2 Предобученные сети

В этой главе

- ✓ Выполнение предобученных моделей распознавания изображений.
- ✓ Введение в GAN и CycleGAN.
- ✓ Модели описания изображений, способные генерировать их текстовые описания.
- ✓ Обмен моделями через Torch Hub.

Мы завершили первую главу обещанием открыть вам во второй потрясающие вещи, и теперь пришло время воплотить его в жизнь. Машинное зрение, безусловно, одна из сфер, на которую появление глубокого обучения оказало наибольшее влияние, и на это есть много причин. Была потребность в классификации или интерпретации содержимого естественных изображений, стали доступны очень большие наборы данных, были изобретены такие новые компоненты, как сверточные слои, работающие на GPU с невероятной точностью и скоростью. Все эти факторы объединились с желанием крупнейших интернет-компаний анализировать фотографии, снимаемые миллионами пользователей на мобильные устройства и размещаемые на платформах этих компаний. Идеальные условия.

Скоро мы научимся использовать результаты лучших исследователей в этой сфере, скачивая и запуская очень интересные модели, уже обученные на общедоступных

масштабных наборах данных. Предобученная нейронная сеть в чем-то подобна программе, которая принимает входные данные и формирует выходные. Поведение такой программы в плане желаемых пар «вход/выход» или нужных свойств определяется архитектурой нейронной сети и примерами, просмотренными ею во время обучения. Готовая модель — быстрый способ ускоренного запуска проекта глубокого обучения за счет использования опыта разработавших модель исследователей и вычислительного времени, затраченного на ее обучение.

В этой главе мы обсудим три часто встречающиеся предобученные модели: модель для маркировки изображения в соответствии с его содержимым, модель, способную формировать новое изображение на основе исходного, и модель, описывающую содержимое изображения простым английским языком. Вы научитесь загружать и запускать эти предобученные модели в PyTorch и познакомитесь с PyTorch Hub — набором инструментов, позволяющим с помощью единого интерфейса легко создавать модели PyTorch, подобные тем, которые мы будем обсуждать. А еще мы затронем источники данных, дадим определения таким терминам, как «метка» и, посетим родео с зебрами.

Если вы переходите на PyTorch с другого фреймворка глубокого обучения и хотели бы непосредственно заняться практикой, можете сразу читать главу 3. Здесь мы рассмотрим скорее развлекательные, чем серьезные вопросы, причем практически общие для всех утилит глубокого обучения. Это не значит, что они неважны! Но если вы уже работали с предобученными моделями в других фреймворках глубокого обучения, то прекрасно знаете, какие обширные возможности они открывают. А если вы уже знакомы с генеративными состязательными сетями (GAN), то вам не будет интересен наш подробный рассказ о них.

Впрочем, мы надеемся, что вы продолжите чтение этой главы, поскольку в ней описываются довольно важные навыки. Умение работать с предобученными моделями с помощью PyTorch — полезный навык и точка! Особенно он полезен, если модель обучена на большом наборе данных. Нам нужно будет привыкнуть к специфике получения и запуска нейронной сети на реальных данных, а также последующей визуализации и оценки результатов ее работы вне зависимости от того, обучали мы ее или нет.

2.1. ПРЕДОБУЧЕННЫЕ СЕТИ ДЛЯ РАСПОЗНАВАНИЯ ТЕМАТИКИ ИЗОБРАЖЕНИЯ

В качестве первого экскурса в глубокое обучение запустим современную глубокую нейронную сеть, предобученную на задаче распознавания образов. В репозиториях исходного кода доступно множество предобученных сетей. Исследователи часто публикуют свой исходный код вместе со статьями, причем нередко в код включаются весовые коэффициенты, полученные при обучении

модели на эталонном наборе данных. С помощью одной из таких моделей можно, например, без особых усилий оснастить веб-сервис возможностями распознавания изображений.

Предобученная сеть, которую мы здесь будем изучать, обучена на подмножестве набора данных ImageNet (http://imagenet.stanford.edu/). ImageNet — очень большой набор данных из более чем 14 миллионов изображений, поддерживаемый Стэнфордским университетом. Все его изображения маркированы при помощи иерархии существительных из набора данных WordNet (http://wordnet.princeton.edu) — большой лексической базы данных английского языка.

