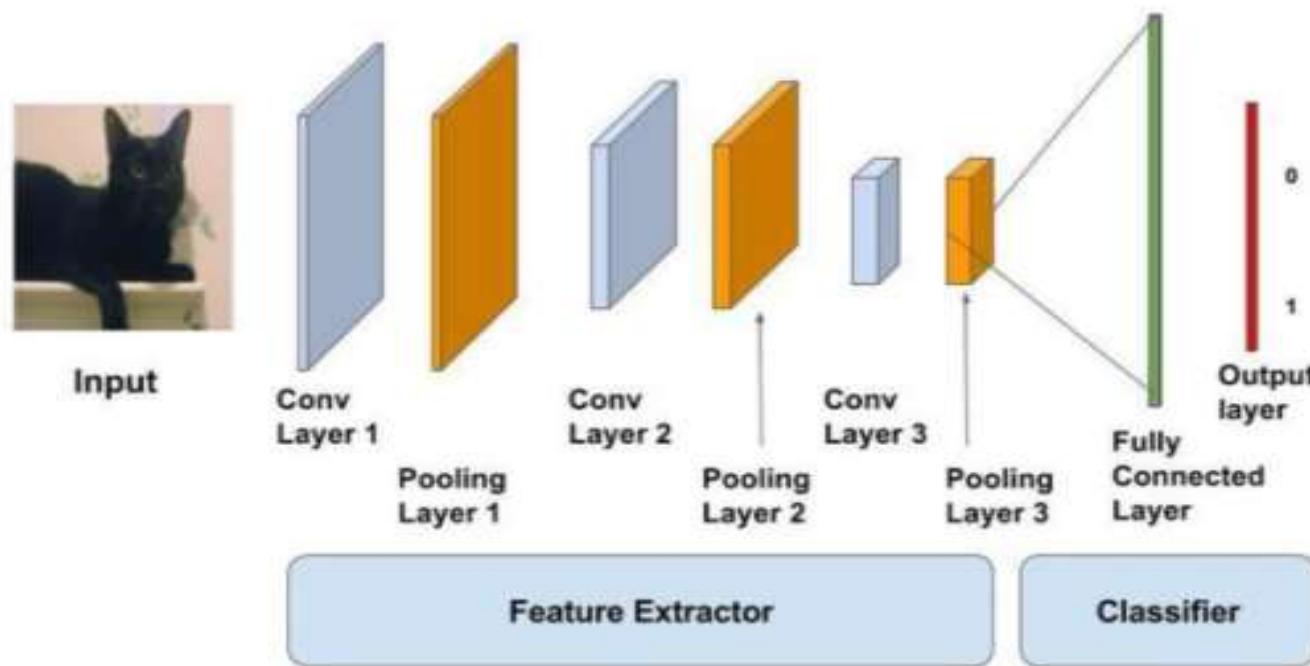
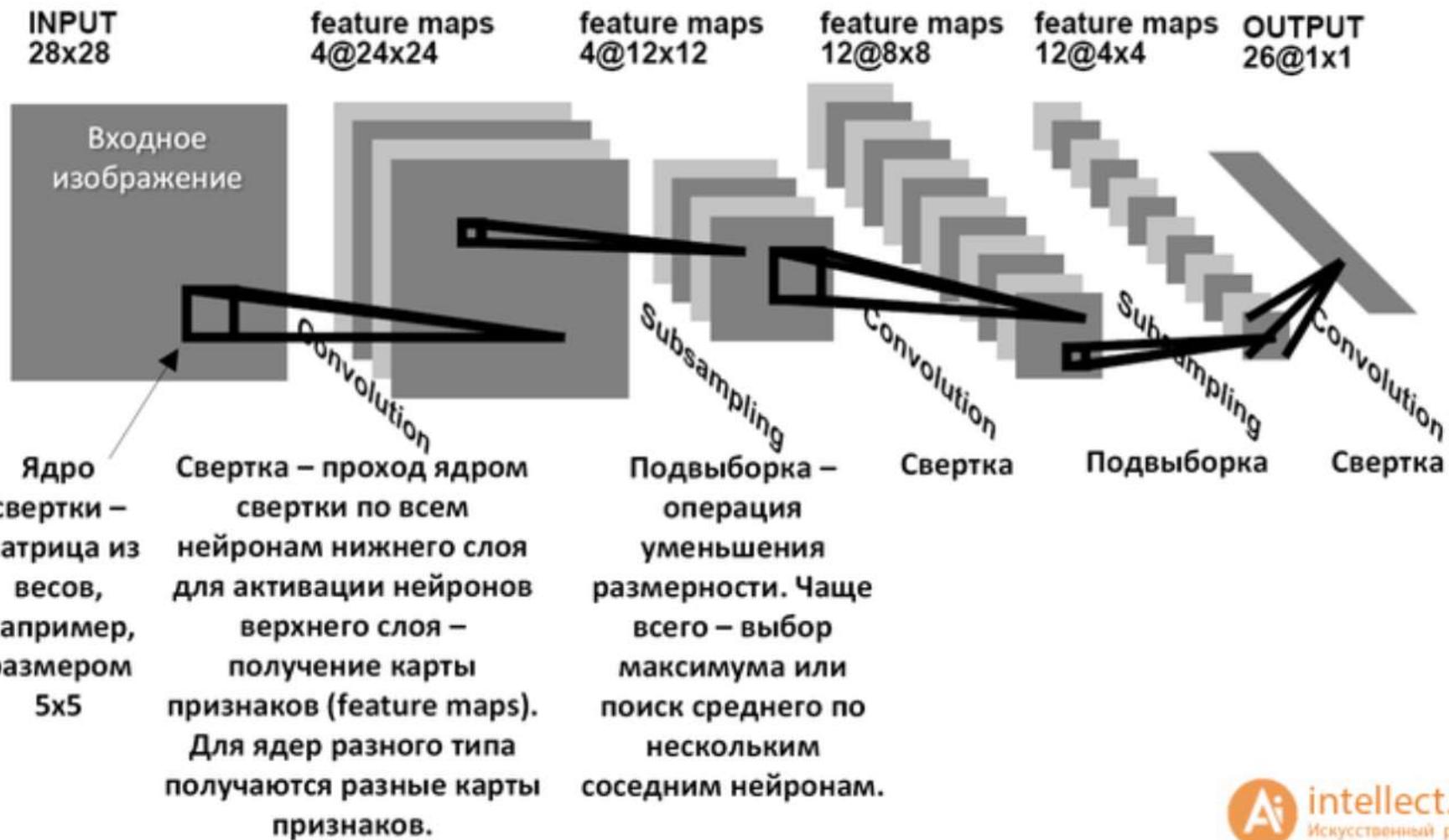


5.17 Архитектура и принципы работы сверточной сети

- Сверточные нейронные сети/Convolutional Neural Networks (CNN) являются одной из форм современных многослойных глубоких нейронных сетей. Здесь и далее приведена обобщенная схема типичной CNN. Первая часть состоит из слоев свертки/convolution и субдискретизации/subsampling (MaxPooling, AveragePooling), которые выступают в качестве экстрактора признаков. Вторая часть состоит из полносвязного слоя, который выполняет нелинейные преобразования извлеченных признаков и действует как классификатор. Выходной сигнал может быть слоем **softmax**, указывающим, есть ли кошка или что-то еще. Следует отметить, что активационная функция **softmax** применяется в большей мере для многопараметрической классификации. Также в качестве выходного может быть использован сигмоидный слой, на выходе которого будет вероятность того, что изображение будет кошкой.



5.17 Архитектура и принципы работы сверточной сети



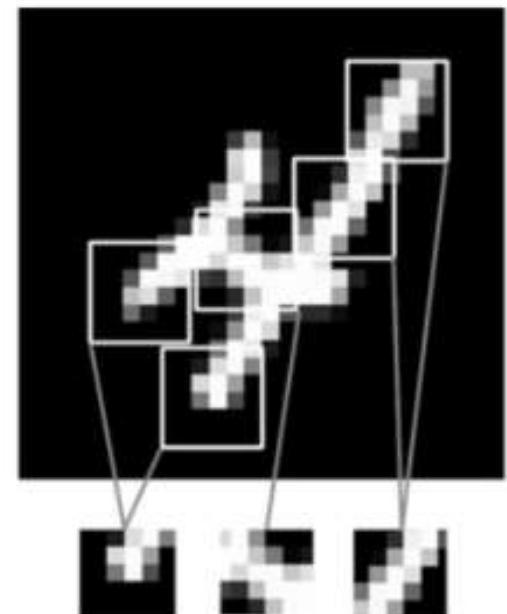
5.17 Архитектура и принципы работы сверточной сети

- В обычном персептроне, который представляет собой полносвязную нейронную сеть, каждый нейрон связан со всеми нейронами предыдущего слоя, причем каждая связь имеет свой персональный весовой коэффициент.
- В свёрточной нейронной сети в операции свёртки используется лишь ограниченная матрица весов небольшого размера, которую «двигают» по всему обрабатываемому слою (в самом начале — непосредственно по входному изображению), формируя после каждого сдвига сигнал активации для нейрона следующего слоя с аналогичной позицией. То есть для различных нейронов выходного слоя используются общие веса — матрица весов, которую также называют набором весов или ядром свёртки. Она построена таким образом, что графически кодирует какой-либо один признак, например, наличие наклонной линии под определенным углом. Тогда следующий слой, получившийся в результате операции свёртки такой матрицей весов, показывает наличие данной наклонной линии в обрабатываемом слое и её координаты, формируя так называемую карту признаков (англ. feature map). Естественно, в свёрточной нейронной сети набор весов не один, а целая гамма, кодирующая всевозможные линии и дуги под разными углами. При этом такие ядра свертки не закладываются исследователем заранее, а формируются самостоятельно путем обучения сети классическим методом обратного распространения ошибки. Проход каждым набором весов формирует свой собственный экземпляр карты признаков, делая нейронную сеть многомерной (много независимых карт признаков на одном слое). Также следует отметить, что при переборе слоя матрицей весов её передвигают обычно не на полный шаг (размер этой матрицы), а на небольшое расстояние. Так, например, при размерности матрицы весов 5×5 её сдвигают на один или два нейрона (пикселя) вместо пяти, чтобы не «перешагнуть» искомый признак.

5.17 Архитектура и принципы работы сверточной сети

5.17.1 Операция свертки

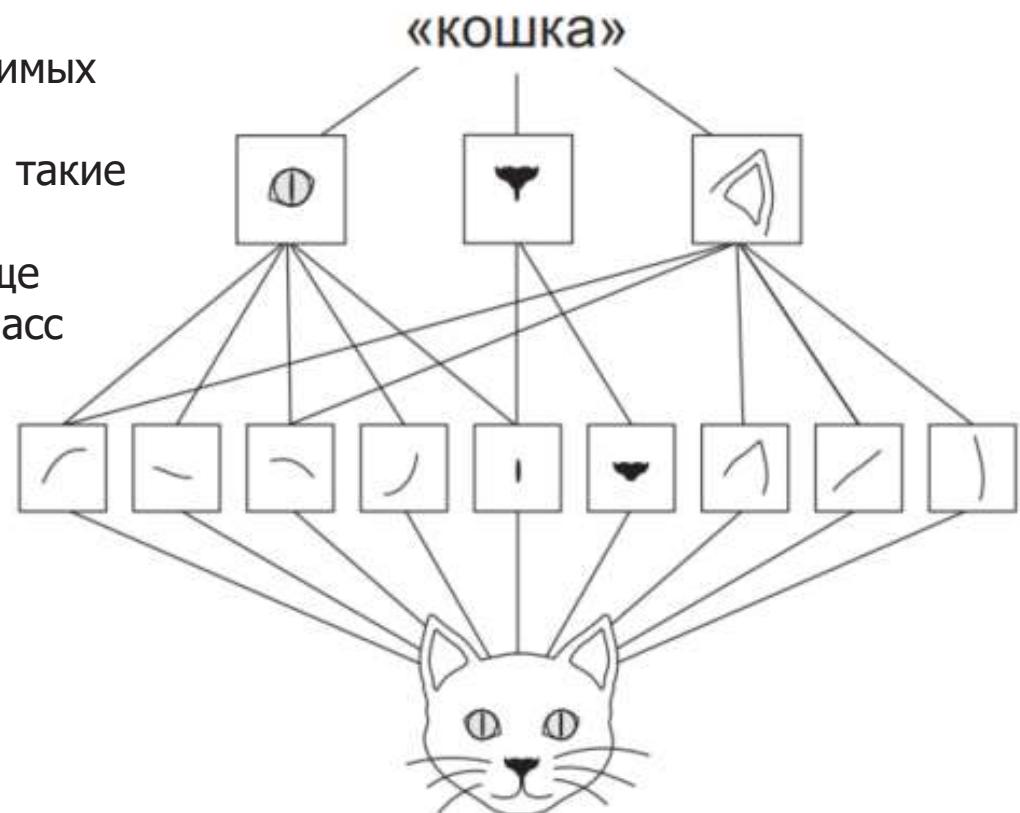
- Основное отличие полносвязного слоя от сверточного заключается в следующем: полносвязные слои (Dense) изучают глобальные шаблоны в пространстве входных признаков (например, в случае с цифрами из набора MNIST это шаблоны, вовлекающие все пиксели), тогда как сверточные слои (Conv2D) изучают локальные шаблоны: в случае с изображениями — шаблоны в небольших двумерных окнах во входных данных, например размерностей 3×3 или 5×5 .
- 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*)



5.17 Архитектура и принципы работы сверточной сети

5.17.1 Операция свертки

- Они могут изучать пространственные иерархии шаблонов. 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*).
- 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 класс «кошка» .



5.17 Архитектура и принципы работы сверточной сети

5.17.1 Операция свертки

- 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 ось глубины имеет размерность 1 (оттенки серого). Операция свертки 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 собой конкретные аспекты входных данных: на верхнем уровне, например, фильтр может соответствовать понятию «присутствие лица на входе».
- В примере MNIST распознавания изображений цифр в оттенках серого 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. Вот что означает термин *карта признаков*: каждое измерение на оси глубины — это признак (или фильтр), а двумерный тензор $\text{output}[:, :, n]$ — это двумерная пространственная *карта ответов* этого фильтра на входных данных.

5.17 Архитектура и принципы работы сверточной сети

5.17.1 Операция свертки

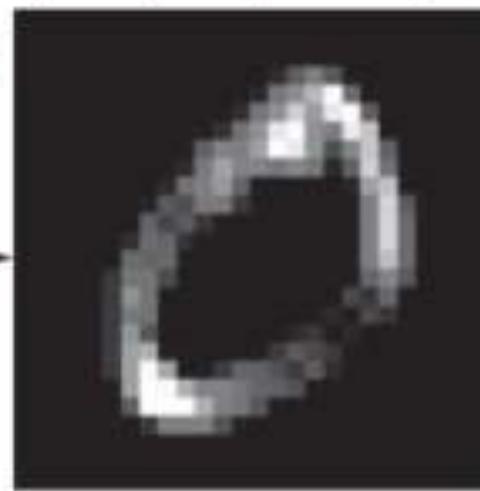
Оригинальный вход



Единственный фильтр



Карта ответов,
выражает в количественной
форме присутствие шаблона
фильтра в разных участках



5.17 Архитектура и принципы работы сверточной сети

5.17.1 Операция свертки

□ Свертки определяются двумя ключевыми параметрами:

- ✓ *Размер шаблонов, извлекаемых из входных данных, — обычно 3×3 или 5×5 .*
- ✓ *Глубина выходной карты признаков — количество фильтров, вычисляемых сверткой.*

□ В Keras эти параметры передаются в слои Conv2D в первых аргументах: Conv2D(выходная_глубина, (высота_окна, ширина_окна))

□ Свертка работает методом скользящего окна: она *двигает* окно с размером $n \times n$ по трехмерной входной карте признаков, останавливается в каждой возможной позиции и извлекает трехмерный шаблон окружающих признаков с формой (высота_окна, ширина_окна, глубина_входа).



