РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ № 3-4

дисциплина: Системы поддержки принятия решений

Группа: НФИбд-01-19

Студенты:

Бармина Ольга Константиновна - 1032192873

Горбунова Ярослава Михайловна - 1032192862

Исаханян Эдуард Тигранович - 1032190607

Евсеева Дарья Олеговна - 1032192860

МОСКВА

2022 г.

Содержание

Обучение нейросети

Ссылка на ноутбук с подробными комментариями:
https://colab.research.google.com/drive/1hosJhKvQv-p90ONy52iWoUWogh8NTKqb?usp=sharing%23scrollTo=8bce8787

Разбор кода проекта:

Устанавливаем необходимые для работы библиотеки и импортируем их

```
!pip install tensorflow numpy matplotlib keras
```

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import keras
%matplotlib inline
```

Данную нейросеть было решено обучать для распознавания рукописных цифр. Для достижения данной цели загружаем датасет mnist и просматриваем количество тренировочных и тестовых данных в наборе данных, а также размеры имеющихся изображений.

```
# загружаем данные датасета mnist - рукописные цифры 0-9
mnist = tf.keras.datasets.mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

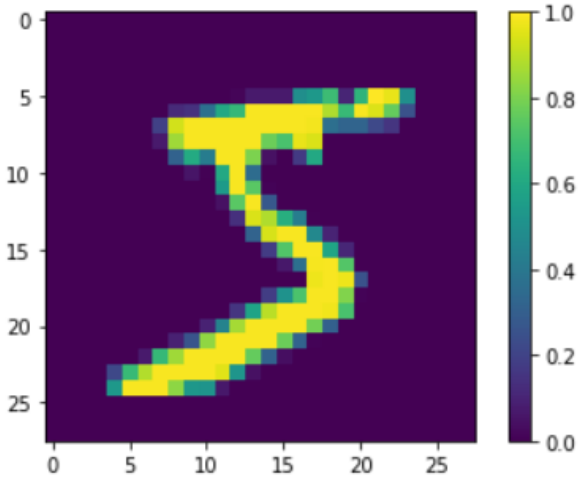
```
# смотрим размеры данных (сначала тренировочных, затем тестовых), видим их количество и для изображений их размер в пикселях
print("Training set (images) shape: {shape}".format(shape=train_images.shape))
print("Training set (labels) shape: {shape}".format(shape=train_labels.shape))

print("Test set (images) shape: {shape}".format(shape=test_images.shape))
print("Test set (labels) shape: {shape}".format(shape=test_labels.shape))
```

```
Training set (images) shape: (60000, 28, 28)
Training set (labels) shape: (60000,)
Test set (images) shape: (10000, 28, 28)
Test set (labels) shape: (10000,)
```

Далее нормализуем наши изображения для последующей работы с ними и проверяем результат нормализации на нулевом изображении тренировочной выборки.

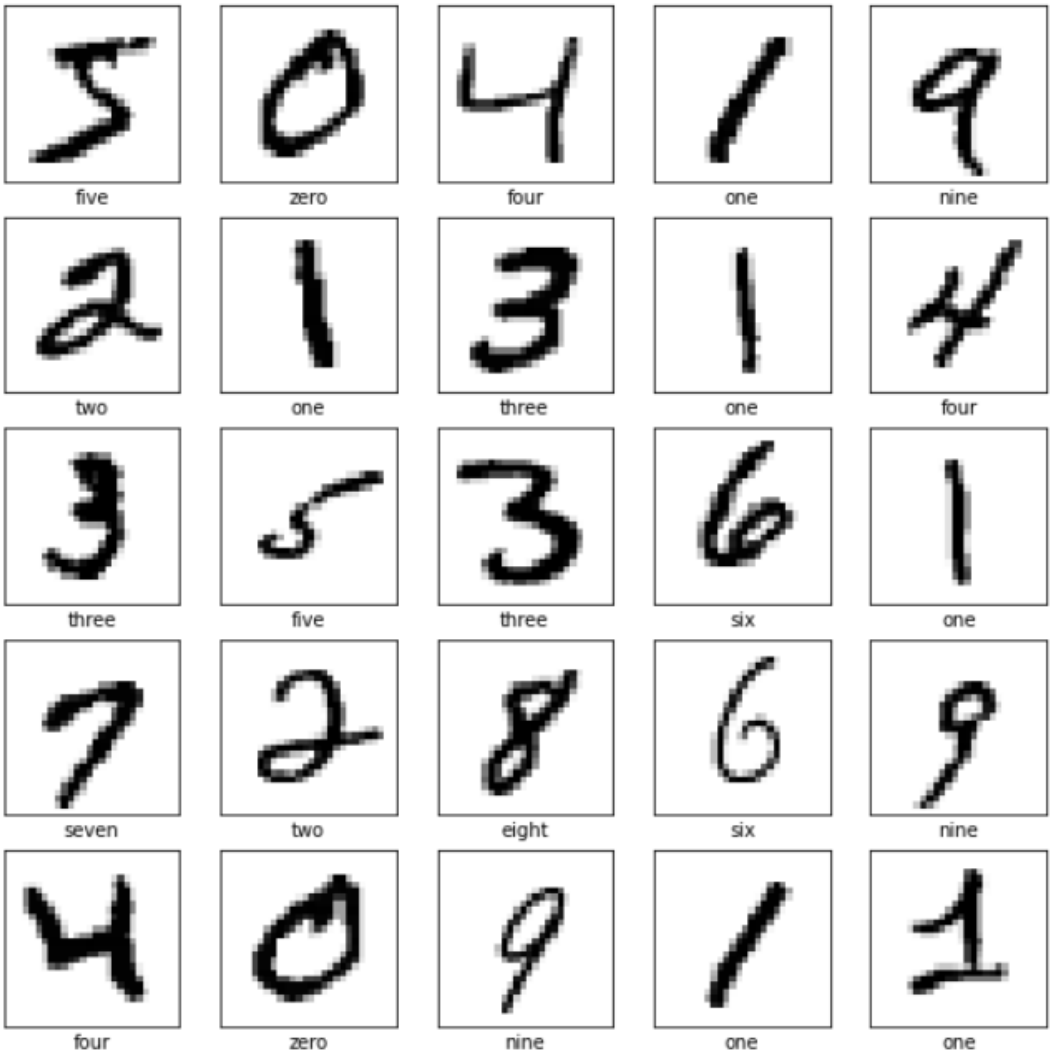
```
# нормализуем изображение
train_images = train_images / 255.0
test_images = test_images / 255.0
```

```
plt.figure()
plt.imshow(train_images[0])
plt.colorbar()
plt.grid(False)
plt.show()
```



После этого введём названия для имеющихся классов, так как в самом наборе они не прописаны и убедимся на первых 25 изображениях тренировочной выборки, что нормализация между названиями и изображениями выполнена успешно.

```
# названия классов не прописаны в датасете, задаем их сами
class_names = ['zero', 'one', 'two', 'three', 'four',
               'five', 'six', 'seven', 'eight', 'nine']
# выведем 25 изображений с подписями, чтобы убедится что нормализация прошла успешно
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.binary)
    plt.xlabel(class_names[train_labels[i]])
plt.show()
```



