5 Глубокое обучение для распознавания образов

Эта глава охватывает следующие темы:

- ✓ суть сверточных нейронных сетей;
- ✓ расширение обучающего набора данных для ослабления эффекта переобучения;
- ✓ использование предварительно обученной сверточной нейронной сети для извлечения признаков;
- ✓ дообучение предварительно обученной сверточной нейронной сети;
- ✓ визуализация процессов обучения сверточных нейронных сетей и принятия ими решений.

Эта глава знакомит со сверточными нейронными сетями — разновидностью моделей глубокого обучения, почти повсеместно используемых в приложениях распознавания образов. Здесь вы научитесь применять сверточные нейронные сети для решения задач классификации изображений, и в частности задач с небольшими наборами обучающих данных, которые наиболее распространены, если только вы не работаете в крупной технологической компании.

5.1. Введение в сверточные нейронные сети

В этом разделе мы погрузимся в теорию сверточных нейронных сетей и выясним причины их успеха в задачах распознавания образов. Но сначала рассмотрим практический пример простой сверточной нейронной сети, в котором сеть используется для классификации изображений цифр из набора MNIST. Эту задачу мы решили

в главе 2, используя полносвязную сеть (ее точность на контрольных данных составила 97,8 %). Несмотря на простоту сверточной нейронной сети, ее точность будет значительно выше полносвязной модели из главы 2.

В листинге 5.1 показано, как выглядит простая сверточная нейронная сеть. Это стек слоев `layer_conv_2d` и `layer_max_pooling_2d`. Как она действует, рассказывается чуть ниже.

Листинг 5.1. Создание небольшой сверточной нейронной сети

```
library(keras)

model <- keras_model_sequential() %>%
  layer_conv_2d(filters = 32, kernel_size = c(3, 3), activation = "relu",
                input_shape = c(28, 28, 1)) %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_conv_2d(filters = 64, kernel_size = c(3, 3),
                              activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_conv_2d(filters = 64, kernel_size = c(3, 3), activation = "relu")
```

Важно отметить, что данная сеть принимает на входе тензоры с формой (`высота_изображения, ширина_изображения, каналы`) (не включая измерение, определяющее пакеты). В данном случае мы настроили сеть на обработку входов с размерами (`28, 28, 1`), соответствующими формату изображений в наборе MNIST, передав аргумент `input_shape=(28, 28, 1)` в первый слой.

Рассмотрим поближе текущую архитектуру сети:

```
> model
_____________________________________________________________________
Layer (type)                     Output Shape          Param #
=====================================================================
conv2d_1 (Conv2D)                (None, 26, 26, 32)    320
_____________________________________________________________________
maxpooling2d_1 (MaxPooling2D)    (None, 13, 13, 32)    0
_____________________________________________________________________
conv2d_2 (Conv2D)                (None, 11, 11, 64)    18496
_____________________________________________________________________
maxpooling2d_2 (MaxPooling2D)    (None, 5, 5, 64)      0
_____________________________________________________________________
conv2d_3 (Conv2D)                (None, 3, 3, 64)      36928
=====================================================================
Total params: 55,744
Trainable params: 55,744
Non-trainable params: 0
```

Как видите, все слои `layer_conv_2d` и `layer_max_pooling_2d` выводят трехмерный тензор с формой (`высота, ширина, каналы`). Измерения ширины и высоты сжимаются с ростом глубины сети. Количество каналов управляется первым аргументом, передаваемым в слои `layer_conv_2d` (32 или 64).

Следующий шаг — передача последнего выходного тензора (с формой `(3, 3, 64)`) на вход полносвязной классифицирующей сети, подобной той, с которой мы уже знакомы, — стека слоев `Dense`. Эти классификаторы обрабатывают векторы — одномерные массивы, тогда как текущий выход является трехмерным тензором. Поэтому мы должны прежде преобразовать трехмерный вывод в одномерный и затем добавить сверху несколько слоев `Dense`.

