ГЛАВА 6

Усовершенствованные варианты архитектуры нейронных сетей

В этой главе мы рассмотрим ряд усовершенствованных методик глубокого обучения, используемых в наши дни. К числу ключевых направлений, которые в последнее время привлекли к себе большое внимание, относятся сегментация изображений, а также локализация и обнаружение объектов. Сегментация изображений играет важную роль при выявлении различных заболеваний и патологий посредством обработки медицинских изображений. Она также находит применение в таких отраслях, как авиастроение, машиностроение и многие другие, где оно может быть использовано для выявления всевозможных дефектов, таких как трещины или другие механические разрушения в оборудовании. Изображения ночного неба могут сегментироваться для обнаружения неизвестных галактик, звезд и планет. Обнаружение и локализация объектов широко используется в местах, требующих постоянного автоматизированного наблюдения за текущей активностью, например в супермаркетах, магазинах, на промышленных предприятиях и т.п. Кроме того, эту технологию можно применять для подсчета количества объектов и людей в определенном месте, а также в системах управления дорожным движением.

Глава начинается с обсуждения традиционных методов сегментации изображений, чтобы вы могли оценить, насколько нейронные сети отличаются от своих традиционных аналогов. После этого мы рассмотрим методы обнаружения и локализации объектов, а затем обсудим генеративно-состязательные сети, которые в последнее время стали очень популярными благодаря возможностям их использования в качестве генеративных моделей для создания синтетических данных. Последние могут использоваться в целях тренировки моделей и логического вывода в тех случаях, ког-

да имеется ограниченный объем данных или их получение связано с большими расходами. Генеративные модели также могут применяться для переноса стиля из одной предметной области в другую. Наконец, мы завершим наше обсуждение рядом рекомендаций относительно того, каким образом можно упростить реализацию моделей в производственной среде, используя сервисные возможности TensorFlow.

Сегментация изображений

Сегментация изображений — это задача компьютерного зрения, включающая разделение изображения на отдельные сегменты, например на совокупности пикселей, обладающих некими общими атрибутами. Что именно может выступать в качестве таких атрибутов, определяется как спецификой предметной области, так и конкретной задачей, но чаще всего основными атрибутами служат интенсивность пикселей, текстура и цвет. В этом разделе мы подробно рассмотрим базовые способы сегментации изображений, такие как пороговые методы на основе гистограмм интенсивностей пикселей, метод водоразделов и другие, которые предназначены для получения предварительной информации о сегментировании изображения, прежде чем их можно будет сегментировать с использованием методов, основанных на глубоком обучении.

Бинарный пороговый метод, основанный на гистограммах интенсивности пикселей

Часто в изображении имеются только две значимые области, представляющие интерес: объект и фон. В подобных сценариях гистограмма интенсивности пикселей представляет распределение вероятности, являющееся бимодальным, т.е. таким, которое характеризуется наличием двух пиков интенсивности пикселей. Простой способ отделения объекта от фона состоит в том, чтобы выбрать пиксель с пороговой интенсивностью и установить интенсивности всех пикселей, которые ниже этого порога, равными 255, а интенсивности пикселей, превышающих пороговое значение, — равными нулю. Эта операция гарантирует, что фон и объект будут представлены белым и черным цветами, не обязательно в указанном порядке. Если изображение представлено функцией $I(x, y)$, а пороговое значение t выбрано на основании гистограммы интенсивности пикселей, то новое сегментированное изображение $I'(x, y)$ можно представить в виде

$$I'(x, y) = 0, \textit{если } I(x, y) > t,$$
$$= 255, \textit{если } I(x, y) \leq t.$$

Если бимодальная гистограмма не делится на две части четкой границей в виде области с нулевой плотностью, то в качестве порога t целесообразно выбрать среднее значение интенсивностей пикселей в пиках бимодальных областей. Если обозначить эти интенсивности как p_1 и p_2, то порог t определится формулой

$$t = \frac{p_1 + p_2}{2}.$$

Альтернативный вариант заключается в выборе порога в виде значения интенсивности пикселей, промежуточного между значениями p_1 и p_2, при котором плотность гистограммы минимальна. Если гистограмма описывается функцией плотности $H(p)$, где $p \in \{0, 1, 2, \ldots, 255\}$ представляет интенсивности пикселей, то

$$t = \underbrace{\operatorname{argmin} H(p)}_{p \in [p_1, p_2]}.$$

Идея бинарного порога может быть расширена на несколько порогов, исходя из гистограммы интенсивности пикселей.

Метод Оцу

Метод Оцу определяет порог, максимизируя дисперсию между различными сегментами изображений. Процедура пороговой бинаризации по методу Оцу включает следующие шаги.

- Вычисляем вероятность интенсивности каждого пикселя в изображении. Если всего возможно N уровней интенсивности пикселей, то нормализованная гистограмма дает нужное распределение вероятностей для данного изображения.

$$P(i) = \frac{count(i)}{N} \quad \forall i \in \{0, 1, 2, \ldots, N-1\}.$$

- Если в изображении имеются два сегмента, C_1 и C_2, разделенных в соответствии с величиной порога t, то множество пикселей $\{0, 1, 2, ..., t\}$ принадлежит к C_1, тогда как множество пикселей $\{t+1, t+2, \ldots, L-1\}$ — к C_2. Дисперсия между этими двумя сегментами определяется суммой квадратов отклонений средних значений кластеров от глобального среднего. Суммы квадратов отклонений вычисляются с использованием весов, равных вероятности каждого кластера.

$$var(C_1 C_2) = P(C_1)(u_1 - u)^2 + P(C_2)(u_2 - u)^2,$$

где u_1 и u_2 — средние значения для кластера 1 и кластера 2; u — общее глобальное среднее:

$$u_1 = \sum_{i=1}^{t} P(i)i,\ u_2 = \sum_{i=t+1}^{L-1} P(i)i,\ u = \sum_{i=0}^{L-1} P(i)i.$$

Вероятность каждого сегмента определяется количеством пикселей в изображении, принадлежащих к данному классу. Вероятность сегмента C_1 пропорциональна количеству пикселей, интенсивность которых меньше или равна пороговой интенсивности t, тогда как вероятность сегмента C_2 пропорциональна количеству пикселей, интенсивность которых превышает порог t. Следовательно,

$$P(C_1) = \sum_{i=0}^{t} P(i),\ P(C_2) = \sum_{i=t+1}^{L-1} P(i).$$

- Внимательно присмотревшись к выражениям для u_1, u_2, $P(C_1)$ и $P(C_2)$, можно заметить, что каждое из них является функцией порога t, тогда как общее среднее u — постоянная величина для данного изображения. Следовательно, межсегментная дисперсия $var(C_1, C_2)$ — функция пороговой интенсивности пикселей t. Порог $\hat{t}$, максимизирующий эту дисперсию, предоставляет максимальное пороговое значение, которое следует использовать для сегментации с помощью метода Оцу:

$$\hat{t} = \underbrace{\text{argmax}\ var(C_1, C_2)}_{t}.$$

  Вместо вычисления производной с последующим приравниванием ее к нулю мы можем получить $\hat{t}$, вычислив дисперсию $var(C_1, C_2)$ при всех значениях $t = \{0, 1, 2, ..., L-1\}$ и выбрав то из них ($\hat{t}$), при котором $var(C_1, C_2)$ достигает максимума.

Метод Оцу может быть расширен для поддержки нескольких сегментов изображения; в этом случае вместо одного порогового значения следует определить $(k-1)$ порогов, где k — число сегментов.

Логика обоих вышеописанных методов, т.е. пороговой бинаризации на основе гистограммы интенсивности пикселей и метода Оцу, проиллюстрирована в листинге 6.1 (рис. 6.1 и 6.2). Вместо того чтобы реализовывать эти алгоритмы с помощью пакета для обработки изображений, мы используем базовую логику для облегчения интерпретации кода. Следует отметить, что описанные процессы сегментации обычно применяют к полутоновым (в градациях серого) изображениям, но сегментацию можно выполнять и по отдельным цветовым каналам.

Листинг 6.1. Реализация пороговой бинаризации на основе гистограммы интенсивности пикселей и метода Оцу средствами Python

```
## Метод пороговой бинаризации на основе гистограммы
## интенсивности пикселей
import cv2
import matplotlib.pyplot as plt
#%matplotlib inline
import numpy as np
img = cv2.imread("coins.jpg")
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
plt.imshow(gray,cmap='gray')
row,col = np.shape(gray)
gray_flat = np.reshape(gray,(row*col,1))[:,0]
ax = plt.subplot(222)
ax.hist(gray_flat,color='gray')
gray_const = []
```

```
## Порог 150 для интенсивности пикселей представляется
## подходящим, поскольку плотность вероятности минимальна
## вблизи этого значения
for i in xrange(len(gray_flat)):
    if gray_flat[i] < 150 :
        gray_const.append(255)
    else:
        gray_const.append(0)
gray_const = np.reshape(np.array(gray_const),(row,col))
bx = plt.subplot(333)
bx.imshow(gray_const,cmap='gray')

## Пороговый метод Оцу

img = cv2.imread("otsu.jpg")
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
plt.imshow(gray,cmap='gray')
row,col = np.shape(gray)
hist_dist = 256*[0]
## Вычисление частоты каждого пикселя в изображении
for i in xrange(row):
    for j in xrange(col):
        hist_dist[gray[i,j]] += 1
# Нормализация частот для получения вероятностей
hist_dist = [c/float(row*col) for c in hist_dist]
plt.plot(hist_dist)
## Вычисление межсегментной дисперсии
def var_c1_c2_func(hist_dist,t):
    u1,u2,p1,p2,u = 0,0,0,0,0
    for i in xrange(t+1):
        u1 += hist_dist[i]*i
        p1 += hist_dist[i]
    for i in xrange(t+1,256):
        u2 += hist_dist[i]*i
        p2 += hist_dist[i]
    for i in xrange(256):
        u += hist_dist[i]*i
    var_c1_c2 = p1*(u1 - u)**2 + p2*(u2 - u)**2
    return var_c1_c2
## Итеративный проход по всем значениям интенсивности пикселей
## в интервале от 0 до 255 и выбор значения, максимизирующего
## дисперсию
variance_list = []
for i in xrange(256):
    var_c1_c2 = var_c1_c2_func(hist_dist,i)
    variance_list.append(var_c1_c2)
## Извлечение порогового значения, максимизирующего дисперсию
t_hat = np.argmax(variance_list)
```

```
## Вычисление сегментированного изображения на основе
## порогового значения t_hat
gray_recons = np.zeros((row,col))
for i in xrange(row):
    for j in xrange(col):
        if gray[i,j] <= t_hat :
            gray_recons[i,j] = 255
        else:
            gray_recons[i,j] = 0
plt.imshow(gray_recons,cmap='gray')
```

```
-- вывод --
```

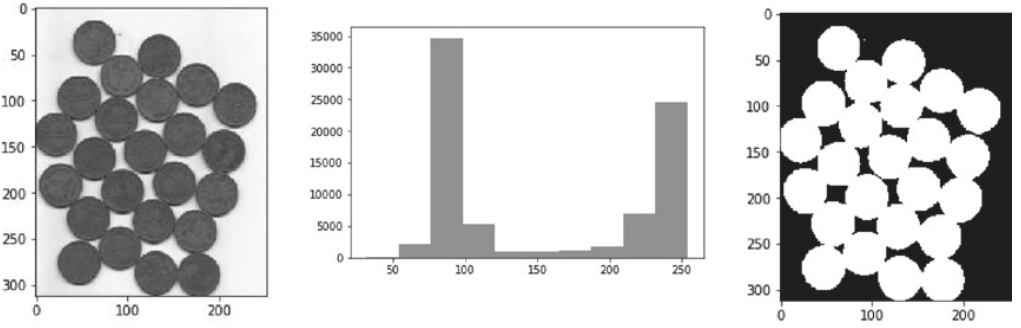


Рис. 6.1. Метод пороговой бинаризации на основе гистограммы интенсивности пикселей

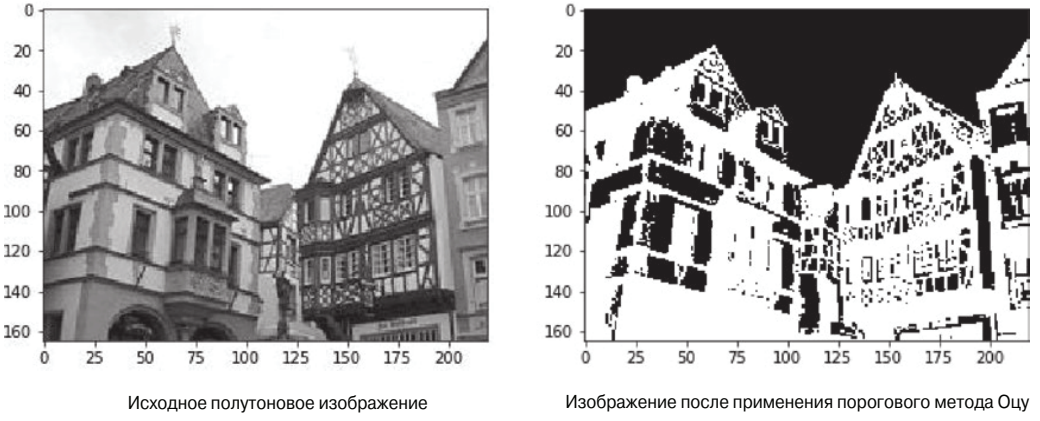


Рис. 6.2. Пороговый метод Оцу

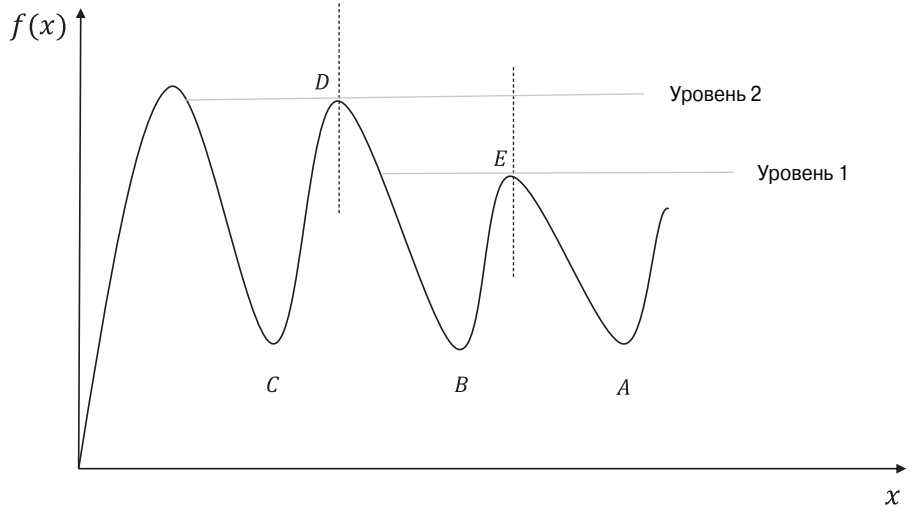
На рис. 6.1 оригинальное изображение монет в градациях серого подверглось пороговой бинаризации с помощью гистограммы интенсивности пикселей, чтобы отделить объекты (т.е. монеты) от фона. Исходя из этого, для порога интенсивности пикселей было выбрано значение 150. Для пикселей с интенсивностью ниже 150

устанавливалась интенсивность 255, используемая для представления объектов, а для пикселей с более высокой интенсивностью устанавливалась нулевая интенсивность, представляющая фон.

На рис. 6.2 приведены результаты применения порогового метода Оцу к изображению для получения двух сегментов, определяемых черным и белым цветами. Черный цвет представляет фон, белый — здание. Оптимальное пороговое значение интенсивности пикселей для этого изображения составило 143.

Сегментация изображений методом водоразделов

Метод водоразделов ориентирован на сегментацию изображения как совокупности топологических областей вблизи локальных минимумов интенсивности пикселей. Если рассматривать интенсивность пикселей как функцию их горизонтальной и вертикальной координат, то этот алгоритм пытается найти области вблизи локальных минимумов, называемых *бассейнами аттракции* (basins of attraction), или *водосборами* (catchment basins). Как только удается идентифицировать эти бассейны, алгоритм пытается выделить их путем построения разделительных границ, или *водоразделов* (watersheds), вдоль высоких пиков или гребней. Чтобы суть идеи стала вам более понятной, обратимся к простой графической иллюстрации того, как работает данный метод (рис. 6.3).



A, B, C — минимумы бассейнов аттракции
D, E — пики или максимумы, где должны быть сооружены водоразделы

Рис. 6.3. Иллюстрация работы метода водоразделов

Если мы начнем наполнять водой водосбор с минимумом в точке B, то по достижении уровня 1 избыток воды сможет переливаться в водосбор A. Чтобы предотвра-

тить подобный перелив, необходимо соорудить дамбу, или *водораздел*, в точке *E*. Соорудив водораздел в точке *E*, мы можем продолжить заливать воду в водосбор *B* до уровня 2, после чего избыток воды начнет переливаться в водосбор *C*. Чтобы предотвратить переливание воды в водосбор *C*, необходимо соорудить водораздел в точке *D*. Используя эту логику, мы можем продолжить процесс создания водоразделов для изоляции водосборов. В этом и заключается суть основной идеи, положенной в основу метода водоразделов. В данном случае мы имеем дело с функцией одной переменной, тогда как в случае полутоновых (в градациях серого) изображений функция, представляющая интенсивность пикселей, будет зависеть от двух переменных: вертикальной и горизонтальной координат.

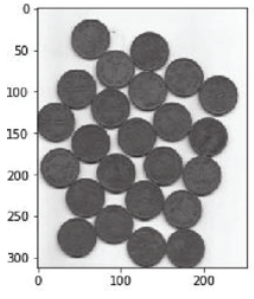
Метод водоразделов оказывается особенно полезным для обнаружения перекрывающихся объектов. Пороговые методы не в состоянии определить четкие границы между объектами. Обратимся к листингу 6.2, в котором приведен пример применения метода водоразделов для сегментации перекрывающихся изображений монет (рис. 6.4).

Листинг 6.2. Сегментация изображений методом водоразделов

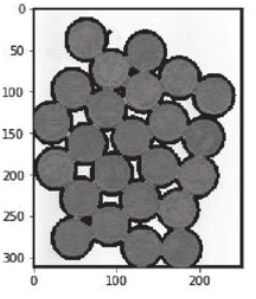
```
import numpy as np
import cv2
import matplotlib.pyplot as plt
from scipy import ndimage
from skimage.feature import peak_local_max
from skimage.morphology import watershed

## Загрузка изображений монет
im = cv2.imread("coins.jpg")
## Преобразование изображения в градации серого
imgray = cv2.cvtColor(im,cv2.COLOR_BGR2GRAY)
plt.imshow(imgray,cmap='gray')
# Порог изображения для его преобразования в бинарное
# изображение по методу Оцу
thresh = cv2.threshold(imgray, 0, 255,
        cv2.THRESH_BINARY | cv2.THRESH_OTSU)[1]
## Обнаружение и отображение контуров
im2, contours, hierarchy =
        cv2.findContours(thresh,cv2.RETR_TREE,
        cv2.CHAIN_APPROX_SIMPLE)
y = cv2.drawContours(imgray, contours, -1, (0,255,0), 3)
## Если мы видим контуры в отображении "y", то монеты имеют
## общий контур, и их невозможно разделить
plt.imshow(y,cmap='gray')
## Следовательно, мы переходим к методу водоразделов,
## чтобы каждая из монет образовала собственный кластер.
## Изменяем разметку преобразуемого изображения, чтобы оно
## состояло только из 0 и 1, поскольку именно такой формат
## должно иметь входное изображение для distance_transform_edt.
```

```
thresh[thresh == 255] = 5
thresh[thresh == 0] = 1
thresh[thresh == 5] = 0
## Функции distance_transform_edt и peak_local_max облегчают
## создание маркеров, обнаруживая точки вблизи центров монет.
## Можно пропустить эти шаги и создать маркер вручную, установив
## один пиксель в каждой монете в случайное число, представляющее
## его кластер.
D = ndimage.distance_transform_edt(thresh)
localMax = peak_local_max(D, indices=False, min_distance=10,
        labels=thresh)
markers = ndimage.label(localMax, structure=np.ones((3, 3)))[0]
# Предоставить матрицу EDT-расстояний и маркеры методу
# водоразделов для обнаружения меток кластеров для каждого
# пикселя. Для каждой монеты соответствующие ей пиксели будут
# заполнены номером кластера.
labels = watershed(-D, markers, mask=thresh)
print("[INFO] {} unique segments found".format(len(np.unique(labels))
        - 1))
# Создание контуров для каждой метки (каждой монеты)
# и их присоединение к картинке
for k in np.unique(labels):
    if k != 0 :
        labels_new = labels.copy()
        labels_new[labels == k] = 255
        labels_new[labels != k] = 0
        labels_new = np.array(labels_new,dtype='uint8')
        im2, contours, hierarchy =cv2.findContours(labels_new,
                cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
        z = cv2.drawContours(imgray,contours, -1, (0,255,0), 3)
        plt.imshow(z,cmap='gray')
```

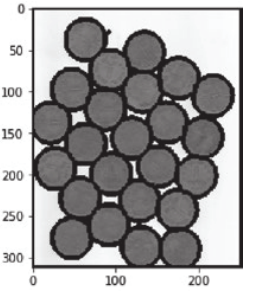
```
-- вывод --
```



Исходное изображение



Контурные границы без модели водоразделов



Контурные границы с моделью водоразделов

Рис. 6.4. Применение метода водоразделов для сегментации изображений

На рис. 6.4 видно, что после применения метода водоразделов края перекрывающихся монет отчетливо выделились, тогда как другие пороговые методы не в состоянии добиться этого эффекта.