5.17 Архитектура и принципы работы сверточной сети

5.17.1 Операция свертки

- Каждый такой трехмерный шаблон затем преобразуется (путем умножения тензора на матрицу весов, получаемую в ходе обучения, которая называется *ядром свертки*) в одномерный вектор с формой (выходная глубина,). Все эти векторы затем собираются в трехмерную выходную карту с формой (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). Например, для окна 3×3 вектор $\text{output}[i, j, :]$ соответствует трехмерному шаблону $\text{input}[i-1:i+2, j-1:j+2, :]$.
- Свертку можно рассматривать как взвешенную сумму между двумя сигналами или функциями. Пример операции свертки на матрице размером 5×5 с ядром размером 3×3 показан ниже. Ядро свертки скользит по всей матрице для получения карты активации.

The diagram illustrates a convolution operation. On the left is a 5x5 input matrix with values:

7	2	3	3	8
4	5	3	8	4
3	3	2	8	4
2	8	7	2	7
5	4	4	5	4

In the center is a 3x3 kernel matrix with values:

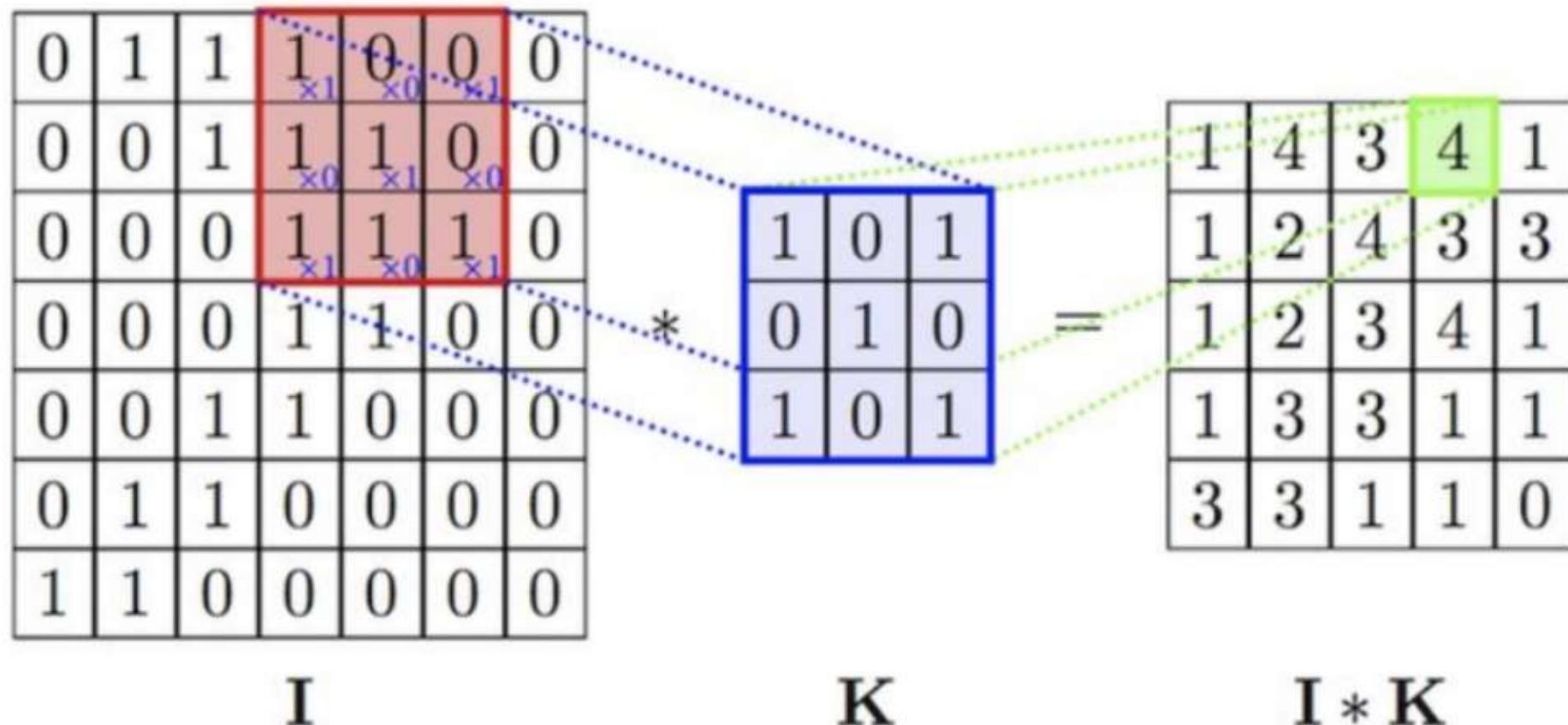
1	0	-1
1	0	-1
1	0	-1

To the right of the multiplication symbol (*) is the result of the convolution step, which is a single value 6. Below the result is the calculation:

$$7 \times 1 + 4 \times 1 + 3 \times 1 + \\ 2 \times 0 + 5 \times 0 + 3 \times 0 + \\ 3 \times -1 + 3 \times -1 + 2 \times -1 \\ = 6$$

5.17 Архитектура и принципы работы сверточной сети

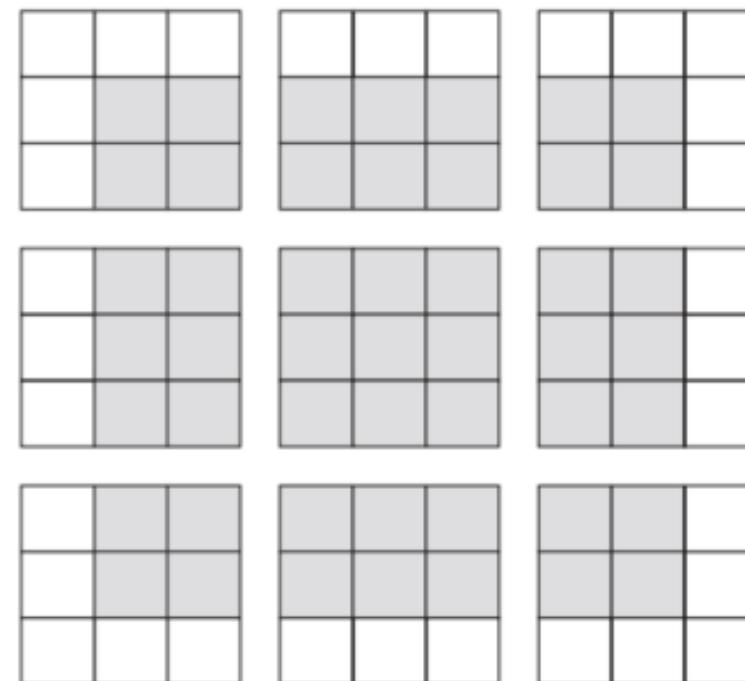
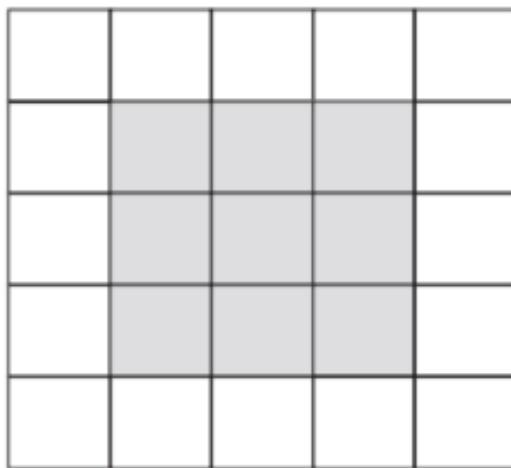
5.17.1 Операция свертки



5.17 Архитектура и принципы работы сверточной сети

5.17.1 Операция свертки

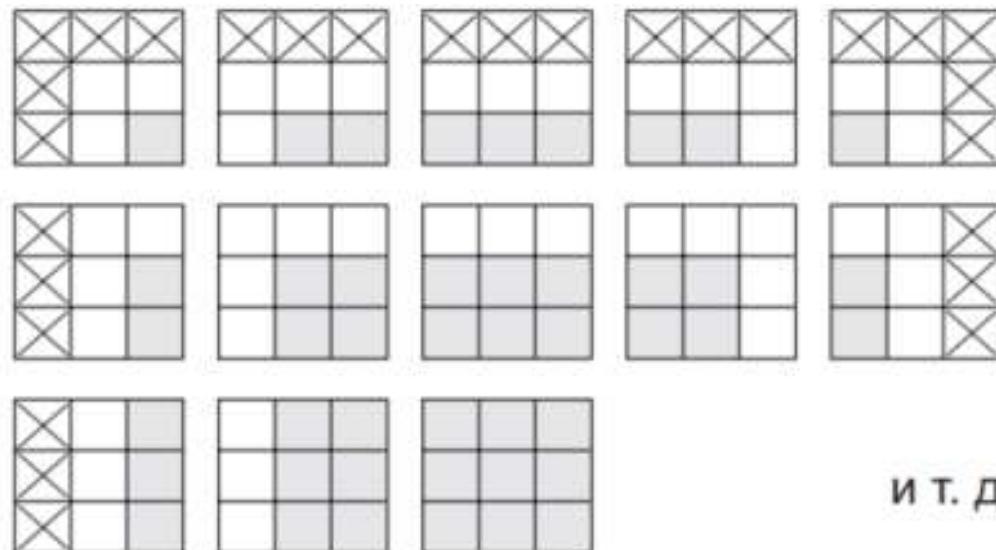
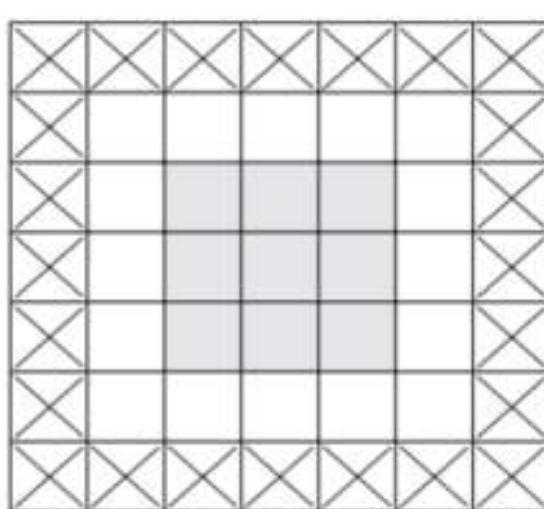
- Рассмотрим карту признаков 5×5 (всего 25 клеток). Существует всего 9 клеток, в которых может находиться центр окна 3×3 , образующих сетку 3×3 . Следовательно, карта выходных признаков будет иметь размер 3×3 . Она получилась немного сжатой: ровно на две клетки вдоль каждого измерения.



5.17 Архитектура и принципы работы сверточной сети

5.17.1 Операция свертки

- Чтобы получить выходную карту признаков с теми же пространственными размерами, что и входная карта, можно использовать *дополнение* (padding). Дополнение заключается в добавлении соответствующего количества строк и столбцов с каждой стороны входной карты признаков, чтобы можно было поместить центр окна свертки в каждую входную клетку. Для окна 3×3 нужно добавить один столбец справа, один столбец слева, одну строку сверху и одну строку снизу. Для окна 5×5 нужно добавить две строки.
- При использовании слоев Conv2D дополнение настраивается с помощью аргумента padding, который принимает два значения: "valid", означающее отсутствие дополнения (будут использоваться только допустимые местоположения окна), и "same", означающее «дополнить так, чтобы выходная карта признаков имела ту же ширину и высоту, что и входная». По умолчанию аргумент padding получает значение "valid".

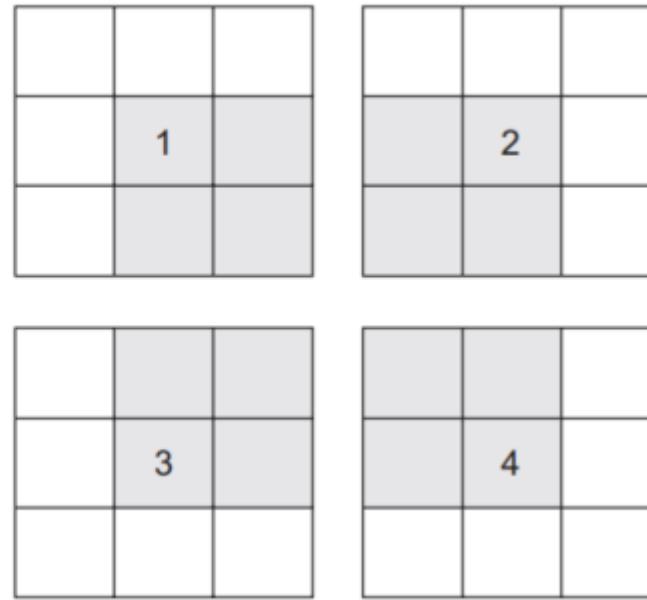
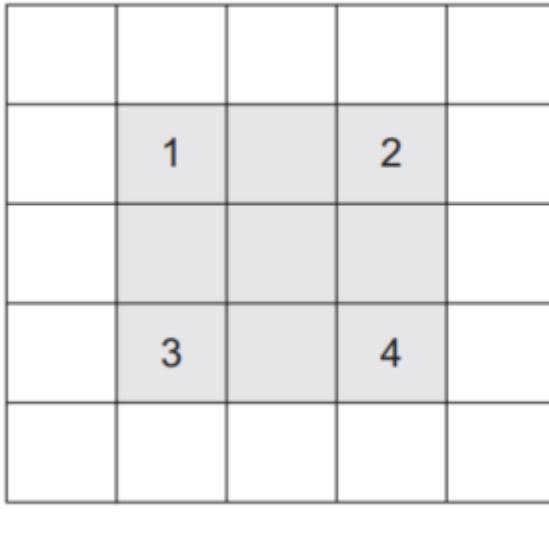


и т. д.