Теперь создадим нашу модель, используя библиотеку TensorFlow, следующим образом. Компилируем её, используя оптимайзер Адам, который позволяет модели обновляться на основании данных и функции потерь, функцию потерь, определяющая точность модели, минимизируется за счёт кросс энтропии, метрику, которая контролирует и тестирует шаги при обучении.

```
# составляем модель и забиваем слои, flatten - входной слой, принимает на вход изображения 28x28,
# dense - сверточный слой на 128 нейронов с функцией активации relu,
# второй слой dense возвращает вектор длиной в кол-о классов с вероятностью что объект относится к каждому классу
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(len(class_names))
])
```

```
# компилируем модель с оптимайзером Адам (позволяет модели обновлятся на основании данных и функции потерь),
# функция, которая определяет насколько точна модель минимизируется за счет кросс энтропии
# метрика контролирует и тестирует шаги при обучении
model.compile(optimizer='adam', loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True), metrics = ['accuracy'])
```

Обучаем готовую модель. Количество эпох может быть выбрано путём проб до момента, пока не наступит переобучение модели, то есть, не начнёт падать качество обучения. В нашем случае было решено обучать на протяжении 10 эпох.

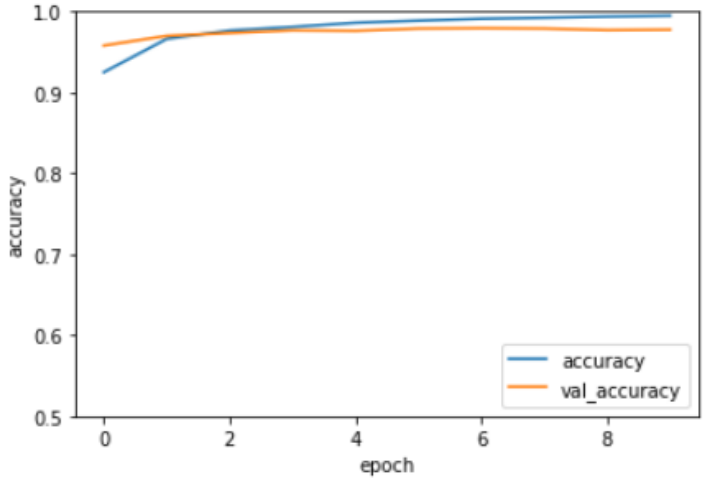
```
# обучаем модель, можем увеличивать кол-о эпох, пока не заметим падение в качестве обучения
history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))
```

```
Epoch 1/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2612 - accuracy: 0.9250 - val_loss: 0.1433 - val_accuracy: 0.9580
Epoch 2/10
1875/1875 [==============================] - 5s 2ms/step - loss: 0.1146 - accuracy: 0.9662 - val_loss: 0.1008 - val_accuracy: 0.9699
Epoch 3/10
1875/1875 [==============================] - 5s 2ms/step - loss: 0.0793 - accuracy: 0.9765 - val_loss: 0.0827 - val_accuracy: 0.9735
Epoch 4/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0600 - accuracy: 0.9809 - val_loss: 0.0764 - val_accuracy: 0.9768
Epoch 5/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0456 - accuracy: 0.9861 - val_loss: 0.0801 - val_accuracy: 0.9762
Epoch 6/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0355 - accuracy: 0.9887 - val_loss: 0.0723 - val_accuracy: 0.9790
Epoch 7/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0282 - accuracy: 0.9911 - val_loss: 0.0699 - val_accuracy: 0.9794
Epoch 8/10
1875/1875 [==============================] - 5s 2ms/step - loss: 0.0248 - accuracy: 0.9924 - val_loss: 0.0711 - val_accuracy: 0.9790
Epoch 9/10
1875/1875 [==============================] - 5s 2ms/step - loss: 0.0200 - accuracy: 0.9939 - val_loss: 0.0828 - val_accuracy: 0.9771
Epoch 10/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0165 - accuracy: 0.9949 - val_loss: 0.0790 - val_accuracy: 0.9776
```

Теперь проанализируем точность обучения нашей модели. Изучая график сравнения полученных значений (accuracy) со значениями на тестовом датасете (val_accuracy) можно заметить, что до наступления первой эпохи точность модели была немного ниже, чем для работы с тестовым датасетом, затем с первой по третью эпохи точности практически одинаковы по значениям, а начиная с третьей эпохи точность модели начинает возрастать и достигает 0,9775999784, что говорит об отличном качестве обучения модели, что позволяет теперь делать весьма точные предсказания принадлежности изображения какому-то из наших классов.

```
# оценим точность обучения, сравнивая полученные значения с тестовым датасетом
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.xlabel('epoch')
plt.ylabel('accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')

test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)
print('Test accuracy:', test_acc)
```

```
313/313 - 0s - loss: 0.0790 - accuracy: 0.9776 - 406ms/epoch - 1ms/step
Test accuracy: 0.9775999784469604
```



Проверим работу предсказания нашей моделью классов, к которым могут относиться тестовые изображения.

```
# на основании обученной модели сделаем предсказание к какому классу относятся изображения
probability_model = tf.keras.Sequential([model, tf.keras.layers.Softmax()])
predictions = probability_model.predict(test_images)
# посмотрим результат предсказаний для 1ого объекта, и посмотрим к какому классу объект принадлежит с большей вероятностью
print(predictions[0])
np.argmax(predictions[0])
```

```
[2.6500420e-11 1.0124619e-11 1.0289765e-08 1.2103928e-04 9.2086495e-14
 1.2451560e-09 6.2051887e-16 9.9987829e-01 3.2048592e-10 7.0983975e-07]
7
```

Для более наглядного анализа напишем функции для отображения двух графиков для каждого из изображений.

Данная функция изображает само изображения и ниже подписывает, к какому классу наша модель отнесла изображение и с какой вероятностью, надпись отображается синим цветом, если предсказанный класс совпадает с действительным классом изображения, красным цветом – в ином случае.

