МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ
ВОЛГОГРАДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
КАФЕДРА «ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ АВТОМАТИЗИРОВАННЫХ СИСТЕМ »

МЕТОДЫ РАСПОЗНАВАНИЯ ОБРАЗОВ

Методические указания

Волгоград
2018

УДК 004.93'1 (075)

Р е ц е н з е н т

зав. кафедрой «Электронно-вычислительные машины и системы»
канд. техн. наук *А. Е. Андреев*

Печатается по решению редакционно-издательского совета
Волгоградского государственного технического университета

Методы распознавания образов : метод. указания / сост.: Н. П. Садовникова, Д. С. Донченко ; ВолгГТУ. – Волгоград, 2018. – 16 с.

Даются основные понятия, связанные с теорией распознавания образов; разбираются примеры решения задач распознавания с помощью языка R. Приводятся варианты.

Предназначены для студентов очной формы обучения направления 09.04.04 «Программная инженерия» и 09.04.01 «Информатика и вычислительная техника».

Содержание

1. Основные понятия теории распознавания образов

Распознавание образов – это наука о методах и алгоритмах классификации объектов различной природы.

Получение символического описания изображений представляет собой задачу перехода от набора простейших признаков изображения, таких, как значения яркости, контурные точки или параметры текстуры, к значительно меньшему набору средств описания, которые могут служить в качестве исходных данных для последующей семантической интерпретации. Типичными графическими символами являются цепочки контурных точек, образующих границу объекта, связанные области постоянной яркости, цвета или текстуры и элементарные фигуры, такие, как прямоугольники, окружности, треугольники. Основной этап при формировании символического описания изображения по массиву элементов или набору простейших признаков заключается в определении геометрических соотношений и связанности между элементами, относительно которых предполагается, что они принадлежат одному классу.

Процесс распознавания можно разделить на шесть основных этапов: снятие информации, предварительная обработка информации, сегментация, описание, распознавание, интерпретация [1].

Снятие информации представляет собой процесс получения визуального изображения.

Предварительная обработка информации заключается в использовании таких методов как понижение шума или улучшение изображения отдельных деталей.

Сегментация — процесс выделения на изображении интересующих объектов. При описании определяются характерные параметры (например,

размеры или форма), необходимые для выделения требуемого объекта на фоне других.

Распознавание представляет собой процесс идентификации объектов. Интерпретация выявляет принадлежность к группе распознаваемых объектов.

Можно выделить следующие наиболее важные направления развития интеллектуальных систем (т.е. систем, решающих задачи, традиционно относимые к интеллектуальной сфере), в которых широко используются методы распознавания образов:

- распознавание символов (печатного и рукописного текстов, банковских чеков и денежных купюр и т.д.);
- распознавание изображений, полученных в различных частотных диапазонах (оптическом, инфракрасном, радиочастотном, звуковом) и анализ сцен;
- распознавание речи;
- медицинская диагностика;
- системы безопасности;
- классификация, кластеризация и поиск в базах данных и знаний.

2. Методы распознавания на основе нейронных сетей

Несмотря на разнообразие методов используемых для решения задач распознавания, нейросетевые модели являются на сегодняшний день самыми распространенными. Связано это с целым набором преимуществ нейронных сетей (НС). Для НС не требуется вручную определять параметры модели, настройка производится автоматически в процессе обучения модели на тренировочных примерах. Архитектура НС и процедуры обучения позволяют выполнить гибкую настройку на конкретную решаемую задачу. Для большинст-

ва НС процедура обучения является эвристическим алгоритмом, что, с одной стороны, обеспечивает приемлемость получаемых решений, а с другой стороны, не требует непомерных вычислительных ресурсов [2]. Нейронные сети ни налагают каких-либо ограничений на тренировочную выборку.

Но несмотря на все достоинства, применение НС к изображениям требует специальных усилий. Изображение должно быть предобработано – приведено к некоторым стандартным условиям. Кроме того, выбор начального представления изображения (это могут быть, например, частотные коэффициенты, главные компоненты, вейвлетные коэффициенты, моменты и т.п.) является отдельной обширной темой, которая не рассматривается в рамках данной работы.

2.1. Сверточные нейронные сети

Свёрточная НС (СНС, Convolutional Neural Network) обеспечивает частичную устойчивость к изменениям масштаба, смещениям, поворотам, смене ракурса и прочим искажениям.

