

8

Классификация изображений

В этой главе

- ✓ Суть сверточных нейронных сетей.
- ✓ Обогащение обучающего набора данных для ослабления эффекта переобучения.
- ✓ Использование предварительно обученной сверточной нейронной сети для извлечения признаков.
- ✓ Дообучение предварительно обученной сверточной нейронной сети.

Компьютерное зрение — одна из самых первых технологий, где глубокое обучение добилось значительных успехов. Оно послужило толчком к первоначальному подъему сферы глубокого обучения в период с 2011 по 2015 годы. Примерно тогда же модели глубокого обучения для компьютерного зрения — *сверточные нейронные сети* — стали показывать удивительно хорошие результаты в состязаниях по классификации изображений. Сначала Дэн Киресан победил в двух специализированных состязаниях: соревнованиях по распознаванию китайских символов ICDAR 2011 и соревнованиях по распознаванию дорожных знаков Германии IJCNN 2011. Затем произошло еще более значимое событие: осенью 2012 года группа Хинтона выиграла крупномасштабное состязание по визуальному распознаванию изображений из набора ImageNet. После этого начали появляться многообещающие результаты в других задачах компьютерного зрения.

Интересно отметить, что первых успехов было недостаточно для того, чтобы сделать глубокое обучение популярным, — на это потребовалось несколько лет. Сообщество исследователей технологий компьютерного зрения потратило много лет на разработку методов, не связанных с нейронными сетями, и не было готово в одночасье отказаться от них только потому, что на пороге появилось что-то новое. В 2013 и 2014 годах многие ученые, работающие в области компьютерного зрения, все еще встречали идею глубокого обучения с большим скептицизмом, и только в 2016-м она наконец заняла доминирующие позиции. Один из авторов вспоминает, как в феврале 2014 года убеждал своего бывшего профессора заняться глубоким обучением. «У этой технологии большое будущее», — говорил он. «А мне кажется, это временное поветрие», — парировал профессор. Но уже в 2016 году вся его лаборатория занималась глубоким обучением. Ничто не остановит идею, время которой пришло.

Каждый день мы взаимодействуем с моделями компьютерного зрения — через Google Photos, поиск изображений Google, камеры наших телефонов, YouTube, программные инструменты оптического распознавания текста и множество других приложений. Также эти модели широко используются в передовых исследованиях в сфере автоматического управления транспортными средствами, робототехники, медицинской диагностики с помощью искусственного интеллекта, автоматических систем кассового обслуживания в магазинах и даже автоматизации сельского хозяйства.

Данная глава знакомит со сверточными нейронными сетями, также известными как *convnets*, — разновидностью моделей глубокого обучения, почти повсеместно используемой в приложениях компьютерного зрения (распознавания образов). Здесь вы научитесь применять сверточные нейронные сети для решения задач классификации изображений, в частности, задач с небольшими наборами обучающих данных, которые распространены наиболее широко (если только вы не работаете в крупной технологической компании).

8.1. Введение в сверточные нейронные сети

В этом разделе мы погрузимся в теорию сверточных нейронных сетей и выясним причины их успеха при решении задач распознавания образов. Но сначала рассмотрим практический пример простой сверточной нейронной сети, классифицирующей изображения рукописных 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 точности полносвязной модели из главы 2.

В листинге 8.1 показано, как выглядит простая сверточная нейронная сеть. Это стек слоев `Conv2D` и `MaxPooling2D`. Как она действует, расскажем чуть позже. Мы построим модель с помощью функционального API, с которым познакомились в предыдущей главе.