Сегментация изображений методом кластеризации с помощью K-средних

Широко известный алгоритм K-средних также может быть использован для сегментации изображений, особенно медицинских. "K" в названии — это параметр алгоритма, определяющий количество кластеров. Алгоритм формирует кластеры, каждый из которых представляется центром кластера, который называется *центроидом* и выбирается на основе признаков входных данных. В случае сегментации изображений такими признаками обычно служат интенсивность пикселя и три его пространственные координаты: горизонтальная, вертикальная и цветовой канал. Поэтому вектор входных признаков можно представить в виде $u \in \mathbb{R}^{4\times 1}$, где

$$u = [I(x, y, z), x, y, z]^{\mathrm{T}}.$$

Можно поступить иначе, проигнорировав пространственные координаты и рассматривая интенсивности пикселей вдоль трех цветовых каналов в качестве входного вектора признаков, т.е.

$$u = [I_R(x, y), I_G(x, y), I_B(x, y)]^{\mathrm{T}},$$

где $I_R(x, y)$, $I_G(x, y)$ и $I_B(x, y)$ представляют интенсивности пикселей вдоль красного (Red), зеленого (Green) и синего (Blue) каналов соответственно для точки с координатами (x, y).

В данном алгоритме используются такие меры расстояния, как L^2 или L^1:

$$D\left(u^{(i)}, u^{(j)} / L^2\right) = \left\|u^{(i)} - u^{(j)}\right\|_2 = \sqrt{\left(u^{(i)} - u^{(j)}\right)^T \left(u^{(i)} - u^{(j)}\right)},$$

$$D\left(u^{(i)}, u^{(j)} / L^1\right) = \left\|u^{(i)} - u^{(j)}\right\|_1.$$

Базовая процедура алгоритма K-средних описана ниже.

- *Шаг 1.* Начинаем с выбранных случайным образом K центроидов кластеров $C_1, C_2, ..., C_k$, соответствующих K кластерам $S_1, S_2, ..., S_k$.
- *Шаг 2.* Вычисляем для каждого пикселя расстояния его вектора признаков $u^{(i)}$ до центроидов кластеров и относим его к кластеру S_j, если центроид C_j находится ближе всех остальных к данному вектору:

$$j = \underbrace{\operatorname{argmin}}_{j} \left\|u^{(i)} - C_j\right\|_2.$$

  Повторяем этот процесс для всех векторов признаков пикселей до тех пор, пока каждый из пикселей не будет приписан к одному из K кластеров.

- *Шаг 3.* Определив центроиды новых кластеров для всех пикселей, вычисляем их заново путем усреднения векторов признаков пикселей в каждом кластере:

$$C_j = \sum_{u^{(i)} \in S_j}^{m} u^{(i)}.$$

Повторяем шаги 2 и 3 в течение нескольких итераций до тех пор, пока центроиды не перестанут изменяться. Посредством этого итеративного процесса мы уменьшаем сумму внутрикластерных расстояний, имеющую следующий вид:

$$L = \sum_{j=1}^{K} \sum_{u^{(i)} \in S_j} \left\| u^{(i)} - C_j \right\|_2.$$

Простая реализация алгоритма K-средних приведена в листинге 6.3, где в качестве признаков используются интенсивности пикселей в трех цветовых каналах. Сегментация изображения осуществляется при $K = 3$. Выход представлен в оттенках серого и поэтому может не отражать фактического качества сегментации. Но если то же самое изображение, которое получено в листинге 6.3, отобразить в цветном формате, то проявятся более тонкие детали сегментации. Следует сделать еще одно важное замечание: минимизируемая функция стоимости или функция потерь, т.е. сумма внутрикластерных расстояний, представляет собой невыпуклую функцию, а это означает, что она может быть подвержена проблемам локальных минимумов. Поэтому имеет смысл запустить процесс сегментации несколько раз с разными начальными значениями центроидов кластеров и выбрать то значение, которое наилучшим образом обеспечивает минимизацию функции стоимости или наиболее подходящую сегментацию.

Листинг 6.3. Сегментация изображения с помощью метода K-средних

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
#np.random.seed(0)
img = cv2.imread("kmeans.jpg")
imgray_ori = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
plt.imshow(imgray_ori,cmap='gray')
## Сохранение размеров изображения
row,col,depth = img.shape
## Свертывание осей строк и столбцов
## для ускорения матричных операций
img_new = np.zeros(shape=(row*col,3))
glob_ind = 0
for i in xrange(row):
    for j in xrange(col):
        u = np.array([img[i,j,0],img[i,j,1],img[i,j,2]])
        img_new[glob_ind,:] = u
        glob_ind += 1
```

```
# Установка количества кластеров
K = 5
# Выполнение алгоритма K-средних
num_iter = 20
for g in xrange(num_iter):
# Определение переменных clusters для сохранения количества кластеров
# и out_dist для сохранения расстояний от центроида
    clusters = np.zeros((row*col,1))
    out_dist = np.zeros((row*col,K))
    centroids = np.random.randint(0,255,size=(K,3))
    for k in xrange(K):
        diff = img_new - centroids[k,:]
        diff_dist = np.linalg.norm(diff,axis=1)
        out_dist[:,k] = diff_dist
# Назначение кластера с минимальным расстоянием до пикселя
    clusters = np.argmin(out_dist,axis=1)
# Пересчет кластеров
    for k1 in np.unique(clusters):
        centroids[k1,:] =
        np.sum(img_new[clusters == k1,:],axis=0)/np.
        sum([clusters == k1])
# Преобразование меток кластеров в двумерное изображение
clusters = np.reshape(clusters,(row,col))
out_image = np.zeros(img.shape)
# Формирование 3D-изображения путем замены меток интенсивностями
# пикселей соответствующих центроидов
for i in xrange(row):
    for j in xrange(col):
        out_image[i,j,0] = centroids[clusters[i,j],0]
        out_image[i,j,1] = centroids[clusters[i,j],1]
        out_image[i,j,2] = centroids[clusters[i,j],2]

out_image = np.array(out_image,dtype="uint8")
# Вывод выходного изображения после его
# преобразования в градации серого.
# Читателям рекомендуется вывести изображение
# как оно есть для большей ясности.
imgray = cv2.cvtColor(out_image,cv2.COLOR_BGR2GRAY)
plt.imshow(imgray,cmap='gray')