Набор данных ImageNet, подобно нескольким другим общедоступным наборам данных, появился в результате научных конкурсов. Такие конкурсы традиционно служат одними из главных площадок соревнования исследователей из различных учреждений и компаний. В частности, с момента своего появления в 2010 году большую популярность обрел конкурс крупномасштабных распознаваний зрительных образов ImageNet (ImageNet Large Scale Visual Recognition Challenge, ILSVRC). Этот конкретный конкурс включает в себя несколько заданий, меняющихся от года к году, например классификацию изображений (выяснение, какие категории объектов содержит изображение), локализацию объектов (определение местоположения объектов на изображении), обнаружение объектов (выявление и маркирование объектов на изображениях), классификацию обстановки (классификация общего плана на изображении) и разбор обстановки (разбиение изображения на области, относящиеся к различным семантическим категориям, например «корова», «дом», «сыр», «шляпа»). В частности, задача классификации изображений состоит в получении на основе входного изображения списка из пяти описывающих содержимое изображения меток (из общего списка в 1000 категорий), отсортированных по степени достоверности.

Обучающий набор данных для ILSVRC состоит из 1,2 миллиона изображений, маркированных одним из 1000 существительных (например, «собака») — *классов* изображений. Мы будем использовать далее в этом значении попеременно термины *«метка»* (*label*) и *«класс»* (*class*). На рис. 2.1 приведены некоторые из изображений ImageNet.

В конечном итоге мы хотим подавать свои собственные изображения на вход предобученной модели, как показано на рис. 2.2. В результате мы получим список прогнозируемых меток для этого изображения, по которым затем сможем понять, что наша модель думает о конкретном изображении. Предсказания для некоторых изображений безошибочны, а для других — нет!

Входное изображение сначала необходимо предварительно обработать, превратив в экземпляр класса многомерного массива `torch.Tensor`. Для изображения RGB с высотой и шириной тензор будет включать три измерения: три цветовых канала и два пространственных измерения заданного размера (в главе 3 мы обсудим подробнее, что представляет собой этот тензор, а пока что можете просто

считать его чем-то наподобие вектора или матрицы чисел с плавающей запятой). Наша модель принимает на входе это обработанное входное изображение и передает его предобученной сети, возвращающей оценки для всех классов. Наивысшая оценка соответствует наиболее вероятному классу в зависимости от веса. После этого каждый класс сопоставляется со своей меткой класса. Выходной сигнал содержится в `torch.Tensor` из 1000 элементов, каждый из которых соответствует оценке для конкретного класса.

Рис. 2.1. Небольшая выборка изображений ImageNet

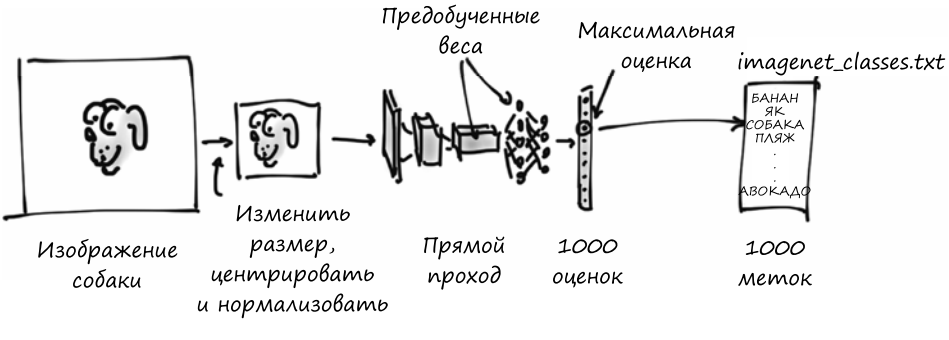

Рис. 2.2. Процесс выполнения вывода

Прежде чем заняться всем этим, необходимо получить саму нейронную сеть, заглянуть ей «под капот», чтобы понять, как она устроена, и разобраться с подготовкой данных для использования моделью.

2.1.1. Получение предобученной сети для распознавания изображений

Как мы уже упоминали, сейчас мы вооружимся нейронной сетью, обученной на ImageNet. Для этого взглянем на проект TorchVision (https://github.com/pytorch/vision), включающий несколько лучших нейросетевых архитектур, предназначенных для машинного зрения: AlexNet (http://mng.bz/lo6z), ResNet (https://arxiv.org/pdf/1512.03385.pdf) и Inception v3 (https://arxiv.org/pdf/1512.00567.pdf). Он также обеспечивает удобный доступ к таким наборам данных, как ImageNet, и прочим инструментам для работы с машинным зрением в PyTorch. Мы обсудим часть из них подробнее далее в этой книге. А пока что просто загрузим и запустим две сети: сначала AlexNet, одну из первых инновационных сетей для распознавания изображений; а затем остаточную сеть (residual network, сокращенно — ResNet), выигравшую, помимо прочего, в 2015 году конкурсы ImageNet по классификации, обнаружению и локализации. Если вы не установили и не настроили PyTorch в главе 1 — самое время это сделать.