Листинг 8.1. Создание небольшой сверточной нейронной сети

```
import keras
from keras import layers

inputs = keras.Input(shape=(28, 28, 1))
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(inputs)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.GlobalAveragePooling2D()(x)
outputs = layers.Dense(10, activation="softmax")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

Важно отметить, что данная сеть принимает на входе тензоры формы (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.

Рассмотрим поближе текущую архитектуру сети (листинг 8.2).

Листинг 8.2. Сводная информация о сети

```
>>> model.summary()
Model: "functional"
```

Layer (type)	Output Shape	Param #
=====		
input_layer (InputLayer)	[(None, 28, 28, 1)]	0
conv2d (Conv2D)	(None, 26, 26, 64)	640
max_pooling2d (MaxPooling2D)	(None, 13, 13, 64)	0
conv2d_1 (Conv2D)	(None, 11, 11, 128)	73,856
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 128)	0
conv2d_2 (Conv2D)	(None, 3, 3, 256)	295,168
global_average_po (GlobalAveragePooling2D)	(None, 256)	0
dense (Dense)	(None, 10)	2,570
=====		
Total params: 372,234 (1.42 MB)		
Trainable params: 372,234 (1.42 MB)		
Non-trainable params: 0 (0.00 B)		

Как видите, все слои Conv2D и MaxPooling2D выводят трехмерный тензор формы (высота, ширина, каналы). Измерения ширины и высоты сжимаются с ростом глубины сети. Количество каналов управляется первым аргументом, передаваемым в слои Conv2D (64, 128 или 256).

Форматы изображений в фреймворках глубокого обучения

Некоторые библиотеки глубокого обучения перемещают каналы на первое место в тензорах изображений (это характерно для большей части экосистемы PyTorch). В ходе работы с такими библиотеками изображения следует передавать в форме (каналы, высота, ширина) вместо (высота, ширина, каналы).

Это всего лишь вопрос соглашения, которое в Keras можно изменить вызовом `keras.config.set_image_data_format("channels_first")` или передачей аргумента `data_format` любому слою свертки или объединения. Но если нет необходимости переупорядочивать положение каналов, то можно оставить значение по умолчанию неизменным.

Последний слой `Conv2D` выдает результат формы $(3, 3, 256)$ — карту признаков 3×3 с 256 каналами. Следующий шаг — передача этого результата 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. Чтобы преодолеть это несоответствие, мы преобразуем трехмерный вывод в одномерный с помощью слоя `GlobalAveragePooling2D`, а затем добавим полносвязные слои `Dense`. Этот слой будет получать среднее значение каждой карты признаков размером 3×3 в тензоре формы $(3, 3, 256)$, в результате чего получится выходной вектор формы $(256,)$. В заключение выполняется классификация по 10 категориям, поэтому последний слой имеет 10 выходов и активацию `softmax`.

Теперь обучим сверточную сеть распознавать цифры MNIST. Мы повторно задействуем код из главы 2 (листинг 8.3). Поскольку модель выполняет классификацию по 10 категориям с активацией `softmax`, используем функцию потерь категориальной перекрестной энтропии, а так как метки являются целыми числами, нам понадобится разреженная версия `sparse_categorical_crossentropy`.

Листинг 8.3. Обучение сверточной нейронной сети на данных из набора MNIST

```
from keras.datasets import mnist

(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images = train_images.reshape((60000, 28, 28, 1))
train_images = train_images.astype("float32") / 255
test_images = test_images.reshape((10000, 28, 28, 1))
test_images = test_images.astype("float32") / 255
model.compile(
    optimizer="adam",
    loss="sparse_categorical_crossentropy",
    metrics=["accuracy"],
)
model.fit(train_images, train_labels, epochs=5, batch_size=64)
```

Оценим модель на контрольных данных (листинг 8.4).

Листинг 8.4. Оценка сверточной сети