5.17 Архитектура и принципы работы сверточной сети

5.17.1 Операция свертки

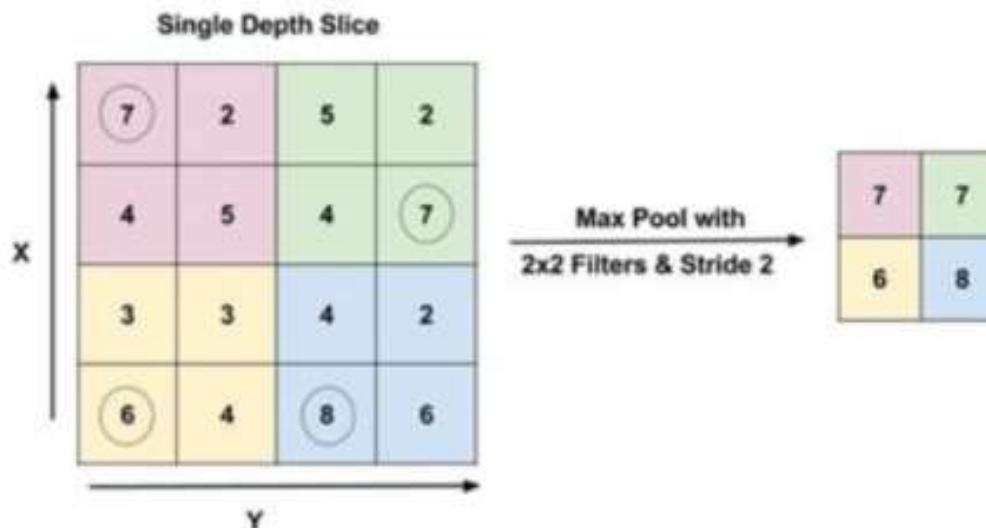
- Другой фактор, который может влиять на размер выходной карты признаков, — *шаг свертки*. До сих пор в объяснениях выше предполагалось, что центральная клетка окна свертки последовательно перемещается в смежные клетки входной карты. Однако в общем случае расстояние между двумя соседними окнами является 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. 5.7 можно видеть, как извлекаются шаблоны 3×3 сверткой с шагом 2 из входной карты 5×5 (без дополнения).
- 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 используются на практике.



5.17 Архитектура и принципы работы сверточной сети

5.17.2 Субдискретизация/subsampling – Max-pooling

- Операция субдискретизации (англ. subsampling, англ. pooling, также переводимая как «операция подвыборки» или операция объединения), выполняет уменьшение размерности сформированных карт признаков (только по ширине и высоте, а не по глубине) путем применения операций максимума или среднего из нескольких соседних нейронов карты. За счёт данной операции, помимо ускорения дальнейших вычислений, сеть становится более инвариантной к масштабу входного изображения. Использование меньшего количества параметров также позволяет избежать переобучения.
- Наиболее распространенной формой пулинга является максимальный пулинг, в котором мы берем фильтр размера и применяем максимальную операцию **max** с определенной частью изображения. На рисунке показана операция MaxPooling2D с размером фильтра 2×2 и шагом 2. Выход представляет собой максимальное значение в области 2×2 , показанной с использованием окруженных цифр. Это существенно уменьшает размер ввода на половину.

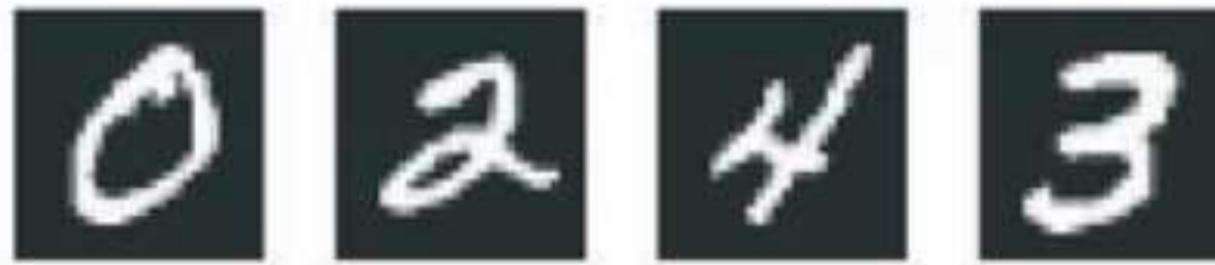


5.17 Архитектура и принципы работы сверточной сети

5.17.3 Пример распознавания цифр из датасета MNIST

Перед нами стоит задача: реализовать классификацию черно-белых изображений рукописных цифр (28×28 пикселов) по 10 категориям (от 0 до 9). Мы будем использовать набор данных MNIST, популярный в сообществе исследователей глубокого обучения, который существует практически столько же, сколько сама область машинного обучения, и широко используется для обучения. Этот набор содержит 60 000 обучающих изображений и 10 000 контрольных изображений, собранных Национальным институтом стандартов и технологий США (National Institute of Standards and Technology — часть NIST в аббревиатуре MNIST).

Образцы изображений MNIST



5.17 Архитектура и принципы работы сверточной сети

5.17.3 Пример распознавания цифр из датасета MNIST

- Создание небольшой сверточной нейронной сети

```
9 from keras import models
10 from keras import layers
11 model = models.Sequential()
12 model.add(layers.Conv2D(32, (3, 3), activation='relu',
13 input_shape=(28, 28, 1)))
14 model.add(layers.MaxPooling2D((2, 2)))
15 model.add(layers.Conv2D(64, (3, 3), activation='relu'))
16 model.add(layers.MaxPooling2D((2, 2)))
17 model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

5.17 Архитектура и принципы работы сверточной сети

5.17.3 Пример распознавания цифр из датасета MNIST

- Как видите, все слои, Conv2D и MaxPooling2D, выводят трехмерный тензор с формой (высота, ширина, каналы). Измерения ширины и высоты сжимаются с ростом глубины сети. Количество каналов управляет первым аргументом, передаваемым в слои Conv2D (32 или 64).

```
>>> 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
=====		
Total params:	55,744	
Trainable params:	55,744	
Non-trainable params:	0	

5.17 Архитектура и принципы работы сверточной сети

5.17.3 Пример распознавания цифр из датасета MNIST

- Добавление полносвязного классификатора поверх сверточной нейронной сети.

```
18 model.add(layers.Flatten())
19 model.add(layers.Dense(64, activation='relu'))
20 model.add(layers.Dense(10, activation='softmax'))
```

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

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

```
>>> 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
```

5.17 Архитектура и принципы работы сверточной сети

5.17.3 Пример распознавания цифр из датасета MNIST

- Выполним чтение и предобработку данных в соответствии с моделью тензора входных данных для сверточной сети.

```
3 "Import from keras dataset Mnist"
4 from keras.datasets import mnist
5 (train_images, train_labels), (test_images, test_labels) = mnist.load_data()
6 "Data preprocessing"
7 train_images = train_images.reshape((60000, 28, 28, 1))
8 train_images = train_images.astype('float32') / 255
9 test_images = test_images.reshape((10000, 28, 28, 1))
10 test_images = test_images.astype('float32') / 255
11
12 "Marking data"
13 from keras.utils import to_categorical
14 train_labels = to_categorical(train_labels)
15 test_labels = to_categorical(test_labels)
```

5.17 Архитектура и принципы работы сверточной сети

5.17.3 Пример распознавания цифр из датасета MNIST

- Откомпилируем построенную сверточную сеть

```
31 "Compiling CNN"
32 model.compile(optimizer='rmsprop',
33                 loss='categorical_crossentropy',
34                 metrics=['accuracy'])
35
```

- Обучим модель на обучающих данных.

```
36 "Fitting (learning) CNN"
37 history = model.fit(train_images, train_labels, epochs=5, batch_size=64)
38
```

- Протестируем точность распознавания рукописных цифр из датасета MNIST построенной сверточной нейросетевой моделью

```
39 "Testing CNN"
40 test_loss, test_acc = model.evaluate(test_images, test_labels)
41 test_acc
42
```

In [11]: test_acc
Out[11]: 0.9889

In [12]: test_loss
Out[12]: 0.03402933020952696

5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью

- В качестве практического примера рассмотрим классификацию изображений собак и кошек из набора данных, содержащего 4000 изображений (2000 кошек, 2000 собак). Мы будем использовать 2000 изображений для обучения, 1000 для проверки и 1000 для контроля.
- Необходимость обучения модели классификации изображений на очень небольшом объеме данных — обычная ситуация, с которой вы наверняка столкнетесь в своей практике, если будете заниматься распознаванием образов с помощью технологий компьютерного зрения на профессиональном уровне.
- Под «небольшим» объемом понимается от нескольких сотен до нескольких десятков тысяч изображений.

5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью

- Будем следовать простой стратегии решения данной задачи:
 - Обучение новой модели с нуля при наличии небольшого объема исходных данных.
Сначала мы обучим маленькую сверточную нейронную сеть на 2000 обучающих образцах без применения регуляризации и расширения обучающего набора данных, чтобы задать базовый уровень достижимого. Она даст нам точность классификации 71 %. С этого момента начнет проявляться эффект переобучения.

5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью

- Затем будет представлен эффективный способ уменьшения степени переобучения в распознавании образов — *расширение данных* (data augmentation). С его помощью мы повысим точность классификации до уровня от 82 %.
- Далее рассмотрим еще два основных приема глубокого обучения на небольших наборах данных: *выделение признаков с использованием предварительно обученной сети* (поможет поднять точность с 90 до 96 %) и *дообучение предварительно обученной сети* (поможет достичь окончательной точности в 97 %).

5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью

- Вместе эти три стратегии
 - обучение малой модели с нуля,
 - выделение признаков с использованием предварительно обученной модели
 - дообучение этой модели
- станут вашим основным набором инструментов для решения задач классификации изображений с обучением на небольших наборах данных.

5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью

- Набор данных «Dogs vs. Cats», который мы будем использовать, не поставляется в составе Keras. Он был создан в ходе состязаний по распознаванию образов в конце 2013-го, когда сверточные нейронные сети еще не заняли лидирующего положения, и доступен на сайте Kaggle. Этот набор можно получить по адресу: www.kaggle.com/c/dogs-vs-cats/data.
- Этот набор содержит 25 000 изображений кошек и собак (по 12 500 для каждого класса) общим объемом 543 Мбайт (в сжатом виде). После загрузки и распаковки архива мы создадим на основе исходного тренировочного размеченного набора новый набор, разделенный на три поднабора: обучающий набор с 1000 образцами каждого класса, проверочный набор с 500 образцами каждого класса и контрольный набор с 500 образцами каждого класса.

5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью

Путь к каталогу с распакованным исходным набором данных

Каталог для сохранения выделенного небольшого набора

```
import os, shutil
```

```
► original_dataset_dir = '/Users/fchollet/Downloads/kaggle_original_data'
```

```
base_dir = '/Users/fchollet/Downloads/cats_and_dogs_small'  
os.mkdir(base_dir)
```

```
train_dir = os.path.join(base_dir, 'train')  
os.mkdir(train_dir)
```

Каталоги для обучающего, проверочного и контрольного поднаборов

```
validation_dir = os.path.join(base_dir, 'validation')  
os.mkdir(validation_dir)
```

```
test_dir = os.path.join(base_dir, 'test')  
os.mkdir(test_dir)
```

```
train_cats_dir = os.path.join(train_dir, 'cats')  
os.mkdir(train_cats_dir)
```

Каталог для обучающих изображений с кошками

```
train_dogs_dir = os.path.join(train_dir, 'dogs')  
os.mkdir(train_dogs_dir)
```

Каталог для обучающих изображений с собаками

5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью

```
validation_cats_dir = os.path.join(validation_dir, 'cats')  
os.mkdir(validation_cats_dir)
```

Каталог для проверочных изображений с кошками

```
validation_dogs_dir = os.path.join(validation_dir, 'dogs')  
os.mkdir(validation_dogs_dir)
```

Каталог для проверочных изображений с собаками

```
test_cats_dir = os.path.join(test_dir, 'cats')  
os.mkdir(test_cats_dir)
```

Каталог для контрольных изображений с кошками

```
test_dogs_dir = os.path.join(test_dir, 'dogs')  
os.mkdir(test_dogs_dir)
```

Каталог для контрольных изображений с собаками

5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью

```
fnames = ['cat.{}.jpg'.format(i) for i in range(1000)]  
for fname in fnames:  
    src = os.path.join(original_dataset_dir, fname)  
    dst = os.path.join(train_cats_dir, fname)  
    shutil.copyfile(src, dst)
```

Копирование первых 1000 изображений с кошками в каталог train_cats_dir

```
fnames = ['cat.{}.jpg'.format(i) for i in range(1000, 1500)]  
for fname in fnames:  
    src = os.path.join(original_dataset_dir, fname)  
    dst = os.path.join(validation_cats_dir, fname)  
    shutil.copyfile(src, dst)
```

Копирование следующих 500 изображений с кошками в каталог validation_cats_dir

```
fnames = ['cat.{}.jpg'.format(i) for i in range(1500, 2000)]  
for fname in fnames:  
    src = os.path.join(original_dataset_dir, fname)  
    dst = os.path.join(test_cats_dir, fname)  
    shutil.copyfile(src, dst)
```

Копирование следующих 500 изображений с кошками в каталог test_cats_dir

5.18 Применение сверточных сетей. Бинарная

классификация цветных RGB изображений

сверточной сетью 5.18.1 Создание малого датасета

```
fnames = ['dog.{}.jpg'.format(i) for i in range(1000)]  
for fname in fnames:  
    src = os.path.join(original_dataset_dir, fname)  
    dst = os.path.join(train_dogs_dir, fname)  
    shutil.copyfile(src, dst)
```

Копирование первых
1000 изображений
с собаками в каталог
train_dogs_dir

```
fnames = ['dog.{}.jpg'.format(i) for i in range(1000, 1500)]  
for fname in fnames:  
    src = os.path.join(original_dataset_dir, fname)  
    dst = os.path.join(validation_dogs_dir, fname)  
    shutil.copyfile(src, dst)
```

Копирование сле-
дующих 500 изо-
бражений с соба-
ками в каталог
validation_dogs_dir

```
fnames = ['dog.{}.jpg'.format(i) for i in range(1500, 2000)]  
for fname in fnames:  
    src = os.path.join(original_dataset_dir, fname)  
    dst = os.path.join(test_dogs_dir, fname)  
    shutil.copyfile(src, dst)
```

Копирование сле-
дующих 500 изо-
бражений с со-
баками в каталог
test_dogs_dir

5.18 Применение сверточных сетей. Бинарная

классификация цветных RGB изображений

сверточной сетью 5.18.1 Создание малого датасета

- Для проверки подсчитаем, сколько изображений оказалось в каждом поднаборе (обучающем/проверочном/контрольном):

```
65 #Проверка - подсчет количества файлов в папке
66 print('total training cat images:', len(os.listdir(train_cats_dir)))
67 print('total training dog images:', len(os.listdir(train_dogs_dir)))
68 print('total validation cat images:', len(os.listdir(validation_cats_dir)))
69 print('total validation dog images:', len(os.listdir(validation_dogs_dir)))
70 print('total test cat images:', len(os.listdir(test_cats_dir)))
71 print('total test dog images:', len(os.listdir(test_dogs_dir)))
72
```

```
In [20]: print('total training cat images:', len(os.listdir(train_cats_dir)))
...: print('total training dog images:', len(os.listdir(train_dogs_dir)))
...: print('total validation cat images:', len(os.listdir(validation_cats_dir)))
...: print('total validation dog images:', len(os.listdir(validation_dogs_dir)))
...: print('total test cat images:', len(os.listdir(test_cats_dir)))
...: print('total test dog images:', len(os.listdir(test_dogs_dir)))
total training cat images: 1000
total training dog images: 1000
total validation cat images: 500
total validation dog images: 500
total test cat images: 500
total test dog images: 500
```

5.18 Применение сверточных сетей. Бинарная

классификация цветных RGB изображений

сверточной сетью 5.18.1 Создание малого датасета

- Также для проверки выведем несколько изображений папок кошек и собак в обучающем наборе:

```
8 import os, shutil
9 import pathlib
10 import random
11 import IPython.display as display
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73 data_train_cats_dir = pathlib.Path(train_cats_dir)
74 data_train_dogs_dir = pathlib.Path(train_dogs_dir)
75
76 cats_image_train = list(data_train_cats_dir.glob('*.*'))
77 cats_image_train = [str(path) for path in cats_image_train]
78
79 #Image train display
80 for n in range(3):
81     image_path = random.choice(cats_image_train)
82     display.display(display.Image(image_path))
83     print(image_path)
84     print()
85
86 dogs_image_train = list(data_train_dogs_dir.glob('*.*'))
87 dogs_image_train = [str(path) for path in dogs_image_train]
88
89 #Image train display
90 for n in range(3):
91     image_path = random.choice(dogs_image_train)
92     display.display(display.Image(image_path))
93     print(image_path)
94     print()
```

5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью

5.18.1 Создание малого датасета

- Примеры изображений кошек в обучающем наборе:



5.18 Применение сверточных сетей. Бинарная

классификация цветных RGB изображений

сверточной сетью 5.18.1 Создание малого датасета

- Примеры изображений собак в обучающем наборе:



5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью 5.18.2 Первая модель сети

- Мы имеем дело с большими изображениями и решаем более сложную задачу, мы сделаем сеть больше по сравнению со сверточной сетью для классификации набора MNIST.
- Она будет иметь на три слоя Conv2D и 2 слоя MaxPooling2D больше. Это увеличит ее емкость и обеспечит дополнительное снижение размеров карт признаков, чтобы они не оказались слишком большими, когда достигнут слоя Flatten.
- С учетом того, что мы начнем с входов, имеющих размер 180×180 (выбор был сделан совершенно произвольно), в конце, точно перед слоем Flatten, получится карта признаков размером 7×7 .