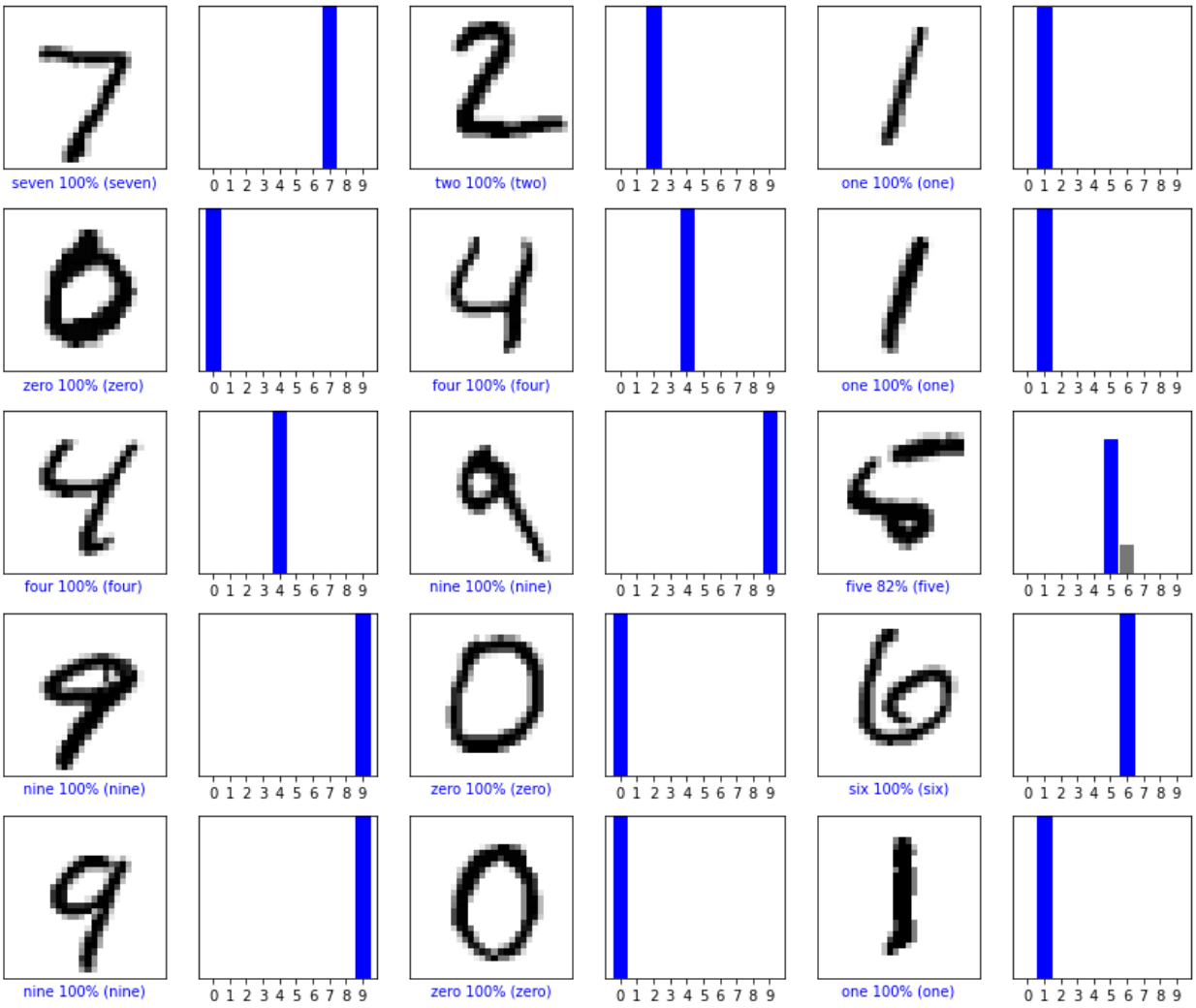
```
# задаем функцию для просмотра изображения и самого вероятного класса с указанием самой вероятности
def plot_image(i, predictions_array, true_label, img):
  true_label, img = true_label[i], img[i]
  plt.grid(False)
  plt.xticks([])
  plt.yticks([])

  plt.imshow(img, cmap=plt.cm.binary)

  predicted_label = np.argmax(predictions_array)
  if predicted_label == true_label:
    color = 'blue'
  else:
    color = 'red'

  plt.xlabel("{} {:2.0f}% ({})".format(class_names[predicted_label],
                                100*np.max(predictions_array),
                                class_names[true_label]),
                                color=color)
```

Данная функция изображает столбчатую диаграмму по распознаванию класса изображения, где по оси X класс из 10 цифр (от 0 до 9), а по оси У вероятность отнесения изображения к классу.

```
# задаем функцию для построения столбчатой диаграммы разброса вероятности принадлежания объекта к классу
def plot_value_array(i, predictions_array, true_label):
  true_label = true_label[i]
  plt.grid(False)
  plt.xticks(range(10))
  plt.yticks([])
  thisplot = plt.bar(range(10), predictions_array, color="#777777")
  plt.ylim([0, 1])
  predicted_label = np.argmax(predictions_array)

  thisplot[predicted_label].set_color('red')
  thisplot[true_label].set_color('blue')
```

Далее воспользуемся двумя, описанными выше функциями, для анализа качества обучения нашей нейросети и изобразим по два графика для первых 15 тестовых изображений.

```
# проверим работу нейросети на нескольких примерах (для 15 изображений, выведенных в виде 5x3)
# выводим класс, к которому изображение отнесла сеть, точность предсказания и в
# скобках настоящий класс изображения
# если они совпадают, то текст синий, если нет - красный
num_rows = 5
num_cols = 3
num_images = num_rows*num_cols
plt.figure(figsize=(2*2*num_cols, 2*num_rows))
for i in range(num_images):
  plt.subplot(num_rows, 2*num_cols, 2*i+1)
  plot_image(i, predictions[i], test_labels, test_images)
  plt.subplot(num_rows, 2*num_cols, 2*i+2)
  plot_value_array(i, predictions[i], test_labels)
plt.tight_layout()
plt.show()
```



Из графиков видно, что модель самым точным образом предсказывает класс изображений, даже для тех случаев, когда даже человеку не всегда легко определить принадлежность написанного (например, нейросеть с точностью 82% определила, что на изображении цифра 5 (третья сверху строка, первое справа изображение), и лишь с 18% может отнести написанное к классу цифр 6).

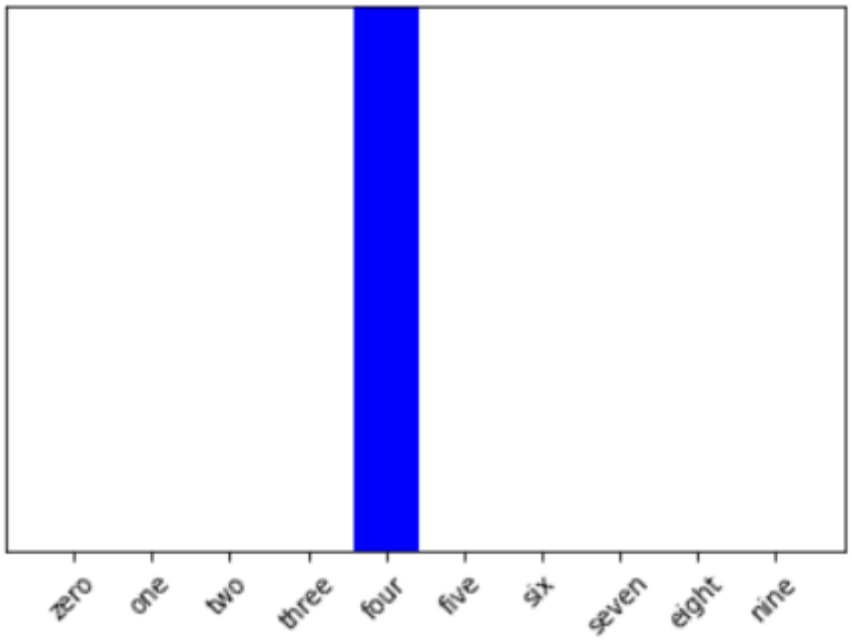
Также более детально можно рассмотреть данную диаграмму для одного изображения (в нашем случае произвольным образом выбрано 4-е).

```
# рассмотрим работу модели с 1 конкретным изображением (например 4м)
img = test_images[4]
# наша модель оптимизирована по целые батчи
# создаем новый батч, состоящий только из этого изображения
img = (np.expand_dims(img,0))
```

```
# ищем вероятности принадлежности к классам и выводим к какому классу изображение принадлежит наиболее вероятно
predictions_single = probability_model.predict(img)
print(predictions_single)
np.argmax(predictions_single[0])
```

```
[[9.8133510e-09 2.2411753e-12 4.3518917e-09 1.8670308e-11 9.9953067e-01
  4.7117536e-11 1.2601764e-09 1.5206474e-07 3.7508854e-09 4.6924566e-04]]
4
```

```
# строим диаграмму распределения вероятностей
plot_value_array(4, predictions_single[0], test_labels)
_ = plt.xticks(range(10), class_names, rotation=45)
plt.show()
```