```
>>> test_loss, test_acc = model.evaluate(test_images, test_labels)
>>> print(f"Test accuracy: {test_acc:.3f}")
Test accuracy: 0.991
```

Полносвязная сеть из главы 2 показала точность 97,8 % на контрольных данных, а простенькая сверточная нейронная сеть — 99,1 %: мы уменьшили количество ошибок на 60 % (относительно). Неплохо!

Но почему такая простая сверточная нейронная сеть работает намного лучше полносвязной модели? Чтобы ответить на этот вопрос, погрузимся в особенности работы слоев Conv2D и MaxPooling2D.

8.1.1. Операция свертывания

Основное отличие полносвязного слоя от сверточного заключается в следующем: слой Dense изучают глобальные шаблоны в пространстве входных признаков (например, в случае с цифрами из набора MNIST это шаблоны, вовлекающие все пиксели), тогда как сверточные слои изучают локальные шаблоны (рис. 8.1). В случае с изображениями это 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, инвариантны в отношении переноса.* После изучения определенного шаблона, находящегося в правом нижнем углу картинки, сверточная нейронная сеть сможет распознавать его повсюду, например, в левом верхнем углу. Полносвязной сети пришлось бы изучить шаблон заново, появившись он в другом месте. А поскольку *видимый мир по своей сути инвариантен в отношении переноса*, это увеличивает эффективность сверточных сетей при решении задач обработки изображений: им требуется меньше обучающих образцов для получения представлений, обладающих мощностью обобщения.
- *Они могут изучать пространственные иерархии шаблонов* (рис. 8.2). Первый сверточный слой будет изучать небольшие локальные шаблоны, такие как края, второй — более крупные шаблоны, состоящие из признаков, возвращаемых

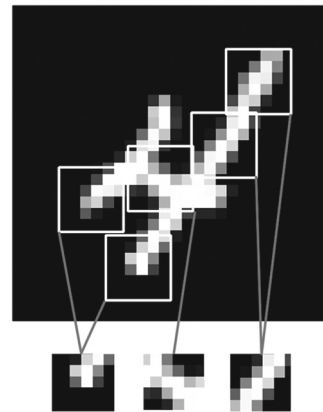


Рис. 8.1. Изображения можно разбить на локальные шаблоны, такие как края, текстуры и т. д.

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*.

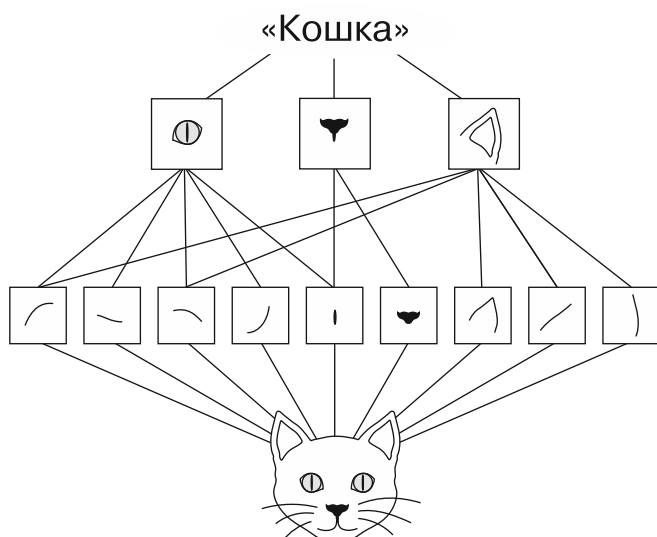


Рис. 8.2. Видимый мир формируется пространственными иерархиями видимых модулей: элементарные линии или текстуры объединяются в простые объекты, такие как глаза или уши, которые, в свою очередь, объединяются в понятия еще более высокого уровня, такие как «кошка»