--- вывод ---
```

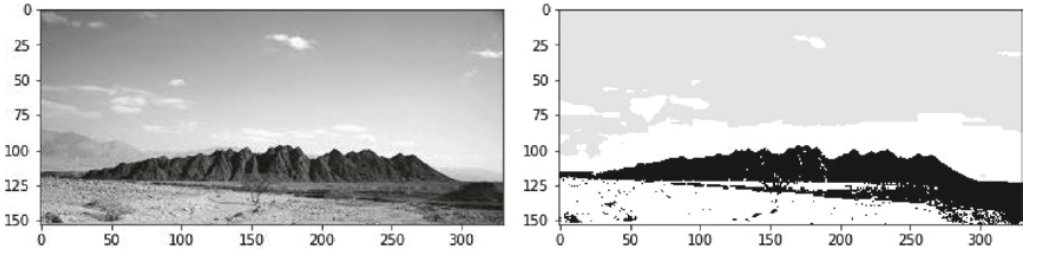


Рис. 6.5. Сегментация посредством алгоритма K-средних

На рис. 6.5 видно, что метод кластеризации с помощью K-средних неплохо справился с сегментацией изображения при $K = 3$.

Семантическая сегментация

В последние годы большую популярность приобрела сегментация изображений с помощью сверточных нейронных сетей. Существенной отличительной чертой этого подхода является аннотирование отнесения каждого пикселя к тому или иному классу объектов, так что процесс обучения подобных сегментирующих сетей полностью контролируется. Несмотря на то что процесс аннотирования изображений требует больших вычислительных затрат, он упрощает задачу, предоставляя *целевой признак* (ground truth) для сравнения. Этим признаком может служить изображение с пикселями, хранящими репрезентативный цвет определенного объекта. Например, если мы работаем с изображениями кошек и собак, имеющими фон, то каждый пиксель изображения может принадлежать к одному из трех классов: кошка, собка и фон. Кроме того, каждый класс объектов обычно представляется репрезентативным цветом, чтобы целевой объект можно было отобразить как сегментированное изображение. А теперь перейдем к рассмотрению сверточных нейронных сетей, способных выполнять семантическую сегментацию.

Метод скользящего окна

С помощью скользящего окна можно извлекать фрагменты из оригинального изображения и передавать их классифицирующей сверточной сети для прогнозирования класса центрального пикселя каждого из фрагментов. Не удивительно, что такой подход требует интенсивных вычислений как на стадии тренировки, так и на стадии тестирования, поскольку классифицирующей CNN-сети должно быть передано по крайней мере N фрагментов на каждое изображение, где N — количество пикселей в изображении.

Работа сети, выполняющей семантическую сегментацию изображений собаки, кошки и фона с использованием скользящего окна, показана на рис. 6.6. Окно скользит по оригинальному изображению, извлекая его фрагменты, которые передаются

классифицирующей сети CNN для классификации центрального пикселя каждого фрагмента. В качестве классифицирующей сети могут использоваться такие предварительно обученные сети, как AlexNet, VGG19, InceptionV3 и др., выходной слой которых генерирует одну из трех меток, соответствующих собакам, кошкам и фону. Дальнейшая подстройка сети может обеспечиваться механизмом обратного распространения ошибки с использованием фрагментов изображений в качестве входа и метки класса центрального пикселя фрагмента в качестве выхода. С точки зрения свертки изображений такая сеть крайне неэффективна, поскольку смежные фрагменты в значительной степени перекрываются и каждый раз подвергаются независимой повторной обработке, что влечет за собой дополнительные накладные расходы. Для преодоления этих недостатков можно использовать полносверточные сети, обсуждению которых посвящен следующий раздел.

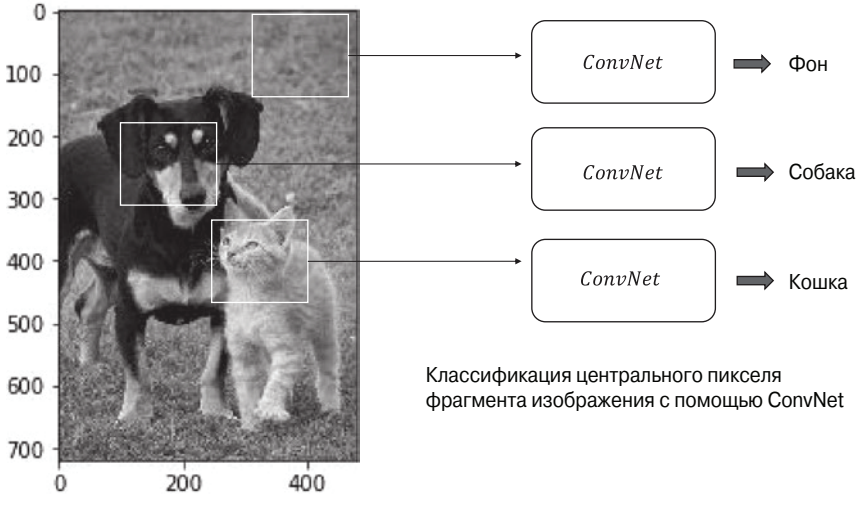


Рис. 6.6. Семантическая сегментация методом скользящего окна

Полносверточная сеть

Полносверточная сеть (fully convolutional network — FCN) состоит из серии сверточных слоев без каких-либо полносвязных слоев. Свертки выбираются таким образом, чтобы размеры изображения, т.е. его ширина и высота, оставались неизменными по мере прохождения сети. Вместо того чтобы оценивать категорию каждого пикселя независимо для каждого извлекаемого фрагмента, как в подходе на основе скользящего окна, полносверточная сеть прогнозирует категории всех пикселей одновременно. Выходной слой такой сети состоит из C карт признаков, где C — количество категорий, включая фон, по которым категоризируется каждый пиксель. Если h и w — высота и ширина оригинала изображения соответственно, то выход состоит из $h'w$ карт признаков в количестве C. Кроме того, для целевого признака (ground truth) должно быть C сегментированных изображений, соответствующих C классам. При

любых пространственных координатах (h_1, w_1) каждая из карт признаков содержит оценку вероятности принадлежности данного пикселя карте признаков, с которой он связан. Эти оценки, содержащиеся в картах признаков для каждого пространственного положения пикселя (h_1, w_1), образуют слой SoftMax для различных классов.

На рис. 6.7 приведена архитектурная схема полносверточной сети. Количество выходных карт признаков, равно как и карт признаков целевых объектов, должно равняться трем, что соответствует трем классам.

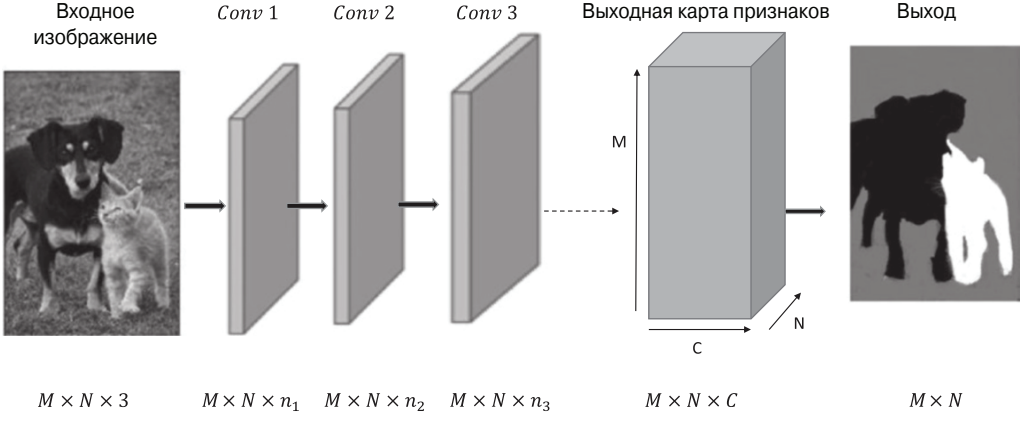


Рис. 6.7. Архитектура полносверточной сети

Если обозначить активацию входной сети или оценку вероятности k-го класса для точки с пространственными координатами (i, j) через $s_k^{(i,j)}$, то вероятность принадлежности пикселя с пространственными координатами (i, j) к k-му классу определяется SoftMax-вероятностью в соответствии с формулой

$$P_k(i, j) = \frac{e^{s_k^{(i,j)}}}{\sum_{k'=1}^{C} e^{s_{k'}^{(i,j)}}}.$$

Кроме того, если $y_k^{(i,j)}$ — метки целевых объектов для k-го класса при пространственных координатах (i, j), то кросс-энтропийные потери для пикселя с пространственными координатами (i, j) можно обозначить как

$$L(i, j) = -\sum_{k=1}^{C} y_k(i, j) \log P_k(i, j).$$

Если высота и ширина изображений, поступающих в сеть, равны M и N соответственно, то общие потери для изображения составляют

$$L = -\sum_{i=0}^{M-1} \sum_{j=0}^{N-1} \sum_{k=1}^{C} y_k(i, j) \log P_k(i, j).$$

Изображения могут передаваться сети в виде мини-пакетов, и тогда в качестве функции потерь или функции стоимости, подлежащей оптимизации в каждой эпохе мини-пакетного обучения посредством градиентного спуска, можно использовать средние потери на одно изображение.

Выходной класс $\hat{k}$ для пикселя с пространственными координатами (i, j) можно определить, взяв класс k, для которого вероятность $P_k^{(i,j)}$ максимальна, т.е.

$$\hat{k} = \underbrace{\text{argmax}}_{k} P_k(i, j).$$

Для получения окончательного сегментированного изображения аналогичные действия должны быть выполнены по отношению к пикселям во всех пространственных точках изображения.

На рис. 6.8 приведены выходные карты признаков сети, предназначенной для сегментации изображений кошек, собак и фона. Как видим, для каждой из трех категорий или классов предусмотрена отдельная карта признаков. Карта признаков имеет те же размеры, что и входное изображение. Для всех трех классов на рисунке отображены входная активация сети, соответствующая вероятность и соответствующая целевая метка для пикселя с пространственными координатами (i, j).

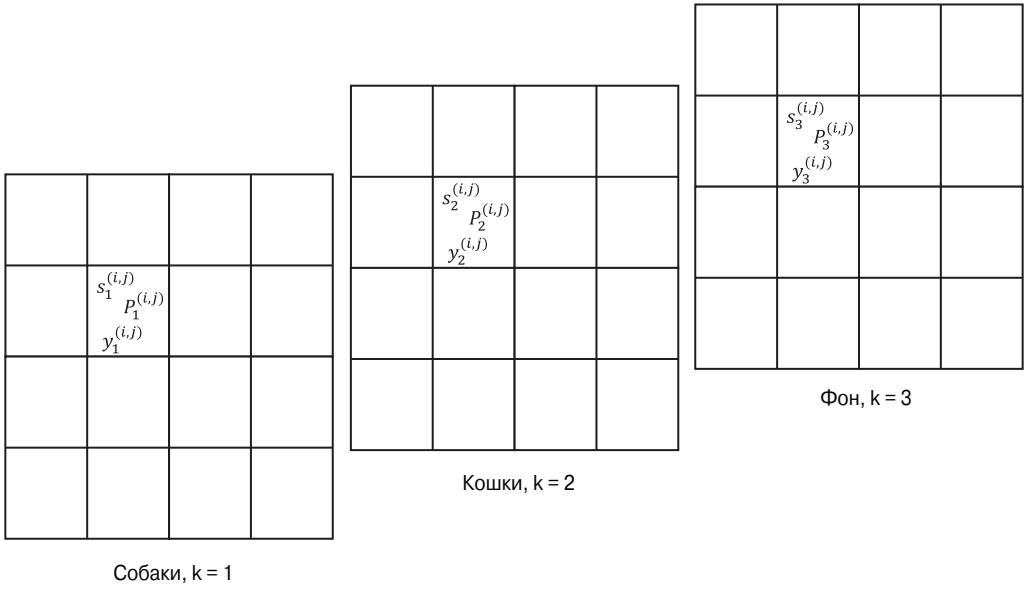


Рис. 6.8. Выходные карты признаков, соответствующие каждому из трех классов: собаки, кошки и фон

В этой сети все свертки сохраняют пространственные размеры входного изображения. Поэтому обработка изображений с высоким разрешением требует интенсивных вычислений, особенно в случае большого количества карт признаков или кана-

лов в каждой свертке. Для решения этой проблемы чаще всего используется другой вариант полносверточной нейронной сети, в котором изображение подвергается сначала понижающей дискретизации в первой половине сети, а затем — повышающей дискретизации во второй половине. Эта модифицированная версия полносверточной сети является следующей темой нашего обсуждения.

Полносверточная сеть с понижающей и повышающей дискретизацией

Вместо того чтобы сохранять неизменными пространственные размеры изображений во всех сверточных слоях, как в предыдущей сети, в этом варианте полносверточной сети используется комбинация сверток, в которой изображение подвергается понижающей дискретизации в первой половине сети, а затем — повышающей дискретизации в последних слоях для восстановления пространственных размеров исходного изображения. Обычно такая сеть состоит из нескольких слоев понижающей дискретизации посредством *шаговых* сверток и/или операций пулинга, за которыми следуют несколько слоев повышающей дискретизации. В рассмотренных до этого случаях свертка либо уменьшала пространственные размеры изображения, либо размеры выходного изображения были равны входным размерам. В данном варианте сети изображения или, скорее, карты признаков сети подвергаются интерполяции, означающей повышение разрешения изображения (upsampling). Архитектурная схема такой сети приведена на рис. 6.9.

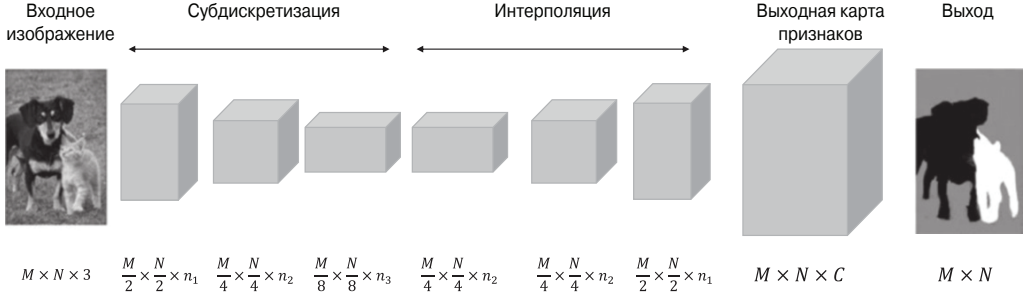


Рис. 6.9. Полносверточная сеть с понижающей и повышающей дискретизацией

Обсуждению методов, которые обычно используются для повышения дискретизации изображений или карт признаков, посвящен следующий раздел.

Повышающая дискретизация

Повышающую дискретизацию (unpooling, upsampling) можно рассматривать как операцию, обратную *понижающей дискретизации*, или *субдискретизации* (pooling, downsampling). Выполняя *субдискретизацию по максимуму* (max pooling) или *субдискретизацию по среднему значению* (average pooling), мы уменьшаем пространственную размерность изображения взятием максимального или среднего значения пикселя

в пределах ядра субдискретизации. Поэтому, если ядро субдискретизации имеет размер 2×2, то пространственные размеры изображения уменьшатся в два раза вдоль каждого измерения. Выполняя повышающую дискретизацию, мы обычно увеличиваем пространственные размеры изображения, повторяя значение пикселя в некоторой окрестности (рис. 6.10, *а*).

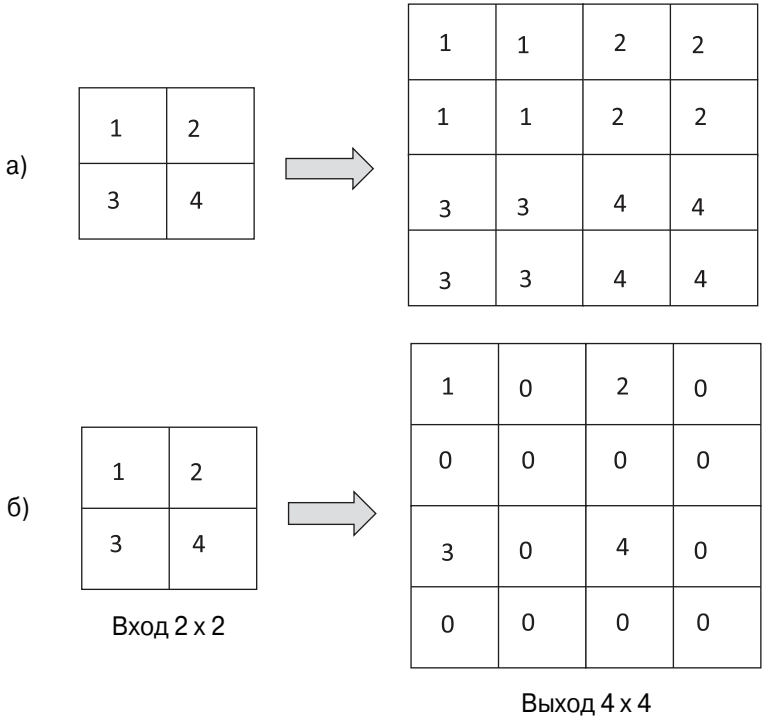


Рис. 6.10. Операция повышения дискретизации

Возможен и другой вариант повышающей дискретизации, когда окрестность заполняется только одним пикселем, тогда как остальные обнуляются (рис. 6.10, *б*).

Повышающая дискретизация по максимуму

Многие полносверточные слои обладают симметрией, поскольку операция субдискретизации, выполняемая в первой половине сети, дополняется соответствующей операцией повышения дискретизации во второй половине для восстановления размера изображения. Всякий раз, когда выполняется субдискретизация, какая-то часть информации о входном изображении теряется из-за того, что ряд соседних пикселей представляется одним репрезентативным элементом. Например, если выполняется субдискретизация по максимальному значению с размером ядра 2×2, то для представления каждой окрестности размером 2×2 на выход передается лишь максимальное значение пикселя, представляющего данную окрестность. На основании выхода невозможно сделать заключение о точном расположении пикселя с максимальным зна-

чением. Следовательно, в этом процессе мы теряем часть информации о входе. В задачах семантической сегментации мы стремимся как можно точнее связать каждый пиксель с его целевой меткой. Однако из-за повышения субдискретизации по максимуму значительная часть информации о границах и других тонких деталях изображения теряется. Одним из способов частичного восстановления этой утерянной пространственной информации на стадии реконструирования изображения посредством повышающей дискретизации является помещение значения входного пикселя в выходное расположение, соответствующее тому расположению, из которого выход понижающей дискретизации по максимуму получил свой вход (рис. 6.11).

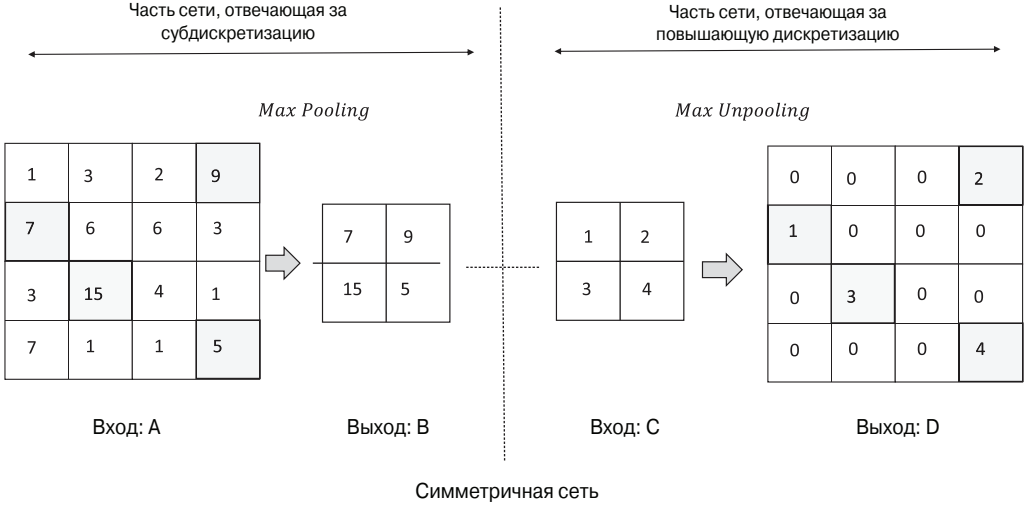


Рис. 6.11. Применение повышающей дискретизации по максимуму в симметричной полносверточной сегментирующей сети

Как показано на рис. 6.11, в процессе выполнения повышающей дискретизации значениями заполняются лишь те расположения в выходной карте *D*, которые соответствуют расположениям максимальных элементов на входе *A*. Этот метод повышения дискретизации обычно называют *повышением дискретизации по максимуму* (max unpooling).

Транспонированная свертка

Операции *интерполяции* (upsampling), выполняемые посредством повышающей дискретизации или повышающей дискретизации по максимуму, относятся к фиксированным преобразованиям. Эти преобразования не включают никаких параметров, которым сеть должна обучиться в процессе тренировки. Соответствующий способ, обеспечивающий возможность обучения сети, заключается в выполнении интерполяции посредством транспонированной свертки, которая во многом подобна уже известным вам операциям свертки. Поскольку транспонированная свертка включает параметры, которым сеть должна обучаться, необходимо научить сеть выполнять ин-

терполяцию таким способом, чтобы обеспечивалось уменьшение функции стоимости, на которой тренируется сеть. Рассмотрим более подробно, как работает транспонированная свертка.

При выполнении шаговой свертки с шагом 2 выходные размеры уменьшаются почти вдвое для каждого пространственного измерения. Поэтапные результаты применения операции свертки с размером ядра 4×4, шагом 2 и шириной поля дополнения нулями, равной 1, к двумерному входу размером 5×5 представлены на рис. 6.12. Ядро скользит поверх входа, и в каждой позиции, над которой оно находится, вычисляется точечное произведение ядра и той части входа, с которой ядро перекрывается.

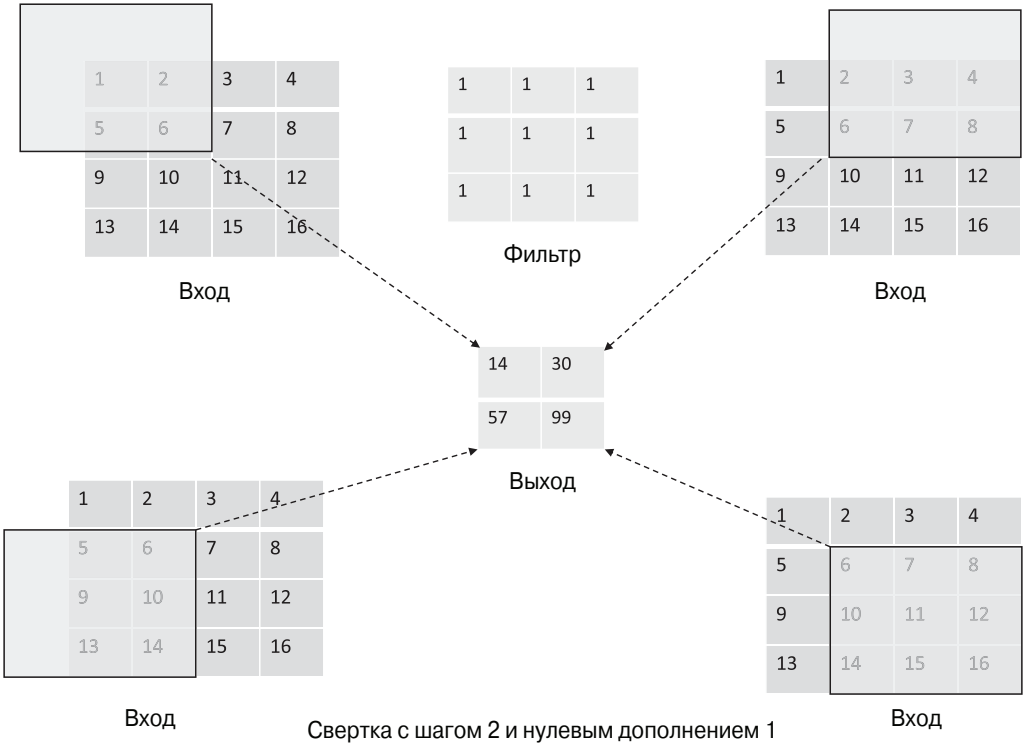


Рис. 6.12. Применение операции шаговой свертки для понижающей дискретизации изображения

Такого же типа логика используется и в случае транспонированной свертки, но вместо субдискретизации шаги больше 1 обеспечивают повышающую дискретизацию. Так, если мы используем шаг 2, то входной размер удваивается в каждом пространственном направлении. На рис. 6.13, *а–в*, показано воздействие операции транспонированной свертки с использованием размера фильтра или ядра, равным 3×3, на вход размером 2×2, в результате чего получается выход размером 4×4. В отличие от обычной свертки, когда мы вычисляем точечное произведение фильтра и части входа, в случае транспонированной свертки значения фильтра при каждом расположении

взвешиваются входным значением из той позиции, в которой находится фильтр, и эти взвешенные значения фильтра заполняют соответствующие расположения в выходе. Выходы для последовательных входных значений вдоль того же пространственного измерения размещаются с зазором, определяемым величиной шага транспонированной свертки. Эти действия выполняются по отношению ко всем входным значениям. Наконец, выходы, соответствующие каждому из входных значений, складываются для получения конечного выхода (рис. 6.13, *в*).

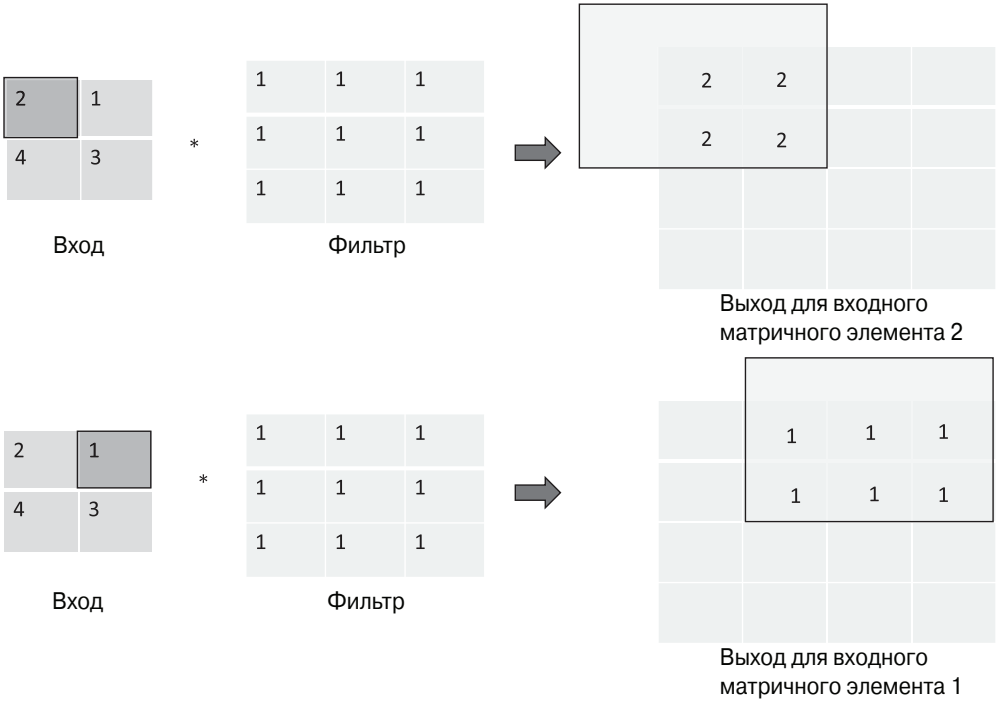


Рис. 6.13, а. Повышающая дискретизация посредством транспонированной свертки

В TensorFlow операцию повышающей дискретизации посредством транспонированной свертки можно выполнить с помощью функции `tf.nm.conv2d_transpose`.

U-Net

Сверточная нейронная сеть U-Net — одна из наиболее эффективных архитектур, применяемых в последнее время для сегментации изображений, особенно медицинских. Архитектура U-Net стала победителем соревнования по отслеживанию клеток *Cell Tracking Challenge*, которое проходило на симпозиуме ISBI (International Symposium on Biomedical Imaging) в 2015 году. Топология сети имеет U-образную форму в направлении от входных слоев к выходным, отсюда и название — **U-Net**. Олаф Роннебергер, Филипп Фишер и Томас Брокс, разработавшие эту сверточную сеть для сегментации изображений, опубликовали подробное описание модели в статье

"U-Net: Convolutional Networks for Biomedical Image Segmentation", доступной по адресу `https://arxiv.org/abs/1505.04597`.

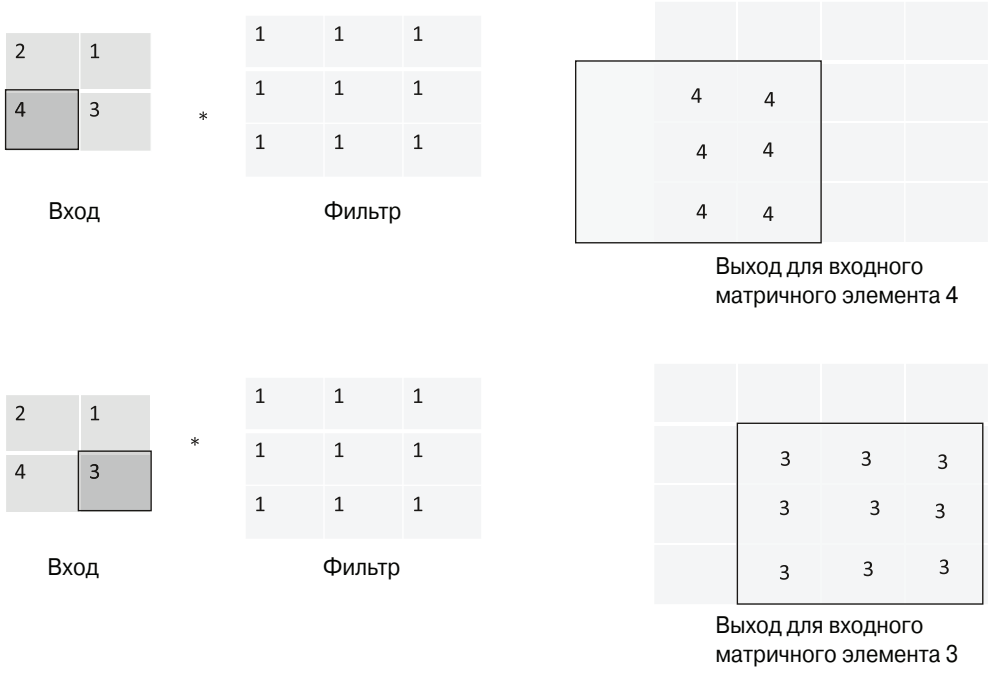


Рис. 6.13, б.

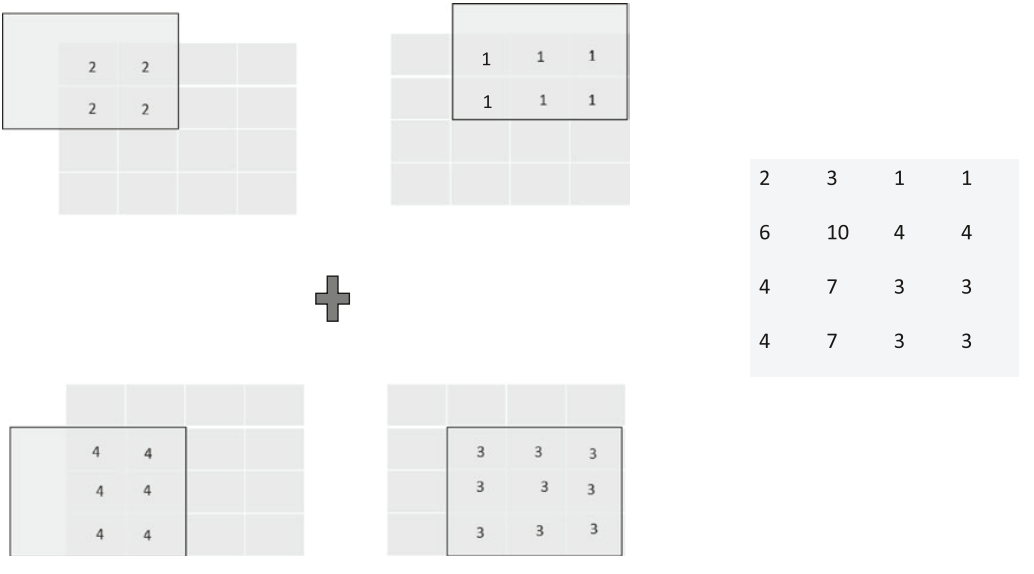


Рис. 6.13, в.

В первой части сети изображения подвергаются понижающей дискретизации посредством сочетания операций свертки и субдискретизации по максимуму. Свертки ассоциируются с попиксельными ReLU-активациями. Каждая операция свертки выполняется с фильтром размером 3×3 и без дополнения нулями, что приводит к редукции двух пикселей в каждом пространственном измерении для выходных карт признаков. Во второй части сети разрешение субдискретизированных изображений увеличивается до достижения последнего слоя, где выходные карты признаков соответствуют конкретным классам сегментируемых объектов. В качестве функции стоимости из расчета на одно изображение должна использоваться попиксельная категориальная кросс-энтропия или логарифмическая функция потерь для классификации, суммированная по всему изображению, как мы видели ранее. Следует сделать одно замечание: выходные карты признаков в U-Net имеют меньшие размеры по сравнению с размерами входных изображений. Например, входное изображение с пространственными размерами 572×572 произведет выходные карты признаков с пространственными размерами 388×388. У кого-то может возникнуть вопрос: а как выполняется попиксельное сравнение классов для вычисления функции потерь в процессе тренировки? Идея очень проста: сегментированные выходные карты признаков сравниваются с целевым сегментированным изображением с размером фрагмента 388×388, извлеченным из центра входного изображения. Основная мысль заключается в том, что при наличии изображений с более высоким разрешением, скажем, 1024×1024, можно создать из него множество случайных изображений с пространственными размерами 572×572 для целей обучения. Целевые изображения также создаются из этих вспомогательных изображений размером 572×572 путем извлечения центрального фрагмента размером 388×388 и снабжения каждого его пиксела меткой соответствующего класса. Это позволяет тренировать сеть на значительных объемах данных, даже если для обучения доступно не так уж много изображений. Архитектурная диаграмма сети U-Net представлена на рис. 6.14.

На этой архитектурной диаграмме видно, что в первой части сети изображения подвергаются свертке и субдискретизации по максимуму для уменьшения пространственных размеров (снижения разрешения), и в то же время глубина каналов, т.е. количество карт признаков, увеличивается. За каждыми двумя последовательными свертками вместе с ассоциированными с ними ReLU-активациями следует операция субдискретизации по максимуму, которая уменьшает размер изображения до 1/4 от первоначального.

Каждая операция субдискретизации по максимуму продвигает сеть далее к следующему набору сверток и вносит вклад в U-образную форму первой части сети. Аналогичным образом слои повышающей дискретизации увеличивают пространственные размеры в два раза вдоль каждого измерения, а значит, размер изображения увеличивается в четыре раза. Кроме того, это замыкает U-образную структуру сети во второй ее части. После каждой повышающей дискретизации изображение проходит через две свертки и ассоциированные с ними ReLU-активации.

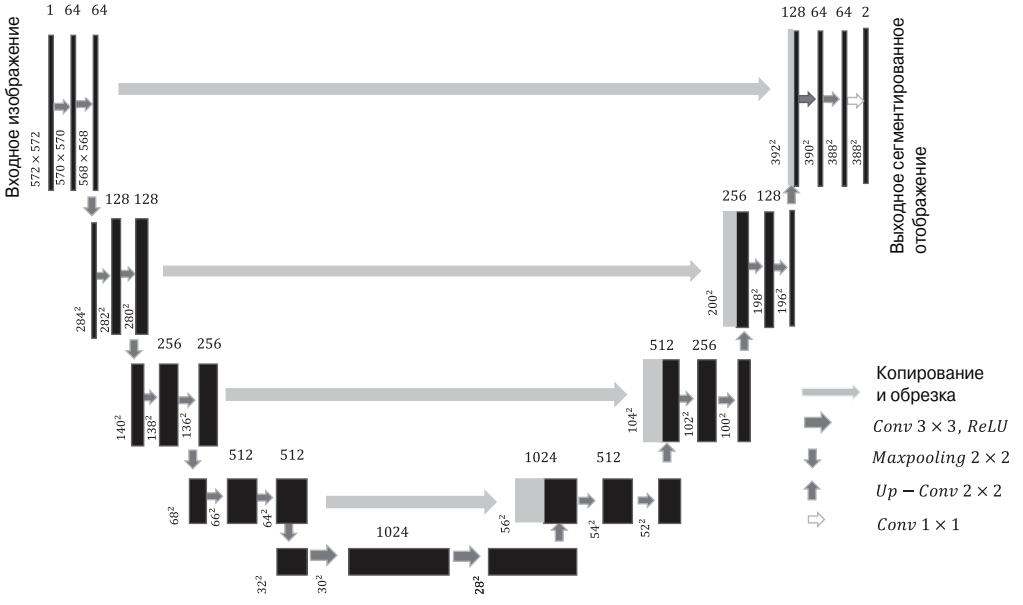


Рис. 6.14. Архитектурная диаграмма U-Net

U-Net — весьма симметричная сеть, коль скоро речь идет об операциях субдискретизации по максимуму и повышающей дискретизации. Однако для соответствующей пары этих операций размеры изображения до и после их выполнения не совпадают, в отличие от других полносверточных слоев. Как ранее уже отмечалось, после выполнения операции субдискретизации по максимуму существенная часть пространственной информации теряется за счет того, что локальная окрестность изображения представляется на выходе одним репрезентативным пикселем. Восстановить эту утерянную пространственную информацию, когда изображение возвращается к своим первоначальным размерам в результате повышающей дискретизации, весьма трудно, а значит, новое изображение также теряет существенную часть информации в области ребер и других тонких аспектов изображения. Как следствие, сегментация получается неоптимальной. Если бы изображение после повышающей дискретизации имело те же пространственные размеры, что и перед выполнением соответствующей операции субдискретизации по максимуму, то, для того чтобы помочь сети восстановить хотя бы часть утерянной информации, было бы достаточно всего лишь присоединить случайное количество карт признаков, прежде чем к выходным картам признаков после повышающей дискретизации будет применена субдискретизация по максимуму. Поскольку в U-Net размеры этих карт признаков не совпадают, сеть обрезает карты признаков перед субдискретизацией, чтобы они имели те же пространственные размеры, что и выходные карты признаков на выходе повышающей дискретизации, и конкатенирует их. Это приводит к лучшей сегментации изображений, поскольку содействует восстановлению некоторой пространственной информации, утерянной в процессе субдискретизации по максимуму. Также следует отметить, что

интерполяцию изображений можно осуществлять любыми из методов, которые мы к этому времени успели рассмотреть, такими как *повышающая дискретизация* (unpooling), *повышающая дискретизация по максимальному значению* (max unpooling) и *транспонированная свертка* (transpose convolution), которая также известна под названием *деконволюция* (deconvolution). К числу заметных преимуществ сегментации с помощью сети U-Net можно отнести следующее.

- Используя всего лишь небольшое количество аннотированных либо помеченных вручную сегментированных изображений, можно генерировать значительные объемы тренировочных данных.
- Сеть U-Net хорошо справляется с сегментацией изображений даже при наличии соприкасающихся объектов одного класса, которые должны быть разделены. Ранее мы уже видели, что разделение соприкасающихся объектов одного класса с помощью традиционных методов обработки изображений — непростая задача, и достижение удовлетворительной сегментации такими методами, как алгоритм водоразделов, требует тщательной разметки входных данных объектными маркерами. Сети U-Net удается обеспечить хорошее разделение соприкасающихся сегментов одного класса за счет введения больших весов неправильной классификации пикселей вблизи границ соприкасающихся сегментов.

Семантическая сегментация с помощью полносвязных нейронных сетей средствами TensorFlow

В этом разделе мы подробно рассмотрим реализацию сети для отделения изображений автомобилей от фона средствами TensorFlow — задача, которая предлагалась на соревнованиях Carvana, проводившихся на платформе Kaggle. Для тренировки сети доступны входные изображения с их *целевыми признаками* (ground truth) сегментации. Мы тренируем модель на 80% тренировочных данных и проверяем работу модели на оставшихся 20%. Для тренировки мы используем полносвязную сверточную сеть, структура которой аналогична структуре U-Net в том, что после прохождения первой половины сети выполняется интерполяция изображения посредством транспонированной свертки. Однако имеются и отличия, одно из которых состоит в том, что пространственные размеры изображения при выполнении свертки не изменяются благодаря использованию дополнения нулями с параметром `SAME`. Вторым отличием является то, что при переходе от потока понижающей дискретизации к потоку повышающей дискретизации никакие связи не опускаются. Детали реализации приведены в листинге 6.4.

Листинг 6.4. Реализация семантической сегментации с помощью полносверточной нейронной сети средствами TensorFlow

