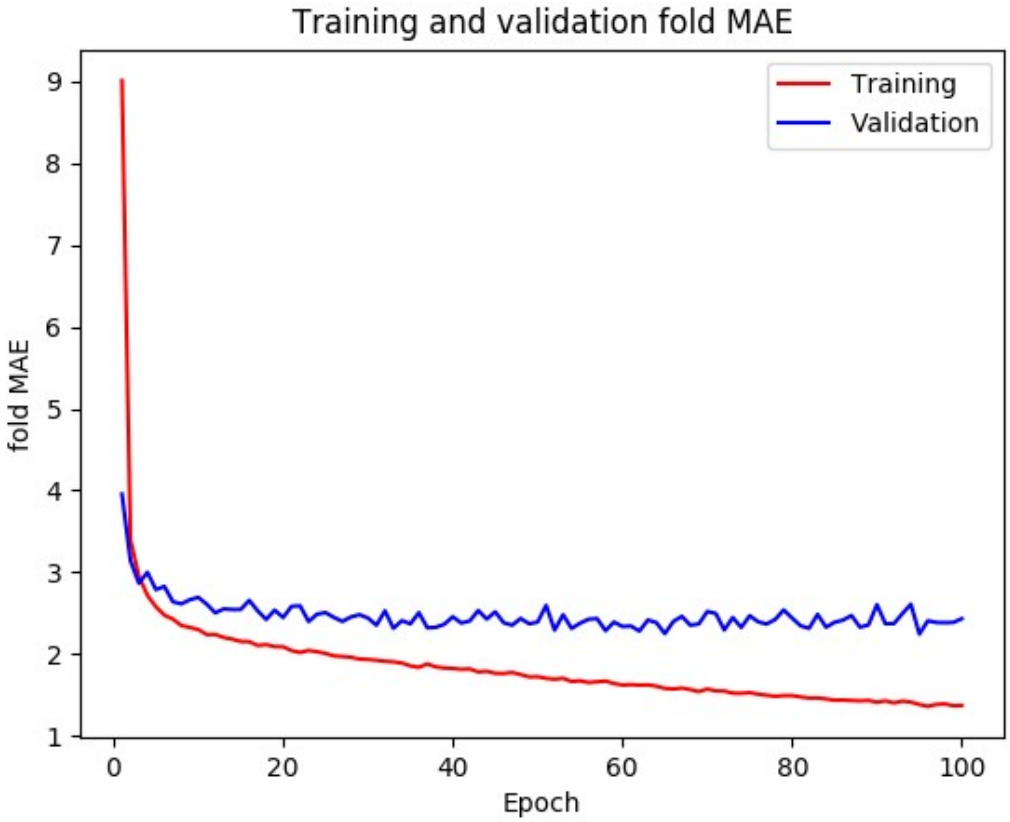


*Рисунок 6 – график среднего значения MAE по 5 блокам*

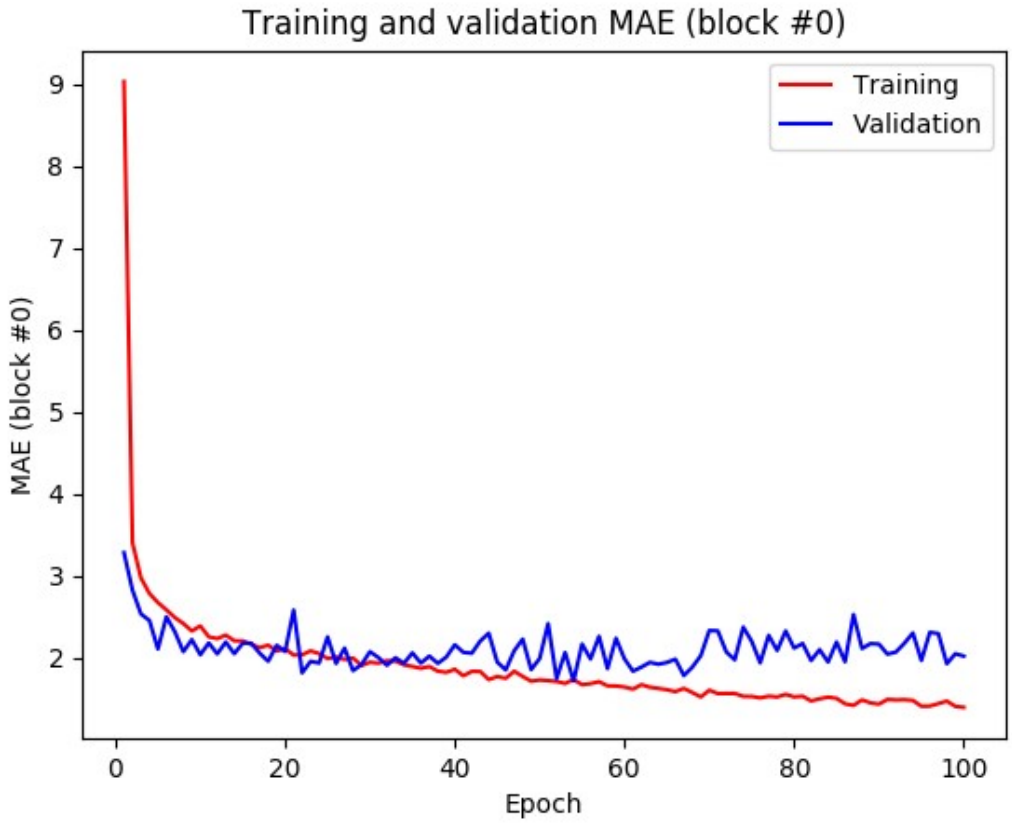


*Рисунок 7 – график значения MAE блока 0 из 5*

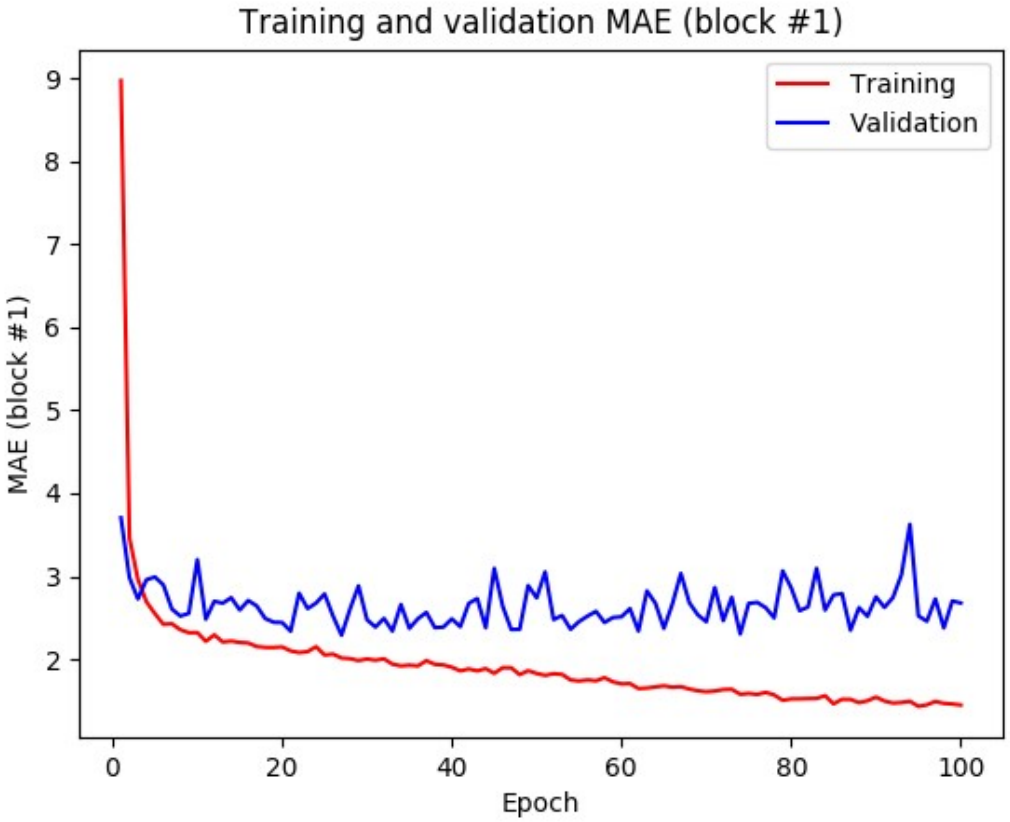


*Рисунок 8 – график значения MAE блока 1 из 5*

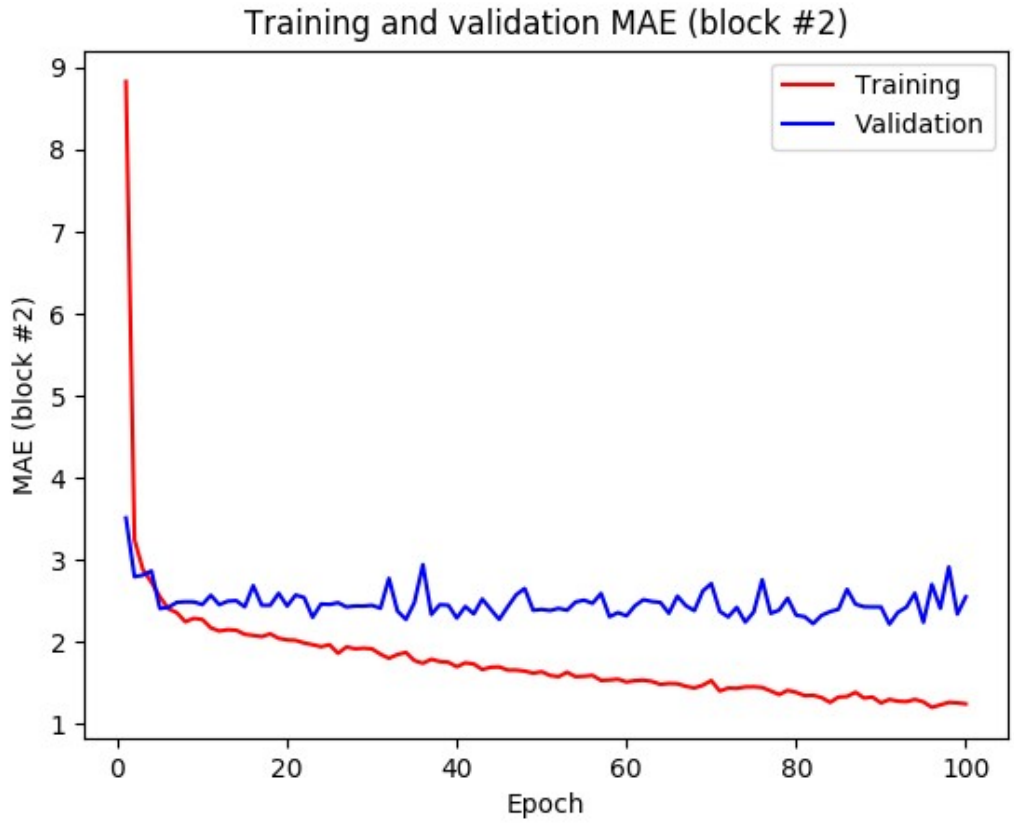


*Рисунок 9 – график значения MAE блока 2 из 5*

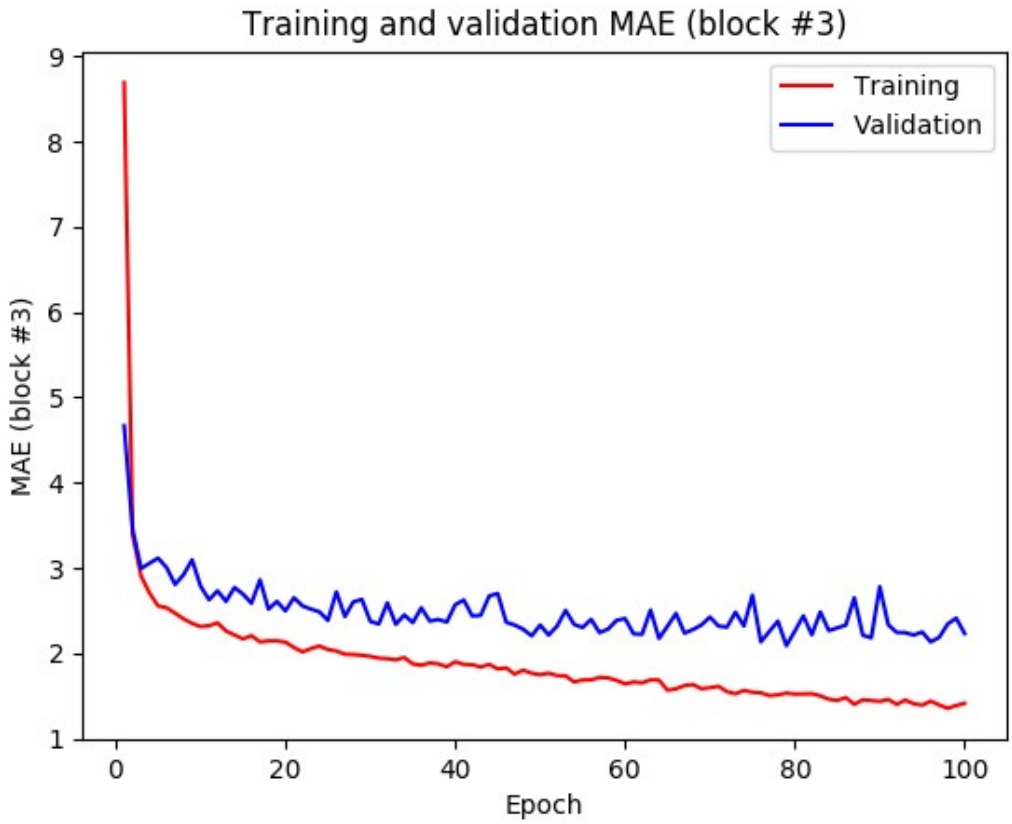


*Рисунок 10 – график значения MAE блока 3 из 5*