Теперь устанавливаем библиотеку h5py, сохраняем нашу модель. После открываем сохранённую модель, проверяем её вид и точность, чтобы убедиться, что сохранение прошло успешно.

```
!pip install h5py
```

```
Requirement already satisfied: h5py in /usr/local/lib/python3.7/dist-packages (3.1.0)
Requirement already satisfied: numpy>=1.14.5 in /usr/local/lib/python3.7/dist-packages (from h5py) (1.19.5)
Requirement already satisfied: cached-property in /usr/local/lib/python3.7/dist-packages (from h5py) (1.5.2)
```

```
# сохраняем модель
model.save('my_model.h5')
```

```
# открываем сохраненную модель и смотрим ее вид
new_model = tf.keras.models.load_model('my_model.h5')
new_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 784)               0

 dense (Dense)               (None, 128)               100480

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0
_________________________________________________________________
```

```
# проверяем точность
loss, acc = new_model.evaluate(test_images, test_labels, verbose=2)
print('Restored model, accuracy: {:5.2f}%'.format(100 * acc))
```

```
313/313 - 0s - loss: 0.0790 - accuracy: 0.9776 - 492ms/epoch - 2ms/step
Restored model, accuracy: 97.76%
```

Вид и точность модели остались невредимы, следовательно, сохранение прошло без потерь данных, наша работа на этом считается оконченной.

Django

Django — это фреймворк для создания веб-приложений с помощью языка программирования Python.

Наш проект состоит из двух проектов pythonProject4, в котором содержатся файлы get.py и post.py, и DjangoPredict (основная часть работы).

