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

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Искусственные нейронные сети»

Тема: Регрессионная модель изменения цен на дома в Бостоне

Студент гр. 7383 _____ Александров Р.А.

Преподаватель _____ Жукова Н.А.

Санкт-Петербург
2020

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.

Постановка задачи.

1. Ознакомиться с задачей регрессии
2. Изучить отличие задачи регрессии от задачи классификации
3. Создать модель
4. Настроить параметры обучения
5. Обучить и оценить модель
6. Ознакомиться с перекрестной проверкой

Требования.

1. Объяснить различия задач классификации и регрессии
2. Изучить влияние кол-ва эпох на результат обучения модели
3. Выявить точку переобучения
4. Применить перекрестную проверку по K блокам при различных K
5. Построить графики ошибки и точности во время обучения для моделей, а также усредненные графики по всем моделям

Теоретическое положение.

Классификационное прогнозирующее моделирование – это задача приближения функции отображения (*f*) от входных переменных (*X*) к дискретным выходным переменным (*y*). Задача классификации требует, чтобы примеры были классифицированы в один или два класса. Для классификационных моделей характерно предсказывать непрерывное значение как вероятность данного примера, принадлежащего каждому выходному классу. Вероятности могут быть интерпретированы как вероятность или достоверность данного примера, принадлежащего каждому классу. Прогнозируемая вероятность может быть преобразована в значение класса путем выбора метки класса, которая имеет наибольшую вероятность.

Прогнозирующее регрессионное моделирование – это задача приближения функции отображения (*f*) от входных переменных (*X*) к непрерывной выходной переменной (*y*). Задача регрессии требует предсказания количества.

Выполнение работы.

В ходе работы была создана и обучена модель нейронной сети, весь код представлен в приложении А. Первоначальные параметры: количество эпох равно 200, K = 4.

Из рис. 1-4 видим, что переобучение модели примерно начинается на 50 эпохе, поэтому уменьшим количество эпох до 50.