Нейронные сети являются по существу математическими моделями для решения проблемы оптимизации. Нейрон — основная вычислительная единица нейронных сетей. Нейрон принимает вход (скажем, x), выполняет некоторые вычисления на нем (например, умножение его на некую переменную w и прибавление другой переменной b), чтобы получить значение ($z = wx + b$). Это значение передается нелинейной функции, называемой функцией активации (f), для получения конечного выхода (активации) нейрона. Существует множество видов функций активации. Одной из популярных функций активации является Сигмоида, которая имеет следующую формулу:

$$y = \frac{1}{1 + e^{-z}}$$

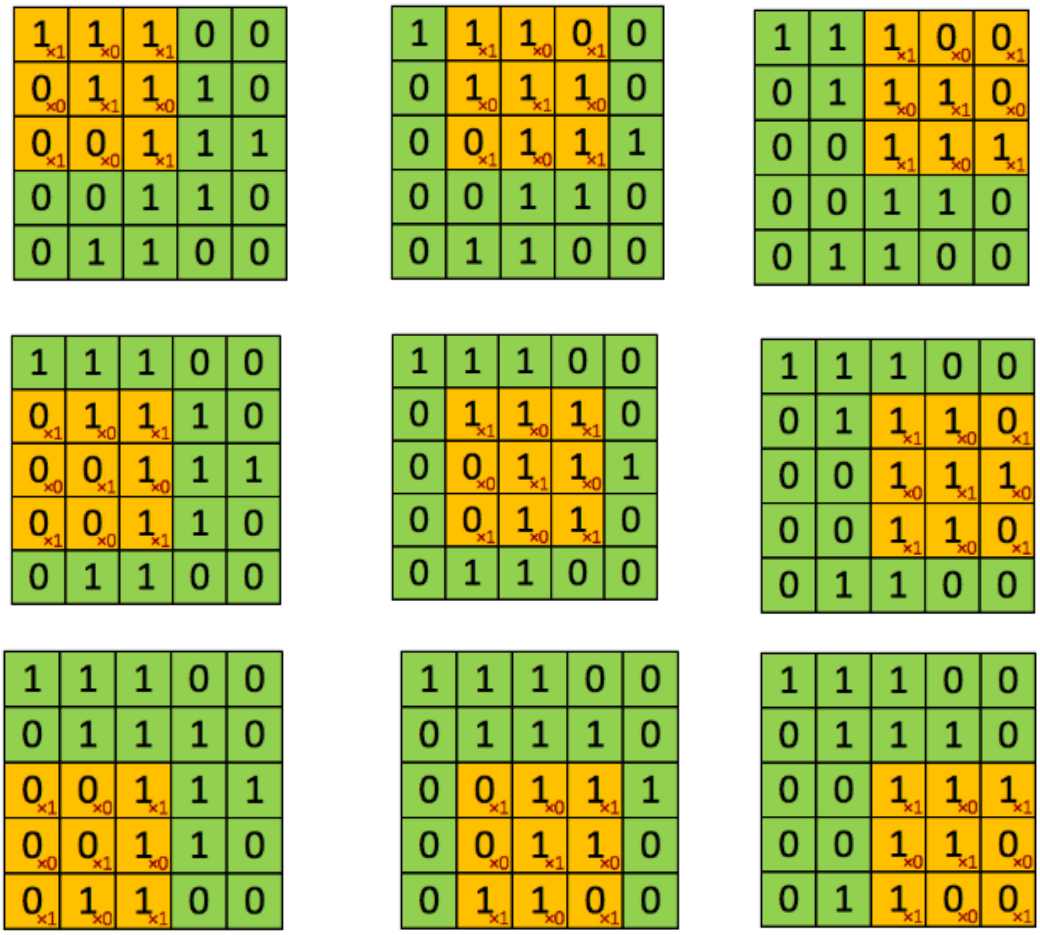
Нейрон, который использует сигмоидную функцию как функцию активации, будет называться сигмовидным нейроном. Нейроны могут иметь и другие названия в зависимости от их функции активации (например, RELU, TanH и т. д.). Один нейрон может быть связан с несколькими нейронами. Несколько связанных между собой нейронов на одном уровне образуют слой. Как правило, все нейроны в одном слое выполняют схожие математические операции, и получают свое название в зависимости от типа операции (за исключением входных и выходных слоев, поскольку они выполняют небольшие математические операции).

