Нейронные сети, регрессия

С.И.Хашин
http://math.ivanovo.ac.ru/dalgebra/Khashin/index.html

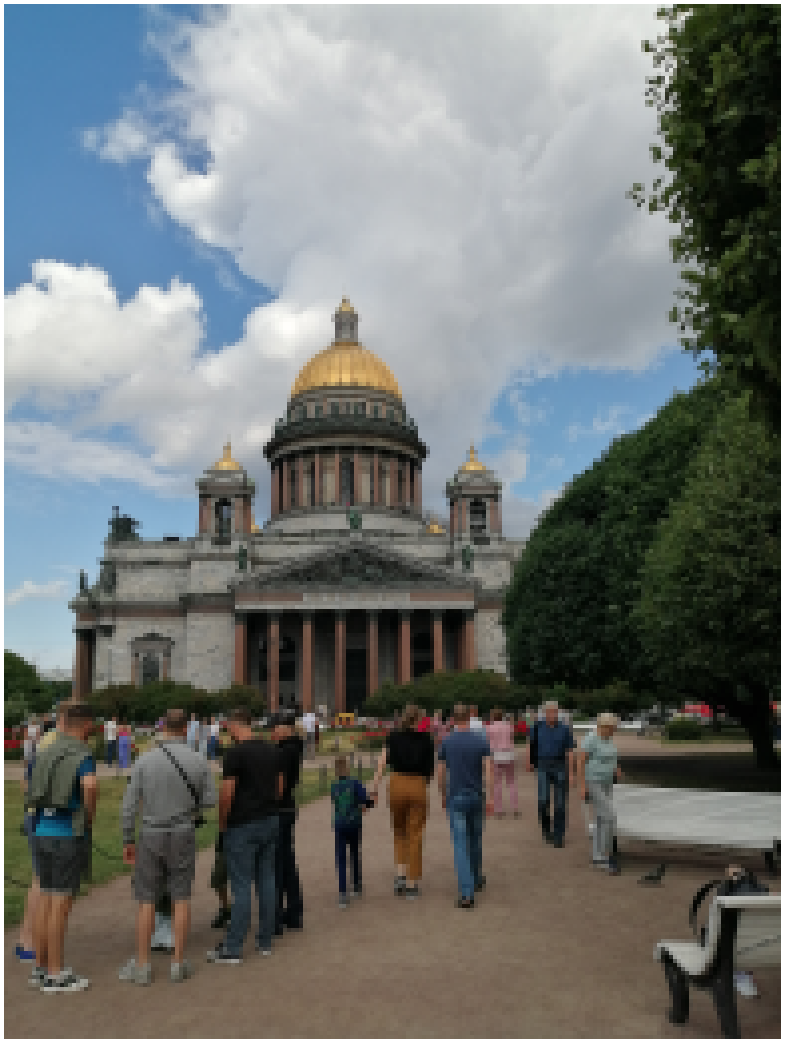
Ивановский государственный университет

.

Иваново-2023

План

Картинка



Чтение изображения

Заголовок:

```
import numpy as np
from PIL import Image
```

Нам требуется изображение, например SpB.jpg. Как его загрузить в память:

```
img = Image.open('SpB.jpg')
a = np.array(img)
print(a.shape)
```

К серой матрице

Из RGB-матрицы в оттенки серого:

$$Y = 0.299\,R + 0.587\,G + 0.114\,B\ .$$

```
def to_gray(a):  # из 3-мерного одна серая матрица
  y=np.round(0.299*a[:,:,0]+0.587*a[:,:,1]+0.114*a[:,:,2])
  return np.array(y, dtype=int)
```

Что получится:

```
a = to_gray(a)
print('a:', a.shape)
```

Создание обучающей матрицы

Возвращаем матрицу из 3-х столбцов. step - шаг по x,y.

```
def array_teach_matriz(a,step=1):
  my, mx = a.shape
  ky, kx = my//step, mx//step
  res = np.zeros((ky*kx, 3), dtype = int)
  idx = 0
  for y in range(0, my, step):
    for x in range(0, mx, step):
      res[idx] = a[y,x], y//step, x//step
      idx += 1
  return res
```