Найти предобученные модели можно в `torchvision.models` (`code/p1ch2/2_pre_trained_networks.ipynb`):

```
# In[1]:
from torchvision import models
```

Взглянем на сами модели:

```
# In[2]:
dir(models)
```

```
# Out[2]:
['AlexNet',
'DenseNet',
'Inception3',
'ResNet',
'SqueezeNet',
'VGG',
...
'alexnet',
'densenet',
'densenet121',
...
'resnet',
'resnet101',
'resnet152',
...
]
```

Названия, начинающиеся с заглавной буквы, относятся к классам Python, реализующим несколько популярных моделей, отличающихся архитектурой, то есть схемой операций между входом и выходом модели. Названия в нижнем регистре — вспомогательные функции, возвращающие созданные на основе этих классов модели, иногда с различными наборами параметров. Например, `resnet101` возвращает экземпляр ResNet со 101 слоем, `resnet18` содержит 18 слоев и т. д. Сначала мы займемся AlexNet.

2.1.2. AlexNet

В 2012 году архитектура AlexNet с большим отрывом от соперников выиграла конкурс ILSVRC с частотой ошибок топ-5 (коэффициент правильных ошибок) (то есть с наличием правильной метки в пяти лучших предсказаниях) в 15,4 %. Для сравнения: занявшая второе место архитектура, не основанная на нейронных сетях, показала результат лишь 26,2 %. Это был поворотный момент в истории машинного зрения: момент, когда сообщество стало осознавать потенциал глубокого обучения для задач машинного зрения. За этим прорывом последовал период непрерывного совершенствования, и теперь у наиболее современных архитектур и методов обучения частота ошибок топ-5 составляет всего 3 %.

По сегодняшним меркам AlexNet довольно небольшая сеть по сравнению с современными моделями. Но в нашем случае она прекрасно подходит, чтобы познакомиться с реально работающей нейронной сетью и научиться запускать ее предобученную версию для нового изображения.

Структура AlexNet приведена на рис. 2.3. Конечно, пока мы еще не знаем всего, что требуется для ее понимания, но кое-что можно сказать уже сейчас.

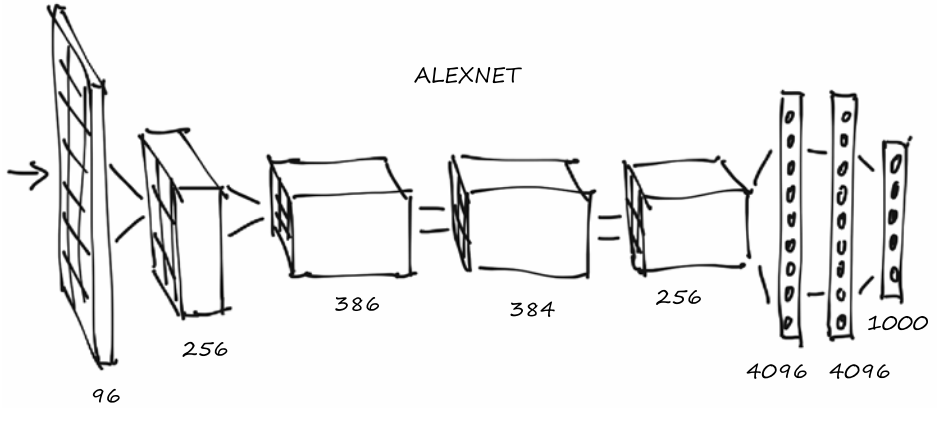

Рис. 2.3. Архитектура AlexNet

Прежде всего, каждый из блоков включает в себя набор операций умножения и сложения, а также небольшое количество прочих функций, как мы увидим в главе 5. Это можно считать своего рода фильтром — функцией, получающей на входе одно или несколько изображений и генерирующей в качестве выходного сигнала другие изображения. Конкретный способ определяется во время обучения на основе *просмотренных* моделью примеров данных, а также желаемых выходных сигналов для них.