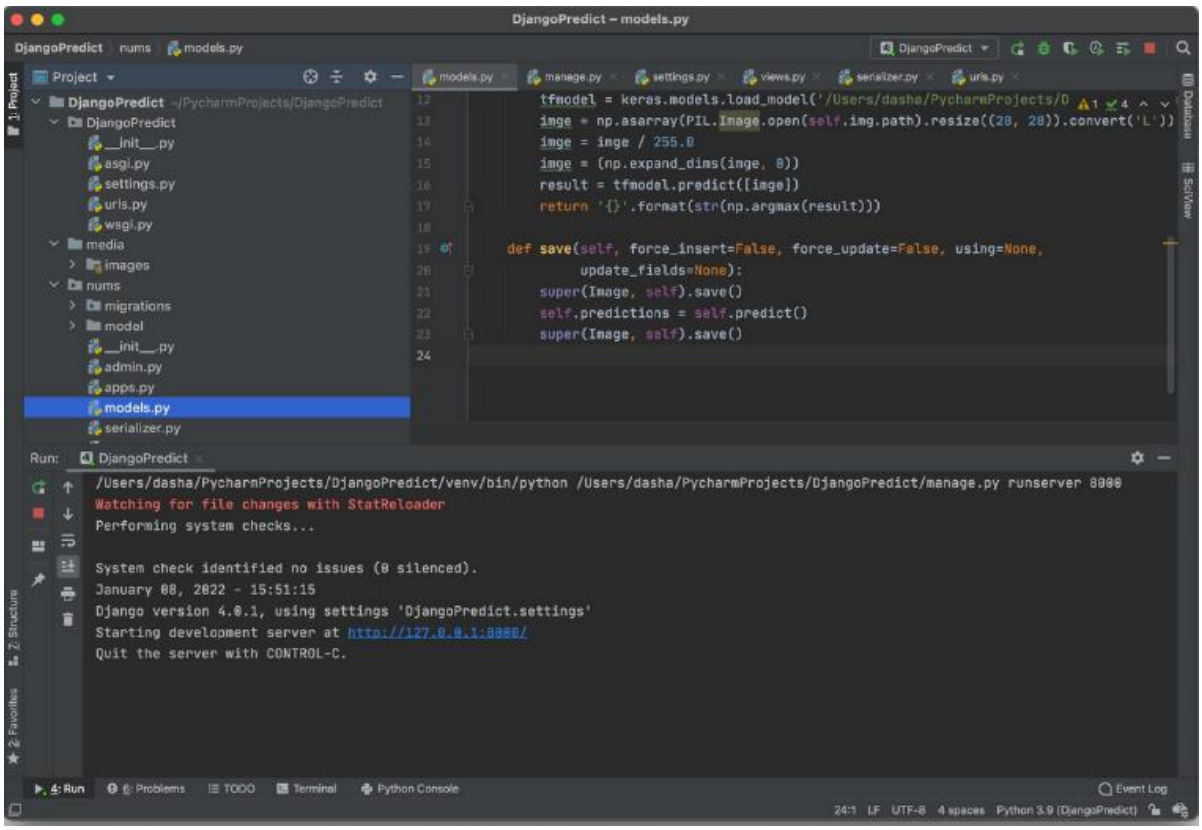
get.py post.py

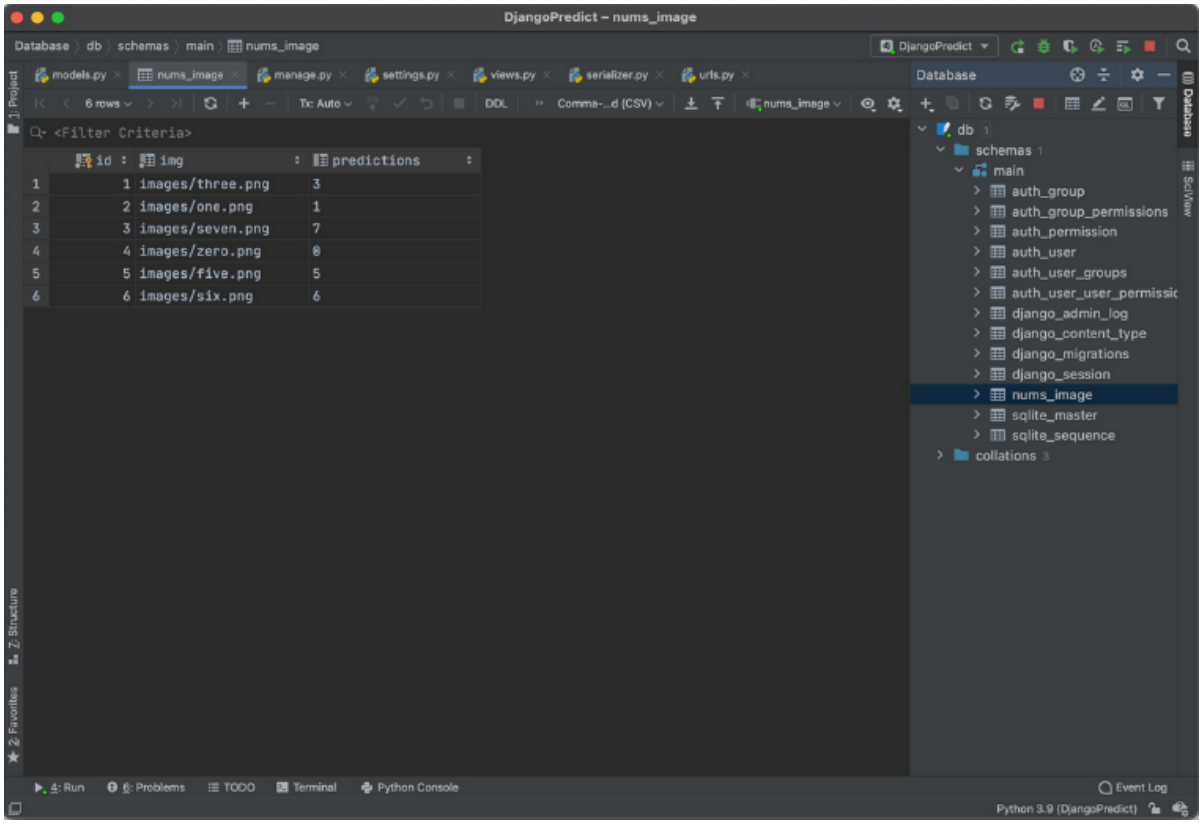
```python
import json
import requests
from PIL import Image
import io

# извлекаем данные об всех предсказаниях
r = requests.get('http://127.0.0.1:8000/predict/')
# преобразовываем данные в формат JSON
s = json.loads(r.text)
for i in range(len(s)):
    # для каждого из них извлекаем данные со страницы фотографии
    asa = requests.get('http://127.0.0.1:8000{}'.format(s[i]['img']))
    b_str = asa.content
    # открываем фотографию из байтовой строки
    image = Image.open(io.BytesIO(b_str))
    # сохряняем в папку
    image.save('/Users/dasha/PycharmProjects/pythonProject4/images/image{}.png'.format(i+1))

```

get.py post.py

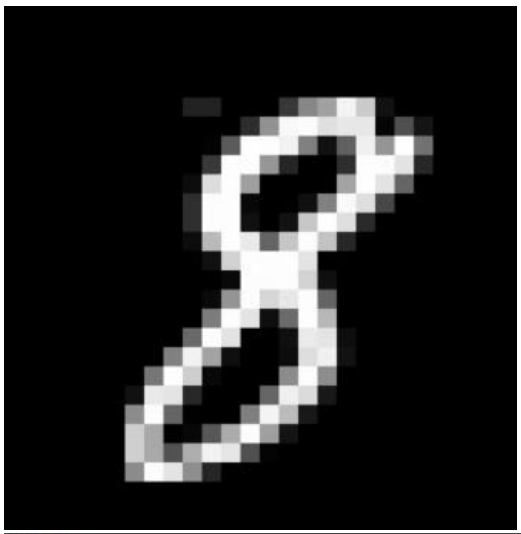
```python
import requests

# указываем путь к фотографии, для которой нужно предсказание
files = {'img': open(r'/Users/dasha/Desktop/six.png', 'rb')}
# отправляем запрос, чтобы получить предсказание
r = requests.post('http://127.0.0.1:8000/predict/', files=files)
# выводим статус запроса
print(r)

```

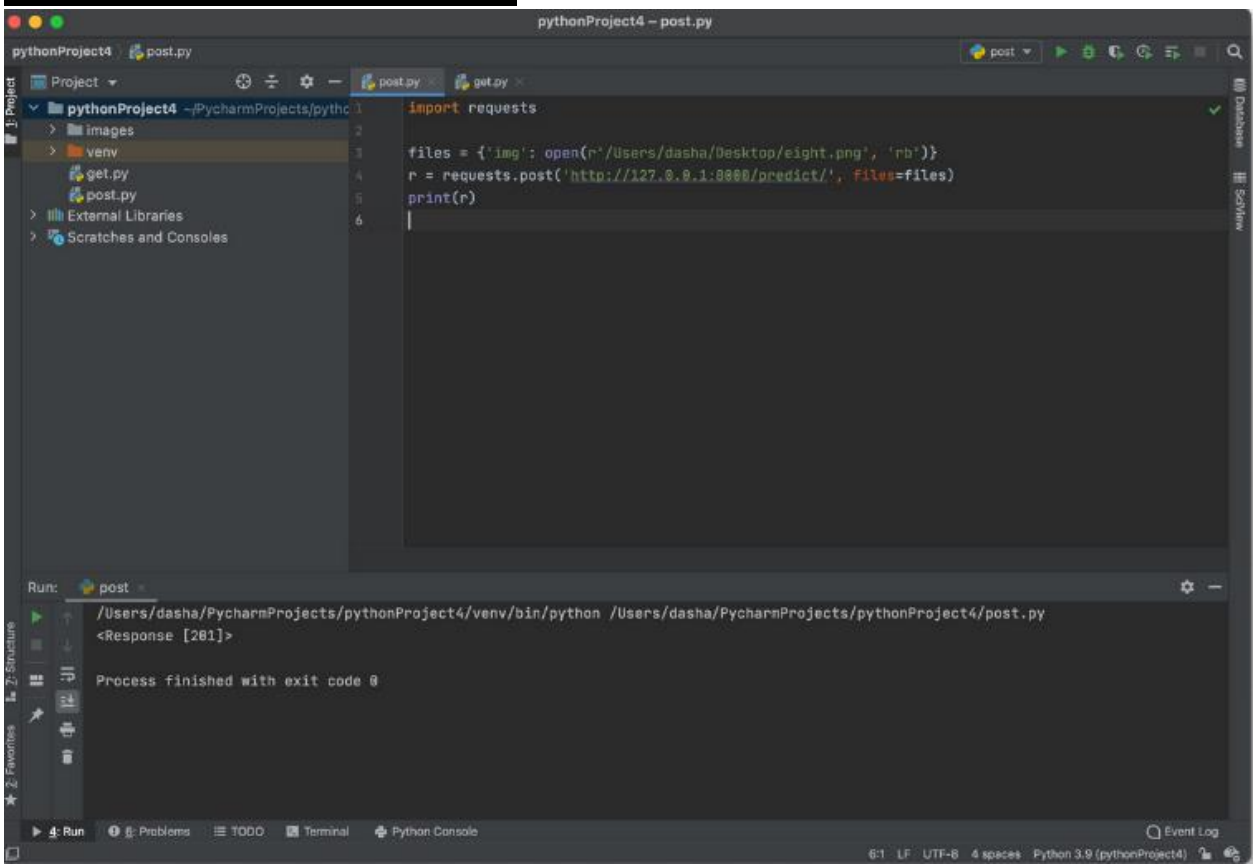
После написания полной работы мы приступаем к запуску нашего основного проекта DjangoPredict и смотрим, что на данный момент находится в базе данных. И видим, что на начальном этапе в базе данных находится 6 элементов (изображений).