Что получится:

```
csv = array_teach_matriz(a, 10)
np.savetxt('01.csv', csv, fmt='%4d', delimiter=',',
           header='a, y, x')
print('csv:', csv.shape)
```

Заголовок файла

```
import numpy as np
import tensorflow as tf
import time
import tkinter as tk

np.set_printoptions(precision=4, linewidth=140,
      suppress=True)
np.random.seed(777)
```

Чтение и нормализация

```
data = np.loadtxt('02.csv', skiprows=1, delimiter=',')
np.random.shuffle(data)
Y = data[:,0]
X = data[:,1:]
# Normalization
Y /= 255 # теперь Y от 0 до 1

maxY = int(max(X[:,0]))
maxX = int(max(X[:,1]))
X[:, 0] /=  maxY # теперь X от 0 до 1
X[:, 1] /=  maxX
X-= 0.5 # теперь X от -0.5 до 0.5
```

Создание изображения

Из обучающих данных (Y,X) восстановим матрицу изображения.
Данные предполагаются нормализованными.

```
def YX_pict(Y, X, my, mx):
  X += 0.5
  pic = np.zeros((my+1, mx+1), dtype=np.uint8)
  #print('pic:', pic.shape)
  for i in range(len(Y)):
    y1 = round(X[i,0]*my)
    x1 = round(X[i,1]*mx)
    pic[y1, x1] = Y[i]*255
  X -= 0.5
  return pic
```

Вывод на экран

Вывод матрицы a на экран в виде серой картинки.

```
def plot_e(c_tk, a):     # Вывод массива a на экран
  cGray = 1+256+256*256
  my, mx = a.shape
  for y in range(my):
    for x in range(mx):
      c_tk.create_rectangle(x,y, x, y + 1,\
          fill="#"+"{:06x}".format(int(a[y][x])*cGray),
          width = 0)
  c_tk.pack();
  c_tk.update();
```

Вывод на экран

Вывод матрицы a на экран в виде серой картинки через tkinter.

```
root = tk.Tk()
c_tk = tk.Canvas(root, height=maxY, width=maxX,
                 bg='white')
plot_e(c_tk, YX_pict(Y, X, maxY, maxX))
```

Модель

```
N1 = 20  # нейронов в 1 слое
N2 = 10  # нейронов в 2 слое

model = tf.keras.Sequential(
  [
    tf.keras.layers.Dense(N1, activation = tf.nn.relu,
                          input_shape=(2,)),
    tf.keras.layers.Dense(N2, activation = tf.nn.tanh),
    tf.keras.layers.Dense(1, activation="sigmoid"),
  ]
)
#model.compile(optimizer='rmsprop', loss='mse', metrics=['m
model.compile(
    optimizer = tf.keras.optimizers.Adam(learning_rate=0.2)
    loss='mse', metrics=['mae'])
print('model:', model.summary())
```

model.summary()

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 20)                60
_________________________________________________________________
dense_1 (Dense)              (None, 10)                210
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 11
=================================================================
Total params: 281
```

Обучение

```
nEpochs = 100  # количество эпох
tm = time.time()
history = model.fit(x=X, y=Y, epochs=nEpochs,
                    #batch_size=len(Y))
                    batch_size=1000)
tm = (time.time() - tm) / nEpochs
res = model.evaluate(X, Y)
print('res=', res)
Y1 = model.predict(X).flatten()
plot_e(c_tk, YX_pict(Y1, X_global, maxY, maxX))
```

callback

```
maxX = maxY = 0           # глобальные переменные
X_global = np.arange(1)
# Отобразим прогресс тренировки
class my_callback(tf.keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs):
    global X_global, maxY, maxX
    if epoch % 100 == 0:
      #print('** callback: ', epoch)
      Y1 = self.model.predict(X_global).flatten()
      plot_e(c_tk, YX_pict(Y1, X_global, maxY, maxX))
  ...

history = model.fit(x=X, y=Y, epochs=nEpochs,
          callbacks=[my_callback()],
          batch_size=1000)
```