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

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Искусственные нейронные сети»

Тема: Регрессионная модель изменения цен на дома в Бостоне

Студент гр. 8382 _____ Янкин Д.О.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2021

Цель работы.

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.

Данный набор содержит относительно немного образцов данных: всего 506, разбитых на 404 обучающих и 102 контрольных образца. И каждый признак во входных данных (например, уровень преступности) имеет свой масштаб. Например, некоторые признаки являются пропорциями и имеют значения между 0 и 1, другие — между 1 и 12 и т. д.

Постановка 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.

Задача классификации заключаются в разделение множества объектов на разные классы. Задача регрессии заключается в предсказании какого-либо значения.

Данные для обучения загружаются из библиотеки Keras:

```
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

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

Затем осуществляется нормализация данных:

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

Определена функция создания модели сети:

```
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
```

Выходной слой сети не имеет функции активации – для данной задачи регрессии нет смысла ограничивать диапазон выходных значений. В качестве функции потерь используется $MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \bar{Y}_i)^2$, Y_i и $\bar{Y}_i$ – истинное и предсказанное значения соответственно. В качестве метрики используется $MAE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \bar{Y}_i)$.

Так как набор данных небольшой, используется перекрестная проверка по K блокам. Данные разбиваются на K частей, одна часть выбирается в качестве валидационной, модель обучается по K-1 других частей. Всего создается K таких идентичных моделей. По полученным K оценкам вычисляется среднее, которое принимается за оценку модели.

Обучена модель из четырех подмоделей на 100 эпохах.

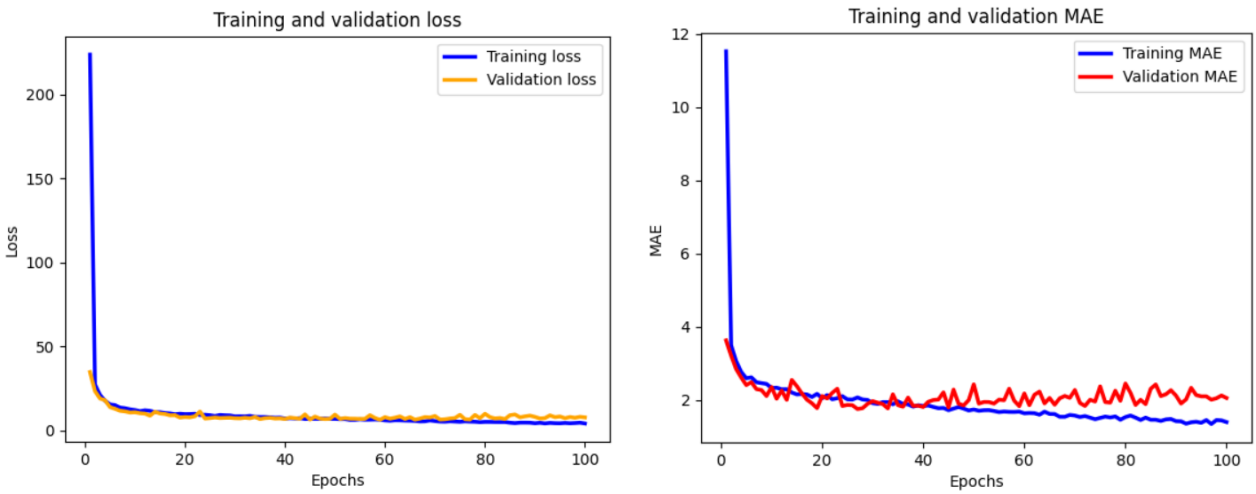