```
## Загрузка необходимых пакетов
import tensorflow as tf
```

```
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
%matplotlib inline
import os
from subprocess import check_output
import numpy as np
from keras.preprocessing.image import array_to_img, img_to_array,
↳load_img,ImageDataGenerator
from scipy.misc import imresize

# Определение субдискретизации - 2 (Conv+ReLU) и 1 Maxpooling.
# При необходимости Maxpooling можно установить в False.

x = tf.placeholder(tf.float32,[None,128,128,3])
y = tf.placeholder(tf.float32,[None,128,128,1])

def down_sample(x,w1,b1,w2,b2,pool=True):
    x = tf.nn.conv2d(x,w1,strides=[1,1,1,1],padding='SAME')
    x = tf.nn.bias_add(x,b1)
    x = tf.nn.relu(x)
    x = tf.nn.conv2d(x,w2,strides=[1,1,1,1],padding='SAME')
    x = tf.nn.bias_add(x,b2)
    x = tf.nn.relu(x)
    if pool:
        y = tf.nn.max_pool(x,ksize=[1,2,2,1],strides=[1,2,2,1],
                padding='SAME')
        return y,x
    else:
        return x

# Определение повышающей дискретизации
def up_sample(x,w,b):
    output_shape = x.get_shape().as_list()
    output_shape[0] = 32
    output_shape[1] *= 2
    output_shape[2] *= 2
    output_shape[1] = np.int(output_shape[1])
    output_shape[2] = np.int(output_shape[2])
    output_shape[3] = w.get_shape().as_list()[2]
    conv_tf = tf.nn.conv2d_transpose(value=x,filter=w,
            output_shape=output_shape,strides=[1,2,2,1],
            padding="SAME")
    conv_tf = tf.nn.bias_add(conv_tf,b)
    return tf.nn.relu(conv_tf)

## Определение весов

weights = {
    'w11': tf.Variable(tf.random_normal([3,3,3,64],mean=0.0,
```

```
        stddev=0.02)),
    'w12': tf.Variable(tf.random_normal([3,3,64,64],mean=0.0,
        stddev=0.02)),
    'w21': tf.Variable(tf.random_normal([3,3,64,128],mean=0.0,
        stddev=0.02)),
    'w22': tf.Variable(tf.random_normal([3,3,128,128],mean=0.0,
        stddev=0.02)),
    'w31': tf.Variable(tf.random_normal([3,3,128,256],mean=0.0,
        stddev=0.02)),
    'w32': tf.Variable(tf.random_normal([3,3,256,256],mean=0.0,
        stddev=0.02)),
    'w41': tf.Variable(tf.random_normal([3,3,256,512],mean=0.0,
        stddev=0.02)),
    'w42': tf.Variable(tf.random_normal([3,3,512,512],mean=0.0,
        stddev=0.02)),
    'w51': tf.Variable(tf.random_normal([3,3,512,1024],mean=0.0,
        stddev=0.02)),
    'w52': tf.Variable(tf.random_normal([3,3,1024,1024],mean=0.0,
        stddev=0.02)),
    'wu1': tf.Variable(tf.random_normal([3,3,1024,1024],mean=0.0,
        stddev=0.02)),
    'wu2': tf.Variable(tf.random_normal([3,3,512,1024],mean=0.0,
        stddev=0.02)),
    'wu3': tf.Variable(tf.random_normal([3,3,256,512],mean=0.0,
        stddev=0.02)),
    'wu4': tf.Variable(tf.random_normal([3,3,128,256],mean=0.0,
        stddev=0.02)),
    'wf': tf.Variable(tf.random_normal([1,1,128,1],mean=0.0,
        stddev=0.02))
}

biases = {
    'b11': tf.Variable(tf.random_normal([64],mean=0.0,stddev=0.02)),
    'b12':tf.Variable(tf.random_normal([64],mean=0.0,stddev=0.02)),
    'b21': tf.Variable(tf.random_normal([128],mean=0.0,stddev=0.02)),
    'b22': tf.Variable(tf.random_normal([128],mean=0.0,stddev=0.02)),
    'b31': tf.Variable(tf.random_normal([256],mean=0.0,stddev=0.02)),
    'b32': tf.Variable(tf.random_normal([256],mean=0.0,stddev=0.02)),
    'b41': tf.Variable(tf.random_normal([512],mean=0.0,stddev=0.02)),
    'b42': tf.Variable(tf.random_normal([512],mean=0.0,stddev=0.02)),
    'b51': tf.Variable(tf.random_normal([1024],mean=0.0,stddev=0.02)),
    'b52': tf.Variable(tf.random_normal([1024],mean=0.0,stddev=0.02)),
    'bu1': tf.Variable(tf.random_normal([1024],mean=0.0,stddev=0.02)),
    'bu2': tf.Variable(tf.random_normal([512],mean=0.0,stddev=0.02)),
    'bu3': tf.Variable(tf.random_normal([256],mean=0.0,stddev=0.02)),
    'bu4': tf.Variable(tf.random_normal([128],mean=0.0,stddev=0.02)),
    'bf': tf.Variable(tf.random_normal([1],mean=0.0,stddev=0.02))
}
```

```
## Создание окончательной модели

def unet_basic(x,weights,biases,dropout=1):

    ## Свертка 1
    out1,res1 = down_sample(x,weights['w11'],biases['b11'],
        weights['w12'],biases['b12'],pool=True)
    out1,res1 = down_sample(out1,weights['w21'],biases['b21'],
        weights['w22'],biases['b22'],pool=True)
    out1,res1 = down_sample(out1,weights['w31'],biases['b31'],
        weights['w32'],biases['b32'],pool=True)
    out1,res1 = down_sample(out1,weights['w41'],biases['b41'],
        weights['w42'],biases['b42'],pool=True)
    out1 = down_sample(out1,weights['w51'],biases['b51'],
        weights['w52'],biases['b52'],pool=False)
    up1 = up_sample(out1,weights['wu1'],biases['bu1'])
    up1 = up_sample(up1,weights['wu2'],biases['bu2'])
    up1 = up_sample(up1,weights['wu3'],biases['bu3'])
    up1 = up_sample(up1,weights['wu4'],biases['bu4'])
    out = tf.nn.conv2d(up1,weights['wf'],strides=[1,1,1,1],
        padding='SAME')
    out = tf.nn.bias_add(out,biases['bf'])
    return out

## Создание генераторов для предобработки изображений и
## формирования пакетов, доступных на этапе выполнения,
## вместо загрузки всех изображений и меток в память.

# Задание необходимых каталогов

# Содержит входные тренировочные данные
data_dir = "/home/santanu/Downloads/Carvana/train/"
# Содержит целевые метки
mask_dir = "/home/santanu/Downloads/Carvana/train_masks/"
all_images = os.listdir(data_dir)
# Отбор изображений для тестирования и валидации
train_images, validation_images = train_test_split(all_images,
        train_size=0.8, test_size=0.2)
# Вспомогательная функция для преобразования изображений из
# градаций серого в RGB
def grey2rgb(img):
    new_img = []
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            new_img.append(list(img[i][j])*3)
    new_img = np.array(new_img).reshape(img.shape[0], img.shape[1], 3)
    return new_img

# Генератор, который мы будем использовать для чтения
```

```
# данных из каталога
def data_gen_small(data_dir, mask_dir, images, batch_size, dims):
    """
    data_dir: место хранения фактических изображений
    mask_dir: место хранения фактических масок
    images: имена файлов изображений, из которых мы хотим
            генерировать пакеты
    batch_size: размер пакета
    dims: размеры, до которых мы хотим довести изображения
    """
    while True:
        ix = np.random.choice(np.arange(len(images)), batch_size)
        imgs = []
        labels = []
        for i in ix:
            # изображения
            original_img = load_img(data_dir + images[i])
            resized_img = imresize(original_img, dims+[3])
            array_img = img_to_array(resized_img)/255
            imgs.append(array_img)

            # маски
            original_mask = load_img(mask_dir +
                    images[i].split(".")[0] + '_mask.gif')
            resized_mask = imresize(original_mask, dims+[3])
            array_mask = img_to_array(resized_mask)/255
            labels.append(array_mask[:, :, 0])
        imgs = np.array(imgs)
        labels = np.array(labels)
        yield imgs, labels.reshape(-1, dims[0], dims[1], 1)

train_gen = data_gen_small(data_dir, mask_dir,
        train_images,32, [128, 128])
validation_gen = data_gen_small(data_dir, mask_dir,
        validation_images,32, [128, 128])

display_step=10
learning_rate=0.0001

keep_prob = tf.placeholder(tf.float32)
logits = unet_basic(x,weights,biases)
flat_logits = tf.reshape(tensor=logits, shape=(-1, 1))
flat_labels = tf.reshape(tensor=y,shape=(-1, 1))
cross_entropies = tf.nn.
        sigmoid_cross_entropy_with_logits(logits=flat_logits,
        labels=flat_labels)

cost = tf.reduce_mean(cross_entropies)
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).
```

```
        minimize(cost)

# Вычисление модели

## Инициализация всех переменных

init = tf.global_variables_initializer()

## Запуск вычислительного графа

with tf.Session() as sess:
    sess.run(init)
    for batch in xrange(500):
        batch_x,batch_y= next(train_gen)
        sess.run(optimizer, feed_dict={x:batch_x,y:batch_y})
        loss = sess.run([cost],feed_dict={x:batch_x,y:batch_y})
        ## Валидация потерь и сохранение результата для
        ## отображения по завершении вычислений
        val_x,val_y = next(validation_gen)
        loss_val = sess.run([cost],feed_dict={x:val_x,y:val_y})
        out_x = sess.run(logits,feed_dict={x:val_x})
        print('batch:',batch,'train loss:',loss,
              'validation loss:',loss_val)

## Чтобы оценить качество сегментации модели, мы отображаем
## сегментацию для пары валидационных изображений.
## Эти валидационные изображения оценивались в последнем
## тренировочном пакете. Заметьте, что модель не тренировалась
## на этих валидационных изображениях.

img = (out_x[1] > 0.5)*1.0
plt.imshow(grey2rgb(img),alpha=0.5)
plt.imshow(val_x[1])
plt.imshow(grey2rgb(val_y[1]), alpha=0.5)
img = (out_x[2] > 0.5)*1.0
plt.imshow(grey2rgb(img),alpha=0.5)
plt.imshow(val_x[2])
plt.imshow(grey2rgb(val_y[2]), alpha=0.5)

-- вывод --

''batch:', 400, 'train loss:', [0.044453222], 'validation loss:',
    [0.058442257])
('batch:', 401, 'train loss:', [0.049510699], 'validation loss:',
    [0.055530164])
('batch:', 402, 'train loss:', [0.048047166], 'validation loss:',
    [0.055518236])
('batch:', 403, 'train loss:', [0.049462996], 'validation loss:',
    [0.049190756])
```