5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью 5.18.2 Первая модель сети

- Так как перед нами стоит задача бинарной классификации, сеть должна заканчиваться единственным признаком - слой Dense с размером 1 и функцией активации sigmoid. Этот признак будет представлять собой вероятность принадлежности рассматриваемого изображения одному из двух классов.

```
inputs = keras.Input(shape=(180, 180, 3))
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
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.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью 5.18.2 Первая модель сети

- Посмотрим, как изменяются размеры карт признаков с каждым последующим слоем:

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[(None, 180, 180, 3)]	0
rescaling (Rescaling)	(None, 180, 180, 3)	0
conv2d_24 (Conv2D)	(None, 178, 178, 32)	896
max_pooling2d_24 (MaxPooling2D)	(None, 89, 89, 32)	0
conv2d_25 (Conv2D)	(None, 87, 87, 64)	18496
max_pooling2d_25 (MaxPooling2D)	(None, 43, 43, 64)	0
conv2d_26 (Conv2D)	(None, 41, 41, 128)	73856
max_pooling2d_26 (MaxPooling2D)	(None, 20, 20, 128)	0
conv2d_27 (Conv2D)	(None, 18, 18, 256)	295168
max_pooling2d_27 (MaxPooling2D)	(None, 9, 9, 256)	0
conv2d_28 (Conv2D)	(None, 7, 7, 256)	590080
flatten_6 (Flatten)	(None, 12544)	0
dense_14 (Dense)	(None, 1)	12545

Total params: 991041 (3.78 MB)
Trainable params: 991041 (3.78 MB)
Non-trainable params: 0 (0.00 Byte)

5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью 5.18.2 Первая модель сети

- На этапе компиляции, как обычно, используем оптимизатор RMSprop. Так как модель заканчивается единственным сигмоидным выходом, используем функцию потерь binary_crossentropy :

```
model.compile(loss='binary_crossentropy',
optimizer="adam", #RMSprop or Adam
metrics=[ 'accuracy'])
```

5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью 5.18.3 Предварительная обработка данных

- Перед передачей в сеть данные должны быть преобразованы в тензоры с вещественными числами. В настоящее время данные хранятся в виде файлов JPEG, поэтому их нужно подготовить для передачи в сеть, выполнив следующие шаги:
 - 1) прочитать файлы с изображениями;
 - 2) декодировать содержимое из формата JPEG в таблицы пикселей RGB;
 - 3) преобразовать их в тензоры с вещественными числами;
 - 4) масштабировать значения пикселей из диапазона [0, 255] в диапазон [0, 1], т.к. сверточным нейронным сетям предпочтительнее передавать небольшие значения);
 - 5) организовать в пакеты (мы будем использовать пакеты по 32 изображения в каждом).

5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью 5.18.3 Ввод изображений и генерация наборов данных

- Во фреймворке Keras имеется вспомогательная функция `image_dataset_from_directory()`, которая позволит быстро настроить конвейер обработки для автоматического преобразования файлов с изображениями в пакеты готовых тензоров .
- Вызов `image_dataset_from_directory(directory)` сначала составит список подкаталогов в каталоге `directory` и предположит, что каждый содержит изображения, принадлежащие одному из классов Затем проиндексирует файлы изображений в каждом подкаталоге и, наконец, создаст и вернет объект `tf.data.Dataset`, подготовленный для чтения файлов, перемешивания, преобразования в тензоры, приведения к общему размеру и упаковки в пакеты

5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью 5.18.3 Ввод изображений и генерация наборов данных

```
from tensorflow.keras.utils import image_dataset_from_directory

train_dataset = image_dataset_from_directory(
    train_dir,
    image_size=(180, 180),
    batch_size=32)
validation_dataset = image_dataset_from_directory(
    validation_dir,
    image_size=(180, 180),
    batch_size=32)
test_dataset = image_dataset_from_directory(
    test_dir,
    image_size=(180, 180),
    batch_size=32)
```

5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью 5.18.4 Обучение модели

- Давайте обучим модель на нашем наборе данных и возьмем аргумент validation_data метода fit() для наблюдения за изменением метрик на этапе проверки с использованием отдельного объекта Dataset:

```
history = model.fit(  
    train_dataset,  
    epochs=30,  
    validation_data=validation_dataset,  
    callbacks=callbacks)
```

5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью 5.18.4 Обучение модели

- В процессе обучения наблюдается сильный эффект переобучения. Это показывает существенная разница в значения как функции потерь, так и точности обучения для обучающей и валидационных выборок.

```
Epoch 9/30
63/63 [=====] - 36s 568ms/step - loss: 0.4058 - accuracy: 0.8080 - val_loss: 0.7349 - val_accuracy: 0.6820
Epoch 10/30
63/63 [=====] - 35s 562ms/step - loss: 0.3481 - accuracy: 0.8400 - val_loss: 0.7287 - val_accuracy: 0.7000
Epoch 11/30
63/63 [=====] - 36s 568ms/step - loss: 0.2686 - accuracy: 0.8835 - val_loss: 0.7310 - val_accuracy: 0.7410
Epoch 12/30
63/63 [=====] - 36s 566ms/step - loss: 0.2314 - accuracy: 0.9055 - val_loss: 0.8448 - val_accuracy: 0.7240
Epoch 13/30
63/63 [=====] - 36s 570ms/step - loss: 0.1459 - accuracy: 0.9455 - val_loss: 0.9275 - val_accuracy: 0.7460
Epoch 14/30
63/63 [=====] - 36s 566ms/step - loss: 0.1143 - accuracy: 0.9550 - val_loss: 1.0620 - val_accuracy: 0.7270
Epoch 15/30
63/63 [=====] - 35s 562ms/step - loss: 0.0856 - accuracy: 0.9670 - val_loss: 1.2065 - val_accuracy: 0.7210
Epoch 16/30
63/63 [=====] - 36s 566ms/step - loss: 0.0587 - accuracy: 0.9750 - val_loss: 1.2683 - val_accuracy: 0.7210
Epoch 17/30
63/63 [=====] - 36s 570ms/step - loss: 0.0596 - accuracy: 0.9780 - val_loss: 1.4429 - val_accuracy: 0.7300
```

5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью 5.18.4 Обучение модели

- Итоговые значения точности составили 1.0000 на обучающей выборке и 0.7560 – на валидационной. Разница составила почти 0.25.

```
Epoch 18/30
63/63 [=====] - 36s 566ms/step - loss: 0.0310 - accuracy: 0.9910 - val_loss: 1.3899 - val_accuracy: 0.7330
Epoch 19/30
63/63 [=====] - 36s 564ms/step - loss: 0.0110 - accuracy: 0.9995 - val_loss: 1.6005 - val_accuracy: 0.7410
Epoch 20/30
63/63 [=====] - 35s 562ms/step - loss: 0.0030 - accuracy: 1.0000 - val_loss: 1.7056 - val_accuracy: 0.7440
Epoch 21/30
63/63 [=====] - 35s 562ms/step - loss: 0.0013 - accuracy: 1.0000 - val_loss: 1.7551 - val_accuracy: 0.7490
Epoch 22/30
63/63 [=====] - 36s 565ms/step - loss: 5.9201e-04 - accuracy: 1.0000 - val_loss: 1.8097 - val_accuracy: 0.7480
Epoch 23/30
63/63 [=====] - 36s 574ms/step - loss: 3.3871e-04 - accuracy: 1.0000 - val_loss: 1.8608 - val_accuracy: 0.7510
Epoch 24/30
63/63 [=====] - 37s 591ms/step - loss: 2.5334e-04 - accuracy: 1.0000 - val_loss: 1.9028 - val_accuracy: 0.7520
Epoch 25/30
63/63 [=====] - 36s 574ms/step - loss: 2.0341e-04 - accuracy: 1.0000 - val_loss: 1.9359 - val_accuracy: 0.7550
Epoch 26/30
63/63 [=====] - 36s 574ms/step - loss: 1.6993e-04 - accuracy: 1.0000 - val_loss: 1.9668 - val_accuracy: 0.7560
Epoch 27/30
63/63 [=====] - 38s 604ms/step - loss: 1.4610e-04 - accuracy: 1.0000 - val_loss: 1.9948 - val_accuracy: 0.7560
Epoch 28/30
63/63 [=====] - 38s 600ms/step - loss: 1.2643e-04 - accuracy: 1.0000 - val_loss: 2.0212 - val_accuracy: 0.7550
Epoch 29/30
63/63 [=====] - 37s 592ms/step - loss: 1.1215e-04 - accuracy: 1.0000 - val_loss: 2.0455 - val_accuracy: 0.7560
Epoch 30/30
63/63 [=====] - 38s 597ms/step - loss: 1.0019e-04 - accuracy: 1.0000 - val_loss: 2.0700 - val_accuracy: 0.7560
```

5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью 5.18.5 Результаты обучения и валидации, создание графиков изменения потерь и точности в процессе обучения

- Создадим графики изменения точности и потерь модели по обучающим и проверочным-валидационным данным в процессе обучения:

```
accuracy = history.history["accuracy"]
val_accuracy = history.history["val_accuracy"]
loss = history.history["loss"]
val_loss = history.history["val_loss"]

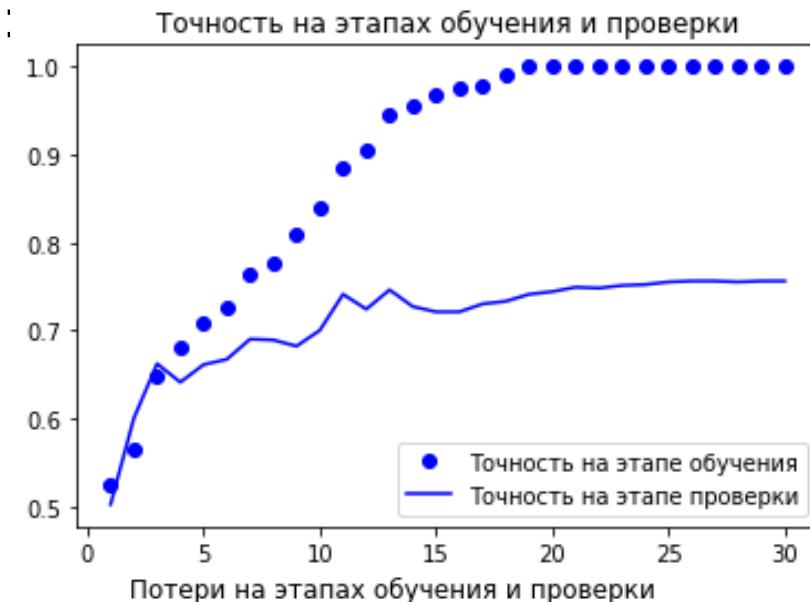
epochs = range(1, len(accuracy) + 1)
plt.plot(epochs, accuracy, "bo", label="Точность на этапе обучения")
plt.plot(epochs, val_accuracy, "b", label="Точность на этапе проверки")
plt.title("Точность на этапах обучения и проверки")
plt.legend()
plt.figure()

plt.plot(epochs, loss, "bo", label="Потери на этапе обучения")
plt.plot(epochs, val_loss, "b", label="Потери на этапе проверки")
plt.title("Потери на этапах обучения и проверки")
plt.legend()
plt.show()
```

5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью 5.18.5 Результаты обучения и валидации, создание графиков изменения потерь и точности в процессе обучения

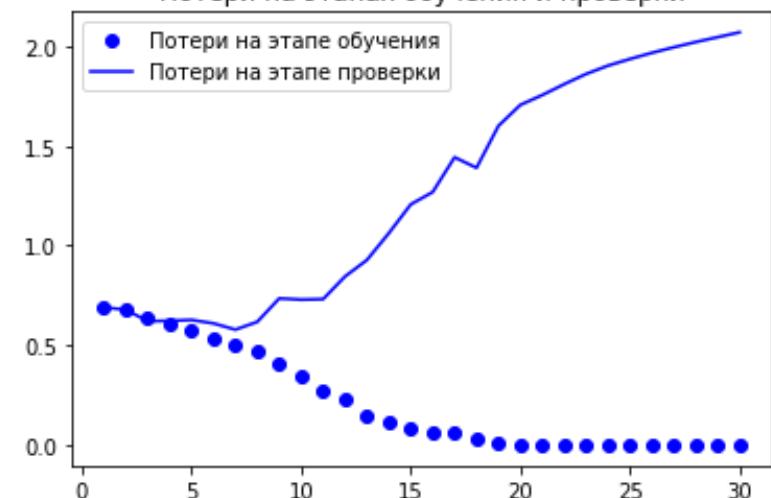
□ Точность на этапах обучения и проверки:

□ График точности почти совпадали до 3-4-й эпох. Затем происходит резкое замедление роста точности на валидационной выборке, при этом точность на обучающей выборке продолжает плавно расти до 1.00.



Потери на этапах обучения и проверки:

□ График функции потерь на обучающей выборке плавно падает почти до 0. На валидационной выборке он падает до 7-й эпохи и затем начинает резко расти. Поскольку у нас относительно немного обучающих образцов (2000), переобучение становится проблемой номер один.



5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью 5.18.4 Обучение модели

- ❑ Если обучать модель с использование механизма обратного вызова, то процесс обучения останавливает на 19-й эпохе.