Свертка применяется к трехмерным тензорам, называемым *картами признаков*, с двумя пространственными осями — *высотой* и *шириной*, а также *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, таких 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*. Выходная карта признаков также является трехмерным тензором: у нее есть ширина и высота. Ее глубина может иметь любую размерность, потому что выходная глубина — это параметр слоя и разные каналы на этой оси глубины больше не соответствуют конкретным цветам, как во входных данных в формате RGB, — скорее, они соответствуют *фильтрам*. Последние 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 размером (28, 28, 1) и выводит карту признаков размером (26, 26, 64): он вычисляет 64 фильтра

по входным данным. Каждый из этих 64 выходных каналов содержит сетку 26×26 значений — *карту ответов* фильтра на входных данных, определяющую ответ этого шаблона фильтра для разных участков входных данных (рис. 8.3). 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* фильтра на входных данных.

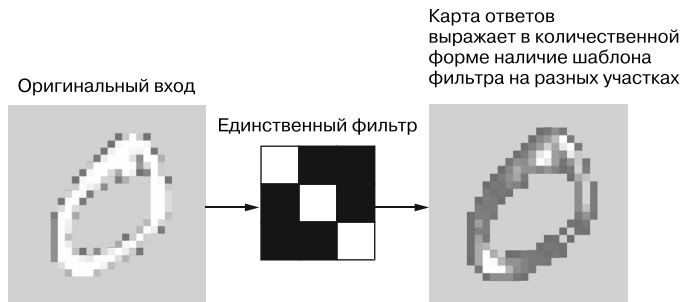


Рис. 8.3. Понятие карты ответов: двумерная карта присутствия шаблона в разных участках входных данных

Свертки определяются двумя ключевыми параметрами:

- *размер шаблонов, извлекаемых из входных данных*, — обычно 3×3 или 5×5 . В данном примере используется размер 3×3 — его выбирают часто;
- *глубина выходной карты признаков* — количество фильтров, вычисляемых сверткой. В данном примере свертка начинается с глубины 32 и заканчивается глубиной 64.

В Keras эти параметры передаются в слой `Conv2D` в первых аргументах: `Conv2D(выходная_глубина, (высота_окна, ширина_окна))`.

Свертка работает методом скользящего окна: она *двигает* окно размером 3×3 или 5×5 по трехмерной входной карте признаков, останавливается в каждой возможной позиции и извлекает трехмерный шаблон окружающих признаков формы (*высота_окна, ширина_окна, глубина_входа*). Затем каждый такой трехмерный шаблон умножением тензора на матрицу весов, получаемую в ходе обучения (она называется *ядром свертки*), преобразуется в одномерный вектор формы (*выходная глубина, .*). Все эти векторы затем собираются в трехмерную выходную карту формы (*высота, ширина, выходная глубина*). Каждое пространственное местоположение в выходной карте признаков соответствует тому же местоположению во входной карте признаков (например, правый нижний угол выхода содержит информацию о правом нижнем угле входа). Так, для окна 3×3 вектор `output[i, j, :]` соответствует трехмерному шаблону `input[i-1:i+2, j-1:j+2, :]`. Целиком процесс изображен на рис. 8.4.

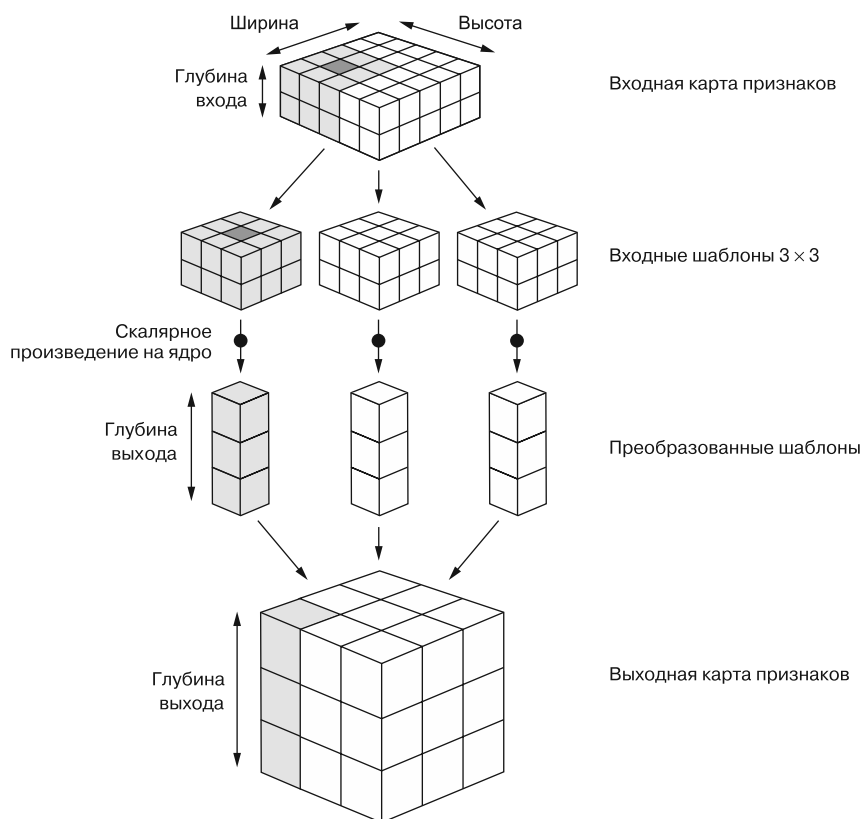