2.1.1 Сверточный слой

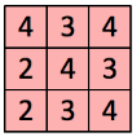
Свертка - это математическая операция, которая используется в одиночной обработке для фильтрации сигналов, поиска шаблонов в сигналах и т. Д. В сверточном слое все нейроны применяют операцию свертки ко входам, поэтому они называются сверточными нейронами. Наиболее важным параметром в сверточном нейроне является размер фильтра (например, слой с размером фильтра 3 * 3). Кроме того, предположим, что вход, который подается на сверточный нейрон, представляет собой входное изображение размером 5 * 5.

Сверточный слой для каждого участка изображения размером 3 * 3 произведет вычисление функции свертки с указанным фильтром. Результатом каждой операции вычисления свертки будет 1 число.

Сверточный фильтр равномерно сдвигается по всему входному изображению, чтобы рассчитать результаты свертки всего изображения, как показано на схеме ниже:



В приведенном примере окно фильтра сдвигалось на 1 пиксель на каждой итерации. В некоторых случаях, шаг может быть увеличен. Результатом вычисления будет матрица с размерами фильтра:



2.1.2 Полносвязный слой

Если каждый нейрон в слое получает вход от всех нейронов в предыдущем слое, то этот слой называется полностью связанным слоем. Цель слоя – классификация, моделирует сложную нелинейную функцию, оптимизируя которую, улучшается качество распознавания.

3 Пример решения задачи распознавания образов

В рамках данной задачи мы рассмотрим простую сверточную нейронную сеть, которая может быть использована для множества задач распознавания образов. Для построения нейронной сети будет использоваться библиотека TensorFlow и язык программирования Python.

Перед началом работы, на системе должны быть установлены:

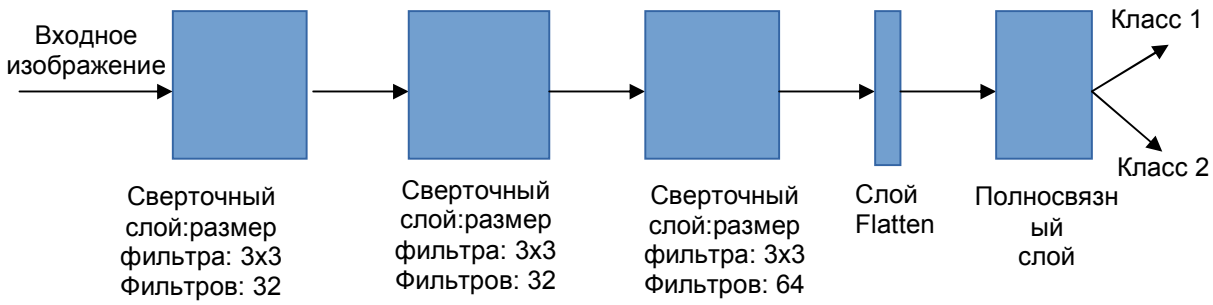
1. Python (https://www.python.org/)
2. Одна из систем управления пакетами Python (например, pip — https://pypi.org/project/pip/)
3. (Опционально) Среда разработки на Python (например, PyCharm Community — https://www.jetbrains.com/pycharm/)
4. Библиотека OpenCV для работы с изображениями (https://docs.opencv.org/)
5. Библиотека TensorFlow (https://www.tensorflow.org/install/)

Также, перед началом работы следует зарегистрироваться на сайте https://www.kaggle.com для скачивания датасетов для обучения нейросети.

3.1 Структура нейронной сети

Данная нейронная сеть имеет простую структуру, и для ее тренировки можно использовать CPU. Большинство нейронных сетей, использующихся для реальных задач распознавания образов, имеют более сложную структуру, и для их обучения используют GPU (т. к. обучение на CPU занимает очень

длительное время). Во время обучения изображения обоих классов (собак / кошек) подаются на сверточный слой, за которым следуют еще два сверточных слоя. После сверточных слоев мы сглаживаем выход и добавляем в него два полностью связанных слоя. Второй полносвязный слой имеет только два выхода, которые представляют вероятность того, что изображение является объектом класса 1 или класса 2.



Слой flatten служит для преобразования результата сверточного слоя в одноразмерный тензор для последующей классификации.

3.2 Входные данные

В качестве примера возьмем готовый датасет из 2000 изображений кошек и собак, которых и будет распознавать нейронная сеть:

https://www.kaggle.com/c/dogs-vs-cats/data (необходимо скачать архив train.zip).