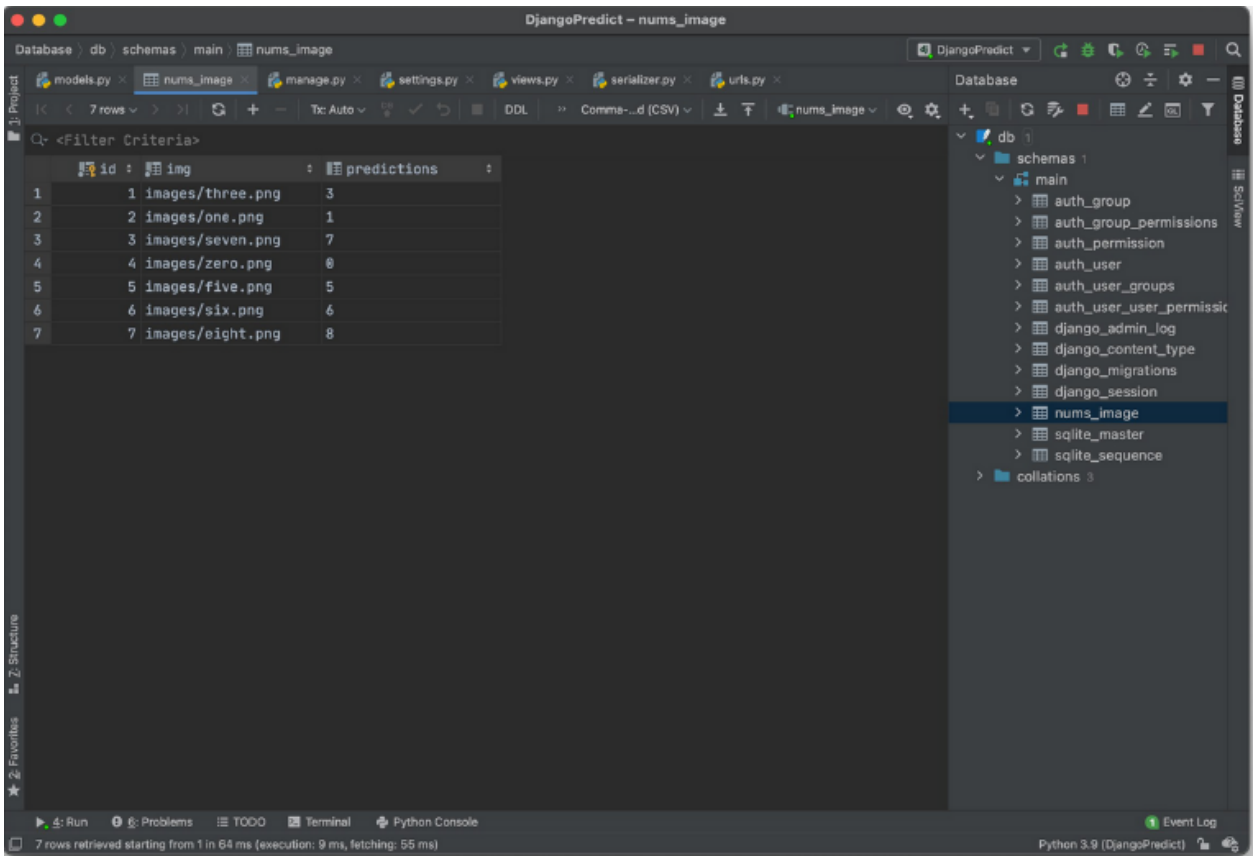


Затем мы наше изображение (см. ниже), для которого необходимо получить предсказание, записываем в post.py проекта pythonProject4 и запускаем файл.