Рис. 8.4. Принцип действия свертки

Обратите внимание на то, что выходные ширина и высота могут отличаться от входных. На то есть две причины:

- эффекты границ, которые могут устраняться дополнением входной карты признаков;
- использование *шага свертки*, определение которого приведем чуть позже.

Рассмотрим эти понятия подробнее.

Эффекты границ и дополнение

Перед нами карта признаков 5×5 (всего 25 клеток). Существует только девять клеток, в которых может находиться центр окна 3×3 , образующих сетку 3×3 (рис. 8.5). Следовательно, карта выходных признаков будет иметь размер 3×3 . Она получилась немного сжатой — ровно на две клетки вдоль каждого измерения. Вы можете увидеть, как проявляется эффект границ, на более раннем примере: изначально у нас имелось 28×28 входов, количество которых после первого сверточного слоя сократилось до 26×26 .

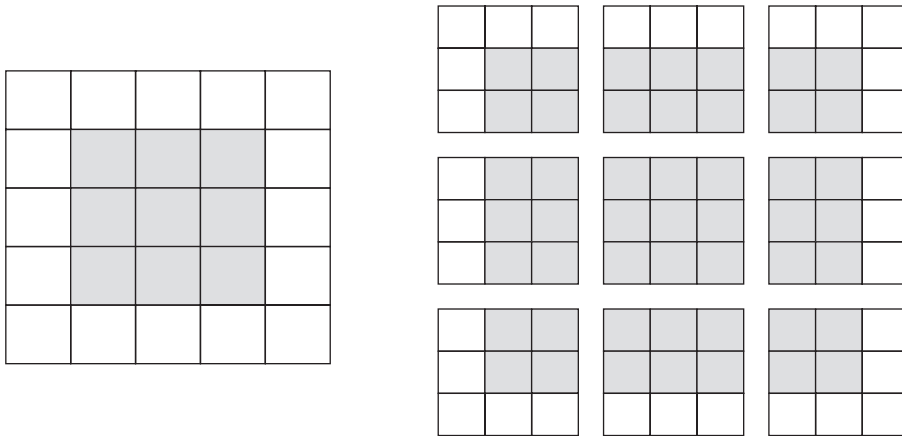


Рис. 8.5. Допустимое местоположение шаблонов 3×3 во входной карте признаков 5×5

Чтобы получить выходную карту признаков с теми же пространственными размерами, что и входная карта, можно использовать *дополнение* (padding). Оно заключается в добавлении с каждой стороны входной карты признаков такого количества строк и столбцов, чтобы можно было поместить центр окна свертки в каждую входную клетку. Для окна 3×3 нужно добавить один столбец справа, один столбец слева и по одной строке сверху и снизу. Для окна 5×5 нужно добавить две строки (рис. 8.6).

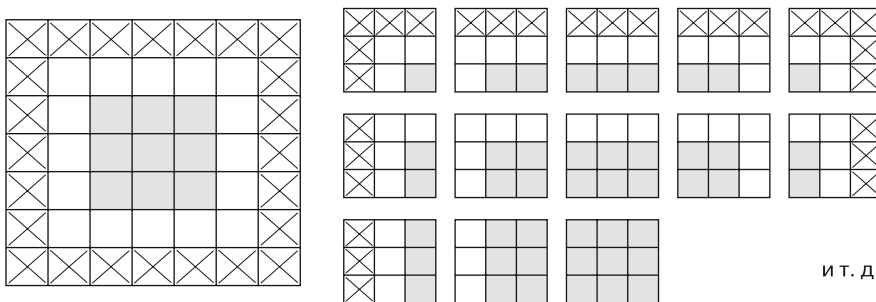


Рис. 8.6. Дополнение входной карты признаков 5×5 для того, чтобы получить 25 шаблонов 3×3

При использовании слоев Conv2D дополнение настраивается с помощью аргумента `padding`, который принимает два значения: `"valid"`, означающее отсутствие дополнения (будут задействованы только допустимые местоположения окна), и `"same"`, означающее «дополнить так, чтобы выходная карта признаков имела те же ширину и высоту, что и входная». По умолчанию аргумент `padding` получает значение `"valid"`.

Шаг свертки

Другой фактор, который может влиять на размер выходной карты признаков, — *шаг свертки*. До сих пор в объяснениях предполагалось, что центральная клетка окна свертки последовательно перемещается в смежные клетки входной карты. Однако в общем случае расстояние между двумя соседними окнами является настраиваемым параметром, который называется *шагом свертки* и по умолчанию равен 1. Также имеется возможность определять *свертки с пробелами* (strided convolutions) — свертки с шагом больше 1. На рис. 8.7 можно видеть, как извлекаются шаблоны 3×3 сверткой с шагом 2 из входной карты 5×5 (без дополнения).