Рисунок 1. Первая модель, первая подмодель

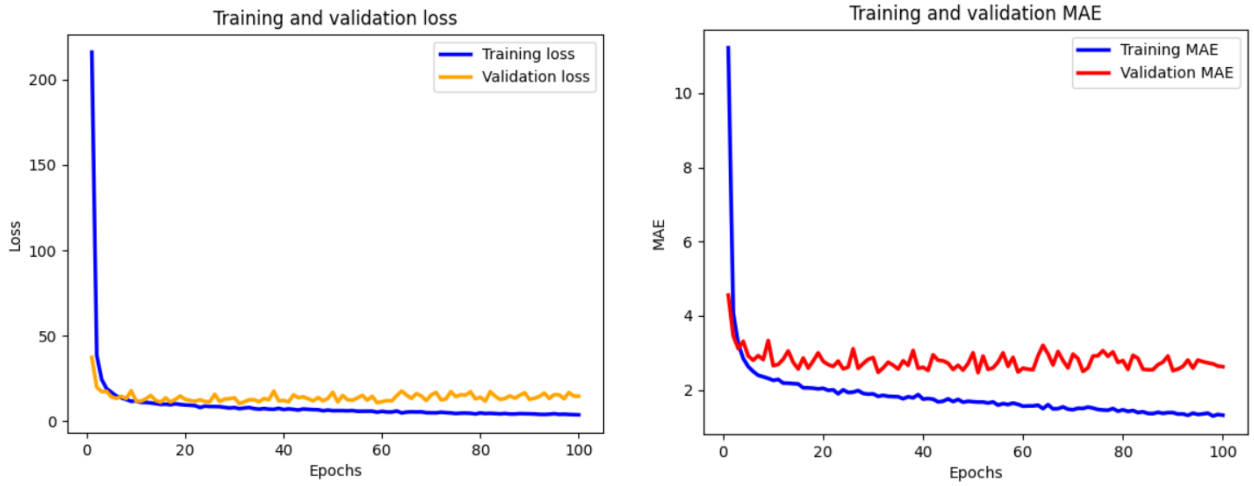

Рисунок 2. Первая модель, вторая подмодель

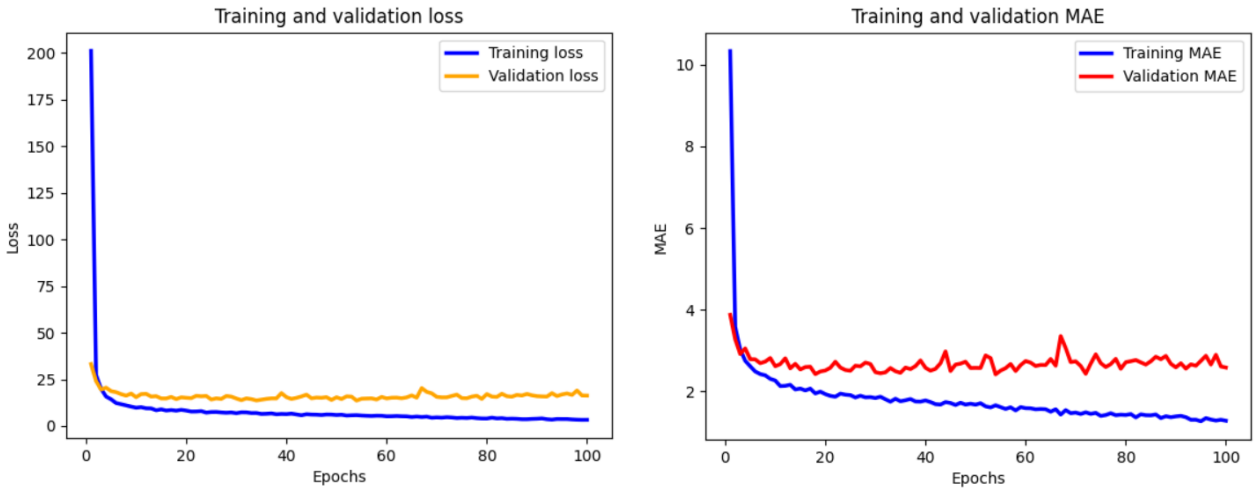

Рисунок 3. Первая модель, третья подмодель

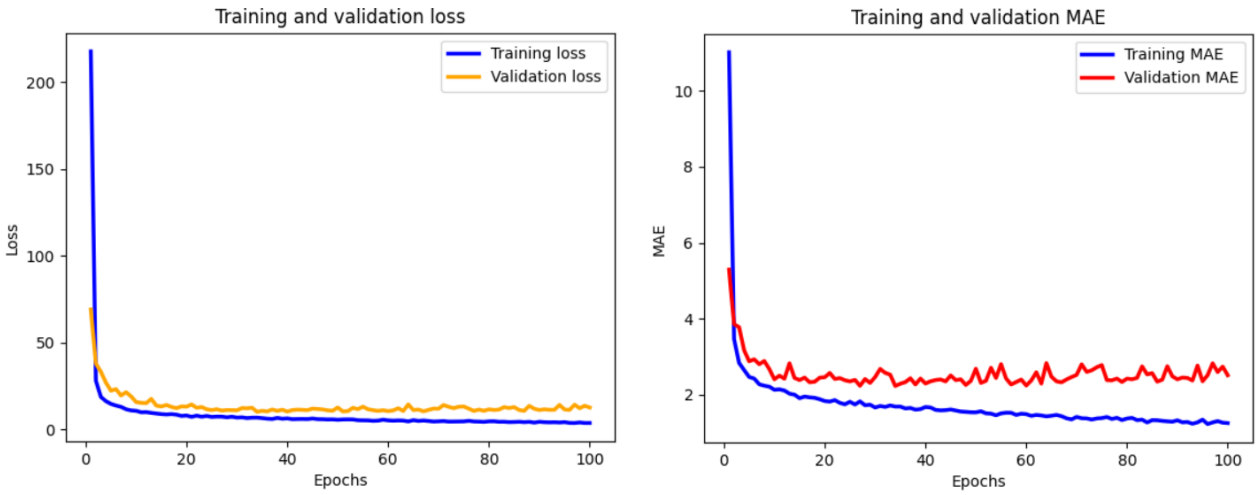

Рисунок 4. Первая модель, четвертая подмодель

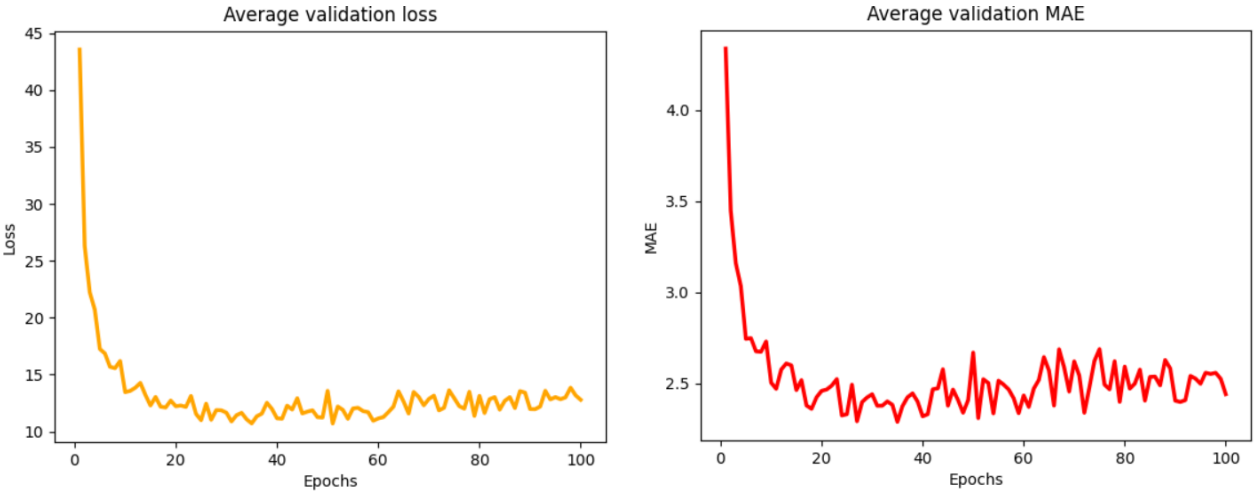

Рисунок 5. Показатели первой модели

Получена оценка модели 2.52, то есть результат модели отличается от истинного на 2520 долларов. Показатели сети перестают улучшаться примерно после 60 эпох, точка переобучения находится примерно там, поэтому в следующей модели число эпох будет уменьшено

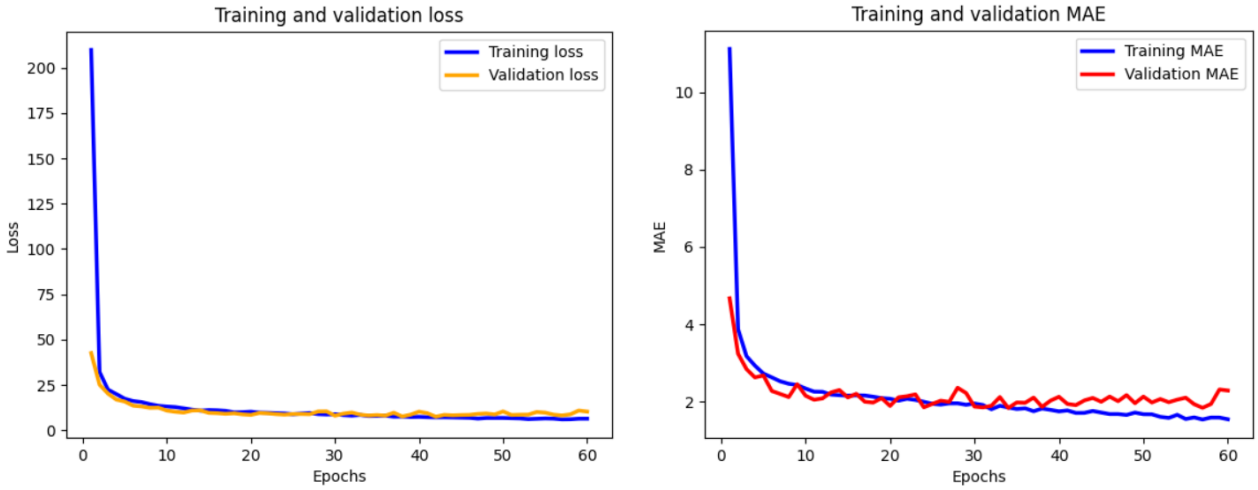