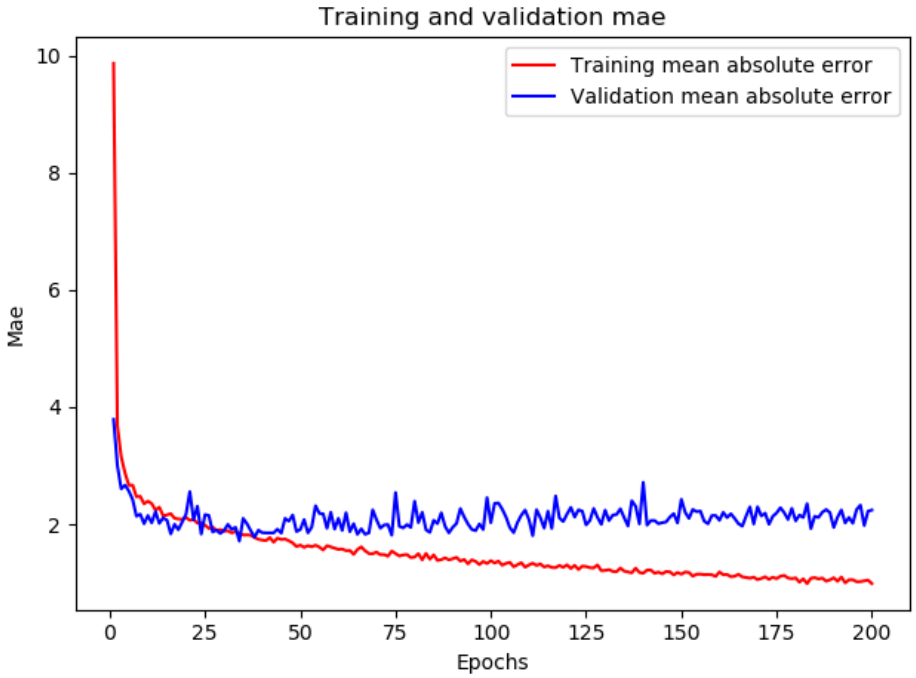

Рисунок 1 – Значение mae для 1 блока

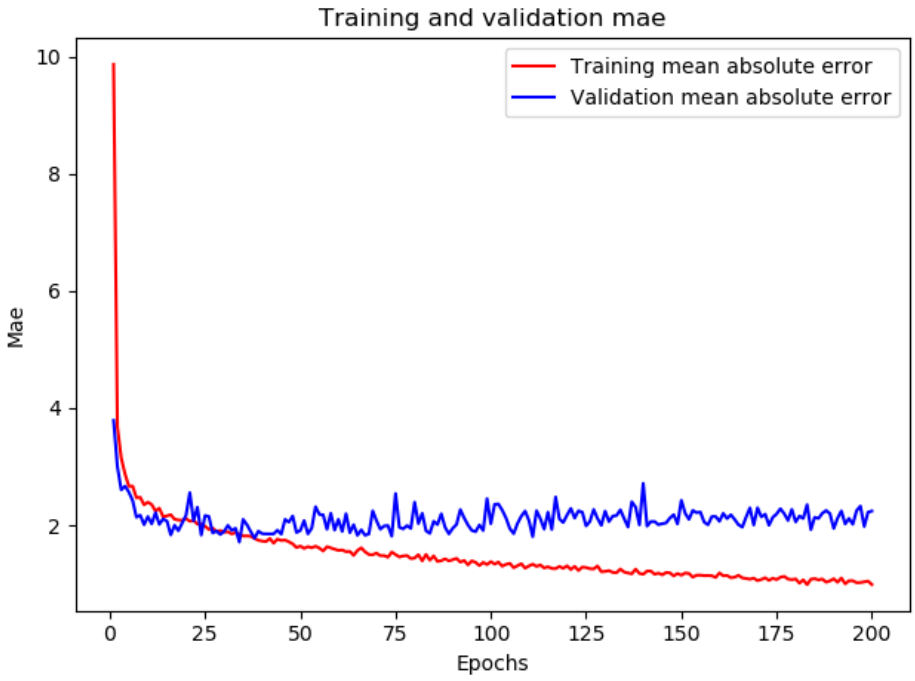

Рисунок 2 – Значение mae для 2 блока

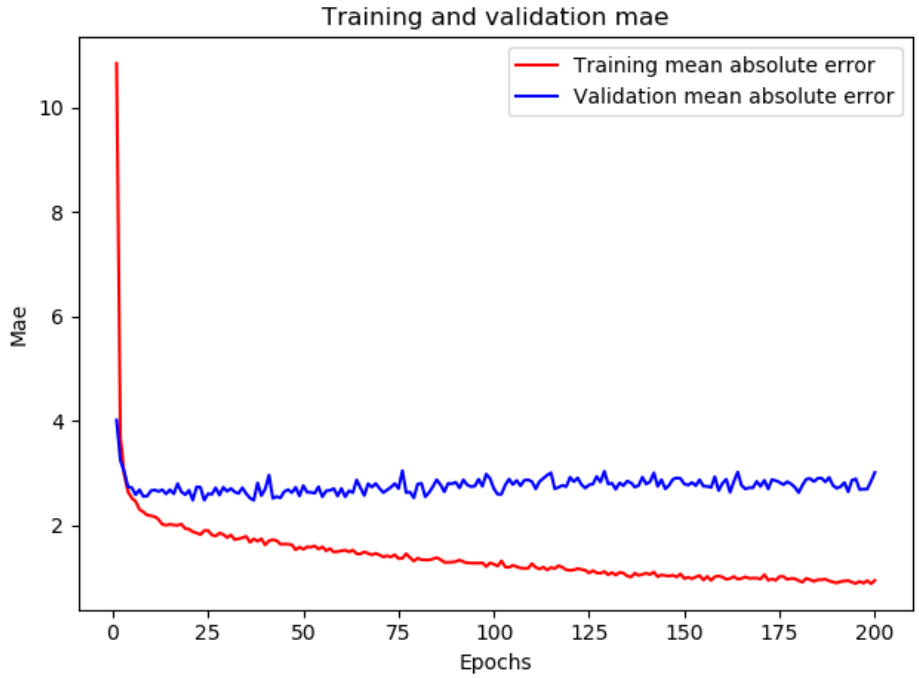

Рисунок 3 – Значение mae для 3 блока

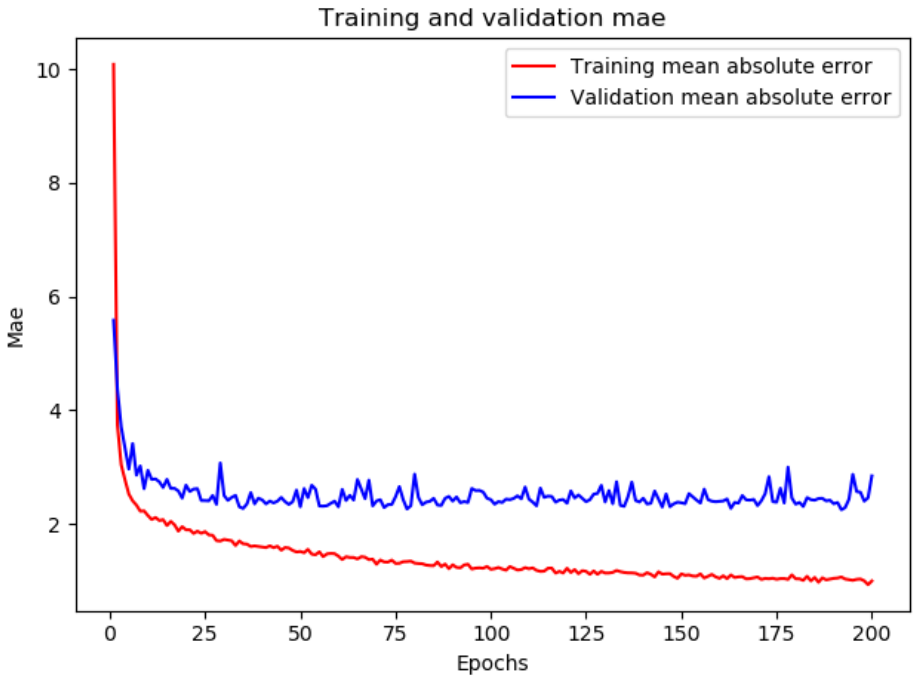

Рисунок 4 – Значение mae для 4 блока

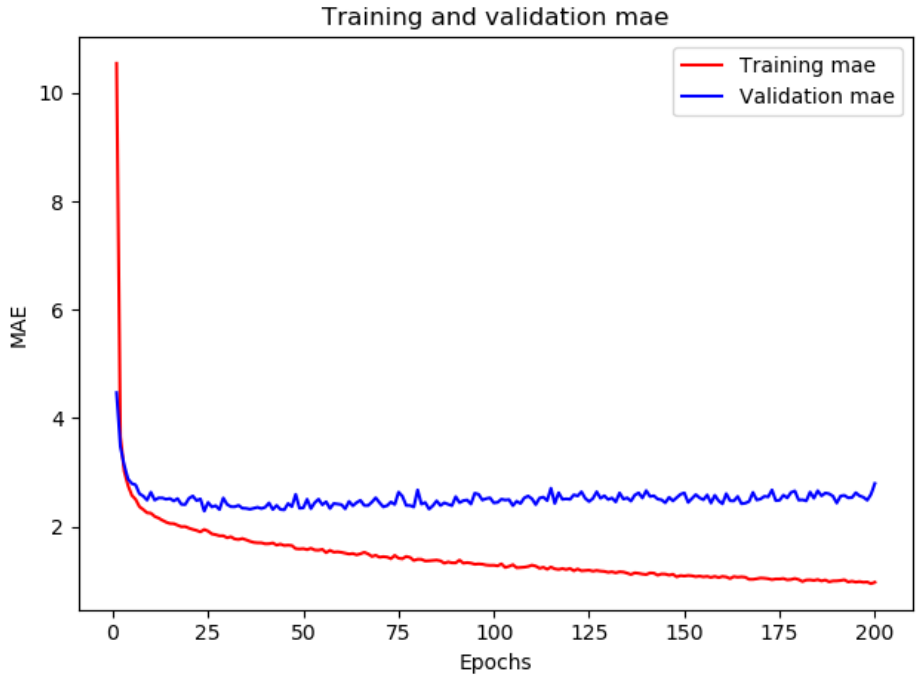

Рисунок 5 – Среднее значение mae

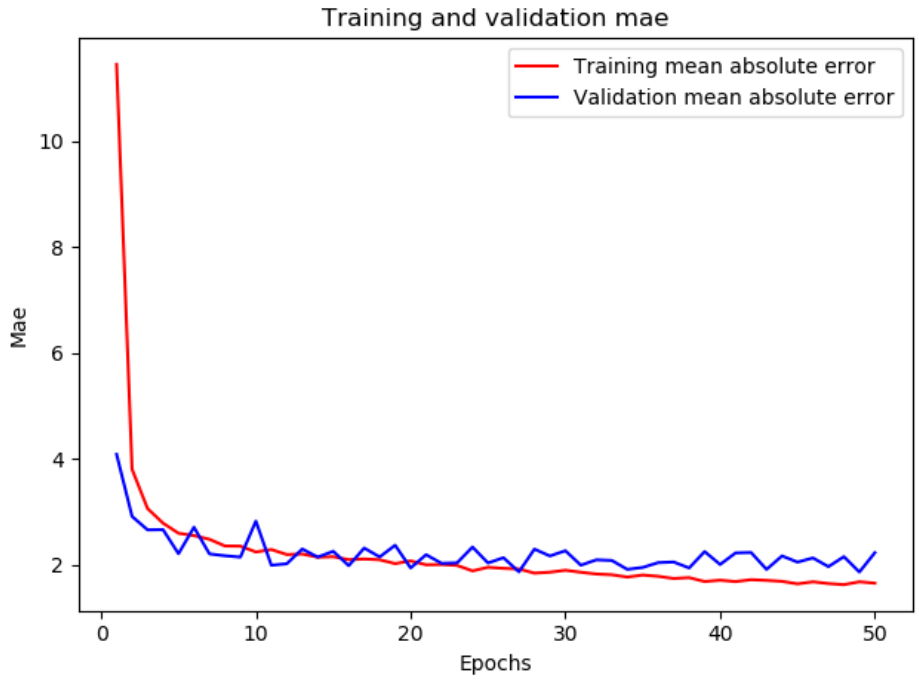

Рисунок 6 – Значение mae для 1 блока

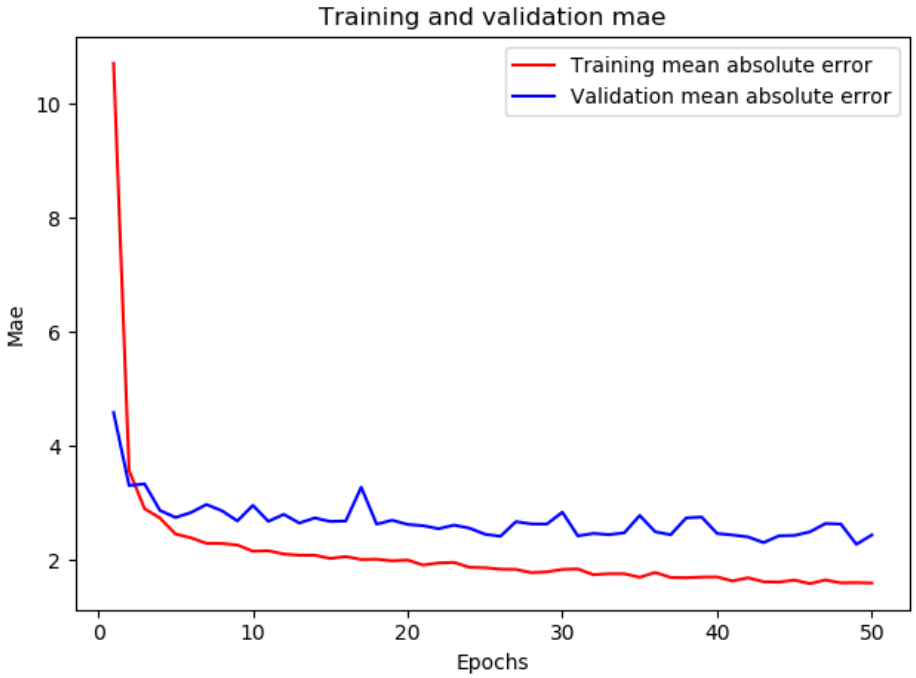

Рисунок 7 – Значение mae для 2 блока