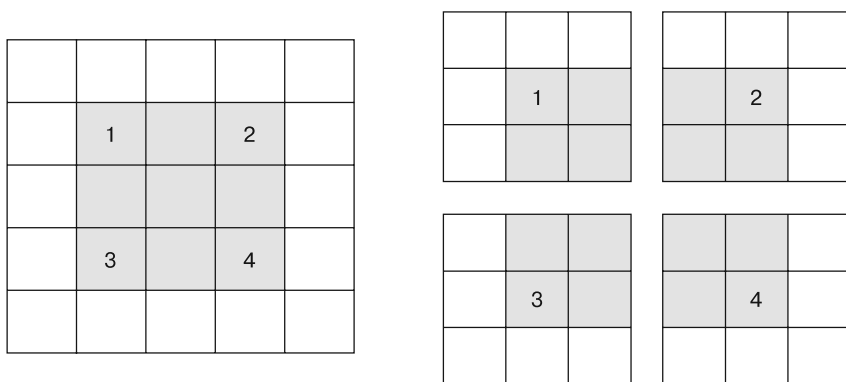


Рис. 8.7. Шаблоны 3×3 свертки с шагом 2×2

Использование шага 2 означает уменьшение ширины и высоты карты признаков за счет уменьшения разрешения в два раза (в дополнение к любым изменениям, вызванным эффектами границ). Свертки с пробелами редко применяются в моделях классификации, но могут пригодиться в моделях некоторых других типов. В следующей главе мы рассмотрим подробнее, как их применить.

В моделях классификации для уменьшения разрешения карты признаков вместо шага часто используется операция *выбора максимального значения из соседних* (max-pooling), которую вы видели в примере первой сверточной нейронной сети. Рассмотрим ее подробнее.

8.1.2. Выбор максимального значения из соседних

Рассматривая пример сверточной нейронной сети, вы могли заметить, что размер карты признаков уменьшается вдвое после каждого слоя **MaxPooling2D**. Например, перед первым слоем **MaxPooling2D** карта признаков имела размер 26×26 , но операция выбора максимального значения из соседних уменьшила ее до 13×13 . Это и есть предназначение данной операции — агрессивное уменьшение разрешения карты признаков, во многом подобное свертке с пробелами.

Операция выбора максимального значения из соседних заключается в следующем: из входной карты признаков извлекается окно, а из него выбирается максимальное значение для каждого канала. Концептуально это напоминает свертку, но вместо преобразования локальных шаблонов с обучением на линейных преобразованиях (ядро свертки) они преобразуются с использованием жестко заданной тензорной операции выбора максимального значения. Главное отличие от свертки состоит в том, что выбор максимального значения из соседних обычно производится с окном 2×2 и шагом 2, чтобы уменьшить разрешение карты признаков в два раза. Собственно свертка, напротив, обычно выполняется с окном 3×3 и без шага (шаг равен 1).

С какой целью вообще снижается разрешение карты признаков? Почему бы просто не убрать слои `MaxPooling2D` и не использовать карты признаков большего размера? Рассмотрим этот вариант. Сверточная основа модели в таком случае будет выглядеть, как в листинге 8.5.

Листинг 8.5. Неверно структурированная сверточная сеть без слоев, выбирающих максимальное значение из соседних