Как правило, входные данные делятся на 3 части:

Данные для обучения (60% датасета)

Данные для валидации (20% датасета)

Данные для тестирования (20% датасета)

3.3 Реализация нейронной сети с использованием TensorFlow

1. Производим импорт библиотек TensorFlow

```
import tensorflow as tf
from tensorflow.data import *
```

2. Считываем входные данные и разбиваем их на три части:

```
classes = ['dogs', 'cats']
num_classes = len(classes)
train_path='training_data'
# validation split
validation_size = 0.2
# batch size
batch_size = 16
data = tf.data.dataset.read_train_sets(train_path, img_size, classes, validation_size=validation_size)
```

3. Добавим функции для вычисления случайных значений весов

```
def create_weights(shape):
    return tf.Variable(tf.truncated_normal(shape, stddev=0.05))
def create_biases(size):
    return tf.Variable(tf.constant(0.05, shape=[size]))
```

4. Построение сверточных слоев в TensorFlow

Для построение сверточного слоя в TensorFlow можно использовать функцию tf.nn.conv2d. Она принимает следующие параметры:

input — выход (активация) из предыдущего слоя. Это должен быть четырехмерный тензор.

filter — обучаемые переменные, определяющие фильтр. На первом шаге эти веса будут определены случайным образом. Это также четырехмерный тензор, чья конкретная форма предопределена как часть сетевого дизайна.

strides — определяет шаг перемещения фильтра при выполнении свертки. В этой функции он должен быть тензором размером > = 4. Мы будем использовать strides = 1.

padding = SAME означает, что мы будем вводить ввод таким образом, чтобы выходные значения x, y были такими же, как и для ввода.

Добавим функцию для построения сверточного слоя:

```
def create_convolutional_layer(input,
                num_input_channels,
                conv_filter_size,
                num_filters):
```

We shall define the weights that will be trained using create_weights function.

```
weights = create_weights(shape=[conv_filter_size, conv_filter_size, num_input_channels, num_filters])
```

```
    ## We create biases using the create_biases function. These are also trained.
    biases = create_biases(num_filters)
    ## Creating the convolutional layer
    layer = tf.nn.conv2d(input=input,
                         filter=weights,
                         strides=[1, 1, 1, 1],
                         padding='SAME')
     layer += biases
     ## We shall be using max-pooling.
    layer = tf.nn.max_pool(value=layer,
                           ksize=[1, 2, 2, 1],
                           strides=[1, 2, 2, 1],
                           padding='SAME')
    ## Output of pooling is fed to Relu which is the activation function for us.
    layer = tf.nn.relu(layer)
     return layer
```

5. Слой Flattening

Выход сверточного слоя представляет собой многомерный тензор. Для классификации необходим одномерный тензор. Это выполняется в слое Flattening. Мы просто используем операцию reshape для создания одномерного тензора:

```
def create_flatten_layer(layer):
    layer_shape = layer.get_shape()
    num_features = layer_shape[1:4].num_elements()
    layer = tf.reshape(layer, [-1, num_features])
     return layer
```

6. Определение функции для создания полносвязного слоя

Объявляем веса и смещения как случайные нормальные распределения. В полностью подключенном слое для каждого входа выполняется стандартная операция z = wx + b.

```
def create_fc_layer(input,
            num_inputs,
            num_outputs,
            use_relu=True):
       #Let's define trainable weights and biases.
    weights = create_weights(shape=[num_inputs, num_outputs])
    biases = create_biases(num_outputs)
     layer = tf.matmul(input, weights) + biases
    if use_relu:
       layer = tf.nn.relu(layer)
     return layer
```

7. Формирование placeholder, который будет содержать входные обучающие изображения

Все входные изображения считываются в файле dataset.py и изменяются до 128 x 128 x 3.

x = tf.placeholder (tf.float32, shape = [None, img_size, img_size, num_channels], name = 'x')

y_true = tf.placeholder (tf.float32, shape = [None, num_classes], name = 'y_true') y_true_cls = tf.argmax (y_true, dimension = 1)

8. Создание структуры нейронной сети

```
layer_conv1 = create_convolutional_layer(input=x,
              num_input_channels=num_channels,
              conv_filter_size=filter_size_conv1,
              num_filters=num_filters_conv1)
 layer_conv2 = create_convolutional_layer(input=layer_conv1,
              num_input_channels=num_filters_conv1,
              conv_filter_size=filter_size_conv2,
              num_filters=num_filters_conv2)
 layer_conv3= create_convolutional_layer(input=layer_conv2,
              num_input_channels=num_filters_conv2,
              conv_filter_size=filter_size_conv3,
              num_filters=num_filters_conv3)
layer_flat = create_flatten_layer(layer_conv3)
 layer_fc1 = create_fc_layer(input=layer_flat,
                  num_inputs=layer_flat.get_shape()[1:4].num_elements(),
                  num_outputs=fc_layer_size,
                  use_relu=True)
 layer_fc2 = create_fc_layer(input=layer_fc1,
                  num_inputs=fc_layer_size,
                  num_outputs=num_classes,
                  use_relu=False)
```

9. Оценка вероятности для каждого класса (применяем функцию softmax к выходам полносвязного слоя)

y_pred = tf.nn.softmax (layer_fc2, name = "y_pred")

y_pred содержит прогнозируемую вероятность каждого класса для каждого входного изображения. Класс, имеющий более высокую вероятность, представляет собой предсказание сети.