На рис. 2.3 входные изображения поступают слева и проходят через пять комплектов фильтров, каждый из которых формирует некоторое количество выходных изображений. С каждым фильтром изображения уменьшаются в размере. Полученные последним комплектом фильтров изображения представляются в виде одномерного вектора из 4096 элементов и классифицируются, в результате чего генерируется 1000 выходных значений вероятности, по одному для каждого выходного класса.

Для запуска архитектуры AlexNet на каком-либо входном изображении необходимо создать экземпляр класса `AlexNet`. Вот таким образом:

```
# In[3]:
alexnet = models.AlexNet()
```

На этом этапе `alexnet` представляет собой объект, подходящий для выполнения архитектуры AlexNet. Понимание всех нюансов данной архитектуры нам сейчас не требуется. Пока что `alexnet` для нас представляет собой просто некий объект — «черный ящик», который можно запускать как функцию. Подав на вход `alexnet` входные данные четко определенного размера, мы выполним прямой проход (forward pass) по сети, при котором входной сигнал пройдет через первый набор нейронов, выходные сигналы которых будут поданы на вход следующего набора нейронов, и так до самого итогового выходного сигнала. На практике это означает, что при наличии объекта `input` нужного типа можно произвести прямой проход с помощью оператора `output = alexnet(input)`.

Но если мы так поступим, то пропустим данные через всю сеть лишь для того, чтобы получить... мусор! А все потому, что сеть не была инициализирована: ее веса, числа, с которыми складываются и на которые умножаются входные сигналы, не были обучены на чем-либо, сеть сама по себе — чистый (или, точнее, *случайный*) лист. Необходимо либо обучить ее с нуля, либо загрузить веса, полученные в результате предыдущего обучения, что мы сейчас и сделаем.

Для этого вернемся к модулю `models`. Мы уже знаем, что названия в верхнем регистре соответствуют классам, реализующим популярные архитектуры, предназначенные для машинного зрения. С другой стороны, названия в нижнем регистре соответствуют функциям, создающим экземпляры моделей с заранее определенным количеством слоев и нейронов, а также, возможно, скачивающие и загружающие в них предобученные веса. Обратите внимание, что эти функции

несущественны, они просто упрощают создание экземпляра модели с соответствующим предобученной сети количеством слоев и нейронов.

2.1.3. ResNet

Сейчас с помощью функции `resnet101` мы создадим экземпляр сверточной нейронной сети из 101 слоя. Для наглядности: до появления остаточных нейронных сетей в 2015-м считалось, что добиться настолько устойчивого обучения при подобной глубине сети чрезвычайно сложно. Остаточные нейронные сети ухитрились сделать это возможным и тем самым побили несколько рекордов за один раз.

Создадим теперь экзепляр данной сети. Для этого мы передадим нашей функции аргумент, который укажет ей скачать веса `resnet101`, обученные на наборе данных ImageNet, включающем 1,2 миллиона изображений и 1000 категорий:

```
# In[4]:
resnet = models.resnet101(pretrained=True)
```

Пока мы смотрим на процесс загрузки, оцените тот факт, что `resnet101` может похвастаться 44,5 миллиона параметров — огромное количество для автоматической оптимизации!

2.1.4. На старт, внимание, почти что марш

Хорошо, что же мы только что сделали? Раз уж нам так интересно, можно взглянуть одним глазом, что представляет собой `resnet101`. Для этого можно вывести на экран значение возвращаемой модели и получить текстовое представление информации, аналогичной той, что мы видели в разделе 2.3, со всеми подробностями о структуре сети. На данный момент нам столько информации не нужно, но по мере чтения книги вы постепенно начнете понимать, что этот код нам говорит:

```
# In[5]:
resnet

# Out[5]:
ResNet(
(conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3),
                bias=False)
(bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
                   track_running_stats=True)
(relu): ReLU(inplace)
(maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1,
                     ceil_mode=False)
(layer1): Sequential(
```

```
 (0): Bottleneck(
...
  )
 )
 (avgpool): AvgPool2d(kernel_size=7, stride=1, padding=0)
 (fc): Linear(in_features=2048, out_features=1000, bias=True)
)
```

Здесь мы видим `modules`, по одному на строку. Обратите внимание, что ничего общего с модулями Python у них нет: это отдельные операции, «кирпичики» нейронной сети. В других фреймворках глубокого обучения их также называют *слоями* (*layers*).