```
inputs = keras.Input(shape=(28, 28, 1))
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(inputs)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.GlobalAveragePooling2D()(x)
outputs = layers.Dense(10, activation="softmax")(x)
model_no_max_pool = keras.Model(inputs=inputs, outputs=outputs)
```

Сводная информация о модели:

```
>>> model_no_max_pool.summary()
Model: "functional_1"
```

Layer (type)	Output Shape	Param #
=====		
input_layer_1 (InputLayer)	[(None, 28, 28, 1)]	0

conv2d_3 (Conv2D)	(None, 26, 26, 64)	320

conv2d_4 (Conv2D)	(None, 24, 24, 128)	73,856

conv2d_5 (Conv2D)	(None, 22, 22, 256)	295,168

global_average_pooling2d_1 (GlobalAveragePooling2D)	(None, 256)	0

dense_1 (Dense)	(None, 10)	2,570
=====		
Total params: 372,234 (1.42 MB)		
Trainable params: 372,234 (1.42 MB)		
Non-trainable params: 0 (0.00 B)		

Что не так в этой конфигурации? Две вещи.

- Она не способствует изучению пространственной иерархии признаков. Окна 3×3 в третьем слое содержат только информацию, поступающую из окон 7×7 в исходных данных. Высокоуровневые шаблоны, изученные с помощью сверточной нейронной сети, будут слишком малы в сравнении с начальными данными, чего может оказаться недостаточно для обучения классификатора цифр (попробуйте распознать цифру, посмотрев на нее через окна 7×7 пикселей!). Нам нужно, чтобы признаки, полученные от последнего сверточного слоя, содержали информацию о совокупности исходных данных.
- Заключительная карта признаков имеет размер 22×22 . Он огромен. Если взять среднее значение каждой карты признаков размером 22×22 , вы потеряете много информации по сравнению с использованием карт признаков 3×3 .

Проще говоря, уменьшение разрешения применяется для уменьшения размеров карт признаков, благодаря чему содержащаяся в них информация становится все менее пространственно распределенной и все более сосредоточенной в каналах, а также для создания иерархий пространственных фильтров за счет того что последовательные слои свертки просматривают изображения через все более крупные окна (с точки зрения долей исходных данных, которые они охватывают).

Обратите внимание на то, что операция выбора максимального значения — не единственный способ уменьшения разрешения. Как вы уже знаете, в предыдущих сверточных слоях можно также использовать шаг свертки. Кроме того, вместо выбора максимального значения можно задействовать операцию выбора среднего значения по соседним элементам (*average pooling*), когда каждый локальный шаблон преобразуется путем взятия среднего значения для каждого канала в шаблоне вместо максимального. Однако операция выбора максимального значения обычно дает лучшие результаты, чем альтернативные решения. Причина в том, что признаки, как правило, кодируют пространственное присутствие некоторого шаблона или понятия в разных клетках карты признаков (отсюда и название — *карта признаков*), соответственно, *максимальное присутствие* признаков намного информативнее *среднего присутствия*. Поэтому для снижения разрешения разумнее сначала получить плотные карты признаков (обычной сверткой без пробелов), а затем рассмотреть максимальные значения признаков в небольших шаблонах, а не разреженные окна из входных данных (сверткой с пробелами) или усредненные шаблоны, которые могут привести к пропуску информации о присутствии.

На данном этапе у вас должно сложиться довольно полное представление об основах сверточных нейронных сетей — картах признаков, операциях свертки и выбора максимального значения по соседним элементам, а также о том, как сконструировать небольшую сверточную нейронную сеть для решения такой простой задачи, как классификация цифр из набора MNIST. Теперь перейдем к более полезным и пригодным на практике применениям.