```
('batch:', 404, 'train loss:', [0.047011156], 'validation loss:',
    [0.051120583])
('batch:', 405, 'train loss:', [0.046235155], 'validation loss:',
    [0.052921098])
('batch:', 406, 'train loss:', [0.051339123], 'validation loss:',
    [0.054767497])
('batch:', 407, 'train loss:', [0.050004266], 'validation loss:',
    [0.052718181])
('batch:', 408, 'train loss:', [0.048425209], 'validation loss:',
    [0.054115709])
('batch:', 409, 'train loss:', [0.05234601], 'validation loss:',
    [0.053246532])
('batch:', 410, 'train loss:', [0.054224499], 'validation loss:',
    [0.05121265])
('batch:', 411, 'train loss:', [0.050268434], 'validation loss:',
    [0.056970511])
('batch:', 412, 'train loss:', [0.046658799], 'validation loss:',
    [0.058863375])
('batch:', 413, 'train loss:', [0.048009872], 'validation loss:',
    [0.049314644])
('batch:', 414, 'train loss:', [0.053399611], 'validation loss:',
    [0.050949663])
('batch:', 415, 'train loss:', [0.047932044], 'validation loss:',
    [0.049477436])
('batch:', 416, 'train loss:', [0.054921247], 'validation loss:',
    [0.059221379])
('batch:', 417, 'train loss:', [0.053222295], 'validation loss:',
    [0.061699588])
('batch:', 418, 'train loss:', [0.047465689], 'validation loss:',
    [0.051628478])
('batch:', 419, 'train loss:', [0.055220582], 'validation loss:',
    [0.056656662])
('batch:', 420, 'train loss:', [0.052862987], 'validation loss:',
    [0.048487194])
('batch:', 421, 'train loss:', [0.052869596], 'validation loss:',
    [0.049040388])
('batch:', 422, 'train loss:', [0.050372943], 'validation loss:',
    [0.052676879])
('batch:', 423, 'train loss:', [0.048104074], 'validation loss:',
    [0.05687784])
('batch:', 424, 'train loss:', [0.050506901], 'validation loss:',
    [0.055646997])
('batch:', 425, 'train loss:', [0.042940177], 'validation loss:',
    [0.047789834])
('batch:', 426, 'train loss:', [0.04780338], 'validation loss:',
    [0.05711592])
('batch:', 427, 'train loss:', [0.051617432], 'validation loss:',
    [0.051806655])
('batch:', 428, 'train loss:', [0.047577277], 'validation loss:',
```

```
    [0.052631289])
('batch:', 429, 'train loss:', [0.048690431], 'validation loss:',
    [0.044696849])
('batch:', 430, 'train loss:', [0.046005826], 'validation loss:',
    [0.050702494])
('batch:', 431, 'train loss:', [0.05022176], 'validation loss:',
    [0.053923506])
('batch:', 432, 'train loss:', [0.041961089], 'validation loss:',
    [0.047880188])
('batch:', 433, 'train loss:', [0.05004932], 'validation loss:',
    [0.057072558])
('batch:', 434, 'train loss:', [0.04603707], 'validation loss:',
    [0.049482994])
('batch:', 435, 'train loss:', [0.047554974], 'validation loss:',
    [0.050586618])
('batch:', 436, 'train loss:', [0.046048313], 'validation loss:',
    [0.047748547])
('batch:', 437, 'train loss:', [0.047006462], 'validation loss:',
    [0.059268739])
('batch:', 438, 'train loss:', [0.045432612], 'validation loss:',
    [0.051733252])
('batch:', 439, 'train loss:', [0.048241541], 'validation loss:',
    [0.04774794])
('batch:', 440, 'train loss:', [0.046124499], 'validation loss:',
    [0.048809234])
('batch:', 441, 'train loss:', [0.049743906], 'validation loss:',
    [0.051254783])
('batch:', 442, 'train loss:', [0.047674596], 'validation loss:',
    [0.048125759])
('batch:', 443, 'train loss:', [0.048984651], 'validation loss:',
    [0.04512443])
('batch:', 444, 'train loss:', [0.045365792], 'validation loss:',
    [0.042732101])
('batch:', 445, 'train loss:', [0.046680171], 'validation loss:',
    [0.050935686])
('batch:', 446, 'train loss:', [0.04224021], 'validation loss:',
    [0.052455597])
('batch:', 447, 'train loss:', [0.045161027], 'validation loss:',
    [0.045499101])
('batch:', 448, 'train loss:', [0.042469904], 'validation loss:',
    [0.050128322])
('batch:', 449, 'train loss:', [0.047899902], 'validation loss:',
    [0.050441738])
('batch:', 450, 'train loss:', [0.043648213], 'validation loss:',
    [0.048811793])
('batch:', 451, 'train loss:', [0.042413067], 'validation loss:',
    [0.051744446])
('batch:', 452, 'train loss:', [0.047555752], 'validation loss:',
    [0.04977461])
```

```
('batch:', 453, 'train loss:', [0.045962822], 'validation loss:',
    [0.047307629])
('batch:', 454, 'train loss:', [0.050115541], 'validation loss:',
    [0.050558448])
('batch:', 455, 'train loss:', [0.045722887], 'validation loss:',
    [0.049715079])
('batch:', 456, 'train loss:', [0.042583987], 'validation loss:',
    [0.048713747])
('batch:', 457, 'train loss:', [0.040946022], 'validation loss:',
    [0.045165032])
('batch:', 458, 'train loss:', [0.045971408], 'validation loss:',
    [0.046652604])
('batch:', 459, 'train loss:', [0.045015588], 'validation loss:',
    [0.055410333])
('batch:', 460, 'train loss:', [0.045542594], 'validation loss:',
    [0.047741935])
('batch:', 461, 'train loss:', [0.04639449], 'validation loss:',
    [0.046171311])
('batch:', 462, 'train loss:', [0.047501944], 'validation loss:',
    [0.046123035])
('batch:', 463, 'train loss:', [0.043643478], 'validation loss:',
    [0.050230302])
('batch:', 464, 'train loss:', [0.040434662], 'validation loss:',
    [0.046641909])
('batch:', 465, 'train loss:', [0.046465941], 'validation loss:',
    [0.054901786])
('batch:', 466, 'train loss:', [0.049838047], 'validation loss:',
    [0.048461676])
('batch:', 467, 'train loss:', [0.043582849], 'validation loss:',
    [0.052996978])
('batch:', 468, 'train loss:', [0.050299261], 'validation loss:',
    [0.048585847])
('batch:', 469, 'train loss:', [0.046049926], 'validation loss:',
    [0.047540378])
('batch:', 470, 'train loss:', [0.042139661], 'validation loss:',
    [0.047782935])
('batch:', 471, 'train loss:', [0.046433724], 'validation loss:',
    [0.049313426])
('batch:', 472, 'train loss:', [0.047063917], 'validation loss:',
    [0.045388222])
('batch:', 473, 'train loss:', [0.045556825], 'validation loss:',
    [0.044953942])
('batch:', 474, 'train loss:', [0.046181824], 'validation loss:',
    [0.045763671])
('batch:', 475, 'train loss:', [0.047123503], 'validation loss:',
    [0.047637179])
('batch:', 476, 'train loss:', [0.046167117], 'validation loss:',
    [0.051462833])
('batch:', 477, 'train loss:', [0.043556783], 'validation loss:',
```