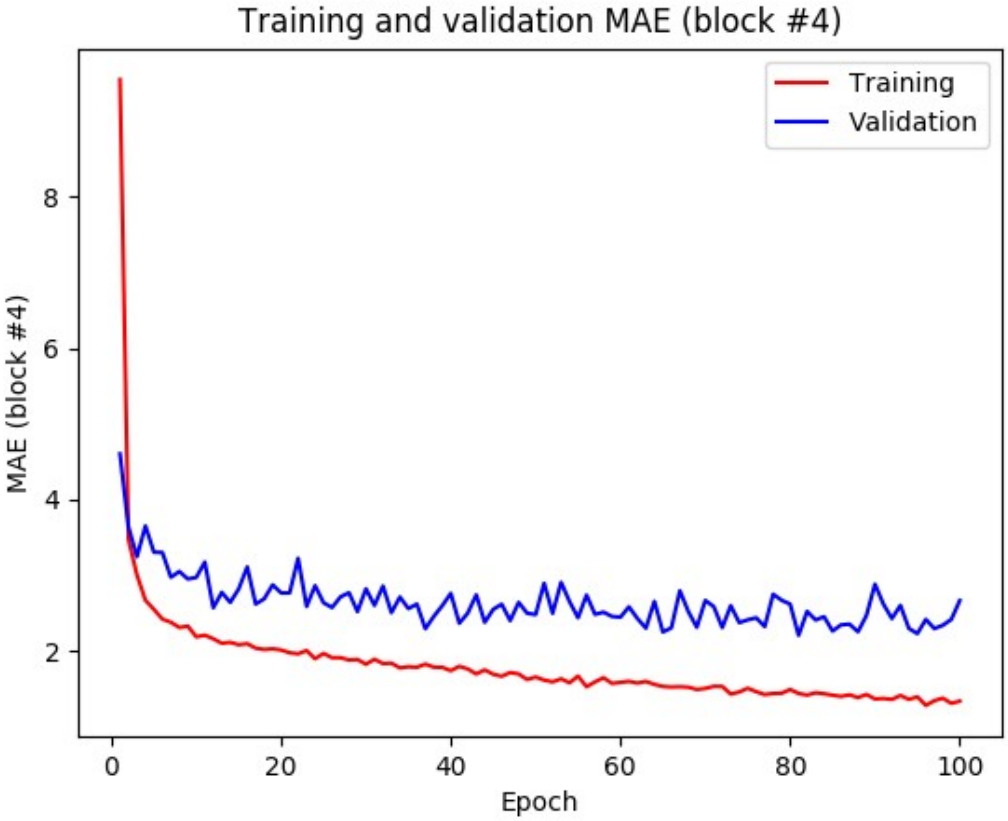


*Рисунок 11 – график значения MAE блока 4 из 5*

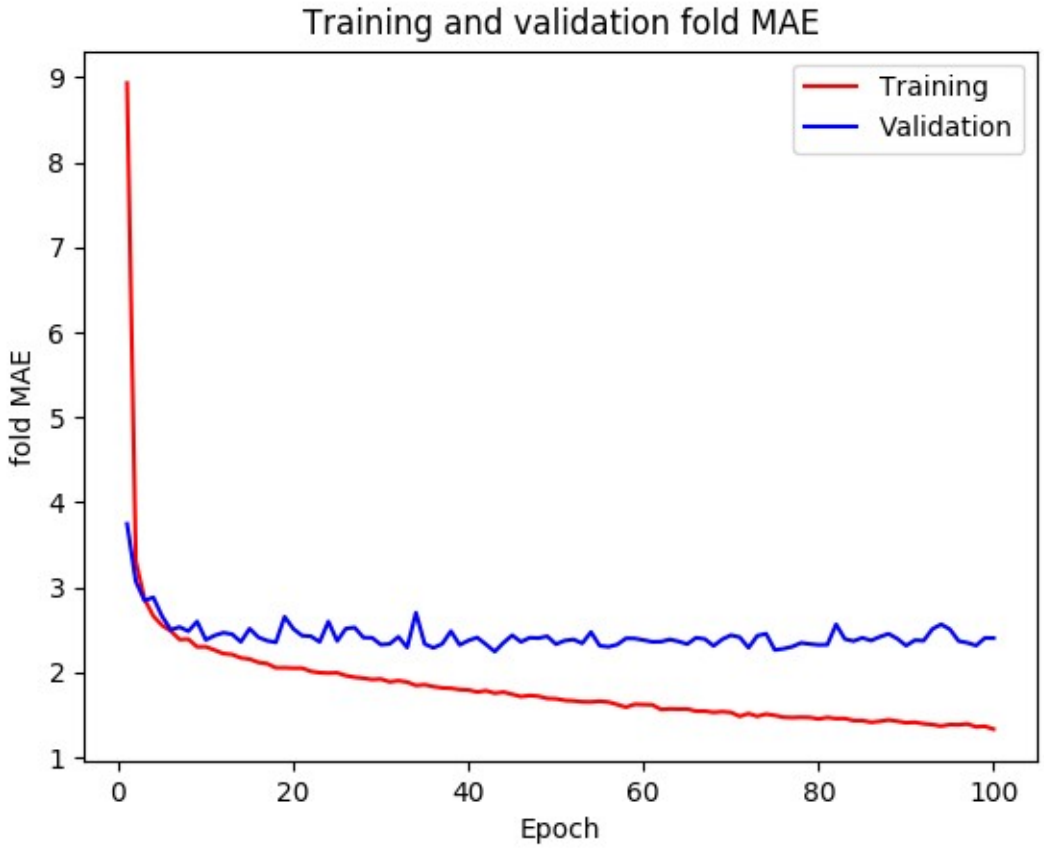


*Рисунок 12 – график среднего значения MAE по 4 блокам*

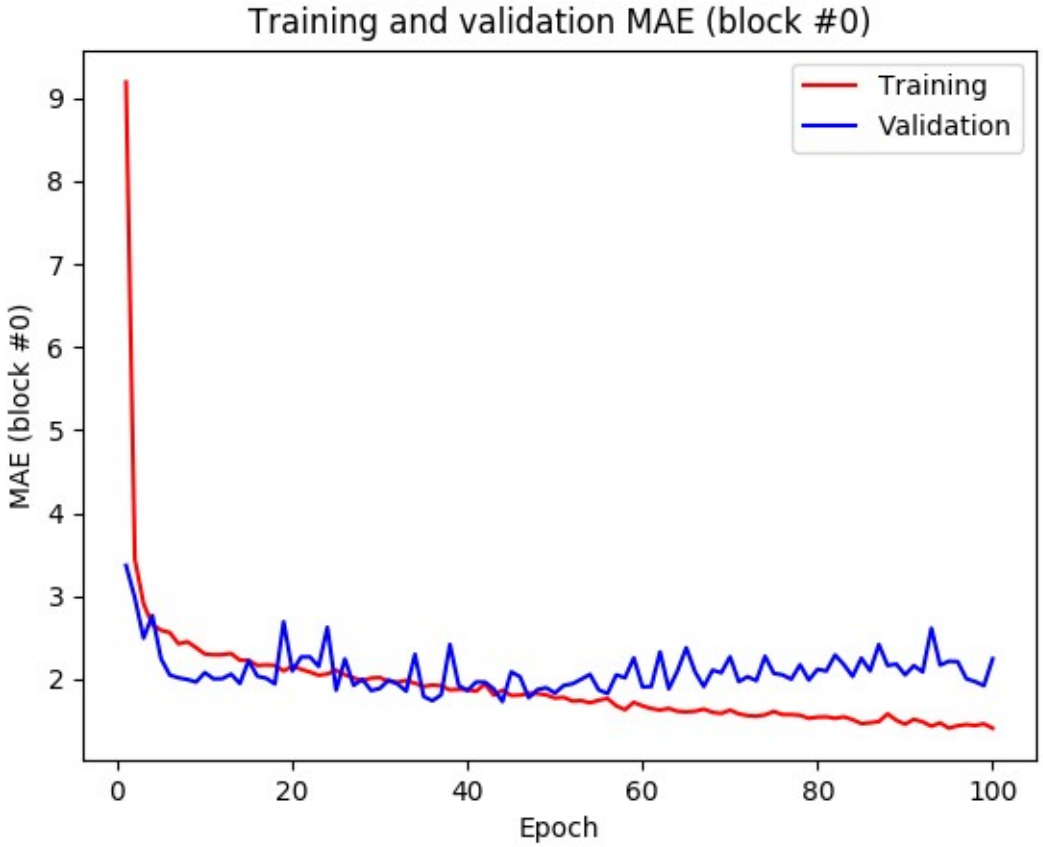


*Рисунок 13 – график значения MAE блока 0 из 6*

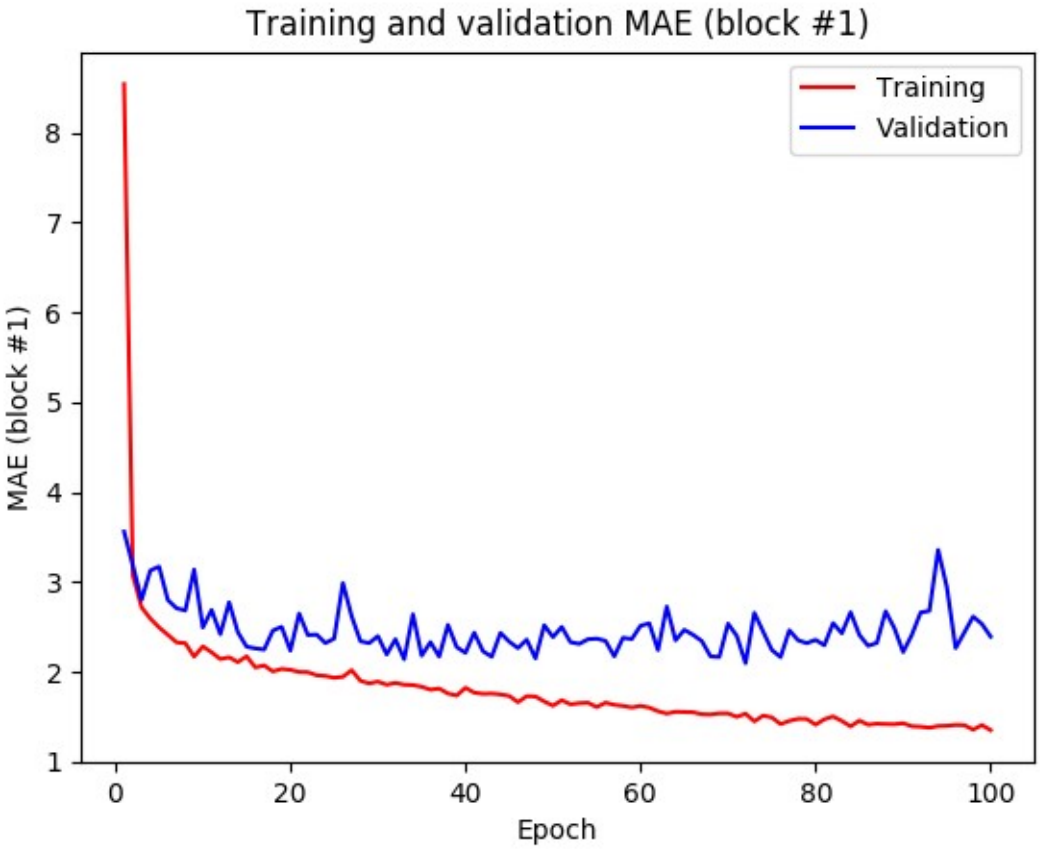


*Рисунок 14 – график значения MAE блока 1 из 6*

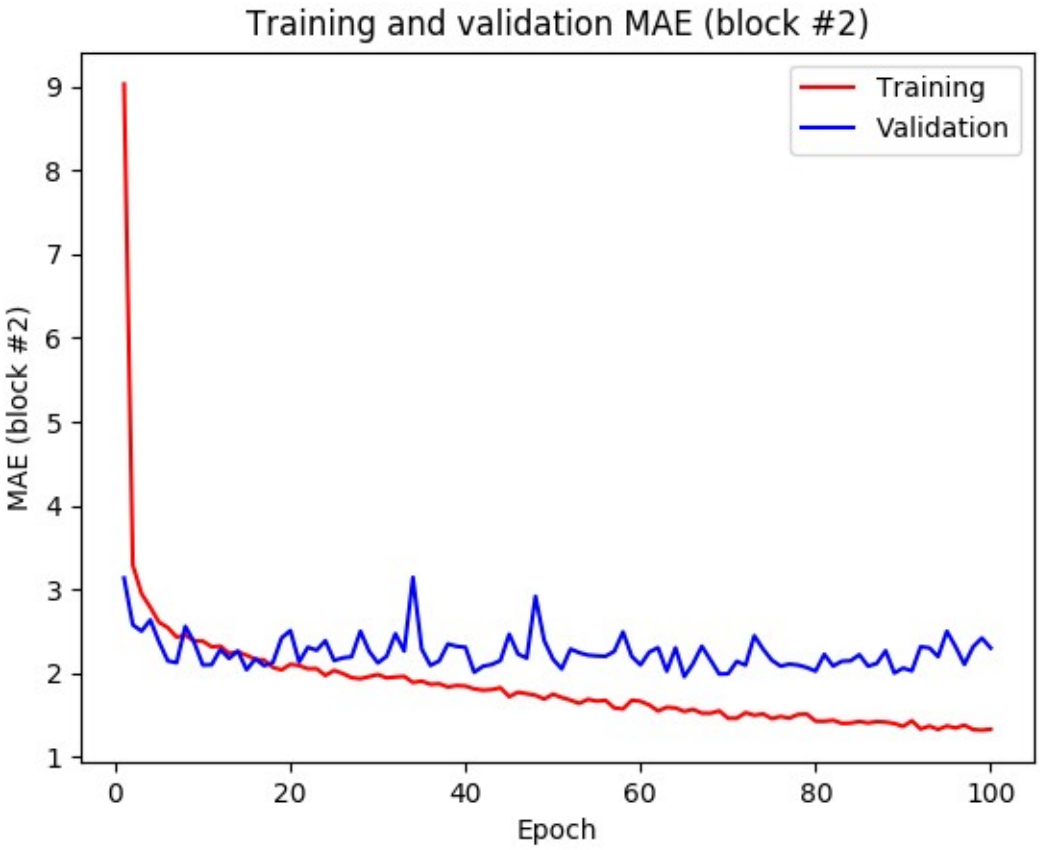


*Рисунок 15 – график значения MAE блока 2 из 6*