Рисунок 6. Вторая модель, первая подмодель

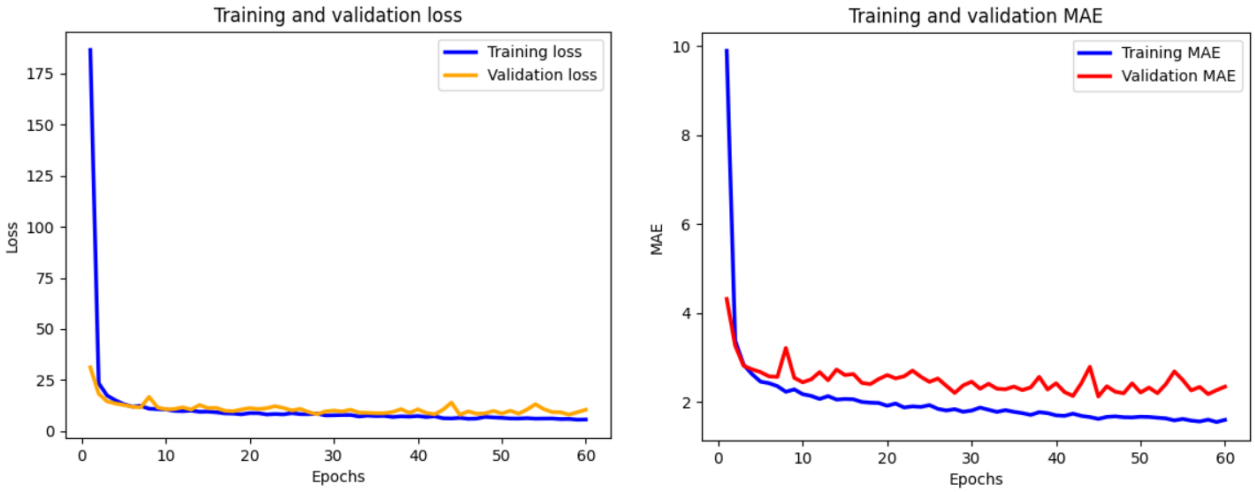

Рисунок 7. Вторая модель, вторая подмодель

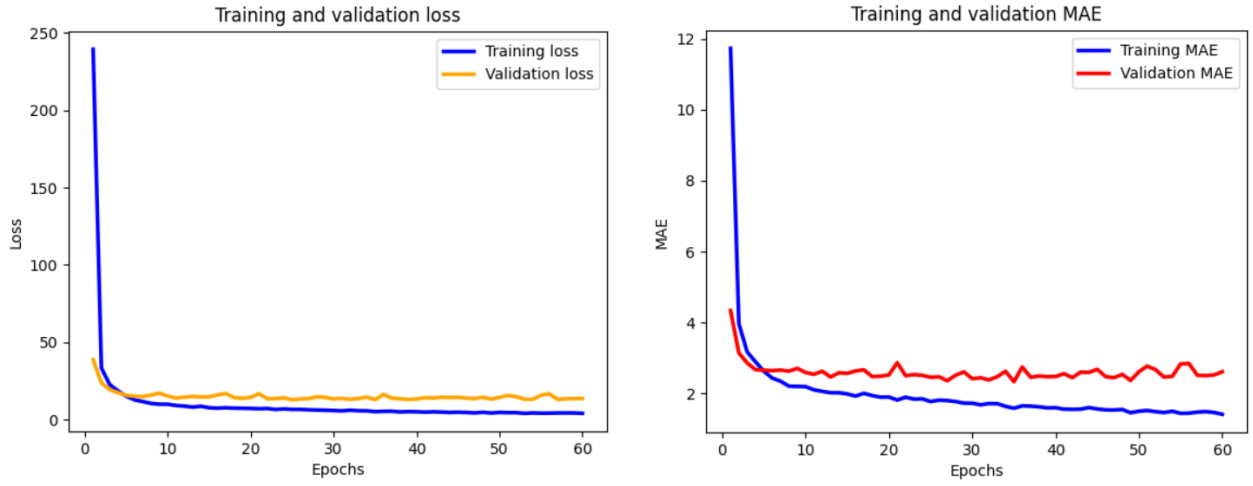

Рисунок 8. Вторая модель, третья подмодель

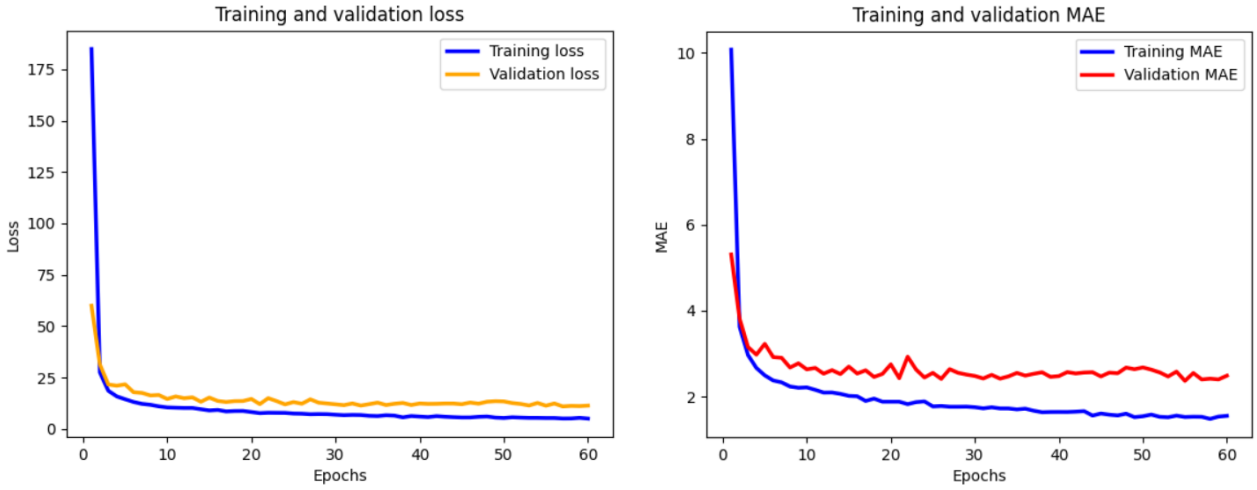

Рисунок 9. Вторая модель, четвертая подмодель

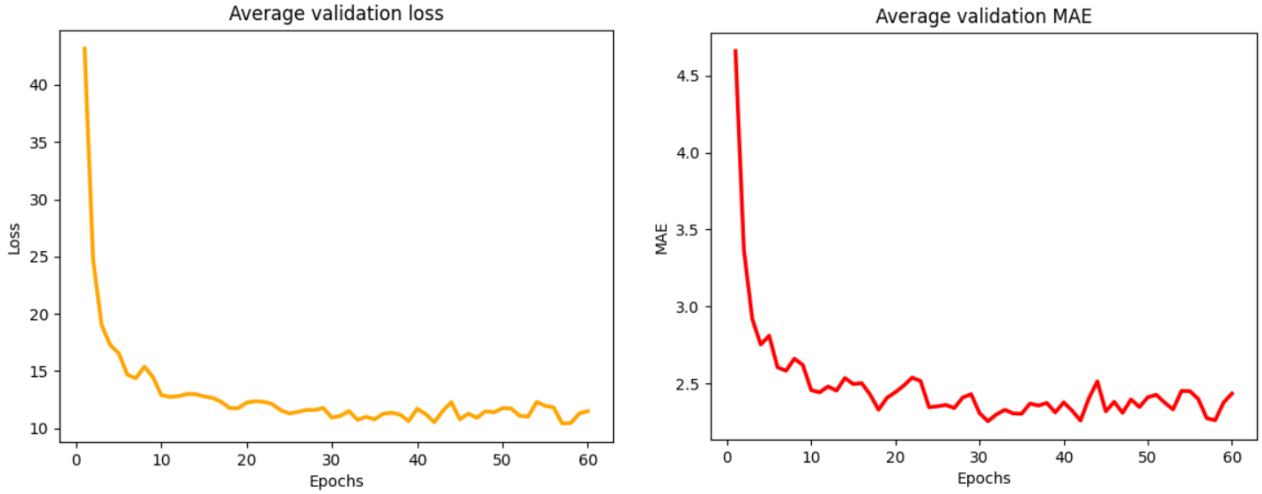

Рисунок 10. Показатели второй модели

При уменьшении числа эпох до 60 получена оценка 2.49, то есть результат стал незначительно лучше.

Далее исследуется влияние числа блоков. Количество подмоделей увеличено до 6.

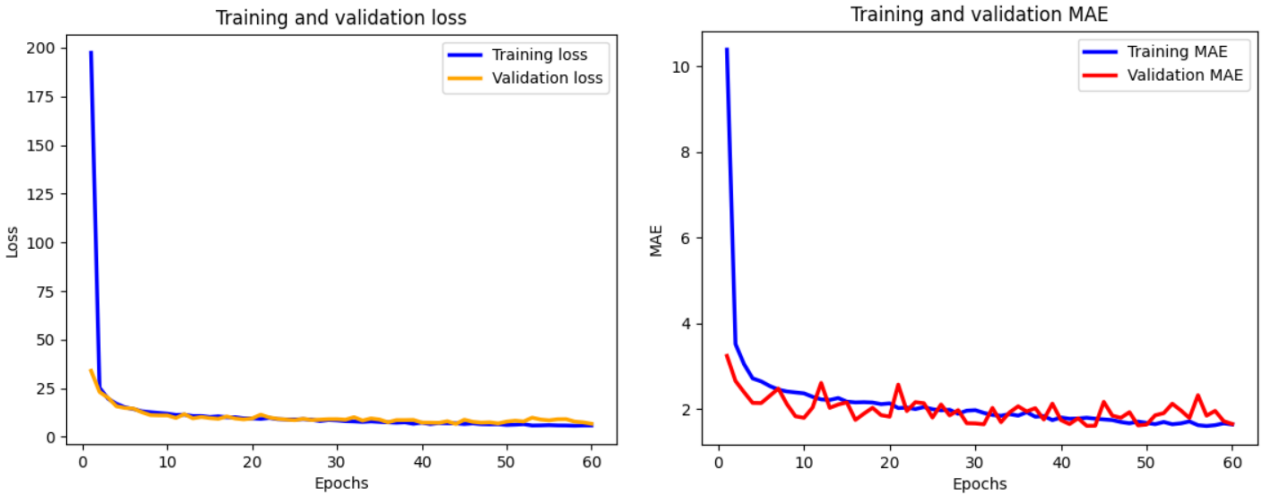

Рисунок 11. Третья модель, первая подмодель

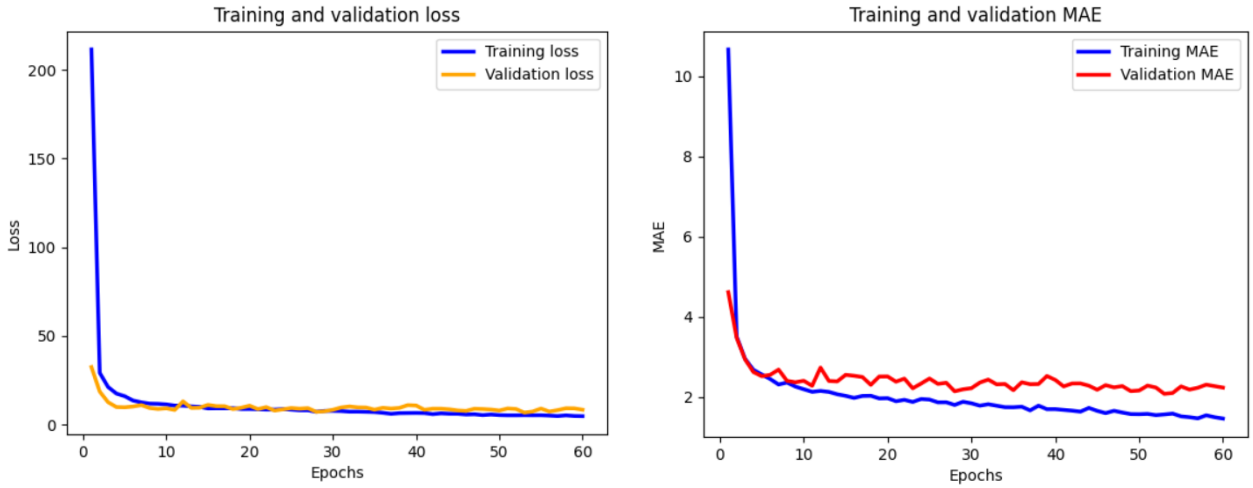