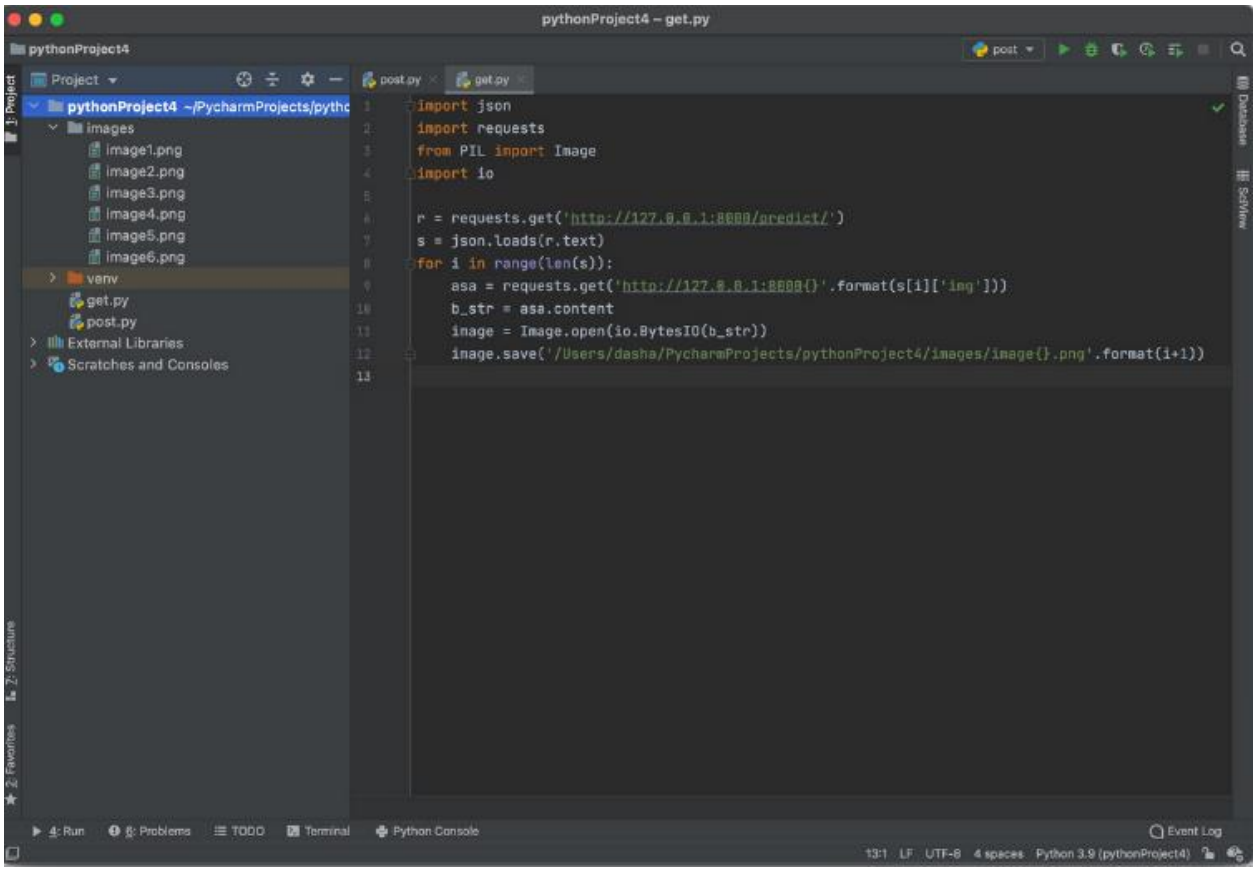




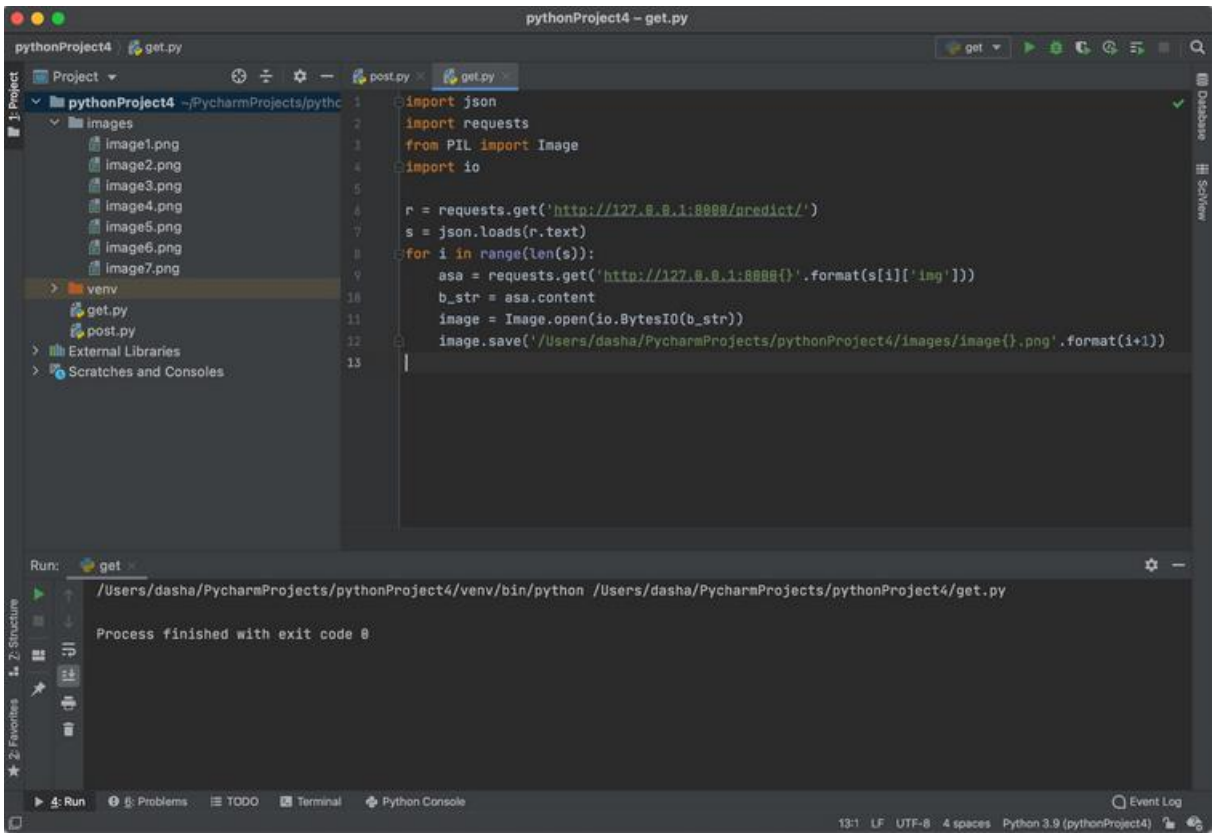
Как результат работы программы мы получаем статус запроса 201 и понимаем, что запрос успешно выполнен (всё удачно создалось). Также проверяем, что теперь находится в базе данных. Мы видим, что количество содержимого базы данных увеличилось на одну позицию (изображений), то есть добавление нашего элемента произошло. При этом предсказание того, что содержится на изображении верное (содержимое – цифра 8, предсказание – 8).



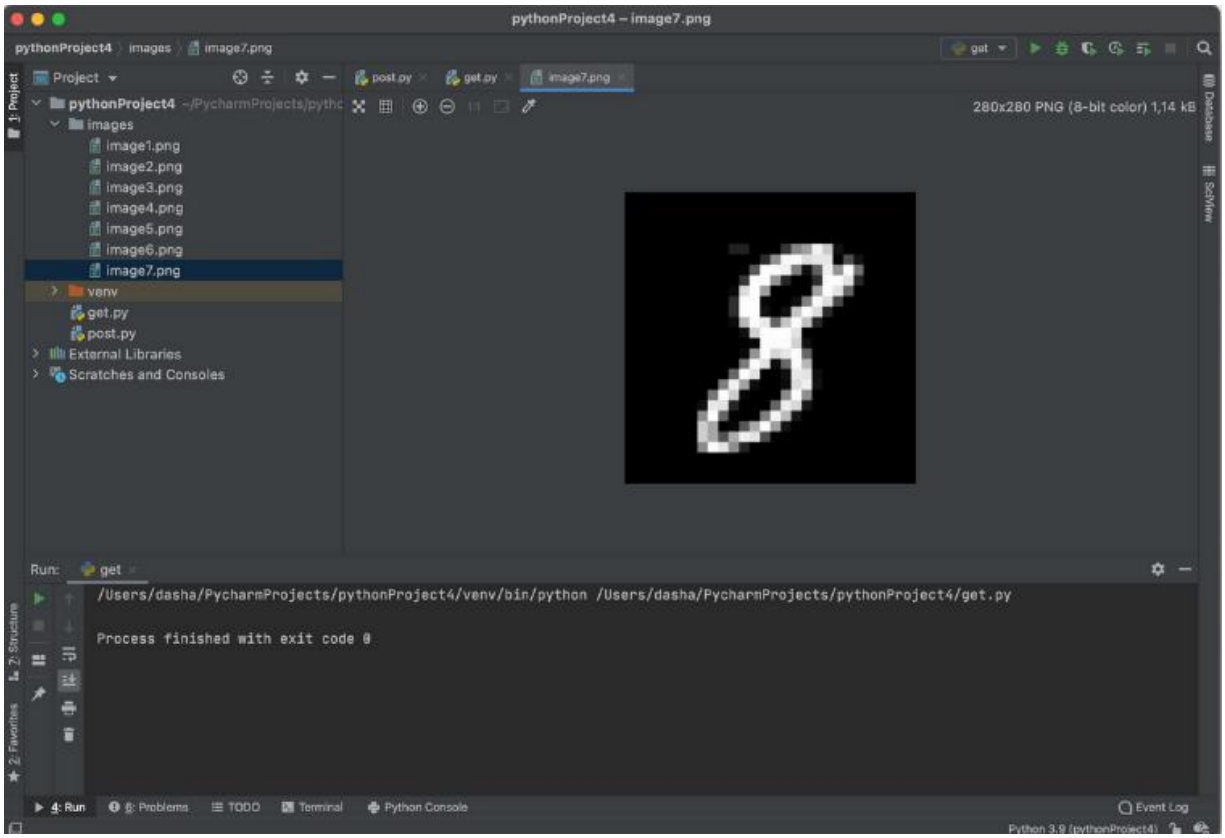
Переходим в программу get.py, видим, что в папке images находится 6 изображений.