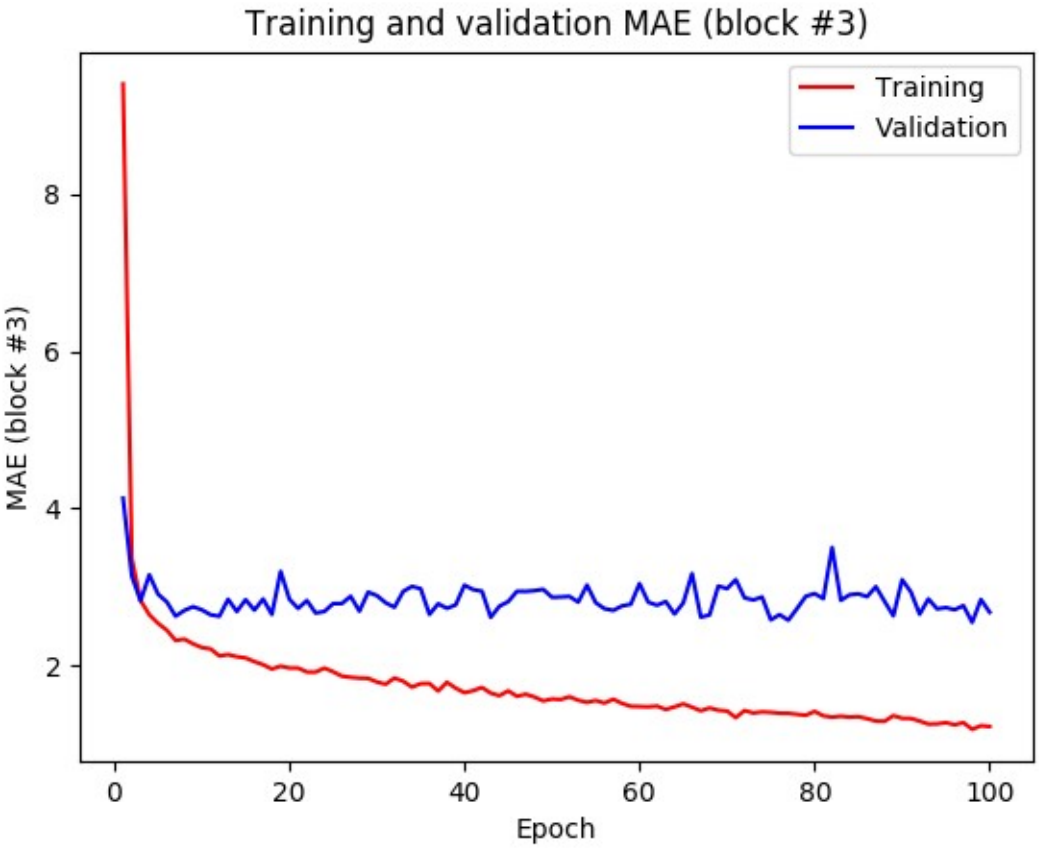


*Рисунок 16 – график значения MAE блока 3 из 6*

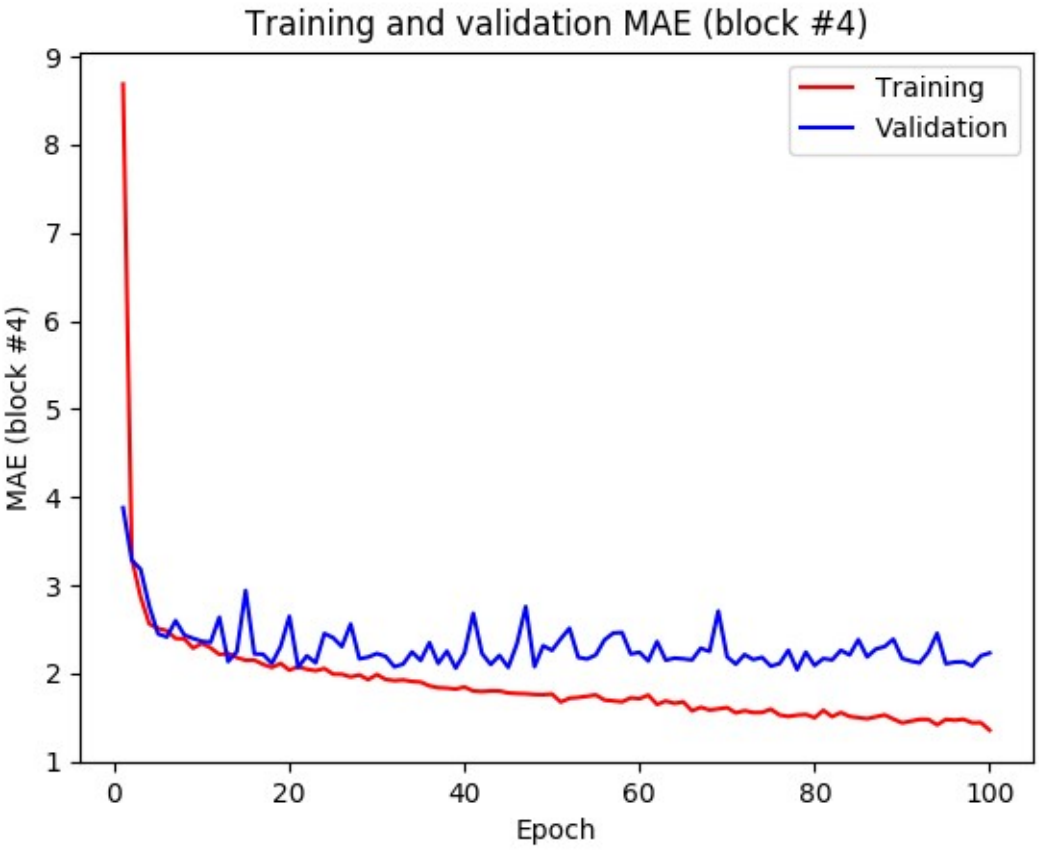


*Рисунок 17 – график значения MAE блока 4 из 6*

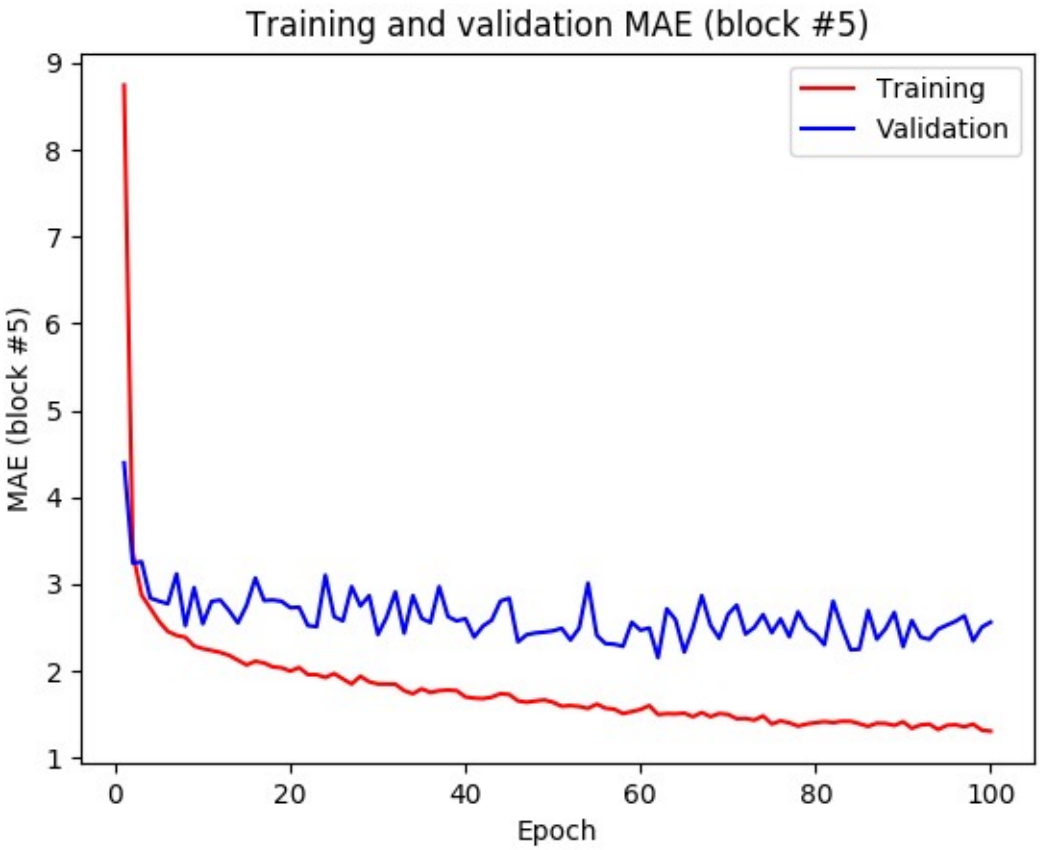


*Рисунок 17 – график значения MAE блока 5 из 6*

**Вывод.**

В ходе выполнения данной работы была изучена задача регрессии с помощью библиотеки Keras и ее отличие от задачи классификации. Была изучена перекрестная проверка, было проведено обучение с ее использованием для разных значений количества блоков.