```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="cats_and_dogs_small_10.2024.h5",
        save_best_only=True,
        monitor="val_loss"),
    keras.callbacks.EarlyStopping(
        monitor='val_loss',
        patience=10),
    keras.callbacks.ReduceLROnPlateau(
        monitor='val_loss',
        factor=0.2,
        patience=5)
]
history = model.fit(
    train_dataset,
    epochs=30,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

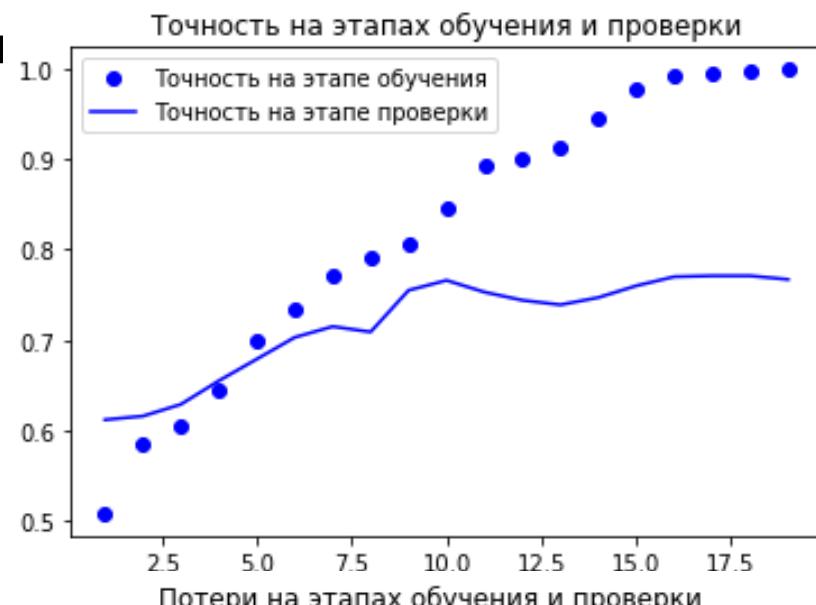
5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью 5.18.4 Обучение модели

- Также наблюдается сильный эффект переобучения, но время обучения существенно сокращается.

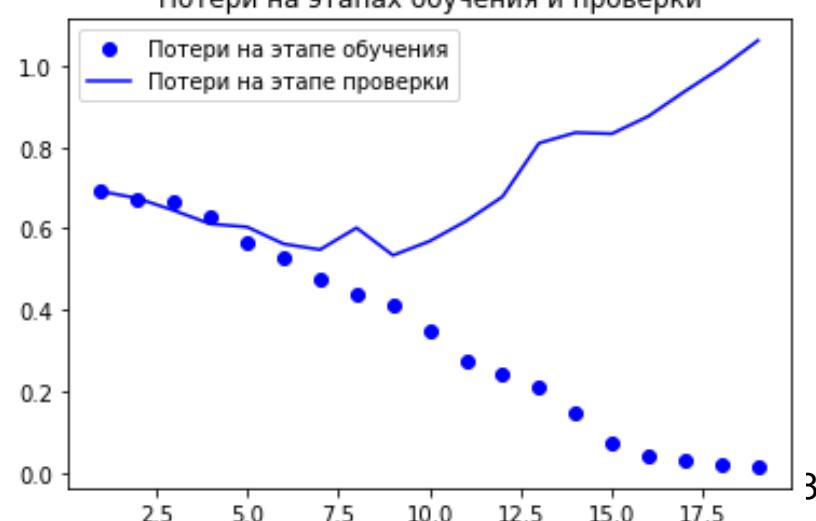
```
Epoch 1/30
63/63 [=====] - 37s 573ms/step - loss: 0.6938 - accuracy: 0.5085 - val_loss: 0.6915 - val_accuracy: 0.6120 - lr: 0.0010
Epoch 2/30
63/63 [=====] - 36s 563ms/step - loss: 0.6735 - accuracy: 0.5860 - val_loss: 0.6743 - val_accuracy: 0.6160 - lr: 0.0010
Epoch 3/30
63/63 [=====] - 36s 566ms/step - loss: 0.6660 - accuracy: 0.6040 - val_loss: 0.6443 - val_accuracy: 0.6290 - lr: 0.0010
Epoch 4/30
63/63 [=====] - 36s 565ms/step - loss: 0.6269 - accuracy: 0.6455 - val_loss: 0.6112 - val_accuracy: 0.6550 - lr: 0.0010
Epoch 5/30
63/63 [=====] - 35s 562ms/step - loss: 0.5674 - accuracy: 0.7000 - val_loss: 0.6040 - val_accuracy: 0.6790 - lr: 0.0010
Epoch 6/30
63/63 [=====] - 36s 563ms/step - loss: 0.5308 - accuracy: 0.7335 - val_loss: 0.5623 - val_accuracy: 0.7030 - lr: 0.0010
Epoch 7/30
63/63 [=====] - 35s 563ms/step - loss: 0.4778 - accuracy: 0.7720 - val_loss: 0.5481 - val_accuracy: 0.7150 - lr: 0.0010
Epoch 8/30
63/63 [=====] - 36s 567ms/step - loss: 0.4388 - accuracy: 0.7915 - val_loss: 0.6018 - val_accuracy: 0.7090 - lr: 0.0010
Epoch 9/30
63/63 [=====] - 36s 568ms/step - loss: 0.4144 - accuracy: 0.8060 - val_loss: 0.5342 - val_accuracy: 0.7550 - lr: 0.0010
Epoch 10/30
63/63 [=====] - 36s 567ms/step - loss: 0.3501 - accuracy: 0.8455 - val_loss: 0.5688 - val_accuracy: 0.7660 - lr: 0.0010
Epoch 11/30
63/63 [=====] - 36s 565ms/step - loss: 0.2755 - accuracy: 0.8930 - val_loss: 0.6183 - val_accuracy: 0.7530 - lr: 0.0010
Epoch 12/30
63/63 [=====] - 36s 562ms/step - loss: 0.2399 - accuracy: 0.8995 - val_loss: 0.6783 - val_accuracy: 0.7440 - lr: 0.0010
Epoch 13/30
63/63 [=====] - 35s 562ms/step - loss: 0.2121 - accuracy: 0.9130 - val_loss: 0.8098 - val_accuracy: 0.7390 - lr: 0.0010
Epoch 14/30
63/63 [=====] - 36s 565ms/step - loss: 0.1453 - accuracy: 0.9455 - val_loss: 0.8358 - val_accuracy: 0.7470 - lr: 0.0010
Epoch 15/30
63/63 [=====] - 36s 565ms/step - loss: 0.0706 - accuracy: 0.9785 - val_loss: 0.8331 - val_accuracy: 0.7600 - lr: 2.0000e-04
Epoch 16/30
63/63 [=====] - 36s 563ms/step - loss: 0.0399 - accuracy: 0.9915 - val_loss: 0.8762 - val_accuracy: 0.7700 - lr: 2.0000e-04
Epoch 17/30
63/63 [=====] - 36s 573ms/step - loss: 0.0284 - accuracy: 0.9955 - val_loss: 0.9376 - val_accuracy: 0.7710 - lr: 2.0000e-04
Epoch 18/30
63/63 [=====] - 35s 563ms/step - loss: 0.0213 - accuracy: 0.9975 - val_loss: 0.9957 - val_accuracy: 0.7710 - lr: 2.0000e-04
Epoch 19/30
```

5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью 5.18.5 Результаты обучения и валидации, создание графиков изменения потерь и точности в процессе обучения

□ Точность на этапах обучения и проверки



□ Потери на этапах обучения и проверки:



5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью 5.18.5 Тестирование модели

- Тестирование обученной модели показывает следующие результаты:

```
32/32 [=====] - 4s 112ms/step - loss: 1.3979 - accuracy: 0.7260
Test accuracy: 0.726
Test loss: 1.398
```

```
test_loss, test_acc = model.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")
print(f"Test Loss: {test_loss:.3f}")
```

5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью 5.18.6 Расширение данных - *data augmentation*

- Причиной переобучения является недостаточное количество образцов для обучения модели, способной обобщать новые данные.
- Имея бесконечный объем данных, можно было бы получить модель, учитывающую все аспекты распределения данных: эффект переобучения никогда не наступил бы.
- Прием расширения данных реализует подход создания дополнительных обучающих данных из имеющихся путем трансформации образцов множеством случайных преобразований, дающих правдоподобные изображения.
- Цель состоит в том, чтобы на этапе обучения модель никогда не увидала одно и то же изображение дважды. Это поможет модели выявить больше особенностей данных и достичь лучшей степени обобщения.
- Сделать подобное в Keras можно путем добавления нескольких слоев аугментации данных в начале модели.

5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью 5.18.6 Расширение данных - data augmentation

- Приведенная далее последовательная модель применяет несколько случайных преобразований к изображениям. Мы добавим ее в нашу модель прямо перед слоем Rescaling.

```
data_augmentation = keras.Sequential(  
    [  
        layers.RandomFlip("horizontal"),  
        layers.RandomRotation(0.1),  
        layers.RandomZoom(0.2),  
    ]  
)
```

- RandomFlip("horizontal") — переворачивает по горизонтали 50 % случайно выбранных изображений;
- RandomRotation(0.1) — поворачивает входные изображения на случайный угол в диапазоне [−10 %, +10 %] (параметр определяет долю полной окружности — в градусах заданный здесь диапазон составит [−36, +36]);
- RandomZoom(0.2) — случайным образом изменяет масштаб изображения, в данном случае в диапазоне [−20 %, +20 %]

5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью 5.18.6 Расширение данных - data augmentation

- Приведенная далее последовательная модель применяет несколько случайных преобразований к изображениям. Мы добавим ее в нашу модель прямо перед слоем Rescaling.

```
inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
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.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью 5.18.6 Расширение данных - data augmentation

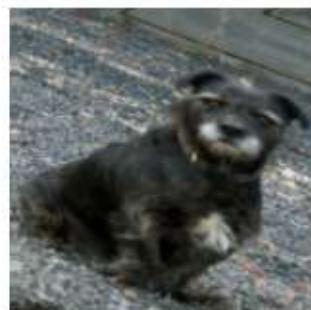
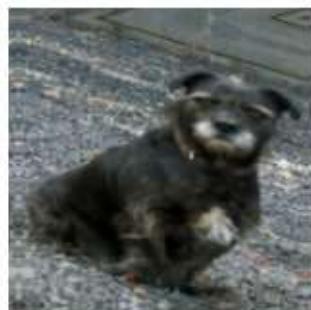
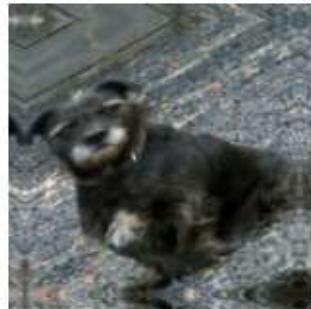
- Давайте посмотрим, как выглядят дополнительные изображения.

```
plt.figure(figsize=(10, 10))
for images, _ in train_dataset.take(1):
    for i in range(9):
        augmented_images = data_augmentation(images)
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(augmented_images[0].numpy().astype("uint8"))
        plt.axis("off")
```

- Следует знать, что слои аугментации изображений случайными преобразованиями неактивны на этапе прогнозирования (когда вызывается метод predict() или evaluate()). Во время оценки модель будет вести себя так, как если бы мы не задействовали аугментацию данных.

5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью 5.18.6 Расширение данных - ***data augmentation***

- Отображение некоторых обучающих изображений, подвергшихся случайнм преобразованиям:



5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью 5.18.6 Расширение данных - ***data augmentation***

- Отображение некоторых обучающих изображений, подвергшихся случайнм преобразованиям:



5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью 5.18.4 Обучение модели

- Обучим модель с использование механизма обратного вызова на 100 эпохах.

```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="cats_and_dogs_small_10.2024.h5",
        save_best_only=True,
        monitor="val_loss"),
    keras.callbacks.EarlyStopping(
        monitor='val_loss',
        patience=20),
    keras.callbacks.ReduceLROnPlateau(
        monitor='val_loss',
        factor=0.2,
        patience=7)
]
history = model.fit(
    train_dataset,
    epochs=100,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью 5.18.4 Обучение модели с использованием генератора пакетов и сохранение модели

- Также наблюдается сильный эффект переобучения.

```
Epoch 15/100
63/63 [=====] - 41s 652ms/step - loss: 0.1106 - accuracy: 0.9620 - val_loss: 1.0694 - val_accuracy: 0.7450 - lr: 0.0010
Epoch 16/100
63/63 [=====] - 40s 642ms/step - loss: 0.0636 - accuracy: 0.9815 - val_loss: 1.0670 - val_accuracy: 0.7530 - lr: 0.0010
Epoch 17/100
63/63 [=====] - 40s 642ms/step - loss: 0.0300 - accuracy: 0.9925 - val_loss: 1.2179 - val_accuracy: 0.7670 - lr: 0.0010
Epoch 18/100
63/63 [=====] - 41s 644ms/step - loss: 0.0115 - accuracy: 0.9980 - val_loss: 1.1464 - val_accuracy: 0.7670 - lr: 2.0000e-04
Epoch 19/100
63/63 [=====] - 40s 633ms/step - loss: 0.0045 - accuracy: 1.0000 - val_loss: 1.1934 - val_accuracy: 0.7670 - lr: 2.0000e-04
Epoch 20/100
63/63 [=====] - 39s 616ms/step - loss: 0.0031 - accuracy: 1.0000 - val_loss: 1.2261 - val_accuracy: 0.7650 - lr: 2.0000e-04
Epoch 21/100
63/63 [=====] - 39s 623ms/step - loss: 0.0024 - accuracy: 1.0000 - val_loss: 1.2538 - val_accuracy: 0.7650 - lr: 2.0000e-04
Epoch 22/100
63/63 [=====] - 39s 622ms/step - loss: 0.0020 - accuracy: 1.0000 - val_loss: 1.2826 - val_accuracy: 0.7670 - lr: 2.0000e-04
Epoch 23/100
63/63 [=====] - 40s 636ms/step - loss: 0.0017 - accuracy: 1.0000 - val_loss: 1.3013 - val_accuracy: 0.7670 - lr: 2.0000e-04
Epoch 24/100
63/63 [=====] - 40s 632ms/step - loss: 0.0014 - accuracy: 1.0000 - val_loss: 1.3298 - val_accuracy: 0.7680 - lr: 2.0000e-04
Epoch 25/100
63/63 [=====] - 40s 642ms/step - loss: 0.0012 - accuracy: 1.0000 - val_loss: 1.3342 - val_accuracy: 0.7670 - lr: 4.0000e-05
Epoch 26/100
63/63 [=====] - 40s 629ms/step - loss: 0.0012 - accuracy: 1.0000 - val_loss: 1.3391 - val_accuracy: 0.7670 - lr: 4.0000e-05
Epoch 27/100
63/63 [=====] - 42s 673ms/step - loss: 0.0011 - accuracy: 1.0000 - val_loss: 1.3441 - val_accuracy: 0.7670 - lr: 4.0000e-05
Epoch 28/100
63/63 [=====] - 41s 647ms/step - loss: 0.0011 - accuracy: 1.0000 - val_loss: 1.3490 - val_accuracy: 0.7670 - lr: 4.0000e-05
Epoch 29/100
63/63 [=====] - 39s 611ms/step - loss: 0.0011 - accuracy: 1.0000 - val_loss: 1.3553 - val_accuracy: 0.7670 - lr: 4.0000e-05
Epoch 30/100
```