```
    [0.044357236])
('batch:', 478, 'train loss:', [0.04773742], 'validation loss:',
    [0.046332739])
('batch:', 479, 'train loss:', [0.04820114], 'validation loss:',
    [0.045707334])
('batch:', 480, 'train loss:', [0.048089884], 'validation loss:',
    [0.052449297])
('batch:', 481, 'train loss:', [0.041174423], 'validation loss:',
    [0.050378591])
('batch:', 482, 'train loss:', [0.049479648], 'validation loss:',
    [0.047861829])
('batch:', 483, 'train loss:', [0.041197944], 'validation loss:',
    [0.051383432])
('batch:', 484, 'train loss:', [0.051363751], 'validation loss:',
    [0.050520841])
('batch:', 485, 'train loss:', [0.047751397], 'validation loss:',
    [0.046632469])
('batch:', 486, 'train loss:', [0.049832929], 'validation loss:',
    [0.048640732])
('batch:', 487, 'train loss:', [0.049518026], 'validation loss:',
    [0.048658002])
('batch:', 488, 'train loss:', [0.051349726], 'validation loss:',
    [0.051405452])
('batch:', 489, 'train loss:', [0.041912809], 'validation loss:',
    [0.046458714])
('batch:', 490, 'train loss:', [0.047130216], 'validation loss:',
    [0.052001398])
('batch:', 491, 'train loss:', [0.041481428], 'validation loss:',
    [0.046243563])
('batch:', 492, 'train loss:', [0.042776003], 'validation loss:',
    [0.042228915])
('batch:', 493, 'train loss:', [0.043606419], 'validation loss:',
    [0.048132997])
('batch:', 494, 'train loss:', [0.047129884], 'validation loss:',
    [0.046108384])
('batch:', 495, 'train loss:', [0.043634158], 'validation loss:',
    [0.046292961])
('batch:', 496, 'train loss:', [0.04454672], 'validation loss:',
    [0.044108659])
('batch:', 497, 'train loss:', [0.048068151], 'validation loss:',
    [0.044547819])
('batch:', 498, 'train loss:', [0.044967934], 'validation loss:',
    [0.047069982])
('batch:', 499, 'train loss:', [0.041554678], 'validation loss:',
    [0.051807735])
```

Средние значения потерь на стадии тренировки и на стадии проверки почти одинаковы, что указывает на отсутствие переобучения модели и ее хорошую обобщаемость. Как показывает сравнение сегментированных изображений с целевыми при-

знаками, представленными на рис. 6.15, *а*, результаты выглядят убедительными. Для этой сети использовались изображения с пространственными размерами 128×128. После увеличения размеров входных изображений до 512×512 точность и качество сегментации значительно улучшились. Поскольку это полносверточная сеть, в которой отсутствуют полностью связанные слои, обработка изображений с новым размером требует внесения в сеть лишь небольших изменений. Чтобы проиллюстрировать тот факт, что изображения с бо́льшим размером чаще всего более предпочтительны в задачах сегментации, поскольку они позволяют захватить больше контекста, на рис. 6.15, *б*, представлен выход сегментации для двух валидационных наборов.

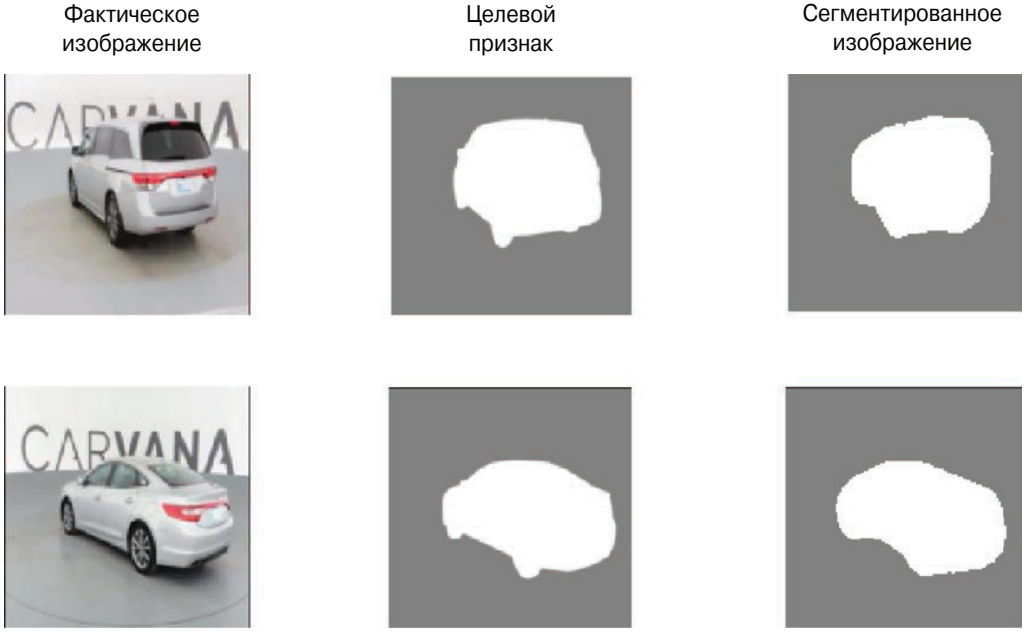


Рис. 6.15, а. Результаты сегментации на валидационном наборе данных, полученные с помощью модели, которая обучалась на изображениях размером 128×128

Сети классификации и локализации изображений

Все классифицирующие модели прогнозируют класс объекта на изображении, но фактически ничего не говорят нам о его местоположении. Для представления местоположения объекта на изображении можно использовать ограничительный прямоугольник. Если изображения аннотированы ограничительными прямоугольниками и доступна информация об их выходном классе, то можно обучить модель предсказывать эти ограничительные прямоугольники вместе с классом объекта. Для представления прямоугольников можно использовать четыре числа, два из которых соответ-

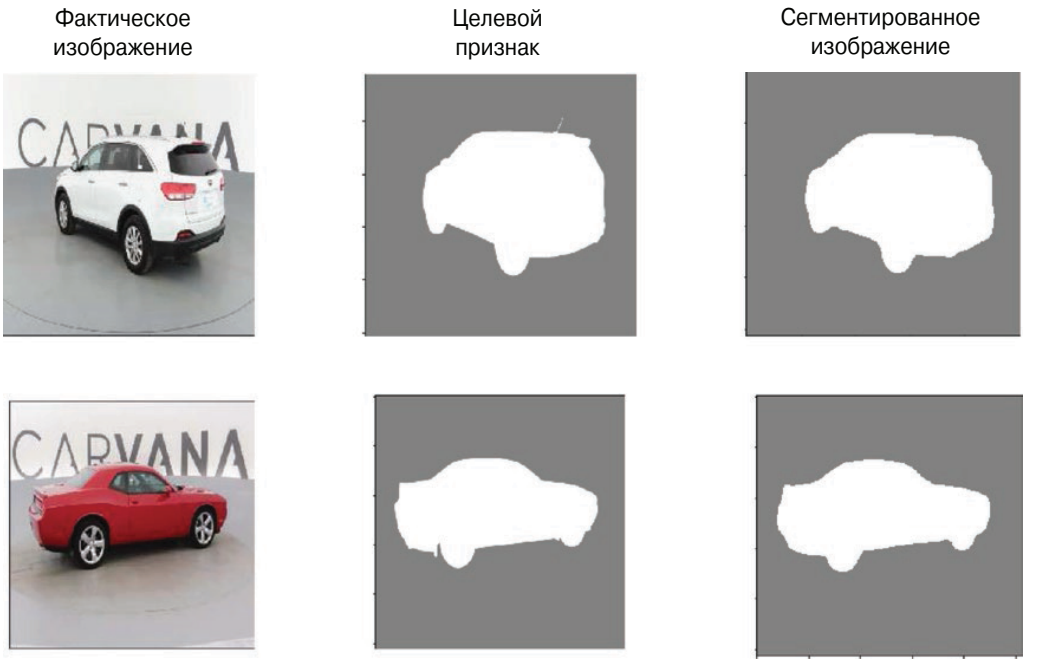


Рис. 6.15, б. Результаты сегментации на валидационном наборе данных, полученные с помощью модели, которая обучалась на изображениях размером 512×512

ствуют пространственным координатам левого верхнего угла, а остальные два — высоту и ширину прямоугольника. В таком случае эти четыре числа могут прогнозироваться с помощью регрессии. Одну сверточную сеть можно использовать для классификации, а другую — для предсказания атрибутов ограничительного прямоугольника посредством регрессии. Однако обычно одна и та же сверточная сеть используется для предсказания как класса объекта, так и местоположения ограничительного прямоугольника. Сеть CNN вплоть до последнего полносвязного слоя останется той же, но на выходе, наряду с элементами, соответствующими классам различных объектов, должны присутствовать четыре дополнительных элемента, соответствующих атрибутам ограничительного прямоугольника. Этот способ предсказания ограничительных прямоугольников вокруг объектов на изображениях называется *локализацией*. На рис. 6.16 представлена классифицирующая и локализующая сеть для обработки изображений кошек и собак. В сети этого типа используется априорное предположение относительно того, что на изображении имеется только один класс объектов.

Функцией стоимости для этой сети должна быть комбинация функции потерь/стоимости по различным классам объектов и регрессионная функция стоимости, связанная с предсказанием атрибутов ограничительного прямоугольника. Поскольку функция стоимости, подлежащая оптимизации, является многозадачной целевой функцией, необходимо определиться с тем, какой вес приписать каждой задаче. Это очень важно, поскольку различные функции стоимости, связанные с данными задача-

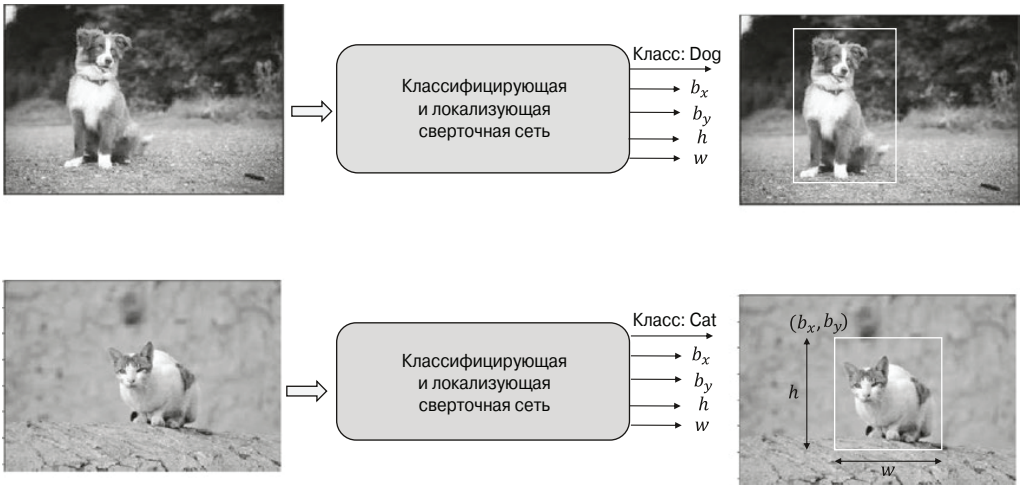


Рис. 6.16. Классифицирующая и локализующая сеть

ми, — скажем, категориальная кросс-энтропия для классификации и среднеквадратическая ошибка для регрессии, — будут иметь различные шкалы, и потому оптимизация может пойти "вразнос", если при формировании полной стоимости эти стоимости не будут взвешены подходящим образом. Стоимости должны нормализоваться путем приведения к общей шкале, после чего им должны быть назначены веса в зависимости от сложности задачи. Обозначим через θ параметры сверточной нейронной сети, относящиеся к n классам и ограничительному прямоугольнику, определяемому четырьмя числами. Пусть выходной класс представлен вектором $y = [y_1 y_2 \dots y_n]^T \in [0, 1]^{n \times 1}$, поскольку каждый из $y_j \in [0, 1]$. Кроме того, пусть параметры ограничительного прямоугольника представлены вектором $s = [s_1 s_2 s_3 s_4]^T$, где s_1 и s_2 обозначают координаты пикселя левого верхнего угла ограничительного прямоугольника, а s_3 и s_4 обозначают его высоту и ширину. Если вектор $p = [p_1 p_2 \dots p_n]^T$ представляет вероятности классов, а вектор $t = [t_1 t_2 t_3 t_4]^T$ — предсказанные атрибуты ограничительного прямоугольника, то функцию стоимости или потерь можно определить следующей формулой:

$$c(\theta) = -\alpha \sum_{j=1}^{n} y_j \log p_j + \beta \sum_{j=1}^{4} (s_j - t_j)^2 .$$

Первый член в этом выражении — категориальная кросс-энтропия для функции активации SoftMax по n классам, второй — регрессионная функция стоимости, связанная с предсказанием атрибутов ограничительного прямоугольника. Параметры α и β — гиперпараметры сети, которые должны настраиваться для получения разумных результатов. Для мини-пакета по m точкам данных функция стоимости может быть выражена в следующем виде:

$$C(\theta) = \frac{1}{m}\left[-\alpha \sum_{i=1}^{m}\sum_{j=1}^{n} y_j^{(i)} \log p_j^{(i)} + \beta \sum_{i=1}^{m}\sum_{j=1}^{4} (s_j^{(i)} - t_j^{(i)})^2 \right],$$

где i представляет различные изображения.

Предыдущую функцию стоимости можно минимизировать методом градиентного спуска. Попутно следует отметить, что, сравнивая функционирование различных версий этой сети с различными значениями параметров (α, β), не следует сравнивать стоимости, связанные с этими сетями, используя их в качестве критериев для выбора наилучшей сети. Вместо этого следует задействовать другие метрики, такие как точность, F1-оценка, площадь под кривой и т.п., для задачи классификации, а метрики наподобие площади области перекрывания предсказанного и целевого ограничительных прямоугольников — для задачи локализации.

Обнаружение объектов

Как правило, изображение содержит не один, а несколько интересующих нас объектов. Существует множество приложений, в которых применяется возможность обнаружения нескольких объектов на изображениях. Например, обнаружение объектов можно использовать для подсчета количества людей в различных отделах магазина с целью анализа покупательского спроса. Таким же способом можно подсчитывать количество автомобилей, остановившихся на красный сигнал светофора, чтобы получить грубую оценку интенсивности дорожного движения на этом перекрестке. Еще одной сферой, в которой обнаружение объектов может принести пользу, являются автоматизированные системы наблюдения на предприятиях, способные подавать предупреждающие сигналы при возникновении событий, создающих опасные ситуации. Можно непрерывно получать изображения критических участков территории завода и выявлять события, создающие ту или иную угрозу безопасности, применяя методы обнаружения объектов на изображениях. Например, если рабочий, обслуживающий оборудование, забудет надеть полагающиеся по технике безопасности перчатки, защитные очки или шлем, то в случае отсутствия этих объектов на изображении будет подан соответствующий сигнал.

Обнаружение нескольких объектов на изображении — это классическая задача компьютерного зрения. Прежде всего, мы не можем использовать классифицирующую или локализующую сеть или любые их варианты, поскольку количество объектов на изображениях может меняться. Чтобы сориентироваться в том, как подступиться к решению задачи обнаружения объектов, начнем с весьма наивного подхода. Мы можем случайным образом выбирать фрагменты существующего изображения, используя простейший метод скользящего окна, и передавать их сети, предварительно обученной классификации и локализации объектов. Рис. 6.17 иллюстрирует применение подхода на основе скользящего окна для обнаружения нескольких объектов на изображении.

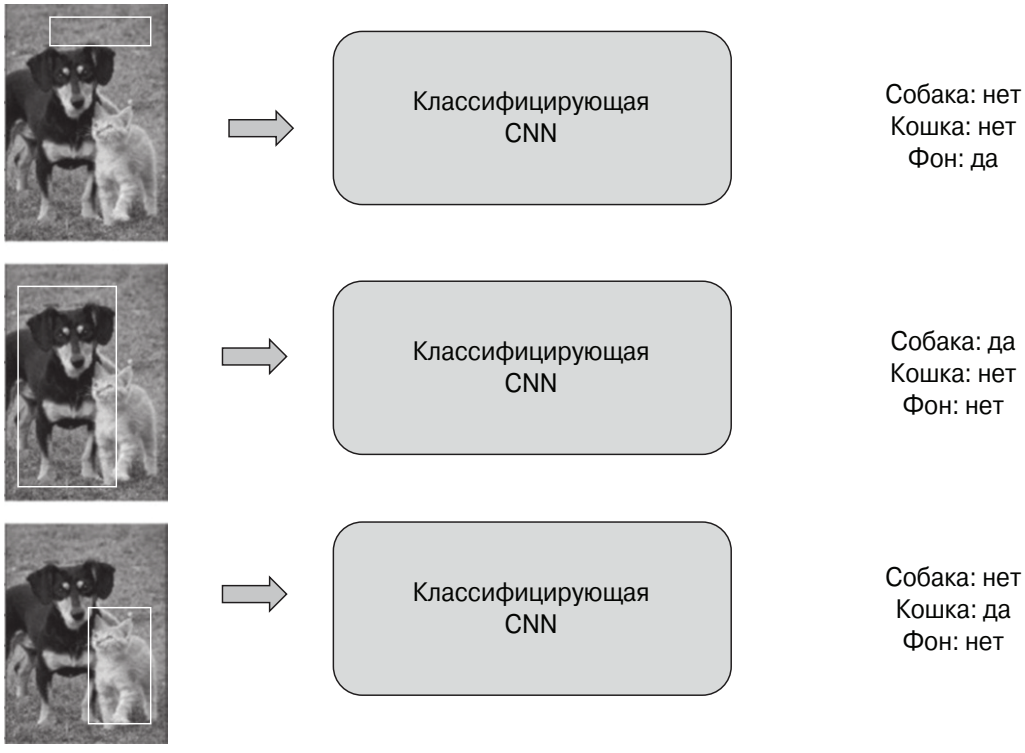


Рис. 6.17. Применение метода скользящего окна для обнаружения объектов

Несмотря на то что этот метод сработает, он либо потребует огромных вычислительных затрат, либо вообще окажется практически нереализуемым, поскольку потребуется проверять тысячи фрагментов изображений в различных местоположениях и масштабах, если подходящая область изображения неизвестна заранее. Существующие улучшенные методы основаны на предложении нескольких вероятных областей расположения объектов и последующей их передаче классифицирующей и локализующей сети. Один из таких методов обнаружения объектов под названием *R-CNN* обсуждается в следующем разделе.

Сеть R-CNN

Буква “R” в названии R-CNN означает *region* (область, зона) и указывает на то, что в данном методе используется предварительная информация о *предполагаемом местоположении* обнаруживаемых объектов. Возможные области обнаружения объектов обычно получают посредством алгоритма так называемого *селективного поиска*. Как правило, селективный поиск выдает около 2000 предложений об областях изображения, которые могут представлять интерес. Для селективного поиска перспективных областей, которые могут содержать объекты, обычно используют традицион-

ные способы обработки изображений. Общее описание процедуры пошагового селективного поиска приведено ниже.

- Генерируются многочисленные области изображения, каждая из которых может принадлежать только к одному классу.
- Меньшие области рекурсивно комбинируются в более крупные посредством *жадного* алгоритма. На каждом этапе сливаются две области, обладающие наибольшим сходством. Этот процесс повторяется до тех пор, пока не останется только одна область. В ходе этого процесса создается иерархия все более крупных областей, что позволяет алгоритму предлагать широкий набор вероятных областей изображения, где могут быть обнаружены объекты. Сгенерированные таким способом области используются в качестве рекомендованных предложений

Вышеупомянутые 2000 областей, представляющие интерес, передаются классифицирующей и локализующей сети для предсказания классов объектов и ассоциированных с ними ограничительных прямоугольников. Классифицирующая сеть представляет собой сверточную нейронную сеть, после которой применяется метод опорных векторов (SVM) для окончательной классификации. Высокоуровневая архитектура R-CNN представлена на рис. 6.18.

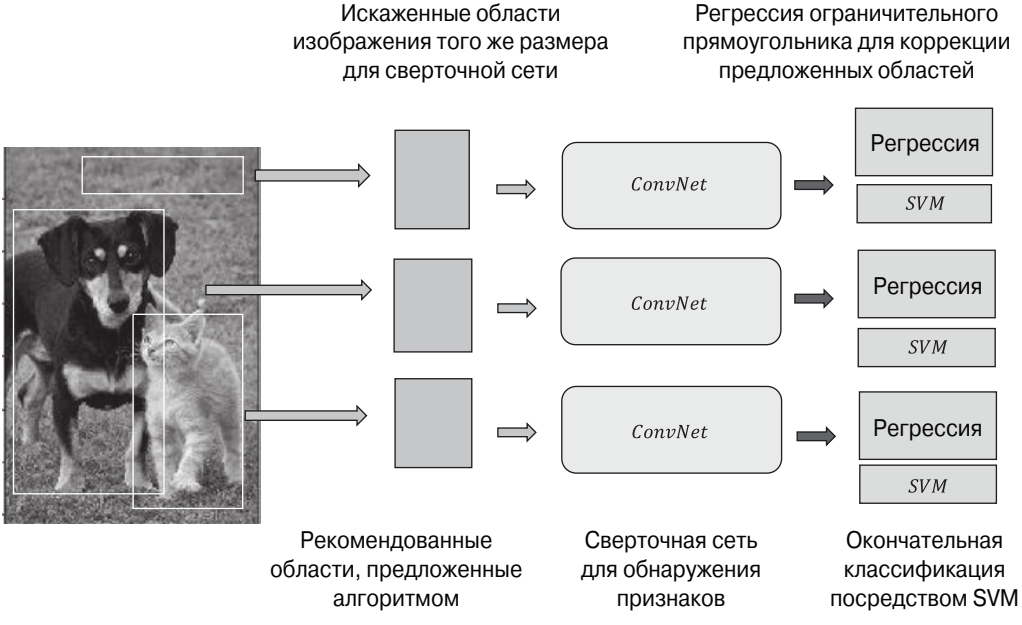


Рис. 6.18. Сеть R-CNN

Шаги высокоуровневой процедуры тренировки R-CNN описаны ниже.

- Берем предобученную сеть ImageNet CNN, такую как AlexNet, и заново тренируем последний полносвязный слой на объектах, подлежащих обнаружению, вместе с их фоном.
- Получаем все области изображений (2000 областей на одно изображение), предложенные селективным поиском, изменяем их размер, чтобы он соответствовал размеру на входе CNN, обрабатываем их посредством CNN, а затем сохраняем признаки на диске для дальнейшей обработки. Обычно на диске сохраняются выходные карты слоя субдискретизации.
- Тренируем SVM для классификации объекта или фона на основании признаков, полученных от CNN. Для каждого класса объектов должен быть предусмотрен свой модуль SVM, который учится отличать конкретный объект от фона.
- Наконец, выполняем регрессию для ограничительного прямоугольника, корректирующую рекомендованные предложения областей.

Несмотря на то что R-CNN хорошо справляется с обнаружением объектов, эта архитектура не лишена ряда недостатков, которые перечислены ниже.

- Одной из проблем R-CNN является огромное количество предлагаемых областей-кандидатов, что замедляет работу сети, поскольку каждая такая область должна независимо пропускаться через сверточные нейронные сети. Кроме того, предлагаемые области фиксированы, и сеть R-CNN не обучается им.
- Предсказания местоположений и ограничительных прямоугольников объектов продуцируются разными моделями, поэтому в процессе тренировки модели сеть не обучается какой-либо специфике локализации объектов на основе тренировочных данных.
- В задаче классификации признаки, сгенерированные на выходе сверточной нейронной сети, используются для тонкой настройки модулей SVM, что увеличивает объем вычислений.

Сети Fast и Faster R-CNN

В сети Fast R-CNN вычислительные трудности, свойственные R-CNN, удается частично преодолеть за счет использования общего пути свертки для всего изображения вплоть до определенного количества слоев, после прохождения которых рекомендованные области изображения проецируются на выходные карты признаков, и соответствующие области извлекаются для дальнейшей обработки посредством полносвязных слоев с их последующей окончательной классификацией. Извлечение соответствующих рекомендованных областей из карт признаков на выходе свертки и изменение их размера для приведения его в соответствие с размером, используемым полносвязным слоем, выполняется посредством операции субдискретизации, известной под названием *ROI-пулинга* (от “region of interest” — интересующая область). Архитектурная диаграмма сети Fast R-CNN представлена на рис. 6.19.

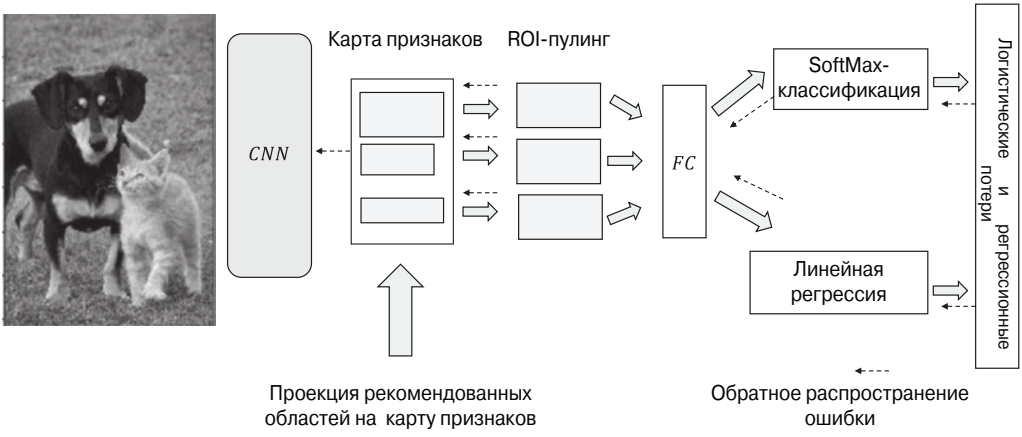


Рис. 6.19. Архитектурная диаграмма сети Fast R-CNN

Сеть Fast R-CNN значительно снижает накладные расходы, связанные с многократным выполнением операции свертки (2000 областей на одно изображение, по числу областей, предоставляемых селективным поиском) в сети R-CNN. Однако рекомендуемые области по-прежнему предлагаются внешними алгоритмами наподобие селективного поиска. В силу этой зависимости от внешних алгоритмов узким местом сети Fast R-CNN является вычисление таких областей, представляющих интерес. Сеть должна ожидать поступления внешних предложений, прежде чем сможет двигаться дальше. Указанного недостатка лишена сеть Faster R-CNN, в которой предложения областей формируются в самой сети, что устраняет зависимость от внешних алгоритмов. Архитектура Faster R-CNN почти совпадает с архитектурой Fast R-CNN, за исключением одного нового элемента — рекомендательной сети, формирующей предложения относительно областей изображения, представляющих интерес, тем самым избавляя от необходимости использовать для этого какие-либо внешние схемы, такие как селективный поиск.

Генеративно-состязательные сети

Генеративно-состязательные сети (generative adversarial networks — GAN) — одно из наибольших достижений в области глубокого обучения за последнее время. Ян Гудфеллоу и его коллеги представили эту архитектуру в 2014 году на конференции NIPS в своем докладе под названием “Generative Adversarial Networks”, с которым можно ознакомиться по адресу `https://arxiv.org/abs/1406.2661`. С тех пор интерес к генеративно-состязательным сетям не ослабевает, и в этом направлении ведутся интенсивные разработки. В действительности Ян Лекун, один из самых известных экспертов в области глубокого обучения, считает появление генеративно-состязательных сетей важнейшим прорывом в глубоком обучении за последние годы. Сети GAN используются в качестве генеративных моделей для получения синтетиче-

ских данных в соответствии с требуемым распределением. Они применяются в целом ряде областей, таких как генерация изображений, улучшение фотографий, абстрактные умозаключения, семантическая сегментация, генерация видео, перенос стиля из одной предметной области в другую, генерация изображений по текстовому описанию и многое другое.

Работа генеративно-состязательных сетей основана на использовании двух агентов *игры с нулевой суммой* (zero-sum game), или *антагонистической игры*, из теории игр. Генеративно-состязательная сеть состоит из двух нейронных сетей: генератора (G) и дискриминатора (D), соперничающих между собой. Генератор (G) пытается сбить дискриминатора (D) с толку, чтобы тот не смог отличать реальные данные из распределения от поддельных, сгенерированных генератором (G). Точно так же дискриминатор (D) учится отличать реальные данные от поддельных, сгенерированных генератором (G). По прошествии определенного времени как дискриминатор, так и генератор совершенствуют свои навыки в решении возложенных на них задач, соревнуясь друг с другом. Получение оптимального решения этой задачи теории игр основывается на концепции *равновесия Нэша* (Nash equilibrium), когда генератор учится производить поддельные данные, имеющие то же распределение, что и оригинальные данные, тогда как дискриминатор создает на выходе вероятность 1/2 как для реальных, так и для поддельных данных.

Прежде всего возникает очевидный вопрос: каким образом конструируются поддельные данные? Их конструирует модель на основе генеративной нейронной сети (G) посредством семплирования шума z из априорного распределения P_z. Если фактические данные x следуют распределению P_x, а поддельные данные $G(z)$, выдаваемые генератором, следуют распределению P_g, то в состоянии равновесия $P_x(x)$ должно совпадать с $P_g(G(z))$, т.е.

$$P_g\left(G\left(z\right)\right) \sim P_x\left(x\right).$$

Поскольку при равновесии распределение поддельных данных почти совпадает с распределением реальных данных, генератор должен научиться семплировать такие поддельные данные, которые будет трудно отличить от реальных. Кроме того, при равновесии дискриминатор (D) должен создавать на выходе значение 1/2 в качестве вероятности для обоих классов — как реальных, так и поддельных данных. Прежде чем мы перейдем к рассмотрению математических выкладок для генеративно-состязательной сети, целесообразно пролить некоторый свет на понятия *игра с нулевой суммой*, *равновесие Нэша* и *минимаксная игра*.

На рис. 6.20 представлена генеративно-состязательная сеть, содержащая две нейронные сети — генератор (G) и дискриминатор (D), которые соревнуются между собой.

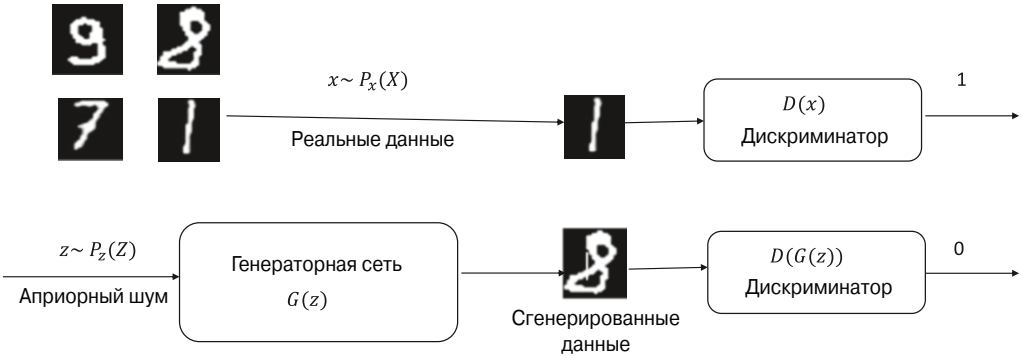


Рис. 6.20. Базовая иллюстрация состязательной сети

Задачи максимина и минимакса

В играх с несколькими участниками каждый из них стремится максимизировать свою выгоду и увеличить свои шансы на выигрыш. Если мы рассматриваем игру с N соперниками, то оптимальная стратегия кандидата i состоит в том, чтобы получить максимальный выигрыш при условии, что остальные $N - 1$ участников играют так, чтобы помешать ему в этом. Стратегия игрока i, при которой его минимальный выигрыш будет наибольшим из всех возможных, независимо от действий остальных игроков, называется *максиминной стратегией*, а наибольший из минимальных выигрышей — *максиминным значением*. Если игрок i придерживается максиминной стратегии, то его выигрыш будет не меньше максиминного значения. Таким образом, максиминная стратегия s_i^* и максиминное значение L_i^* можно выразить в следующем виде:

$$s_i^* = \underbrace{\operatorname{argmax}}_{s_i} \underbrace{\min}_{s_{-i}} L_i(s_i, s_{-i}),$$

$$L_i^* = \underbrace{\max}_{s_i} \underbrace{\min}_{s_{-i}} L_i(s_i, s_{-i}).$$

Максиминная стратегия игрока i легче поддается интерпретации, если считать, что ему известны ходы противника и он знает, что они будут пытаться минимизировать его наибольший выигрыш из всех возможных при каждом его ходе. Следовательно, при данном предположении игрок i должен делать ход, который принесет максимальный из всех возможных минимальных выигрышей при каждом его ходе.

Объяснить минимаксную стратегию в этой парадигме легче, чем в соответствующих ей технических терминах. В соответствии с *минимаксной стратегией* игрок i должен предполагать, что другие игроки, обозначенные как $-i$, будут позволять ему получать лишь наименьший из всех возможных максимальных выигрышей при каждом из их ходов. В этом случае для игрока i логично выбирать ход, который принесет ему максимальный из всех возможных выигрышей, которые другие игроки установи-

ли для i при каждом своем ходе. В условиях минимаксной стратегии выигрыш игрока i задается формулой