# Приложение А

```
import numpy as np
import matplotlib.pyplot as plt
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

def draw_plot(label, train, val, blocks_num):
    plt.clf()
    arg = range(1, len(train) + 1)
    plt.plot(arg, train, 'r', label='Training')
    plt.plot(arg, val, 'b', label='Validation')
    plt.title('Training and validation ' + label)
    plt.xlabel('Epoch')
    plt.ylabel(label)
    plt.legend()
    plt.savefig(str(blocks_num) + '_fold_' + label + '.png')

def normalize(data):
    mean = data.mean(axis=0)
    data -= mean
    std = data.std(axis=0)
    data /= std

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

def fold(data, targets, k, num_epochs = 100):
    num_val_samples = len(data) // k
    fold_history = {'mae': [], 'val_mae': []}
    for i in range(k):
        print('processing fold #', i)
        val_data = data[i * num_val_samples: (i + 1) * num_val_samples]
        val_targets = targets[i * num_val_samples: (i + 1) * num_val_samples]
        train_data = np.concatenate(
            [data[:i * num_val_samples], data[(i + 1) * num_val_samples:]],
axis=0)
        train_targets = np.concatenate(
            [targets[:i * num_val_samples], targets[(i + 1) *
num_val_samples:]], axis=0)
        model = build_model((train_data.shape[1],))
        H = model.fit(train_data, train_targets, epochs=num_epochs,
batch_size=1, verbose=0,
            validation_data=(val_data, val_targets))
        mae = H.history['mean_absolute_error']
        val_mae = H.history['val_mean_absolute_error']
        loss = H.history['loss']
        val_loss = H.history['val_loss']
        fold_history['mae'].append(mae)
        fold_history['val_mae'].append(val_mae)
        draw_plot('MAE (block #' + str(i) + ')', mae, val_mae, k)
    draw_plot('fold MAE',
        [np.mean([x[i] for x in fold_history['mae']]) for i in
range(num_epochs)],
        [np.mean([x[i] for x in fold_history['val_mae']]) for i in
range(num_epochs)], k)
```

```
(train_data, train_targets), (test_data, test_targets) =
boston_housing.load_data()
# All the data available is both train and test, default separation is
conditional
data = np.concatenate([train_data, test_data], axis=0)
targets = np.concatenate([train_targets, test_targets], axis=0)
normalize(data)
fold(data, targets, 5)
```