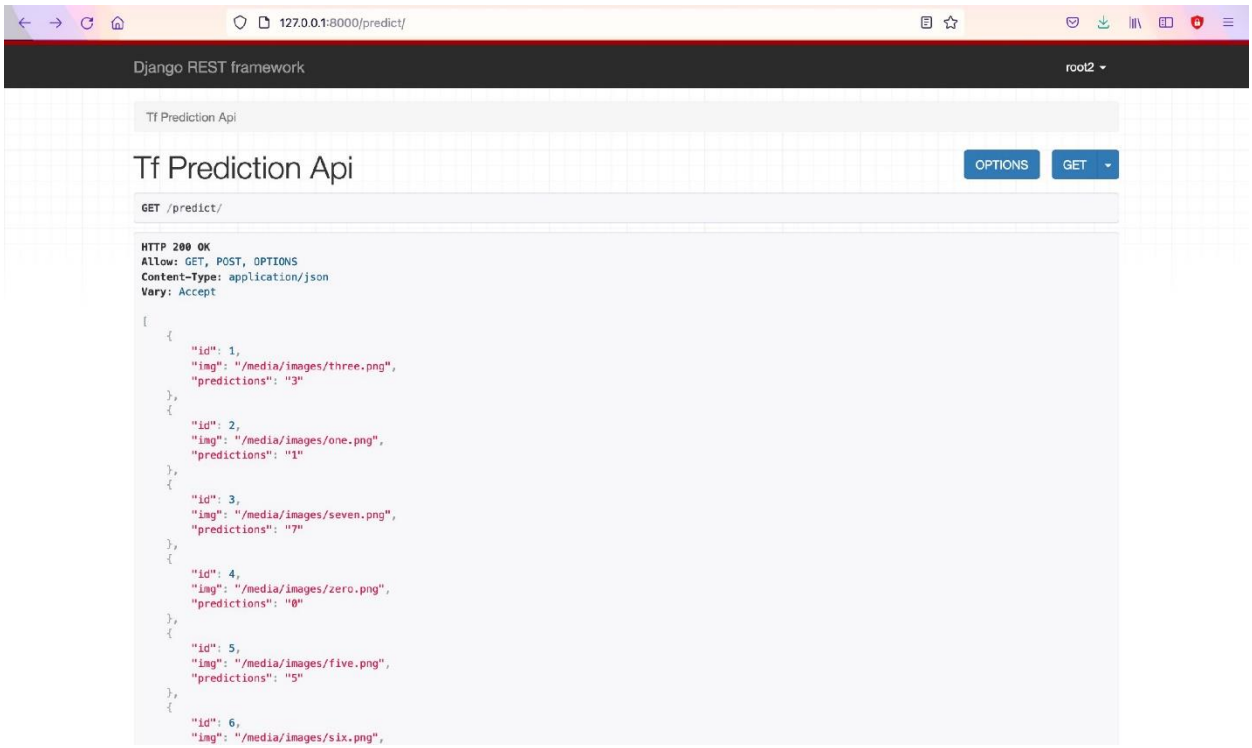
После запуска в папку images добавляется также седьмое изображение.

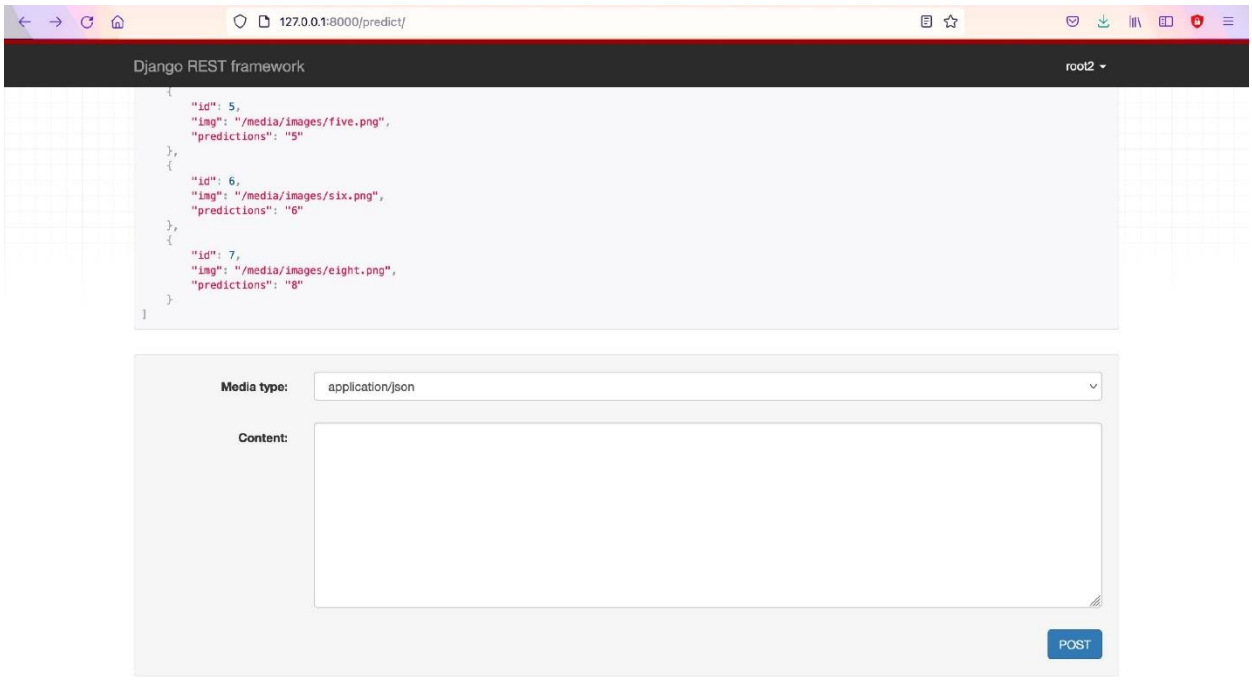


Для того, чтобы убедиться, что отработка прошла успешно, откроем добавленное изображение и проверим, что это именно наше исходное изображение. При выполнении этих действий мы убеждаемся в верности работы нашего проекта.



Также можно увидеть, как выглядит созданный Api в браузере.





Более подробные комментарии ко всему проекту можно увидеть в коде отдельных программ (файлы с расширением .py).