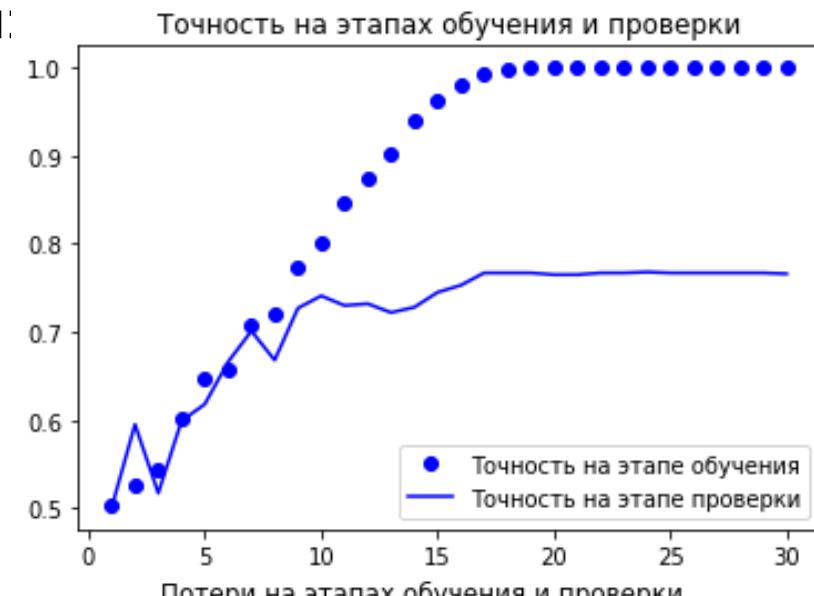
5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью 5.18.5 Результаты обучения и валидации, создание графиков изменения потерь и точности в процессе обучения

- Тестирование обученной модели показывает следующие результаты. Точность на этапе тестирования немного выросла с 0.7260 до 0.7390. Кроме того, разрыв между точностью на валидационной и тестовой выборках сократился.

```
32/32 [=====] - 4s 114ms/step - loss: 1.5078 - accuracy: 0.7390
Test accuracy: 0.739
Test loss: 1.508
```

5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью 5.18.5 Результаты обучения и валидации, создание графиков изменения потерь и точности в процессе обучения

- Точность на этапах обучения и проверки:



- Потери на этапах обучения и проверки:



5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью

5.18.7 Вторая модель данных с регуляризацией

```
inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
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.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.Dropout(0.5)(x) ← Регуляризация с помощью dropout
x = layers.Flatten()(x) ← Регуляризация с помощью  $\|I_1 + I_2\|$ 
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

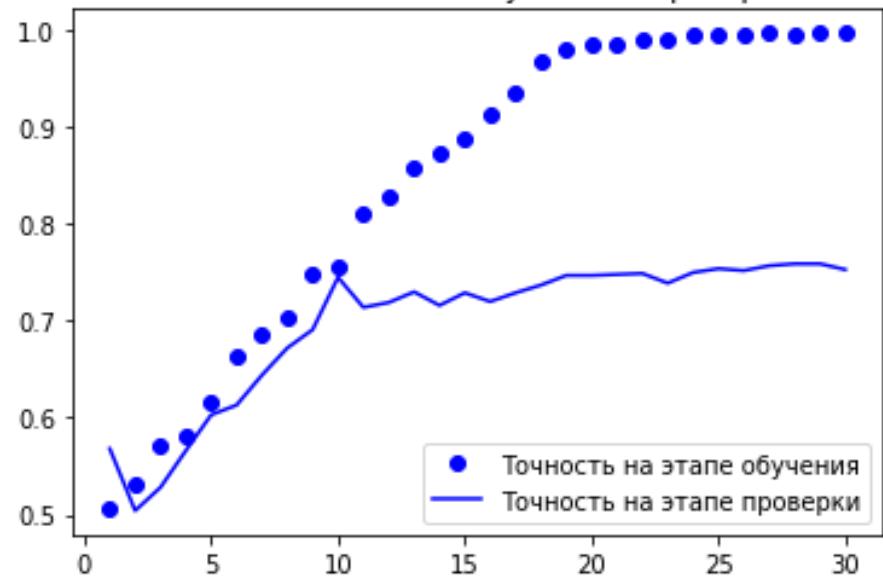
Регуляризация с помощью
dropout

5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью 5.18.8 Результаты обучения и валидации, графики потерь и точности процесса обучения

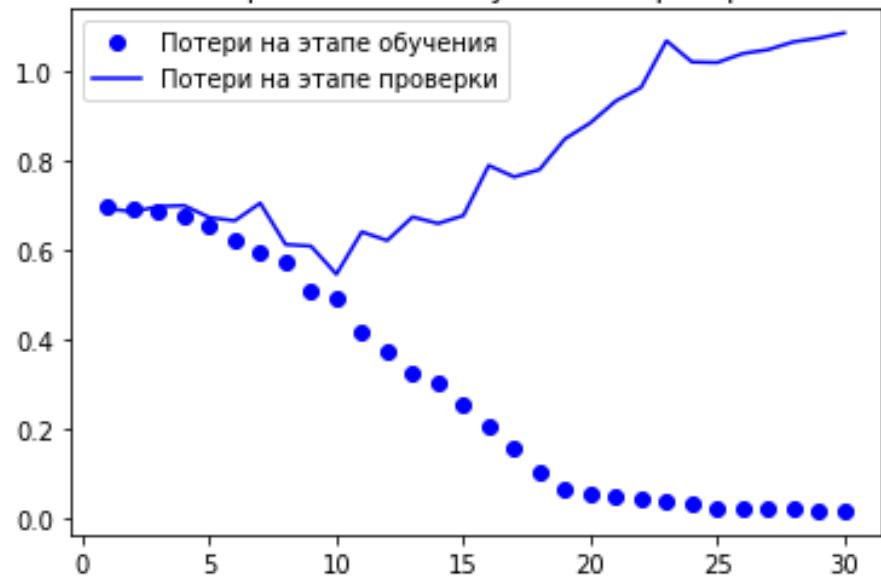
- Результаты для модели с регуляризацией dropout, 100 эпох ранний останов на 32-й эпохе

```
Epoch 28/100
63/63 [=====] - 37s 592ms/step - loss: 0.0200 - accuracy: 0.9960 - val_loss: 1.0641 - val_accuracy: 0.7590 - lr: 4.0000e-05
Epoch 29/100
63/63 [=====] - 37s 592ms/step - loss: 0.0167 - accuracy: 0.9975 - val_loss: 1.0725 - val_accuracy: 0.7590 - lr: 4.0000e-05
Epoch 30/100
63/63 [=====] - 38s 599ms/step - loss: 0.0184 - accuracy: 0.9970 - val_loss: 1.0841 - val_accuracy: 0.7530 - lr: 4.0000e-05
32/32 [=====] - 4s 115ms/step - loss: 1.1914 - accuracy: 0.7500
Test accuracy: 0.750
Test loss: 1.191
```

Точность на этапах обучения и проверки



Потери на этапах обучения и проверки

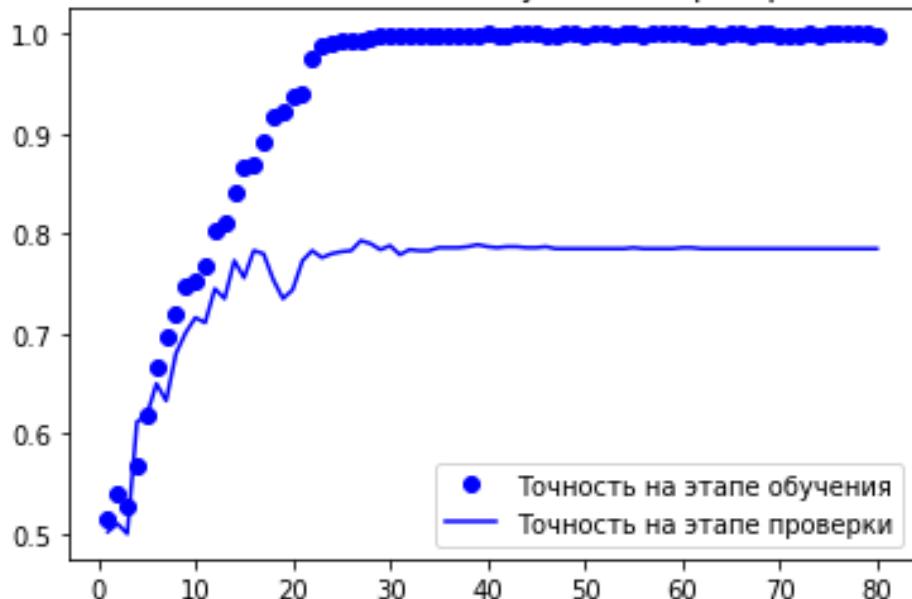


5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью 5.18.8 Результаты обучения и валидации, графики потерь и точности процесса обучения

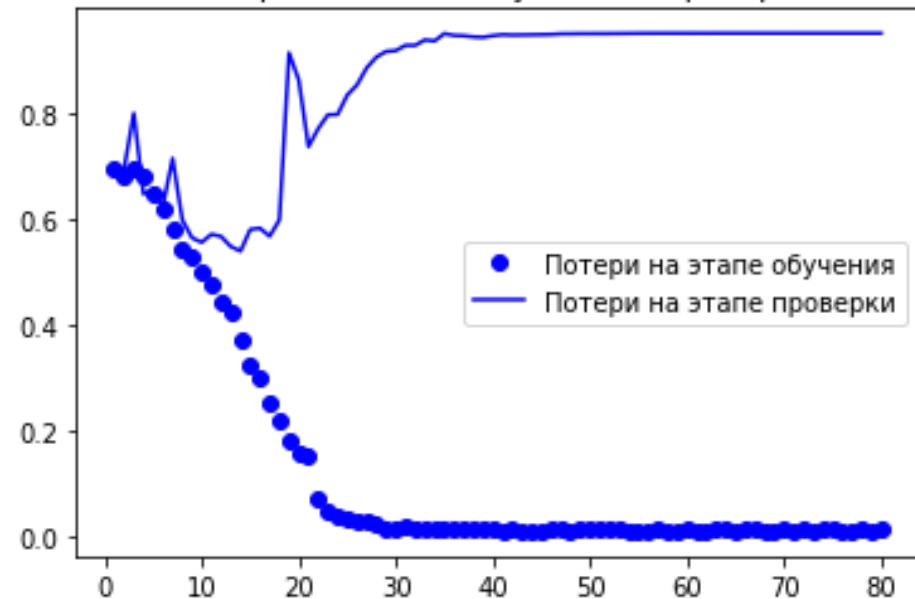
- Результаты для модели с регуляризацией dropout, 80 эпох

```
Epoch 79/80
63/63 [=====] - 36s 575ms/step - loss: 0.0095 - accuracy: 1.0000 - val_loss: 0.9494 - val_accuracy: 0.7850 - lr: 5.1200e-10
Epoch 80/80
63/63 [=====] - 36s 572ms/step - loss: 0.0103 - accuracy: 0.9985 - val_loss: 0.9494 - val_accuracy: 0.7850 - lr: 5.1200e-10
32/32 [=====] - 4s 113ms/step - loss: 1.1785 - accuracy: 0.7500
Test accuracy: 0.750
Test loss: 1.178
```

Точность на этапах обучения и проверки



Потери на этапах обучения и проверки



5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью 5.18.7 Вторая модель данных с регуляризацией

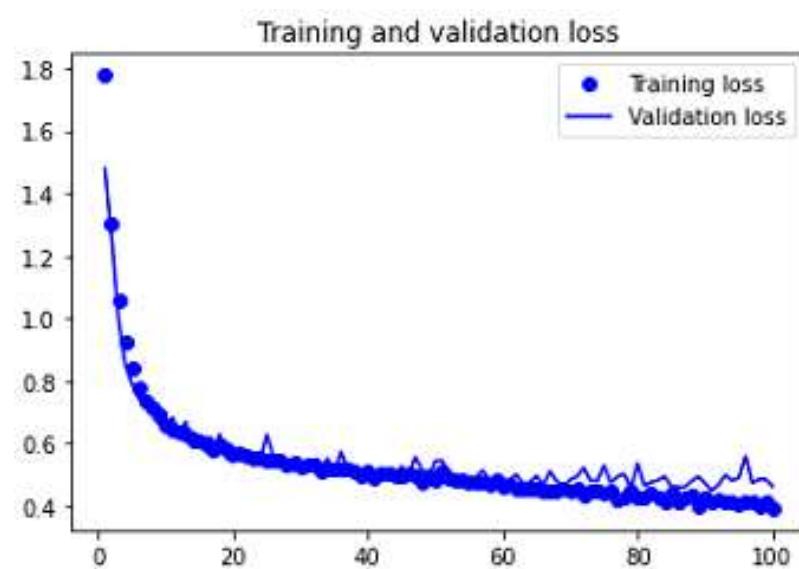
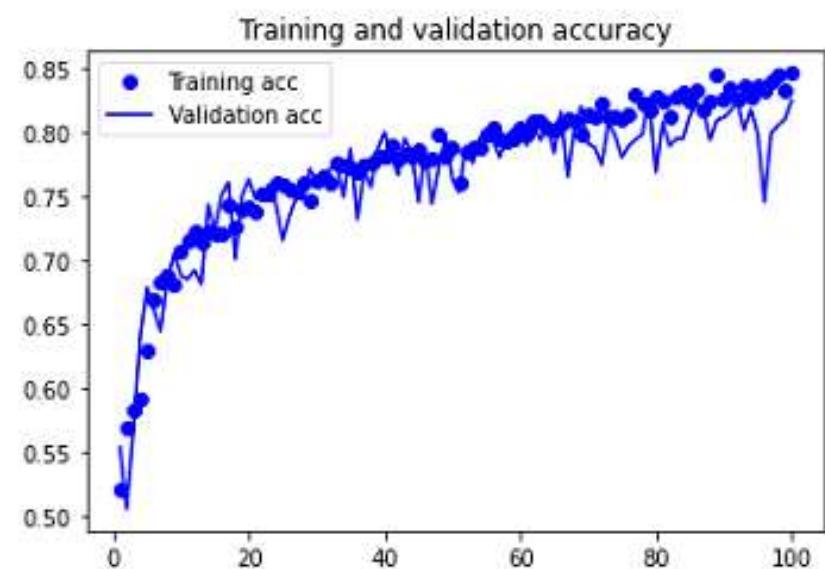
```
inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
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.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.Dropout(0.5)(x) ←
x = layers.Flatten()(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

Регуляризация с помощью
dropout

5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью 5.18.8 Результаты обучения и валидации, графики потерь и точности процесса обучения