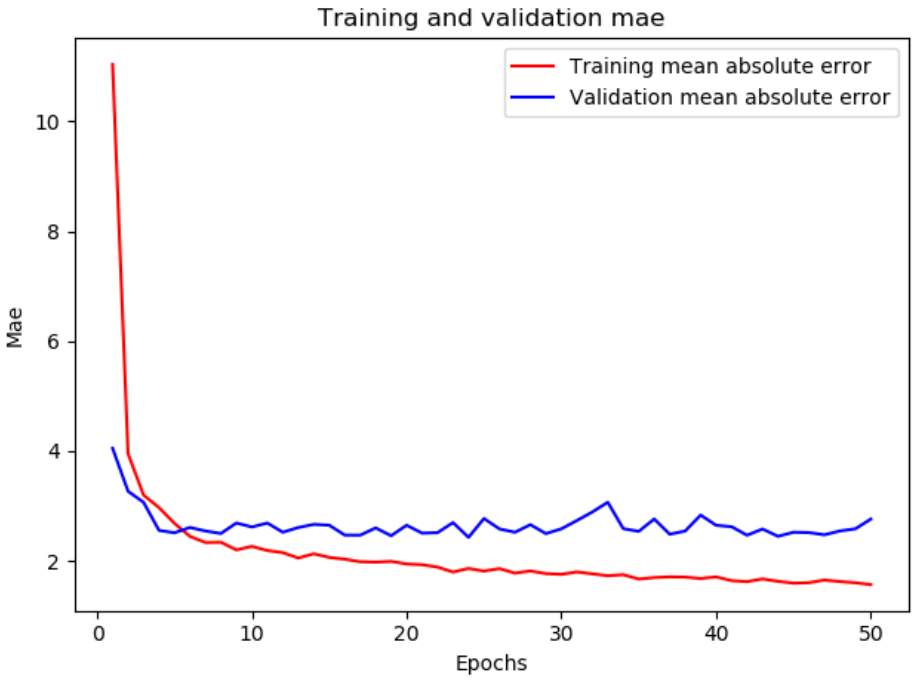

Рисунок 8 – Значение mae для 3 блока

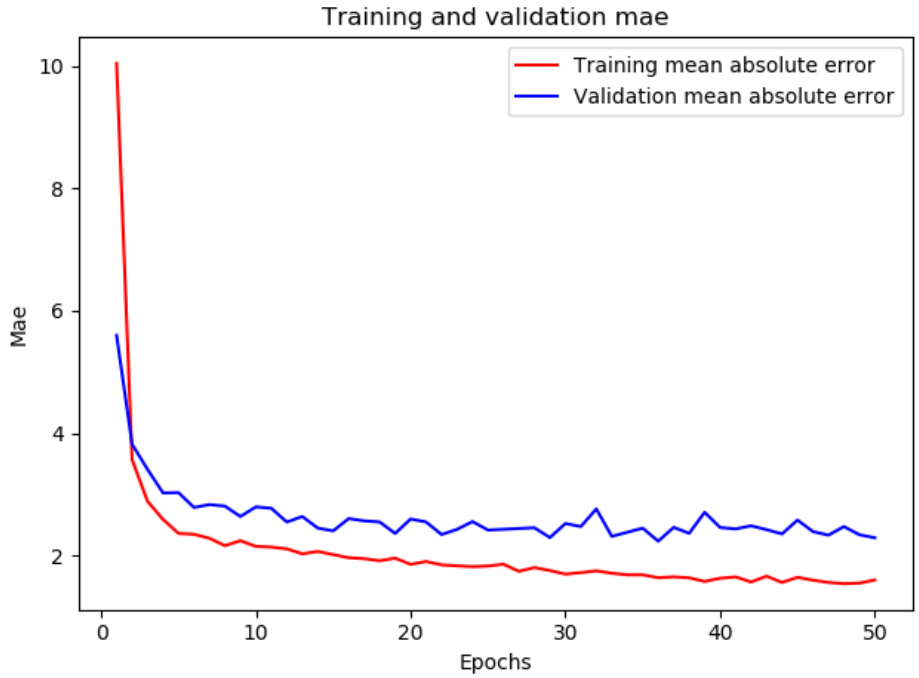

Рисунок 9 – Значение mae ae для 4 блока

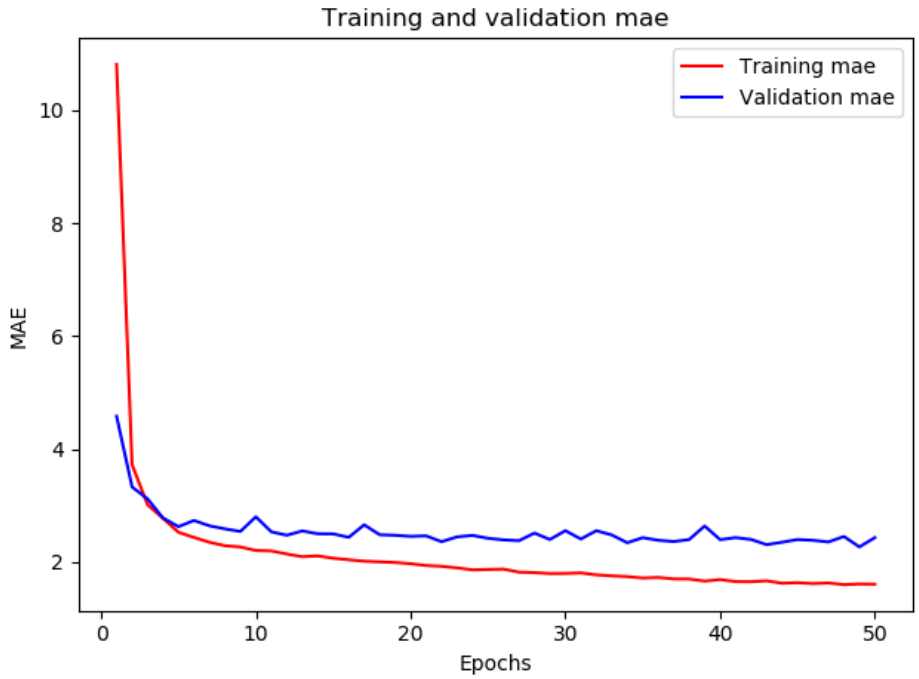

Рисунок 10 – Среднее значение mae

Выводы.

В ходе работы были выявлены различия между задачами классификации регрессии, изучено влияние количества эпох на результат обучения модели, выявлена точка переобучения, применена перекрёстная проверка по K-блокам и

построены необходимые графики. Наименьшее значение mae наблюдается в модели с 3 блоками.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