Рисунок 12. Третья модель, вторая подмодель

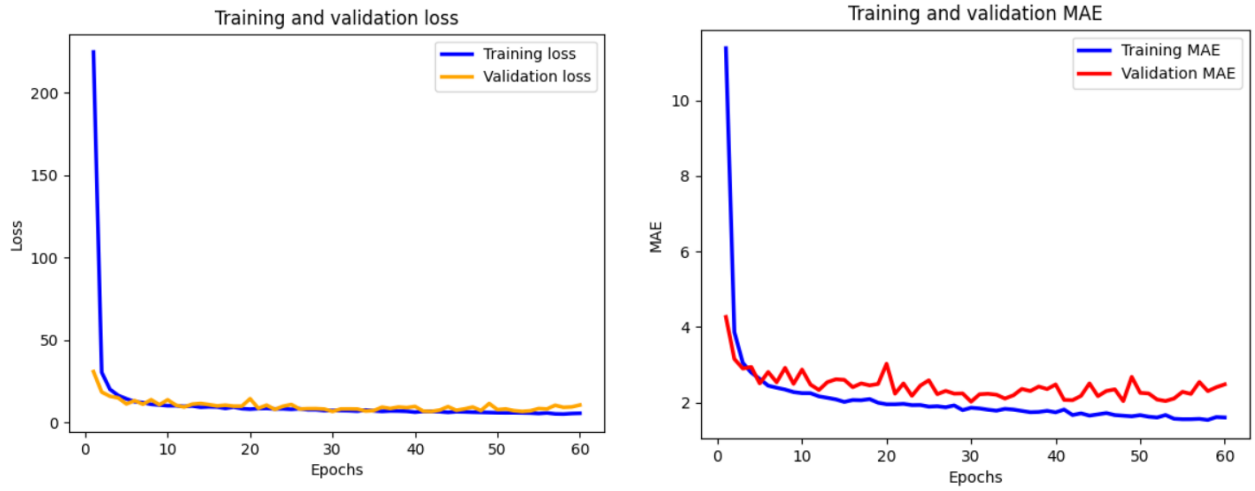

Рисунок 13. Третья модель, третья подмодель

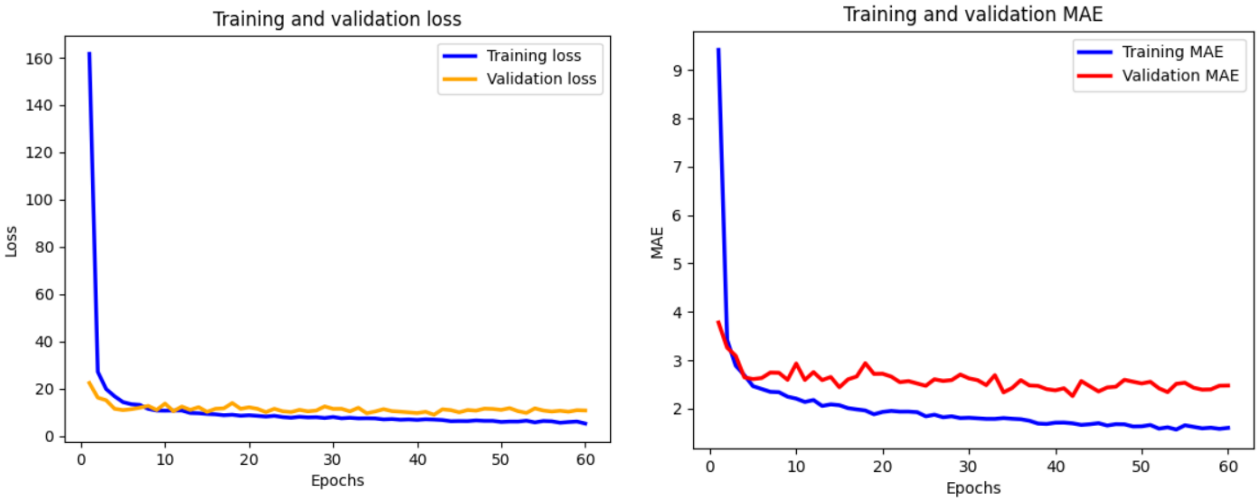

Рисунок 14. Третья модель, четвертая подмодель

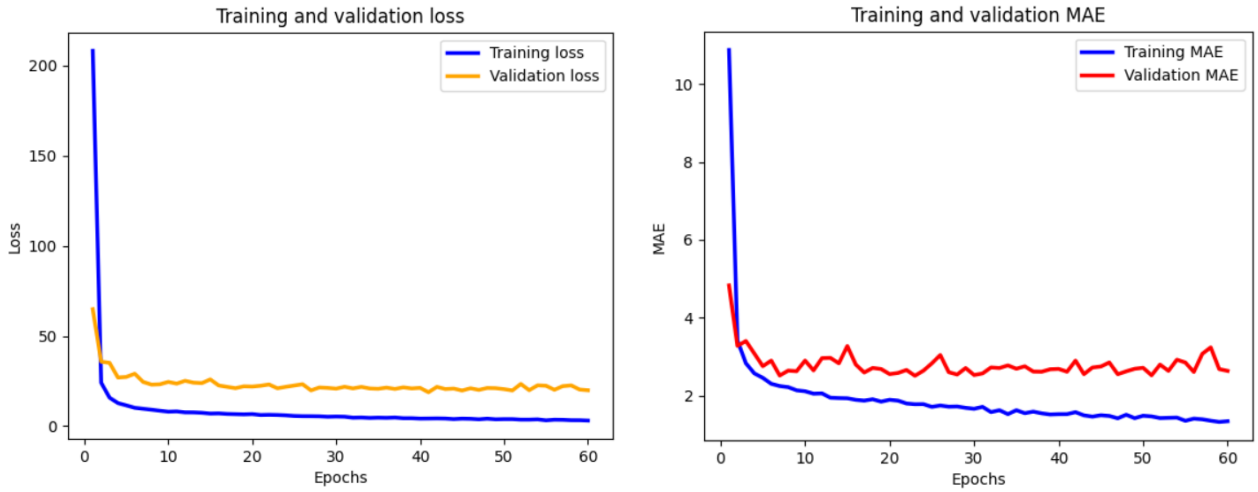

Рисунок 15. Третья модель, пятая подмодель

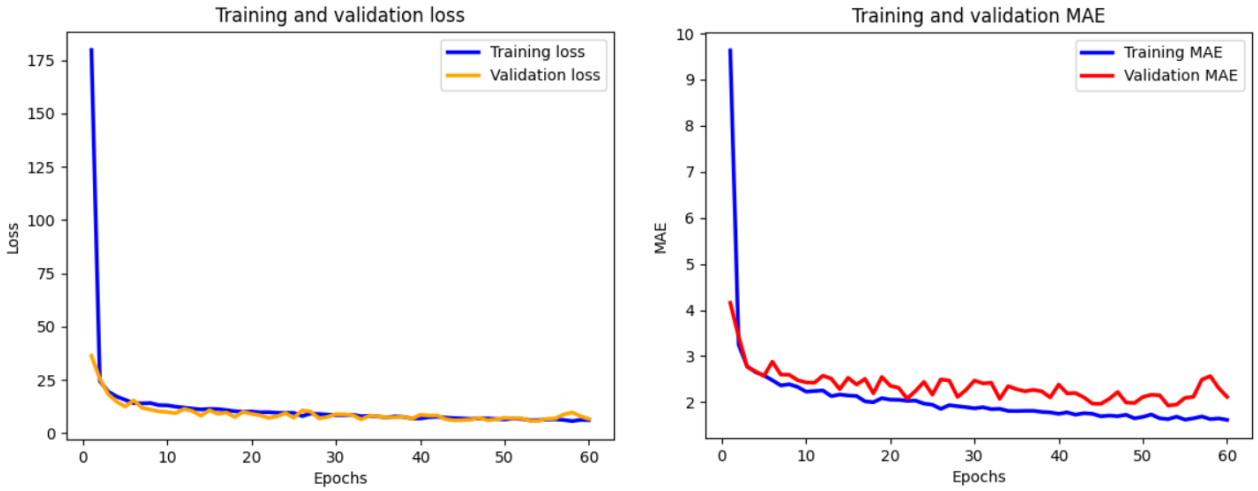

Рисунок 16. Третья модель, шестая подмодель

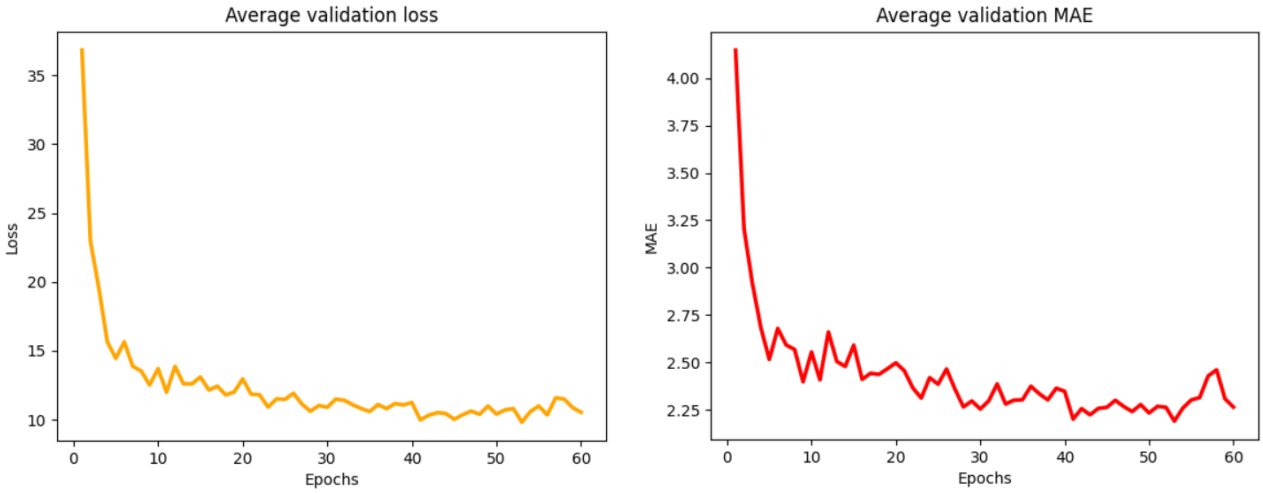

Рисунок 17. Показатели третьей модели

Для данной модели получена оценка 2.43, результат улучшился. Увеличение количества моделей снижает негативное влияние случаев неудачного обучения, но слишком сильно увеличивать этот параметр не стоит, чтобы количество тестовых данных оставалось на приемлемом уровне.

Выводы.

В ходе лабораторной работы была построена модель нейронной сети для решения задачи регрессии: предсказания стоимости домов на основании некоторых параметров. Для этого был использован метод перекрестной проверки на K блоков, исследовано влияние количества блоков, эпох обучения, найдена точка переобучения.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

```
import pandas
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt

dataframe = pandas.read_csv("sonar.csv", header=None)
dataset = dataframe.values
X = dataset[:, 0:30].astype(float)
Y = dataset[:, 60]

encoder = LabelEncoder()
encoder.fit(Y)
encoded_Y = encoder.transform(Y)

model = Sequential()
model.add(Dense(60, input_dim=30, activation='relu'))
model.add(Dense(15, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
history = model.fit(X, encoded_Y, epochs=100, batch_size=10,
validation_split=0.1)

history_dict = history.history

loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, 'bo', label='Training loss')
plt.plot(epochs, val_loss_values, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

plt.clf()
acc_values = history_dict['accuracy']
val_acc_values = history_dict['val_accuracy']
plt.plot(epochs, acc_values, 'bo', label='Training acc')
plt.plot(epochs, val_acc_values, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
```

```
plt.legend()
plt.show()
```