Листинг 5.2. Добавление классификатора поверх сверточной нейронной сети

```
model <- model %>%
  layer_flatten() %>%
  layer_dense(units = 64, activation = "relu") %>%
  layer_dense(units = 10, activation = "softmax")
```

Мы реализуем 10-видовую классификацию, используя конечный слой с 10 выходами и функцией активации `softmax`. Вот как выглядит сеть теперь:

```
>>> model.summary()
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
maxpooling2d_1 (MaxPooling2D)  (None, 13, 13, 32)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
maxpooling2d_2 (MaxPooling2D)  (None, 5, 5, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 3, 3, 64)          36928
_________________________________________________________________
flatten_1 (Flatten)          (None, 576)               0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                36928
_________________________________________________________________
dense_2 (Dense)              (None, 10)                650
=================================================================
Total params: 93,322
Trainable params: 93,322
Non-trainable params: 0
```

Как видите, выходы `(3, 3, 64)` преобразуются в векторы с формой `(576,)` перед передачей двум слоям `Dense`.

Теперь давайте обучим сеть, передав ей цифры из набора MNIST. Далее мы будем повторно использовать большое количество программного кода из главы 2.

Листинг 5.3. Обучение сверточной нейронной сети на данных из набора MNIST

```
mnist <- dataset_mnist()
c(c(train_images, train_labels), c(test_images, test_labels)) %<-% mnist
```

```
train_images <- array_reshape(train_images, c(60000, 28, 28, 1))
train_images <- train_images / 255
test_images <- array_reshape(test_images, c(10000, 28, 28, 1))
test_images <- test_images / 255

train_labels <- to_categorical(train_labels)
test_labels <- to_categorical(test_labels)

model %>% compile(
  optimizer = "rmsprop",
  loss = "categorical_crossentropy",
  metrics = c("accuracy")
)

model %>% fit(
  train_images, train_labels,
  epochs = 5, batch_size=64
)
```

Оценим модель на контрольных данных:

```
> results <- model %>% evaluate(test_images, test_labels)
> results
$loss
[1] 0.02563557

$acc
[1] 0.993
```

Полносвязная сеть из главы 2 показала точность 97,8 % на контрольных данных, а простенькая сверточная нейронная сеть показала точность 99,3 %: мы уменьшили процент ошибок на 68 % (относительно). Неплохо!

Но почему такая простая сверточная нейронная сеть работает настолько хорошо в сравнении с полносвязной моделью? Чтобы ответить на этот вопрос, погрузимся в особенности работы слоев `layer_conv_2d` и `layer_max_pooling_2d`.

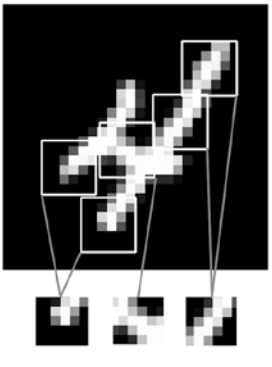

Рис. 5.1. Изображения можно разбить на локальные шаблоны, такие как края, текстуры и т. д.

5.1.1. Операция свертывания

Основное отличие полносвязного слоя от сверточного заключается в следующем: слои `Dense` изучают глобальные шаблоны в пространстве входных признаков (например, в случае с цифрами из набора MNIST это шаблоны, вовлекающие все пикселы), тогда как сверточные слои изучают локальные шаблоны (рис. 5.1), в случае с изображениями —

шаблоны в небольших двумерных окнах во входных данных. В предыдущем примере все такие окна имели размеры 3×3.

Эта ключевая характеристика наделяет сверточные нейронные сети двумя важными свойствами:

- *Шаблоны, которые они изучают, являются инвариантными в отношении переноса*. После изучения определенного шаблона в правом нижнем углу картинки сверточная нейронная сеть сможет распознавать его повсюду: например, в левом верхнем углу. Полносвязной сети пришлось бы изучить шаблон заново, если он появляется в другом месте. Это увеличивает эффективность сверточных сетей в задачах обработки изображений (потому что *видимый мир по своей сути является инвариантным в отношении переноса*): таким сетям требуется меньше обучающих образцов для получения представлений, обладающих силой обобщения.
- *Они могут изучать пространственные иерархии шаблонов* (рис. 5.2). Первый сверточный слой будет изучать небольшие локальные шаблоны, такие как края, второй — более крупные шаблоны, состоящие из признаков, возвращаемых первым слоем, и т. д. Это позволяет сверточным нейронным сетям эффективно изучать все более сложные и абстрактные визуальные представления (потому что *видимый мир по своей сути является пространственно-иерархическим*).