$$L_i^* = \underbrace{\min}_{s_{-i}} \underbrace{\max}_{s_i} L_i(s_i, s_{-i}).$$

Обратите внимание на то, что окончательные выигрыши или потери игроков, когда все они сделали свои ходы, могут отличаться от максиминного или минимаксного значений.

Попробуем пояснить максиминную задачу интуитивно понятным примером, когда два агента, A и B, соревнуются между собой в максимизации своих выигрышей в ходе игры. Кроме того, предположим, что A может сделать три хода, L_1, L_2, L_3, тогда как B — два хода, M_1 и M_2. Соответствующая таблица выигрышей приведена на рис. 6.21. В каждой ячейке первое значение соответствует выигрышу игрока A, второе — выигрышу игрока B.

		B	
		M_1	M_2
A	L_1	$(6, 2)$	$(4, -41)$
	L_2	$(10, 0.5)$	$(-20.5, 2)$
	L_3	$(-201, 4.1)$	$(8, 8)$

Рис. 6.21. Максиминная и минимаксная стратегии для двух игроков

Сначала предположим, что оба игрока придерживаются максиминной стратегии, т.е. они должны делать ходы, максимизирующие их выигрыш, ожидая, что противник будет стараться по возможности минимизировать их выигрыш.

Для игрока А максиминная стратегия будет заключаться в выборе хода L_1, при котором минимальным значением A будет 4. Если A выберет ход L_2, то он рискует проиграть –20,5, тогда как при выборе хода L_3 он может получить еще худший результат: –201. Поэтому максиминнным значением для A является максимальное из всех возможных минимальных значений в каждой строке, т.е. 4, соответствующее стратегии L_1.

Для игрока B максиминная стратегия будет заключаться в выборе хода M_1, поскольку в этом случае минимальное значение, которое получит B, равно 0,5. Если B выберет ход M_2, то он рискует получить значение –41. Поэтому максиминным значением для B является максимальное из всех возможных минимальных значений в столбцах, т.е. значение 0,5, соответствующее ходу M_1.

Предположим, что оба игрока, A и B, делали бы свои ходы в соответствии со своими максиминными стратегиями, т.е. (L_1, M_1). В этом случае выигрышем A будет 6, а

выигрышем B — 2. Мы видим, что максиминные значения отличаются от фактических значений выигрышей, коль скоро игроки придерживаются своих максиминных стратегий.

Рассмотрим теперь ситуацию, когда оба игрока хотят придерживаться минимаксной стратегии. В соответствии с этим каждый игрок выбирает стратегию, приводящую его к максимальному выигрышу, которым является минимальное из всех возможных максимальных значений выигрыша, определяемых противником на каждом из его ходов.

Рассмотрим минимаксное значение и минимаксную стратегию игрока A. Если игрок B выберет ход M_1, то максимум, который может получить A, будет равен 10, а если B выберет ход M_2, то максимум, который может получить A, равен 8. Очевидно, что в каждом из своих ходов B позволит A получить лишь минимальный из всех возможных максимальных выигрышей, и поэтому с позиций игрока B минимаксное значение, на которое может надеяться A, равно 8, соответствующее его ходу L_2.

Точно так же минимаксным значением игрока B будет минимальное из всех возможных максимальных значений для B при каждом ходе A, т.е. минимальное из значений 2 и 8. Следовательно, минимаксное значение B равно 2.

Следует подчеркнуть, что для любого игрока минимаксное значение всегда больше максиминного, а не равно ему, исключительно в силу того, как определены максиминное и минимаксное значения.

Игра с нулевой суммой

В теории игр *игра с нулевой суммой* (zero-sum game) — это математическая формулировка ситуации, в которой выигрыши или проигрыши одного участника равны проигрышам или выигрышам остальных участников. Поэтому для системы в целом чистый выигрыш или проигрыш группы участников равен нулю. Рассмотрим игру с нулевой суммой, в которой участвуют два игрока, A и B. Игра с нулевой суммой может быть представлена структурой, которая называется *матрицей выигрышей*, или *платежной матрицей* (payoff matrix).

На рис. 6.22 приведена матрица выигрышей для двух игроков, в которой каждая клетка представляет выигрыш игрока A для каждой комбинации ходов игроков A и B. Поскольку это игра с нулевой суммой, выигрыш B не фигурирует в явном виде, так как он представляет собой взятый с отрицательным знаком выигрыш игрока A. Пусть игрок A играет максиминную игру. Он будет выбирать максимальное из минимальных значений в каждой строке, а значит, выберет стратегию L_3 с соответствующим выигрышем, равным максимальному из значений {–6, –10, 6}, т.е. 6. Выигрыш 6 соответствует ходу M_2 для B. Точно так же, если бы игрок A придерживался минимаксной стратегии, он был бы вынужден получить выигрыш, равный минимальному из максимальных выигрышей в каждом столбце, т.е. для каждого из ходов B. В этом случае выигрышем игрока A было бы минимальное из значений {8, 6, 12}, т.е. 6, со-

ответствующее минимаксной стратегии L_3. Опять-таки, этот выигрыш 6 соответствует ходу M_2 для B. Поэтому, как мы видим, в случае игры с нулевой суммой максиминный выигрыш участника равен его минимаксному выигрышу.

		B		
		M_1	M_2	M_3
A	L_1	4	-2	-6
	L_2	0	-10	12
	L_3	8	6	10

Рис. 6.22. Матрица выигрышей для игры с нулевой суммой

Рассмотрим максиминный выигрыш игрока B. Максиминным выигрышем игрока B является максимальное из минимальных значений игрока B на каждом ходе, т.е. максимальное из значений (–8, –6, –12) = –6, которое соответствует ходу M_2. Кроме того, это значение соответствует ходу L_3 для игрока A. Точно так же минимаксный выигрыш игрока B равен минимальному из максимальных выигрышей, который B может получить за каждый из ходов игрока A, т.е. минимальному из значений (6, 10, –6) = –6. Опять-таки, для игрока B минимаксное значение совпадает с максиминным и соответствует ходу M_2. Соответствующим ходом игрока A в этом сценарии также является L_3.

Итак, вот выводы, которые можно сделать в отношении игры с нулевой суммой.

- Независимо от того, придерживаются ли игроки A и B максиминной или минимаксной стратегии, они придут к ходам L_3 и M_2 соответственно, приносящим выигрыши 6 для игрока A и –6 для игрока B. Кроме того, минимаксное и максиминное значения для игроков совпадают с фактическими значениями выигрышей, которые игрок получает, если соблюдается минимаксная стратегия.
- Предыдущий пункт подводит к одному очень важному факту: в игре с нулевой суммой минимаксная стратегия для одного игрока определяла бы фактические стратегии обоих игроков, если бы они оба использовали чисто минимаксную или максиминную стратегию. Следовательно, ходы обоих игроков можно определить, рассматривая только ходы игрока A или игрока B. Если мы рассматриваем минимаксную стратегию игрока A, то в нее вписываются ходы обоих игроков. Если выгода от выигрыша для A равна $U(S_1, S_2)$, то ходы A и B, т.е. S_1 и S_2 соответственно, можно определить, применяя минимаксную стратегию только игрока A или только игрока B.

Минимакс и седловые точки

Для минимаксных задач в играх с нулевой суммой, включающих двух игроков, A и B, выигрыш $U(x, y)$ игрока A можно записать в виде

$$\hat{U} = \underbrace{\min}_{y} \underbrace{\max}_{x} U(x, y),$$

где x обозначает ход игрока A, y — ход игрока B.

Кроме того, значения x, y, соответствующие $\hat{U}$, являются равновесными стратегиями игроков A и B соответственно, т.е. игроки не изменят ходы, если продолжат следовать минимаксной или максиминной стратегии. Для игры с нулевой суммой и двумя участниками минимаксная и максиминная стратегии будут давать одни и те же результаты, а значит, это равновесие истинно, если игроки играют, используя либо минимаксную, либо максиминную стратегию. И поскольку минимаксное значение равно максиминному, то порядок, в котором определяется минимаксная или максимальная стратегия, не имеет значения. Мы могли бы с равным успехом позволить игрокам A и B независимо выбирать свои наилучшие стратегии для каждой стратегии противника, и тогда мы увидели бы, что для игры с нулевой суммой одна из комбинаций стратегий будет перекрываться. Это условие перекрывания является наилучшей стратегией для обоих игроков, A и B, и идентично их минимаксной стратегии. Это условие является также *равновесием Нэша* для игры.

Вплоть до этого момента мы сохраняли стратегии дискретными для облегчения интерпретации с помощью матрицы выигрышей. Для сети GAN стратегии являются непрерывными параметрами генератора и дискриминатора нейронных сетей, и поэтому, прежде чем мы подробно рассмотрим *функцию выгоды* (payoff utility function) для сети GAN, имеет смысл рассмотреть функцию выгоды $f(x, y)$ для игрока A, зависящую от двух непрерывных переменных x и y. Далее, пусть x будет ходом игрока A, а y — ходом игрока B. Нам нужно найти точку равновесия, которая также является минимаксом или максимином функции выгоды любого игрока. Выигрыш, соответствующий минимаксу игрока A, будет представлять как стратегию A, так и стратегию B. Поскольку для игры с нулевой суммой для двух участников минимаксное и максиминное значения совпадают, порядок минимакса не имеет значения, т.е.

$$\underbrace{\min}_{y} \underbrace{\max}_{x} f(x, y) = \underbrace{\max}_{x} \underbrace{\min}_{y} f(x, y) = \underbrace{\min}_{y} f(x, y).$$
$$\underbrace{\max}_{x}$$

Для непрерывной функции это возможно только в том случае, если решение предыдущей функции является седловой точкой. *Седловая точка* (saddle point) — это точка, в которой градиент по каждой переменной равен нулю. Однако она не является локальным минимумом или максимумом. Вместо этого она представляет локальный минимум в одних направлениях входного вектора и локальный максимум в других. Поэтому седловую точку можно найти только с помощью производных более высо-

кого порядка. Для многомерной функции $f(x)$ с $\forall x \in \mathbb{R}^{n\times 1}$ мы можем определить седловую точку с помощью следующего теста.

- Вычисляем градиент $f(x)$ по вектору x, т.е. $\nabla_x f(x)$, и приравниваем его к нулю.
- Вычисляем гессиан $\nabla_x^2 f(x)$ функции $f(x)$, т.е. матрицу производных второго порядка в каждой из точек, в которых вектор градиента $\nabla_x f(x)$ равен нулю. Если гессиан имеет как положительные, так и отрицательные собственные значения в оцениваемой точке, то данная точка — седловая.

Возвращаясь к функции выгоды двух переменных $f(x, y)$ для игрока A, определим ее следующим образом, чтобы проиллюстрировать пример:

$$f(x, y) = x^2 - y^2.$$

Следовательно, функцией выгоды для игрока B автоматически станет $f(x, y) = -x^2 + y^2$.

Исследуем, обеспечивает ли функция выгоды равновесие, если оба игрока следуют минимаксной или максиминной стратегии в игре с нулевой суммой. Игра будет иметь равновесие, вне которого игроки не смогут улучшить свои выигрыши, поскольку их стратегии оптимальны, если функция $f(x, y)$ имеет седловую точку. Условием равновесия является равновесие Нэша для игры.

Приравнивая градиент $f(x, y)$ к нулю, получаем

$$\nabla f(x, y) = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} 2x \\ -2y \end{bmatrix} = 0 \Rightarrow (x, y) = (0, 0).$$

Вычисляем гессиан данной функции:

$$\nabla^2 f(x, y) = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & -2 \end{bmatrix}.$$

Гессиан функции равен $\begin{bmatrix} 2 & 0 \\ 0 & -2 \end{bmatrix}$ для любого значения (x, y), включая значение $(x, y) = (0, 0)$. Поскольку гессиан имеет как положительные, так и отрицательные собственные значения, т.е. 2 и –2, точка $(x, y) = (0, 0)$ является седловой. В условиях равновесия стратегия для игрока A должна заключаться в том, чтобы установить $x = 0$, а для игрока B — в том, чтобы установить $y = 0$ в минимаксной или максиминной игре с нулевой суммой.

Функция стоимости и тренировка сети GAN

В генеративно-состязательных сетях генераторная и дискриминаторная сети пытаются превзойти друг друга, следуя минимаксной стратегии в игре с нулевой суммой. В этом случае ходы являются параметрами, которые выбирает сеть. Прежде всего упростим нотацию, используя для параметров обозначения, представляющие сами модели, т.е. G — для параметров генератора; D — для параметров дискриминатора. Определим рамки, которыми ограничивается выгода для каждой из сетей, определяемая их функциями выгоды. Дискриминатор будет пытаться корректно классифицировать как поддельные синтетически сгенерированные значения, так и реальные. Иными словами, он будет пытаться максимизировать функцию выгоды

$$U(D,\ G) = E_{x\sim P_x(x)}\left[\log D(x)\right] + E_{z\sim P_z(z)}\left[\log\left(1 - D\left(G(z)\right)\right)\right],$$

где x обозначает реальные образцы данных, извлеченные из распределения вероятности $P_x(x)$, а z — шум, извлеченный из априорного зашумленного распределения $P_z(z)$.

Кроме того, дискриминатор пытается вывести 1 для реального образца данных x и 0 для созданных генератором поддельных или синтетических данных, базирующихся на зашумленных образцах z. Таким образом, дискриминатору хотелось бы следовать стратегии, которая максимально приближает $D(x)$ к 1, т.е. делает значение $\log D(x)$ близким к нулю. Чем ближе $D(x)$ к 1, тем меньше значение $\log D(x)$, а значит, тем меньше выгода, которую получит дискриминатор. Точно так же дискриминатору хотелось бы захватывать поддельные или синтетические данные, устанавливая их вероятность близкой к нулю, т.е. устанавливать $D(G(z))$ как можно ближе к нулю, чтобы идентифицировать данные как поддельное изображение. Если $D(G(z))$ близко к нулю, то выражение $\left[\log\left(1 - D\left(G(z)\right)\right)\right]$ также стремится к нулю. Если значение $D(G(z))$ отклоняется от нуля, то выигрыш дискриминатора уменьшается, поскольку уменьшается $\log\left(1 - D\left(G(z)\right)\right)$. Дискриминатору хотелось бы добиться этого по всему распределению x и z, а значит, и по отношению к членам, выражающим математическое ожидание или среднее в его функции выигрыша. Разумеется, влияние генератора G на функцию выигрыша D проявляется через $G(z)$, т.е. второй член, и поэтому он также будет пытаться делать ходы, которые минимизируют выигрыш D. Чем больше выигрыш D, тем хуже ситуация для G. Поэтому мы можем считать, что G имеет ту же функцию выгоды, что и D, только с отрицательным знаком, что превращает эту игру в игру с нулевой суммой, в которой выигрыш G дается формулой

$$V(D,\ G) = -E_{x\sim P_x(x)}\left[\log D(x)\right] - E_{z\sim P_z(z)}\left[\log\left(1 - D\left(G(z)\right)\right)\right].$$

Генератор G будет пытаться выбирать свои параметры таким образом, чтобы максимизировать $V(D, G)$, т.е. будет пытаться генерировать поддельные образцы данных $G(z)$, чтобы обманутый дискриминатор классифицировал их нулевой меткой. Иными словами, он хочет, чтобы дискриминатор считал $G(z)$ реальными данными и назначал

им высокую вероятность. Высокие значения $D(G(z))$, удаленные от нуля, приведут к тому, что $\log\big(1-D\big(G(z)\big)\big)$ будет иметь большое отрицательное значение, а значит, выражение $-E_{z\sim P_z(z)}\Big[\log\big(1-D\big(G(z)\big)\big)\Big]$ будет иметь большую положительную величину, тем самым увеличивая выигрыш генератора. К сожалению, генератор не сможет влиять на первый член в $V(D, G)$, включающий реальные данные, поскольку он не включает параметров G.

Модели генератора G и дискриминатора D тренируются путем предоставления им возможности следовать минимаксной стратегии в игре с нулевой суммой. Дискриминатор будет пытаться максимизировать свою функцию выигрыша $U(D, G)$ и достигнуть ее минимаксного значения:

$$u^* = \underbrace{\min}_{D}\underbrace{\max}_{G} E_{x\sim P_x(x)}\big[\log D(x)\big] + E_{z\sim P_z(z)}\Big[\log\big(1-D\big(G(z)\big)\big)\Big].$$

Точно так же генератор G будет пытаться максимизировать свою функцию $V(D, G)$, выбирая стратегию

$$v^* = \underbrace{\min}_{D}\underbrace{\max}_{G} - E_{x\sim P_x(x)}\big[\log D(x)\big] - E_{z\sim P_z(z)}\Big[\log\big(1-D\big(G(z)\big)\big)\Big].$$

Поскольку G никак не влияет на первый член, то

$$v^* = \underbrace{\min}_{D}\underbrace{\max}_{G} - E_{z\sim P_z(z)}\big[\log D(x)\big].$$

Как мы уже видели, в игре с нулевой суммой и двумя участниками отсутствует необходимость в рассмотрении двух минимаксных стратегий по отдельности, так как обе они могут быть получены путем рассмотрения минимаксной стратегии для функции выигрыша одного из участников. Рассматривая формулировку минимаксной стратегии дискриминатора, мы получаем для значения выигрыша дискриминатора при равновесии (равновесии Нэша) следующее выражение:

$$u^* = \underbrace{\min}_{\substack{G\\ \max\\ D}} E_{x\sim P_x(x)}\big[\log D(x)\big] + E_{z\sim P_z(z)}\Big[\log\big(1-D\big(G(z)\big)\big)\Big].$$

Значения $\hat{G}$ и $\hat{D}$ при равновесном значении u^* — это оптимальные значения параметров для обеих сетей, изменением которых ни одна из них не сможет увеличить свои очки. Кроме того, $(\hat{G}, \hat{D})$ дает седловую точку функции выгоды D:

$$E_{x\sim P_x(x)}\big[\log D(x)\big] + E_{z\sim P_z(z)}\Big[\log\big(1-D\big(G(z)\big)\big)\Big].$$

Предыдущие формулировки можно упростить, разбив оптимизацию на две части, т.е. позволить D максимизировать свою функцию выгоды по своим параметрам, а G — минимизировать функцию выгоды D по собственным параметрам на каждом ходе:

$$\underbrace{\max}_{D} E_{x\sim P_x(x)}\big[\log D(x)\big] + E_{z\sim P_z(z)}\Big[\log\big(1-D\big(G(z)\big)\big)\Big],$$

$$\underbrace{\min}_{G} E_{z\sim P_z(z)}\left[\log\left(1-D\left(G(z)\right)\right)\right].$$

Каждый из них, оптимизируя собственную функцию стоимости, рассматривает ходы противника как фиксированные. Подобный итеративный способ оптимизации есть не что иное, как метод градиентного спуска для вычисления седловой точки. Поскольку пакеты машинного обучения главным образом кодируются для минимизации, а не максимизации функций, следует умножить целевую функцию дискриминатора на –1 и позволить ему минимизировать ее, а не максимизировать.

Ниже приведен псевдокод реализации мини-пакетного подхода, который обычно используется для тренировки GAN-сетей и базируется на предыдущих эвристиках.

- for N (количество эпох):
- for k (шаги):
  - Извлечь m образцов $\left\{z^{(1)}, z^{(2)}, \ldots, z^{(m)}\right\}$ из зашумленного распределения $z \sim P_z(z)$.
  - Извлечь m образцов $\left\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\right\}$ из распределения данных $x \sim P_x(x)$.
  - Обновить параметры дискриминатора D, используя стохастический градиентный спуск. Если обозначить параметры дискриминатора D как θ_D, то θ_D обновляются в соответствии со следующей формулой:

$$\theta_D \rightarrow \theta_D - \nabla_{\theta_D}\left[-\frac{1}{m}\sum_{i=1}^{m}\left(\log D\left(x^{(i)}\right)\right)+\log\left(1-D\left(G(z^{(i)})\right)\right)\right].$$

- end
  - Извлечь m образцов $\left\{z^{(1)}, z^{(2)}, \ldots, z^{(m)}\right\}$ из зашумленного распределения $z \sim P_z(z)$.
  - Обновить генератор G методом стохастического градиентного спуска. Если обозначить параметры дискриминатора G как θ_G, то θ_G обновляются в соответствии со следующей формулой:

$$\theta_G \rightarrow \theta_G - \nabla_{\theta_G}\left[-\frac{1}{m}\sum_{i=1}^{m}\log\left(1-D\left(G(z^{(i)})\right)\right)\right].$$

- end

Затухающие градиенты для генератора

Как правило, образцы данных, получаемые на начальных стадиях тренировки генератора, заметно отличаются от оригинальных данных, поэтому дискриминатору не составляет труда пометить их как поддельные. Следствием этого является близость к нулю значений $D(G(z))$, что приводит к насыщению градиента

$\nabla_{\theta_G}\left[\frac{1}{m}\sum_{i=1}^{m}\log\left(1-D\left(G\left(z^{(i)}\right)\right)\right)\right]$, в результате чего мы сталкиваемся с проблемой затухающих градиентов для параметров сети генератора G. Чтобы избавиться от этой проблемы, следует вместо минимизации функции $E_{z-P_z(z)}\left[\log\left(1-D\left(G(z)\right)\right)\right]$ максимизировать функцию $E_{z-P_z(z)}\left[\log G(z)\right]$ или, применительно к градиентному спуску, минимизировать функцию $E_{z-P_z(z)}\left[-\log G(z)\right]$. После такой замены метод тренировки уже не является строго минимаксной игрой, а становится разумным приближением, позволяющим преодолеть проблему насыщения на ранней стадии тренировки.

Реализация сети GAN средствами TensorFlow

В данном разделе приведен пример сети GAN, обученной на изображениях MNIST, в которой генератор пытается создать поддельные синтетические изображения в стиле MNIST, в то время как дискриминатор пытается пометить такие изображения как поддельные, но при этом сохраняет способность определять реальные данные как подлинные. По завершении тренировки мы семплируем несколько образцов синтетических изображений и проверяем, напоминают ли они реальные. Генератором служит простая нейронная сеть прямого распространения с тремя скрытыми слоями, за которыми следует выходной слой, состоящий из 784 элементов, соответствующих 784 пикселям изображения MNIST. В качестве функции активации выходных элементов выбрана не сигмоида, а гиперболический тангенс как менее подверженный проблемам затухающих градиентов. Выходные значения функции активации в виде гиперболического тангенса заключены в пределах от –1 до 1, в связи с чем реальные изображения MNIST нормализуются до значений между –1 и 1, чтобы как синтетические, так и реальные MNIST-изображения были приведены к одному диапазону. Сеть дискриминатора также представляет собой нейронную сеть прямого распространения с тремя скрытыми слоями и сигмоидным выходным элементом для бинарной классификации реальных изображений MNIST и синтетических, полученных с помощью генератора. Входом генератора является 100-мерный входной вектор, семплированный из равномерного зашумленного распределения, со значениями в интервале от –1 до 1 для каждого измерения. Детали реализации описаны в листинге 6.5.

Листинг 6.5. Реализация генеративно-состязательной сети