Cost-функция (функция оптимизации издержек) вычисляется с помощью функции reduce_mean.

```
y_pred_cls = tf.argmax (y_pred, dimension = 1)
cross_entropy = tf.nn.softmax_cross_entropy_with_logits(logits=layer_fc2, labels=y_true)
cost = tf.reduce_mean(cross_entropy)
```

10. Расчет градиента и оптимизация веса

```
def train(num_iteration):
    global total_iterations
        for i in range(total_iterations,
                total_iterations + num_iteration):
         x_batch, y_true_batch, _, cls_batch = data.train.next_batch(batch_size)
        x_valid_batch, y_valid_batch, _, valid_cls_batch = data.valid.next_batch(batch_size)
              feed_dict_tr = {x: x_batch,
                      y_true: y_true_batch}
        feed_dict_val = {x: x_valid_batch,
                         y_true: y_valid_batch}
         session.run(optimizer, feed_dict=feed_dict_tr)
         if i % int(data.train.num_examples/batch_size) == 0:
           val_loss = session.run(cost, feed_dict=feed_dict_val)
           epoch = int(i / int(data.train.num_examples/batch_size))
                   show_progress(epoch, feed_dict_tr, feed_dict_val, val_loss)
           saver.save(session, 'dogs-cats-model')
       total_iterations += num_iteration
```

11. Распознавание

```
image = cv2.imread(filename)
# Resizing the image to our desired size and
# preprocessing will be done exactly as done during training
image = cv2.resize(image, (image_size, image_size), cv2.INTER_LINEAR)
images.append(image)
images = np.array(images, dtype=np.uint8)
images = images.astype('float32')
images = np.multiply(images, 1.0/255.0)
#The input to the network is of shape [None image_size image_size num_channels]. Hence we reshape.
x_batch = images.reshape(1, image_size,image_size,num_channels)
  graph = tf.get_default_graph()
 y_pred = graph.get_tensor_by_name("y_pred:0")
 ## Let's feed the images to the input placeholders
x= graph.get_tensor_by_name("x:0")
y_true = graph.get_tensor_by_name("y_true:0")
y_test_images = np.zeros((1, 2))
```

```
 feed_dict_testing = {x: x_batch, y_true: y_test_images}
result=sess.run(y_pred, feed_dict=feed_dict_testing)
```

12. Запустить новый образ собаки / кошки

```
python predict.py test_dog.jpg
[[ 0.99398661  0.00601341]]
```

4. Задача для самостоятельного решения

Необходимо обучить созданную нейронную сеть на новом наборе данных.

1) Изменить исходный набор данных, например, увеличив их объем за счет поворота, увеличения масштаба изображения и пр.

2) Использовать произвольный набор данных для новой задачи распознания.

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. Горелик А.Л., Скрипкин В.А. Методы распознавания//– М: Высшая школа, 1984, – 208 с.

2. Тропченко А.А., Тропченко А.Ю. Методы вторичной обработки и распознавания изображений. Учебное пособие. – СПб: Университет ИТМО, 2015. – 215 с.

Учебное издание

Составители:

Наталья Петровна **Садовникова**
Дмитрий Сергеевич **Донченко**

МЕТОДЫ РАСПОЗНАВАНИЯ ОБРАЗОВ

Методические указания

Темплан 2018 г. (учебно-методическая литература). Поз. № 216.
Подписано в печать 31.08.2018. Формат 60x84 1/16. Бумага офсетная.
Гарнитура Times. Печать офсетная. Усл. печ. л. 0,93.
Тираж 10 экз. Заказ

Волгоградский государственный технический университет.
400005, г. Волгоград, просп. им. В. И. Ленина, 28, корп. 1.

Отпечатано в типографии ИУНЛ ВолгГТУ.
400005, г. Волгоград, просп. им. В. И. Ленина, 28, корп. 7.