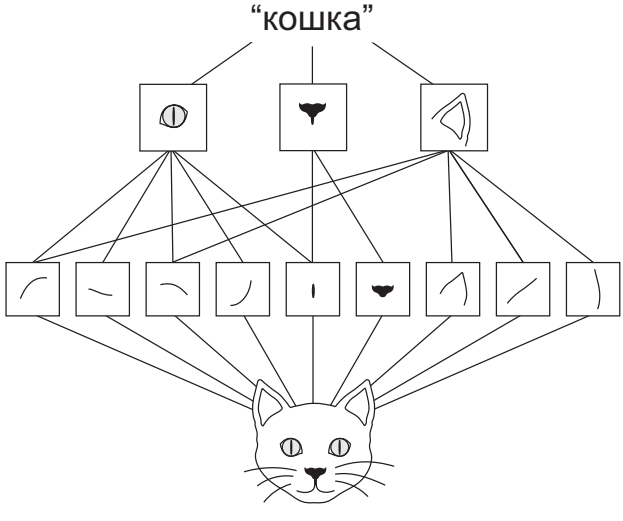

Рис. 5.2. Видимый мир формируется пространственными иерархиями видимых модулей: гиперлокальные края объединяются в локальные объекты, такие как глаза или уши, которые, в свою очередь, объединяются в понятия еще более высокого уровня, такие как «кошка»

Свертка применяется к трехмерным тензорам, называемым *картами признаков*, с двумя пространственными осями (высота и ширина), а также с *осью глубины* (или

осью каналов). Для изображений в формате RGB размерность оси глубины равна 3, потому что имеется три канала цвета: красный (red), зеленый (green) и синий (blue). Для черно-белых изображений, как в наборе MNIST, ось глубины имеет размерность 1 (оттенки серого). Операция свертывания извлекает шаблоны из своей входной карты признаков и применяет одинаковые преобразования ко всем шаблонам, производя *выходную карту признаков*. Эта выходная карта признаков также является трехмерным тензором: она имеет ширину и высоту. Ее глубина может иметь любую размерность, потому что выходная глубина является параметром слоя, и разные каналы на этой оси глубины больше не соответствуют конкретным цветам, как во входных данных в формате RGB; скорее, они соответствуют *фильтрам*. Фильтры представляют конкретные аспекты входных данных: на верхнем уровне, например, фильтр может соответствовать понятию «присутствие лица на входе».

В примере MNIST первый сверточный слой принимает карту признаков с размером `(28, 28, 1)` и выводит карту признаков с размером `(26, 26, 32)`: он вычисляет 32 фильтра по входным данным. Каждый из этих 32 выходных каналов содержит сетку 26 × 26 значений — *карту ответов* фильтра на входных данных, определяющую ответ этого шаблона фильтра для разных участков входных данных (рис. 5.3). Вот что означает термин *карта признаков*: каждое измерение на оси глубины — это признак (или фильтр), а двумерный тензор `output[:, :, n]` — это двумерная пространственная *карта ответов* этого фильтра на входных данных.

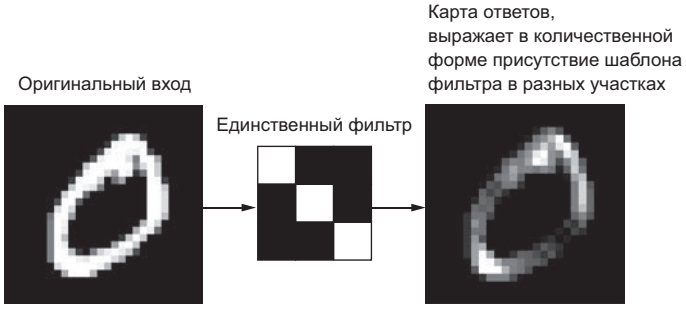

Рис. 5.3. Понятие карты ответов: двумерная карта присутствия шаблона в разных участках входных данных

Свертки определяются двумя ключевыми параметрами:

- *размером шаблонов, извлекаемых из входных данных* — обычно 3 × 3 или 5 × 5. В данном примере используется размер 3 × 3, что является распространенным выбором;
- *глубиной выходной карты признаков* — количеством фильтров, вычисляемых сверткой. В данном примере свертка начинается с глубины 32 и заканчивается глубиной 64.

В Keras эти параметры передаются в слой в первых аргументах: `layer_conv_2d (выходная_глубина, с (высота_окна, ширина_окна))`.