- Результаты для модели с регуляризацией dropout и $L_1 + L_2$, 100 эпох

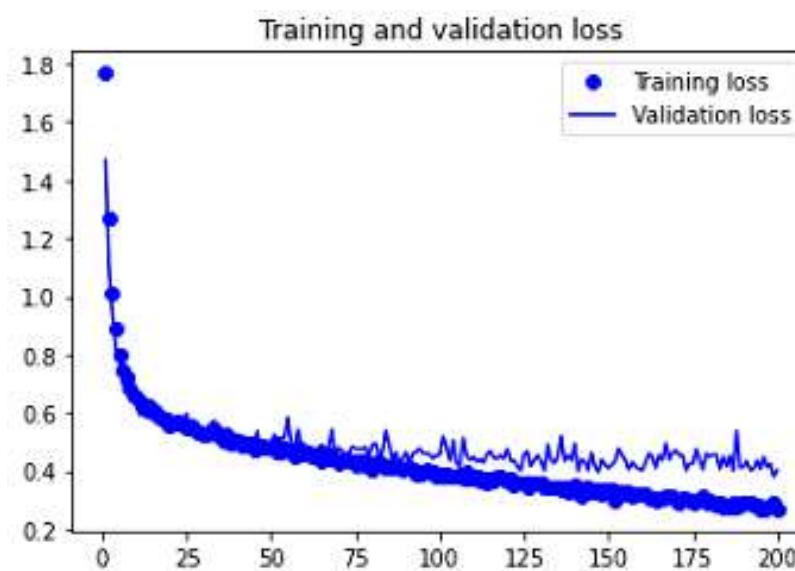
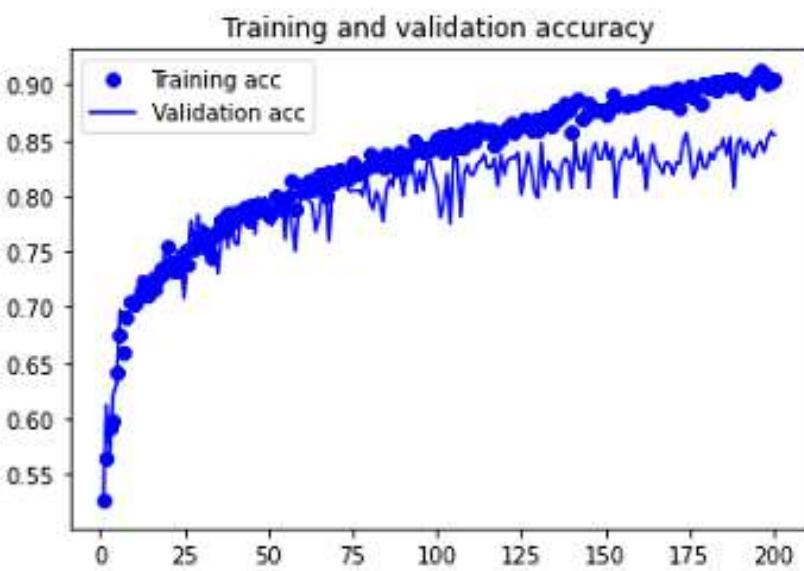
100/100 [=====] - 22s 218ms/step - loss: 0.3924 - acc: 0.8456 - val_loss: 0.4615 - val_acc: 0.8242



5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью 5.18.8 Результаты обучения и валидации, графики потерь и точности процесса обучения

- Результаты для модели с регуляризацией dropout и $L_1 + L_2$, 200 эпох

```
[100/100 [=====] - 24s 242ms/step - loss: 0.2688 - acc: 0.9045 - val_loss: 0.4023 - val_acc: 0.8547
```

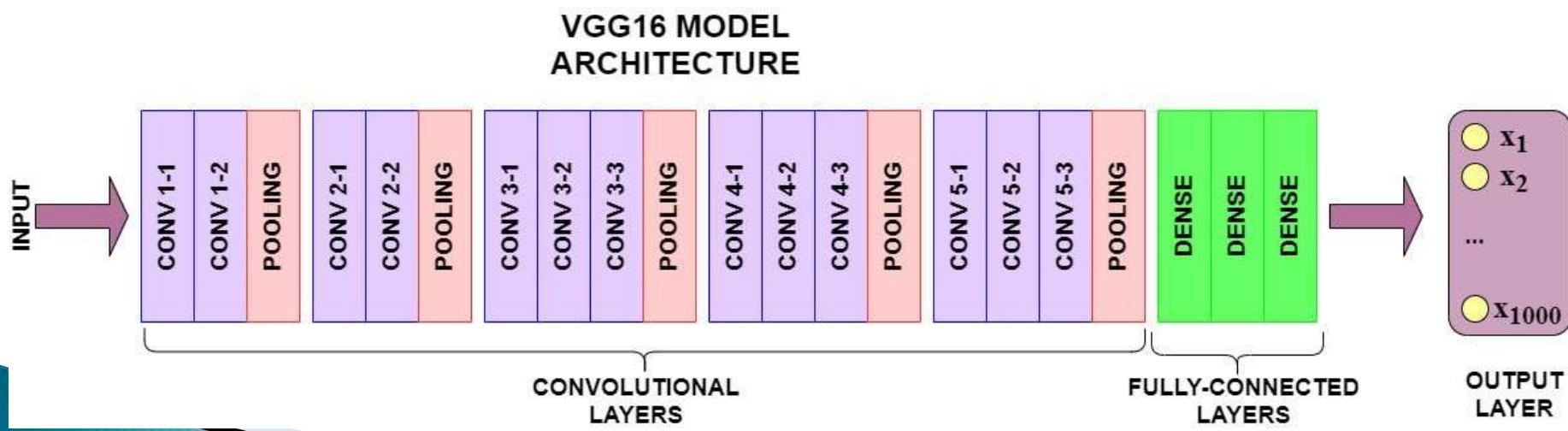


5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью 5.18.8 Использование предварительно обученной модели

- Типичным и эффективным подходом к глубокому обучению на небольших наборах изображений является использование предварительно обученной модели. Предварительно обученная модель — это сохраненная модель, прежде обученная на большом наборе данных, обычно в рамках масштабной задачи классификации изображений. Если исходный набор данных достаточно велик и достаточно обобщен, то пространственная иерархия признаков, изученных моделью, может с успехом выступать в роли обобщенной модели видимого мира и использоваться во многих задачах компьютерного зрения, даже если новые задачи будут связаны с совершенно иными классами, отличными от встречавшихся в оригинальной задаче.
- Например, можно обучить сеть на изображениях из ImageNet (где подавляющее большинство классов — животные и бытовые предметы) и затем использовать ее для идентификации чего-то иного.
- Такая переносимость изученных признаков между разными задачами — главное преимущество глубокого обучения перед многими более старыми приемами поверхностного обучения, которое делает глубокое обучение очень эффективным инструментом для решения задач с малым объемом данных.

5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью 5.18.8 Использование предварительно обученной модели

- Для нашего случая мы возьмем за основу сверточную нейронную сеть, обученную на наборе ImageNet (1,4 миллиона изображений, классифицированных по 1000 разных классов). Коллекция ImageNet содержит множество изображений разных животных, включая разновидности кошек и собак, а значит, можно рассчитывать, что модель, обученная на этой коллекции, прекрасно справится с задачей классификации изображений кошек и собак.
- Мы воспользуемся архитектурой VGG16, разработанной Кареном Симоняном и Эндрю Циссерманом в 2014 году.

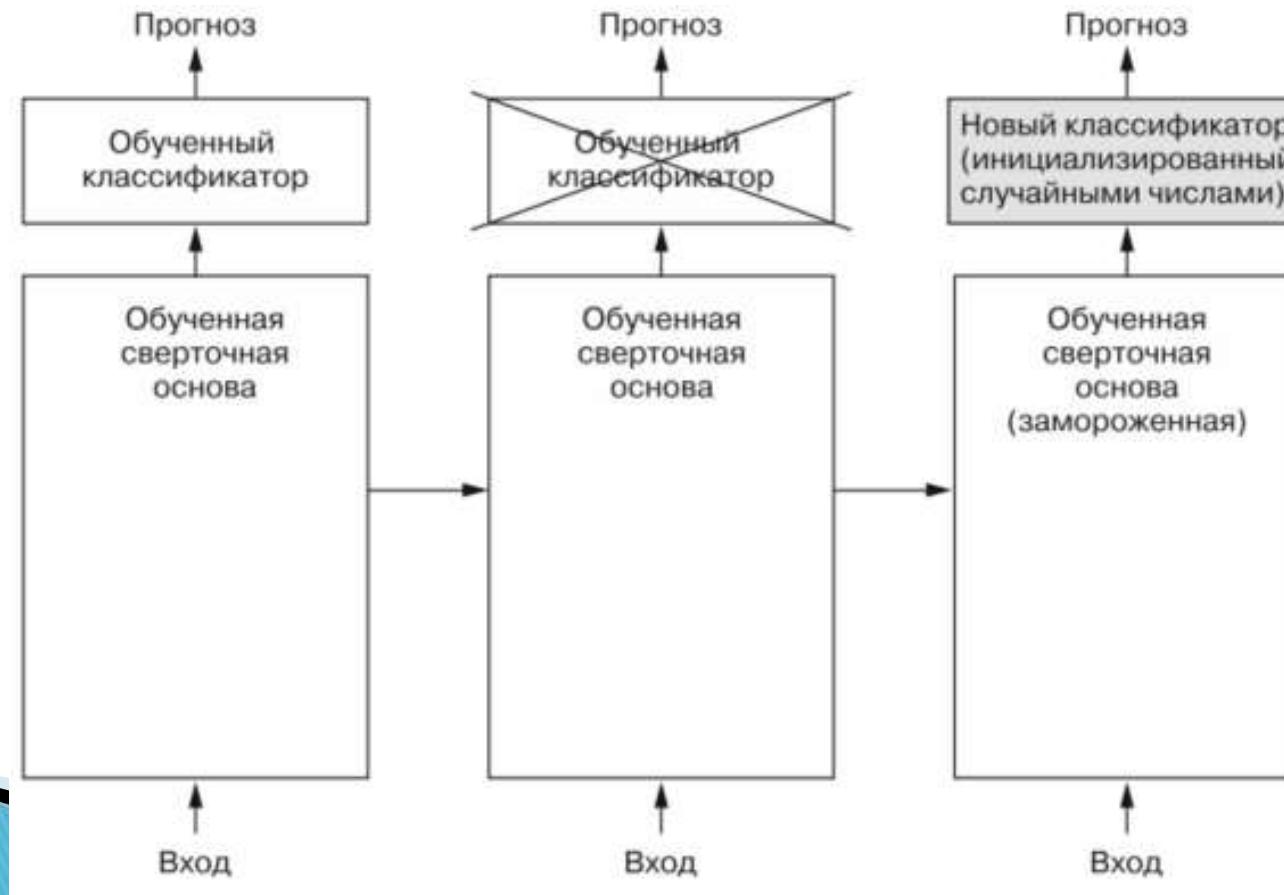


5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью 5.18.8 Использование предварительно обученной модели - выделение признаков

- Выделение признаков заключается в использовании представлений, изученных предварительно обученной моделью, для выделения признаков из новых образцов. Эти признаки затем пропускаются через новый классификатор, обучаемый с нуля.
- Как было показано выше, сверточные нейронные сети, используемые для классификации изображений, состоят из двух частей: они начинаются с последовательности слоев выбора значений и свертки и заканчиваются полносвязным классификатором. Первая часть называется сверточной основой (convolutional base) модели.

5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью 5.18.8 Использование предварительно обученной модели - выделение признаков

- В случае со сверточными нейронными сетями процесс выделения признаков заключается в том, чтобы взять сверточную основу предварительно обученной сети, пропустить через нее новые данные и на основе вывода обучить новый классификатор.



5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью 5.18.8 Использование предварительно обученной модели - выделение признаков

- Почему повторно используется только сверточная основа? Нельзя ли снова взять полносвязный классификатор? В общем случае этого следует избегать Причина в том, что представления, полученные сверточной основой, обычно более универсальны, а значит, более пригодны для повторного использования: карты признаков сверточной нейронной сети — это карты присутствия на изображениях обобщенных понятий, которые могут пригодиться независимо от конкретной задачи распознавания образов. Но представления, изученные классификатором, обязательно будут характерны для набора классов, на котором обучалась модель: они будут содержать только информацию о вероятности присутствия того или иного класса на изображении.
- Отмечу также, что уровень обобщенности (и, соответственно, пригодности к повторному использованию) представлений, выделенных конкретными сверточными слоями, зависит от глубины слоя в модели. Слои, следующие первыми, выделяют локальные, наиболее обобщенные карты признаков (таких как визуальные границы, цвет и текстура), тогда как слои, располагающиеся дальше (или выше), выделяют более абстрактные понятия (такие как «глаз кошки» или «глаз собаки») Поэтому, если новый набор данных существенно отличается от набора, на котором обучалась оригинальная модель, возможно, большего успеха получится добиться, если использовать только несколько первых слоев модели, а не всю сверточную основу

5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью 5.18.8 Использование предварительно обученной модели - выделение признаков

- Создадим экземпляр сверточной основы VGG16.

```
conv_base = keras.applications.vgg16.VGG16(  
    weights="imagenet",  
    include_top=False,  
    input_shape=(180, 180, 3))
```

- Здесь конструктору передаются три аргумента:
 - аргумент `weights` определяет источник весов для инициализации модели;
 - аргумент `include_top` определяет необходимость подключения к сети полносвязного классификатора. По умолчанию полносвязный классификатор определяет принадлежность изображения к 1000 классам. Так как мы намереваемся использовать свой полносвязный классификатор (только с двумя классами, `cat` и `dog`), мы не будем подключать его;
 - аргумент `input_shape` определяет форму тензоров с изображениями, которые будут подаваться на вход сети. Это необязательный аргумент: если опустить его, сеть сможет обрабатывать изображения любого размера. В нашем примере мы передаем его, чтобы иметь возможность видеть (в следующей сводке), как уменьшается размер карт признаков с каждым новым слоем свертки и объединения.

5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью 5.18.8 Использование предварительно обученной модели - выделение признаков

- Далее приводится информация о сверточной основе VGG16 Она напоминает простые сверточные нейронные сети, уже знакомые вам.
- Заключительная карта признаков имеет форму (5, 5, 512). Поверх нее мы положим полно связанный классификатор.

`conv_base.summary()`

Model: "vgg16"		
Layer (type)	Output Shape	Param #
input_11 (InputLayer)	[None, 180, 180, 3]	0
block1_conv1 (Conv2D)	(None, 180, 180, 64)	1792
block1_conv2 (Conv2D)	(None, 180, 180, 64)	36928
block1_pool (MaxPooling2D)	(None, 90, 90, 64)	0
block2_conv1 (Conv2D)	(None, 90, 90, 128)	73856
block2_conv2 (Conv2D)	(None, 90, 90, 128)	147584
block2_pool (MaxPooling2D)	(None, 45, 45, 128)	0
block3_conv1 (Conv2D)	(None, 45, 45, 256)	295168
block3_conv2 (Conv2D)	(None, 45, 45, 256)	590080
block3_conv3 (Conv2D)	(None, 45, 45, 256)	590080

block3_pool (MaxPooling2D)	(None, 22, 22, 256)	0
block4_conv1 (Conv2D)	(None, 22, 22, 512)	1180160
block4_conv2 (Conv2D)	(None, 22, 22, 512)	2359808
block4_conv3 (Conv2D)	(None, 22, 22, 512)	2359808
block4_pool (MaxPooling2D)	(None, 11, 11, 512)	0
block5_conv1 (Conv2D)	(None, 11, 11, 512)	2359808
block5_conv2 (Conv2D)	(None, 11, 11, 512)	2359808
block5_conv3 (Conv2D)	(None, 11, 11, 512)	2359808
block5_pool (MaxPooling2D)	(None, 5, 5, 512)	0

5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью 5.18.8 Использование предварительно обученной модели - выделение признаков

- Далее можно пойти двумя путями:
 - пропустить наш набор данных через сверточную основу, записать получившийся массив NumPy на диск и затем использовать его как входные данные для отдельного полносвязного классификатора. Это быстрое и незатратное решение, потому что требует запускать сверточную основу только один раз для каждого входного изображения, а сверточная основа — самая дорогостоящая часть конвейера. Однако по той же причине этот прием не позволит использовать прием обогащения данных;
 - дополнить имеющуюся модель (conv_base) слоями Dense и пропустить все входные данные. Этот путь позволяет использовать обогащение данных, потому что каждое изображение проходит через сверточную основу каждый раз, когда попадает в модель. Однако по той же причине этот путь намного более затратный, чем первый.
- Мы охватим оба приема. Сначала рассмотрим код, реализующий первый прием: запись вывода conv_base в ответ на передачу наших данных и его использование в роли входных данных новой модели.