Если прокрутить вниз, можно увидеть множество модулей `Bottleneck`, один за другим (всего 101!), содержащих операции свертки и прочие модули. Именно так и устроена типичная глубокая нейронная сеть для машинного зрения: более или менее последовательный каскад фильтров и нелинейных функций, завершающийся слоем (`fc`), генерирующим оценки для каждого из 1000 выходных классов (`out_features`).

Переменную `resnet` можно вызывать как функцию, при этом она принимает на входе одно или несколько изображений и генерирует соответствующее количество оценок для каждого из классов ImageNet. Перед этим, впрочем, необходимо предварительно привести входные изображения к нужному размеру, а их значения (цвета) примерно в один числовой диапазон. Для этого модуль `torchvision` предоставляет преобразования (`transforms`), позволяющие быстро описывать конвейеры простейших операций предварительной обработки:

```
# In[6]:
from torchvision import transforms
preprocess = transforms.Compose([
  transforms.Resize(256),
  transforms.CenterCrop(224),
  transforms.ToTensor(),
  transforms.Normalize(
    mean=[0.485, 0.456, 0.406],
    std=[0.229, 0.224, 0.225]
)])
```

Здесь мы описали функцию `preprocess`, масштабирующую входное изображение до размера 256 × 256, обрезающую его до 224 × 224 по центру, преобразующую в тензор (многомерный массив PyTorch: в данном случае трехмерный массив, содержащий цвет, высоту и ширину) и нормализующую его компоненты RGB (красный, зеленый, синий) до заданных среднего значения и стандартного отклонения. Если мы хотим получить от сети осмысленные ответы, все это должно соответствовать данным, полученным сетью во время обучения. Мы обсудим преобразования подробнее, когда будем создавать свои собственные модели распознавания изображений в подразделе 7.1.3.

Возьмем теперь изображение нашей любимой собаки (`bobby.jpg` из репозитория GitHub), проведем предварительную обработку и посмотрим, что о нем думает модель ResNet. Начнем с загрузки изображения из локальной файловой системы с помощью Pillow (https://pillow.readthedocs.io/en/stable) — модуля Python для обработки изображений:

```
# In[7]:
from PIL import Image
img = Image.open("../data/p1ch2/bobby.jpg")
```

Если вы работаете с блокнотом Jupyter, то, чтобы посмотреть в нем это изображение, необходимо выполнить следующую команду (изображение будет показано на месте `<PIL.JpegImagePlugin…>`):

```
# In[8]:
img
# Out[8]:
<PIL.JpegImagePlugin.JpegImageFile image mode=RGB size=1280x720 at
  0x1B1601360B8>
```

Либо можно вызвать метод `show`, при этом всплывет окошко с программой просмотра, в которой будет показано изображение с рис. 2.4.

Рис. 2.4. Бобби, наше особенное входное изображение

Далее можно пропустить это изображение через наш конвейер предварительной обработки:

```
# In[9]:
img_t = preprocess(img)
```

При этом можно изменять размер изображения, обрезать его и нормализовывать входной тензор так, как нужно сети. Мы поговорим об этом подробнее в следующих двух главах; а пока что держитесь крепче:

```
# In[10]:
import torch
batch_t = torch.unsqueeze(img_t, 0)
```

Все готово к запуску модели.

2.1.5. Марш!

Процесс выполнения обученной модели на новых данных в сфере глубокого обучения называется *выводом* (*inference*). Для выполнения вывода необходимо перевести сеть в режим `eval`:

```
# In[11]:
resnet.eval()
```

```
# Out[11]:
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3),
                  bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
                  track_running_stats=True)
  (relu): ReLU(inplace)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1,
                  ceil_mode=False)
  (layer1): Sequential(
   (0): Bottleneck(
...
    )
   )
  (avgpool): AvgPool2d(kernel_size=7, stride=1, padding=0)
  (fc): Linear(in_features=2048, out_features=1000, bias=True)
 )
```

Если забыть сделать это, некоторые предобученные модели, например включающие *нормализацию по мини-батчам* и *дропаут*, не дадут никаких осмысленных результатов просто по причине их внутреннего устройства. Теперь, после установки режима `eval`, можно выполнять вывод:

```
# In[12]:
out = resnet(batch_t)
out
# Out[12]:
tensor([[ -3.4803, -1.6618, -2.4515, -3.2662, -3.2466, -1.3611,
          -2.0465, -2.5112, -1.3043, -2.8900, -1.6862, -1.3055,
...
          2.8674, -3.7442, 1.5085, -3.2500, -2.4894, -0.3354,
          0.1286, -1.1355, 3.3969, 4.4584]])
```