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

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

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

train_data -= mean
train_data /= std
test_data -= mean
test_data /= std

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

k = 4
num_val_samples = len(train_data) // k
num_epochs = 50
all_scores = []
val_mae_histories = []
mae_histories = []
epochs = range(1, num_epochs + 1)

for i in range(k):
    print('processing fold #', i)

    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()

    history = model.fit(partial_train_data, partial_train_targets, epochs=num_epochs, batch_size=1, verbose=0, validation_data=(val_data, val_targets))
    history_dict = history.history

    mae_hist = history_dict['mae']
    mae_histories.append(mae_hist)

    val_mae_hist = history_dict['val_mae']
    val_mae_histories.append(val_mae_hist)

    plt.plot(epochs, mae_hist, 'r', label='Training mean absolute error')
    plt.plot(epochs, val_mae_hist, 'b', label='Validation mean absolute error')
    plt.title('Training and validation mae')
    plt.xlabel('Epochs')
    plt.ylabel('Mae')
    plt.legend()
    plt.show()

    val_mse, val_mae = model.evaluate(val_data, val_targets, verbose=0)

    all_scores.append(val_mae)

print(np.mean(all_scores))
plt.clf()
plt.plot(epochs, np.mean(mae_histories, axis=0), 'r', label='Training mae')
plt.plot(epochs, np.mean(val_mae_histories, axis=0), 'b', label='Validation mae')
plt.title('Training and validation mae')
plt.xlabel('Epochs')
plt.ylabel('MAE')
plt.legend()
plt.show()
```