```
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

## Размерность априорного зашумленного сигнала выбрана равной 100.
## Генератор имеет последовательно 150 и 300 скрытых элементов,
## которые предшествуют 784 выходным элемента, соответствующим
```

```
## изображению размером 28x28.

h1_dim = 150
h2_dim = 300
dim = 100
batch_size = 256
#-------------------------------------------------------------------
# Определяем генератор - берем шум и преобразуем его в изображения
#-------------------------------------------------------------------
def generator_(z_noise):
    w1 = tf.Variable(tf.truncated_normal([dim,h1_dim],
            stddev=0.1),name="w1_g", dtype=tf.float32)
    b1 = tf.Variable(tf.zeros([h1_dim]), name="b1_g",
            dtype=tf.float32)
    h1 = tf.nn.relu(tf.matmul(z_noise, w1) + b1)
    w2 = tf.Variable(tf.truncated_normal([h1_dim,h2_dim],
            stddev=0.1), name="w2_g",dtype=tf.float32)
    b2 = tf.Variable(tf.zeros([h2_dim]), name="b2_g",
            dtype=tf.float32)
    h2 = tf.nn.relu(tf.matmul(h1, w2) + b2)
    w3 = tf.Variable(tf.truncated_normal([h2_dim,28*28],
            stddev=0.1), name="w3_g", dtype=tf.float32)
    b3 = tf.Variable(tf.zeros([28*28]), name="b3_g",
            dtype=tf.float32)
    h3 = tf.matmul(h2, w3) + b3
    out_gen = tf.nn.tanh(h3)
    weights_g = [w1, b1, w2, b2, w3, b3]
    return out_gen,weights_g
#-------------------------------------------------------------------
# Определяем дискриминатор - получаем от генератора как реальные,
# так и поддельные синтетические изображения, и классифицируем их
#-------------------------------------------------------------------
def discriminator_(x,out_gen,keep_prob):
    x_all = tf.concat([x,out_gen], 0)
    w1 = tf.Variable(tf.truncated_normal([28*28, h2_dim],
            stddev=0.1), name="w1_d",dtype=tf.float32)
    b1 = tf.Variable(tf.zeros([h2_dim]), name="b1_d", dtype=tf.float32)
    h1 = tf.nn.dropout(tf.nn.relu(tf.matmul(x_all, w1) + b1), keep_prob)
    w2 = tf.Variable(tf.truncated_normal([h2_dim, h1_dim],
            stddev=0.1), name="w2_d",dtype=tf.float32)
    b2 = tf.Variable(tf.zeros([h1_dim]), name="b2_d", dtype=tf.float32)
    h2 = tf.nn.dropout(tf.nn.relu(tf.matmul(h1,w2) + b2), keep_prob)
    w3 = tf.Variable(tf.truncated_normal([h1_dim, 1], stddev=0.1),
            name="w3_d", dtype=tf.float32)
    b3 = tf.Variable(tf.zeros([1]), name="d_b3", dtype=tf.float32)
    h3 = tf.matmul(h2, w3) + b3
    y_data = tf.nn.sigmoid(tf.slice(h3, [0, 0], [batch_size, -1],
            name=None))
    y_fake = tf.nn.sigmoid(tf.slice(h3, [batch_size, 0], [-1, -1],
```

```
            name=None))
    weights_d = [w1, b1, w2, b2, w3, b3]
    return y_data,y_fake,weights_d
#---------------------------------------------------------------
# Чтение данных MNIST
#---------------------------------------------------------------
mnist = input_data.read_data_sets('MNIST_data', one_hot=True)
#---------------------------------------------------------------
# Определение различных операций и переменных TensorFlow,
# функции стоимости и оптимизатора
#---------------------------------------------------------------
# Заполнители
x = tf.placeholder(tf.float32, [batch_size, 28*28], name="x_data")
z_noise = tf.placeholder(tf.float32, [batch_size,dim], name="z_prior")
# Вероятность исключения (дропаута)
keep_prob = tf.placeholder(tf.float32, name="keep_prob")
# Определение выходных операций и весов для генератора
out_gen,weights_g = generator_(z_noise)
# Определение операций и весов для дискриминатора
y_data, y_fake,weights_d = discriminator_(x,out_gen,keep_prob)
# Функции стоимости для дискриминатора и генератора
discr_loss = - (tf.log(y_data) + tf.log(1 - y_fake))
gen_loss = - tf.log(y_fake)
optimizer = tf.train.AdamOptimizer(0.0001)
d_trainer = optimizer.minimize(discr_loss,var_list=weights_d)
g_trainer = optimizer.minimize(gen_loss,var_list=weights_g)
init = tf.global_variables_initializer()
saver = tf.train.Saver()

#---------------------------------------------------------------
# Вызов графа TensorFlow и начало тренировки
#---------------------------------------------------------------

sess = tf.Session()
sess.run(init)
z_sample = np.random.uniform(-1, 1,
    size=(batch_size,dim)).astype(np.float32)

for i in range(60000):
    batch_x, _ = mnist.train.next_batch(batch_size)
    x_value = 2*batch_x.astype(np.float32) - 1
    z_value = np.random.uniform(-1, 1,
        size=(batch_size,dim)).astype(np.float32)
    sess.run(d_trainer,feed_dict={x:x_value,
        z_noise:z_value,keep_prob:0.7})
    sess.run(g_trainer,feed_dict={x:x_value,
        z_noise:z_value,keep_prob:0.7})
    [c1,c2] = sess.run([discr_loss,gen_loss],feed_dict={x:x_value,
        z_noise:z_value,keep_prob:0.7})
```

```
    print ('iter:',i,'cost of discriminator',c1, 'cost of generator',c2)
#---------------------------------------------------------------------
# Генерирование пакета поддельных синтетических изображений
#---------------------------------------------------------------------
out_val_img = sess.run(out_gen,feed_dict={z_noise:z_sample})
saver.save(sess, " newgan1",global_step=i)
#---------------------------------------------------------------------
# Вывод сгенерированных изображений
#---------------------------------------------------------------------
imgs = 0.5*(out_val_img + 1)
for k in range(36):
    plt.subplot(6,6,k+1)
    image = np.reshape(imgs[k],(28,28))
    plt.imshow(image,cmap='gray')

-- вывод --
```

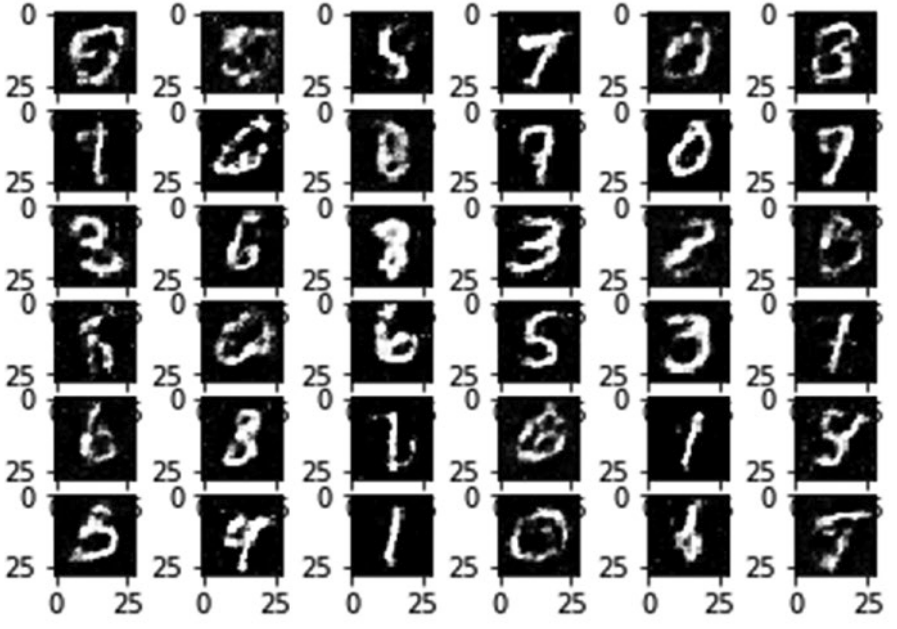


Рис. 6.23. Синтетические цифры, сгенерированные сетью GAN

На рис. 6.23 видно, что генератор GAN способен производить изображения, похожие на цифры из набора данных MNIST. Для получения результатов такого качества модель GAN тренировалась на 60000 мини-пакетах размером 256. Хочу подчеркнуть, что по сравнению с другими нейронными сетями сети GAN сравнительно хуже поддаются обучению. Поэтому для получения желаемых результатов могут потребоваться многочисленные эксперименты, сопровождаемые настройкой сети.

Развертывание моделей TensorFlow в производственной среде

Для экспорта обученной модели TensorFlow в производственную среду можно воспользоваться возможностями системы TensorFlow Serving, назначение которой — упрощение развертывания моделей машинного обучения в производственных условиях. Сервис TensorFlow Serving, как говорит само его название, обеспечивает хостинг модели в производственной среде и предоставляет приложения с локальным доступом к ней. Ниже приведено общее описание пошаговой процедуры загрузки модели TensorFlow в производственную среду.

- Модель TensorFlow необходимо обучить, активизировав вычислительный граф TensorFlow в активной сессии.
- Для экспорта обученной модели можно использовать модуль `SavedModelBuilder` TensorFlow. С его помощью вы сможете сохранить копию модели в безопасном месте, откуда ее можно будет легко загрузить в случае необходимости. При вызове модуля `SavedModelBuilder` требуется указать путь экспорта. Если указанного пути экспорта не существует, модуль `SavedModelBuilder` создаст требуемый каталог. Также существует возможность указать номер версии модели с помощью флага `FLAGS.model_version`.
- Для связывания определения метаграфа TensorFlow и других переменных с экспортируемой моделью можно использовать метод `SavedModelBuilder.add_meta_graph_and_variable()`. Опция `signature_def_map` этого метода позволяет создавать отображения для различных сигнатур, предоставляемых пользователем. Сигнатуры позволяют указать входы и выходы тензоров, которые потребуются при отправке модели входных данных для предсказания и получения от нее выходных данных в виде предсказаний. Например, можно создать сигнатуру для классификации и сигнатуру для предсказания и привязать их к параметру `signature_def_map`. В случае мультиклассификационной модели можно определить сигнатуру классификации, чтобы получать тензор изображения в качестве входа и вероятность класса в качестве выхода. Точно так же можно определить сигнатуру предсказания, чтобы получать тензор изображения в качестве входа и необработанную оценку класса в качестве выхода. Экспортируя модели TensorFlow, можете использовать в качестве образца фрагмент кода, доступный по следующему адресу:

  `https://github.com/tensorflow/serving/blob/master/tensorflow_serving/example/mnist_saved_model.py`

- Для загрузки экспортированной модели можно использовать стандартный сервер моделей TensorFlow или сервер, скомпилированный на локальной машине. Более подробное описание этого процесса вы найдете по ссылке, предоставляемой на сайте TensorFlow по следующему адресу:

  `https://www.tensorflow.org/serving/serving_basic`

В листинге 6.6, *а*, приведен код базовой реализации, предназначенной для сохранения модели TensorFlow и ее последующего использования в целях предсказания на стадии тестирования. Этот код имеет много общего с кодом, который использовался бы при развертывании модели в производственной среде.

Листинг 6.6, а. Пример сохранения модели в TensorFlow

```
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data

batch_size,learning_rate = 256,0.001
epochs = 10
total_iter = 1000

x = tf.placeholder(tf.float32,[None,784],name='x')
y = tf.placeholder(tf.float32,[None,10],name='y')

W = tf.Variable(tf.random_normal([784,10],mean=0,stddev=0.02),name='W')
b = tf.Variable(tf.random_normal([10],mean=0,stddev=0.02),name='b')
logits = tf.add(tf.matmul(x,W),b,name='logits')
pred = tf.nn.softmax(logits,name='pred')
correct_prediction = tf.equal(tf.argmax(y,1),tf.argmax(pred,1),
        name='correct_prediction')
accuracy = tf.reduce_mean(tf.cast(correct_prediction,tf.float32),
        name='accuracy')

mnist = input_data.read_data_sets('MNIST_data', one_hot=True)
batches = (mnist.train.num_examples//batch_size)
saver = tf.train.Saver()

cost = tf.reduce_mean(tf.nn.
        sigmoid_cross_entropy_with_logits(logits=logits,labels=y))
optimizer_ = tf.train.AdamOptimizer(learning_rate).minimize(cost)
init = tf.global_variables_initializer()
with tf.Session() as sess:
    sess.run(init)
    for step in range(total_iter):
        batch_x,batch_y = mnist.train.next_batch(batch_size)
        sess.run(optimizer_,feed_dict={x:batch_x,y:batch_y})
        c = sess.run(cost,feed_dict={x:batch_x,y:batch_y})
        print ('Loss in iteration ' + str(step) + '= ' + str(c))
        if step % 100 == 0 :
                saver.save(sess,'/home/santanu/model_basic',
                global_step=step)
    saver.save(sess,'/home/santanu/model_basic',global_step=step)
    val_x,val_y = mnist.test.next_batch(batch_size)
    print('Accuracy:',sess.run(accuracy,feed_dict={x:val_x,y:val_y}))

-- вывод --
```

```
Loss in iteration 991= 0.0870551
Loss in iteration 992= 0.0821354
Loss in iteration 993= 0.0925385
Loss in iteration 994= 0.0902953
Loss in iteration 995= 0.0883076
Loss in iteration 996= 0.0936614
Loss in iteration 997= 0.077705
Loss in iteration 998= 0.0851475
Loss in iteration 999= 0.0802716
('Accuracy:', 0.91796875)
```

Важным моментом в вышеприведенном коде является создание экземпляра `saver` класса `tf.train.Save()`. Вызов метода `save` объекта `saver` в сеансе TensorFlow сохраняет весь метаграф сессии в указанном расположении. Это очень важно, поскольку переменные TensorFlow существуют лишь в сеансе TensorFlow, и этот метод обеспечивает возможность извлечения модели, созданной в некотором сеансе, в более позднее время для использования в целях предсказания, подстройки модели и т.п.

Модель сохраняется в указанном расположении с использованием указанного имени, т.е. `model_basic`, и включает следующие три файла:

- *model_basic-9999.meta*;
- *model_basic-9999.index*;
- *model_basic-9999.data-00000-of-00001*.

Составляющая "9999" в именах файлов обозначает номер шага и была добавлена, поскольку мы указали параметр `global_step` при вызове метода `save`. Это облегчает определение версии модели, так как мы можем быть заинтересованы в сохранении нескольких копий модели, созданных на различных шагах. Однако TensorFlow поддерживает лишь последние четыре версии.

Файл *model_basic-9999.meta* содержит граф сохраненного сеанса, тогда как файлы *model_basic-9999.data-00000-of-00001* и *model_basic-9999.index* образуют файл контрольной точки, содержащий значения всех переменных. Кроме того, создается общий контрольный файл, содержащий информацию обо всех доступных файлах контрольных точек.

Перейдем к рассмотрению процесса восстановления модели. Мы могли бы создать сеть, определив все переменные и операции вручную, как при создании первоначального варианта сети. Но эти определения уже содержатся в файле *model_basic-9999.meta*, поэтому их можно импортировать в текущий сеанс с помощью метода `import_meta_graph`, что и сделано в листинге 6.6, *б*. Все, что необходимо сделать после загрузки метаграфа, — это загрузить значения различных параметров. Для этого предназначен метод `restore` экземпляра `saver`. Как только мы это сделаем, на переменные можно будет ссылаться по их именам. Например, имена `pred` и `accuracy` непосредственно используются для ссылки на тензоры, которые далее применяются для предсказания новых данных. Точно таким же способом могут быть

восстановлены и заполнители, имена которых будут использоваться для передачи данных различным операциям.

В листинге 6.6, *б*, приведен код реализации, предназначенной для восстановления модели TensorFlow и ее использования для выполнения предсказаний и проверки точности сохраненной обученной модели.

Листинг 6.6, б. Пример восстановления сохраненной модели в TensorFlow

```
batch_size = 256
with tf.Session() as sess:
    init = tf.global_variables_initializer()
    sess.run(init)
    new_saver = tf.train.import_meta_graph(
            '/home/santanu/model_basic-999.meta')
    new_saver.restore(sess,tf.train.latest_checkpoint('./'))
    graph = tf.get_default_graph()
    pred = graph.get_tensor_by_name("pred:0")
    accuracy = graph.get_tensor_by_name("accuracy:0")
    x = graph.get_tensor_by_name("x:0")
    y = graph.get_tensor_by_name("y:0")
    val_x,val_y = mnist.test.next_batch(batch_size)
    pred_out = sess.run(pred,feed_dict={x:val_x})
    accuracy_out = sess.run(accuracy,feed_dict={x:val_x,y:val_y})
    print 'Accuracy on Test dataset:',accuracy_out

-- вывод --

Accuracy on Test dataset: 0.871094
```

Резюме

Эта глава завершающая. Несмотря на повышенную сложность изложенных в ней концепций и моделей, используемые в ней подходы уже обсуждались в предыдущих главах. Прочитав эту главу, вы должны чувствовать себя более уверенно при реализации широкого круга моделей, обсуждавшихся в данной книге, равно как и при реализации других моделей и методов, которые предлагаются в непрерывно развивающейся сфере глубокого обучения. Следите за публикациями и новыми разработками признанных экспертов в области глубокого обучения — это наилучший способ идти в ногу со временем и постоянно быть в курсе всех инноваций. И коль скоро это так, то вашего внимания в первую очередь заслуживают, в частности, работы таких авторитетов, как Джеффри Хинтон, Ян Лекун, Йошуа Бенджио и Ян Гудфеллоу. Кроме того, я считаю, что для получения наилучших результатов вы должны стремиться совершенствовать свои знания в области математики и глубокого обучения как науки, а не просто использовать соответствующие методы в качестве черного ящика. На этом я заканчиваю и выражаю благодарность всем читателям.