5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью 5.18.8 Использование предварительно обученной модели - выделение признаков

- Сначала выделим признаки в массив NumPy, вызвав метод predict() модели conv_base для обучающих, проверочных и контрольных данных
Для этого выполним обход наших наборов данных и выделим признаки VGG16.
- Важно отметить, что predict() принимает только изображения, без меток, а наш объект набора данных выдает пакеты, содержащие как изображения, так и их метки
Более того, модель VGG16 принимает данные, предварительно обработанные с помощью функции keras.applications.vgg16.preprocess_input, которая приводит значения пикселей в соответствующий диапазон.

```
def get_features_and_labels(dataset):  
    all_features = []  
    all_labels = []  
    for images, labels in dataset:  
        preprocessed_images = keras.applications.vgg16.preprocess_input(images)  
        features = conv_base.predict(preprocessed_images)  
        all_features.append(features)  
        all_labels.append(labels)  
    return np.concatenate(all_features), np.concatenate(all_labels)  
  
train_features, train_labels = get_features_and_labels(train_dataset)  
val_features, val_labels = get_features_and_labels(validation_dataset)  
test_features, test_labels = get_features_and_labels(test_dataset)
```

5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью 5.18.8 Использование предварительно обученной модели - выделение признаков

- Теперь можно определить полносвязный классификатор (обратите внимание, что для регуляризации здесь используется прием прореживания) и обучить его на только что записанных данных и метках.

```
inputs = keras.Input(shape=(5, 5, 512))
#x = data_augmentation(inputs)
#x = layers.Rescaling(1./255)(inputs)
x = layers.Flatten()(inputs)
x = layers.Dropout(0.5)(x)
x = layers.Dense(256)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)

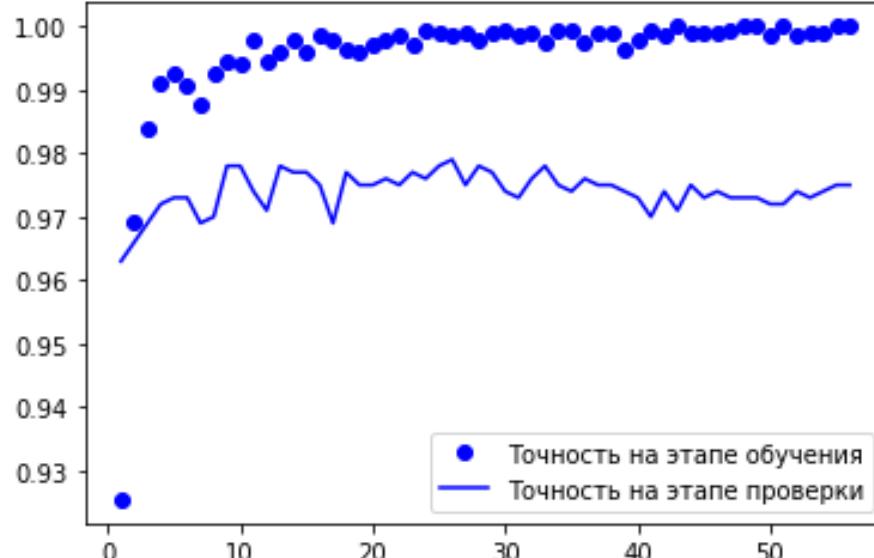
model.compile(loss='binary_crossentropy',
optimizer="adam",
metrics=['accuracy'])
```

5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью 5.18.8 Результаты обучения и валидации, графики потерь и точности процесса обучения

- Результаты для модели с регуляризацией dropout, 80 эпох.
- Мы достигли точности, близкой к 97 %, — более высокой, чем в предыдущем разделе, где обучали небольшую модель с нуля.
- Кроме того, графики показывают, что почти с самого начала стал проявляться эффект переобучения, несмотря на выбор довольно большого коэффициента прореживания.

```
Epoch 55/80
63/63 [=====] - 2s 36ms/step - loss: 1.5541e-04 - accuracy: 1.0000 - val_loss: 0.9968 - val_accuracy: 0.9750 - lr: 4.0000e-05
Epoch 56/80
63/63 [=====] - 2s 36ms/step - loss: 8.3511e-06 - accuracy: 1.0000 - val_loss: 0.9969 - val_accuracy: 0.9750 - lr: 4.0000e-05
```

Точность на этапах обучения и проверки



Потери на этапах обучения и проверки



5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью 5.18.8 Выделение признаков с аугментацией данных

- Теперь рассмотрим второй прием выделения признаков, более медленный и затратный, но позволяющий использовать обогащение данных в процессе обучения, — объединение модели conv_base с новым полно связанным классификатором и ее полноценное обучение
- Для этого мы сначала заморозим сверточную основу. Замораживание одного или нескольких слоев предотвращает изменение весовых коэффициентов в них в процессе обучения. Если этого не сделать, представления, прежде изученные сверточной основой, изменятся в процессе обучения на новых данных. Так как слои Dense сверху инициализируются случайными значениями, в сети могут произойти существенные изменения весов, фактически разрушив представления, полученные ранее.

5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью 5.18.8 Выделение признаков с аугментацией данных данных

- В Keras, чтобы заморозить сеть, нужно передать атрибут trainable со значением False. При передаче в атрибуте trainable значения False список обучаемых весов слоя или модели очищается.

```
conv_base = keras.applications.vgg16.VGG16(  
    weights="imagenet",  
    include_top=False)  
conv_base.trainable = False
```

- Теперь создадим новую модель, объединяющую следующее:
 - 1 Этап обогащения данных
 - 2 Замороженную сверточную основу
 - 3 Полносвязный классификатор
- Аугментация данных выполняется также как это делалось ранее.

5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью 5.18.8 Выделение признаков с аугментацией данных

- Добавим к сверточной части VGG16 полно связанный классификатор после слоя аугментации и откомпилируем модель.

```
inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
x = keras.applications.vgg16.preprocess_input(x)
x = conv_base(x)
x = layers.Flatten()(x)
x = layers.Dense(256)(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs, outputs)

model.compile(loss='binary_crossentropy',
optimizer="adam",
metrics=[ 'accuracy'])
```

5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью 5.18.4 Обучение модели

- Обучим модель на основе предобученной сети VGG16 с использованием механизма обратного вызова на 80 эпохах.

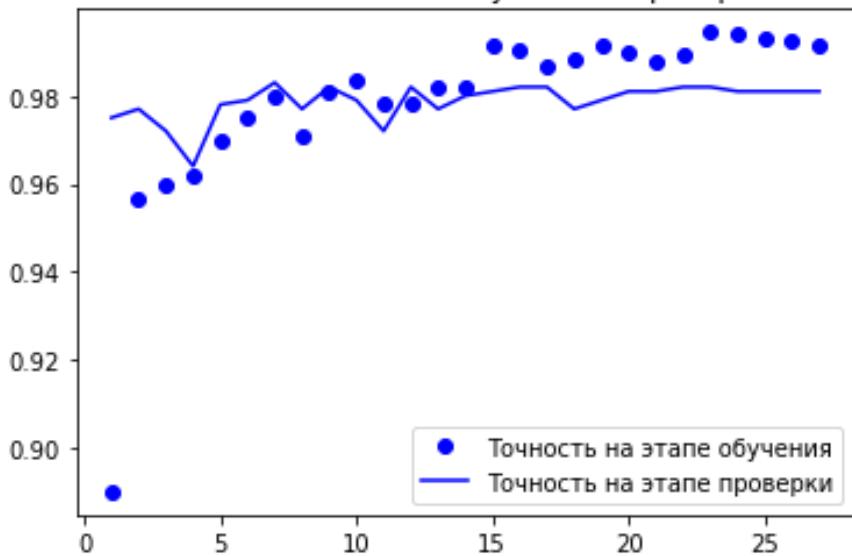
```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="cats_and_dogs_small_10.2024.h5",
        save_best_only=True,
        monitor="val_loss"),
    keras.callbacks.EarlyStopping(
        monitor='val_loss',
        patience=20),
    keras.callbacks.ReduceLROnPlateau(
        monitor='val_loss',
        factor=0.2,
        patience=7)
]
history = model.fit(
    train_dataset,
    epochs=80,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

5.18 Применение сверточных сетей. Бинарная классификация цветных RGB изображений сверточной сетью 5.18.8 Результаты обучения и валидации, графики потерь и точности процесса обучения

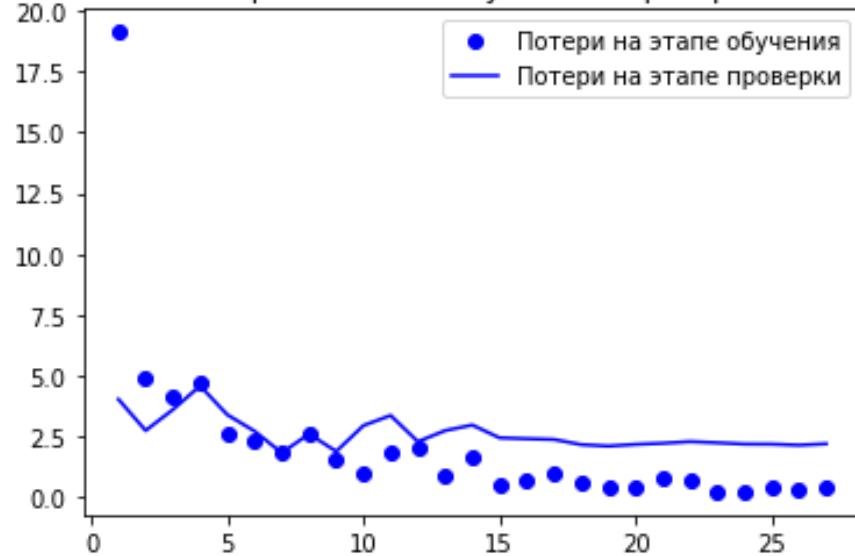
- Результаты для модели на основе предобученной сети VGG16 с использованием аугментации и регуляризацией dropout, 80 эпох дают следующее.
- Мы достигли точности для валидационной выборки 98.1%, для тестовой – 97.6% — более высокой, чем в предыдущем разделе.
- Графики показывают, что эффект переобучения практически отсутствует.

```
Epoch 25/80
63/63 [=====] - 187s 3s/step - loss: 0.4673 - accuracy: 0.9930 - val_loss: 2.1917 - val_accuracy: 0.9810 - lr: 4.0000e-05
Epoch 26/80
63/63 [=====] - 186s 3s/step - loss: 0.3447 - accuracy: 0.9925 - val_loss: 2.1513 - val_accuracy: 0.9810 - lr: 4.0000e-05
Epoch 27/80
63/63 [=====] - 187s 3s/step - loss: 0.4628 - accuracy: 0.9915 - val_loss: 2.2051 - val_accuracy: 0.9810 - lr: 4.0000e-05
32/32 [=====] - 61s 2s/step - loss: 2.5937 - accuracy: 0.9760
Test accuracy: 0.976
Test loss: 2.594
```

Точность на этапах обучения и проверки



Потери на этапах обучения и проверки



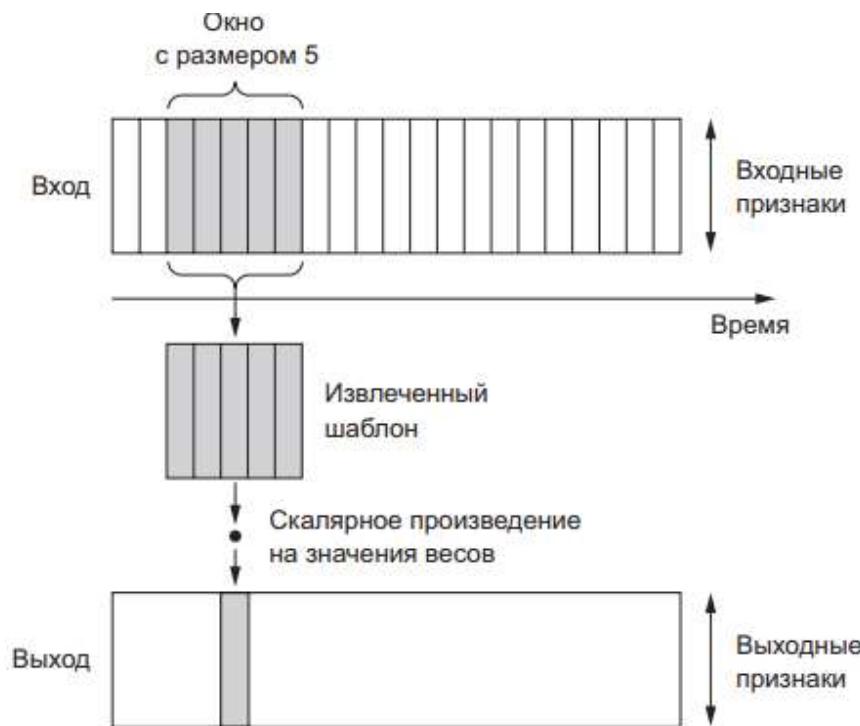
5.19 Нейросетевая обработка текстовых и последовательностных данных 15.9.2 Обработка с помощью одномерной сверточной нейронной сети

- Аналогично двумерным сверткам, извлекающим двумерные шаблоны из тензоров с изображениями и применяющими идентичные преобразования к каждому такому шаблону, можно использовать одномерные свертки для извлечения одномерных шаблонов (подпоследовательностей) из последовательностей, в том числе текстовых.
- Такие одномерные сверточные слои способны распознавать локальные шаблоны в последовательности. Поскольку к каждому шаблону применяются одни и те же преобразования, тот или иной шаблон, найденный в некоторой позиции в предложении, позднее может быть опознан в другой позиции, что делает преобразования, выполняемые одномерными сверточными сетями, инвариантными (во времени).

5.19 Нейросетевая обработка текстовых и последовательностных данных 15.9.2 Обработка с помощью одномерной сверточной нейронной сети

- Например, одномерная сверточная сеть, обрабатывающая последовательность символов или числовых значений и использующая окно свертки с размером 5, способна запоминать слова, фрагменты слов или подпоследовательности чисел длиной до 5 символов и распознавать их в любом контексте во входной последовательности, — то есть одномерная сверточная сеть, обрабатывающая текст или числовую последовательность поэлементно, способна изучить морфологию слов или паттерны числовых последовательностей.

В одномерной сверточной нейронной сети каждый временной интервал извлекается из временного шаблона во входной последовательности.



5.19 Нейросетевая обработка текстовых и последовательностных данных 15.9.2 Обработка с помощью одномерной сверточной нейронной сети

- В двумерных сверточных сетях применяются операции выбора соседних значений, такие как выбор среднего или максимального значения из двумерных данных, которые используются в сверточных сетях для уменьшения разрешения тензоров с изображениями.
- Двумерные операции выбора имеют одномерный эквивалент, извлекающий одномерные шаблоны (подпоследовательности) из входных данных и возвращающий максимальное (выбор максимального значения из соседних) или среднее значение (выбор среднего значения из соседних). Так же как в двумерных сверточных сетях, эта операция используется для уменьшения длины одномерного входа (*снижения разрешения*).

5.19 Нейросетевая обработка текстовых и последовательностных данных 15.9.2 Обработка с помощью одномерной сверточной нейронной сети

- В Keras одномерные сверточные сети создаются с помощью слоя Conv1D, интерфейс которого напоминает интерфейс слоя Conv2D. Он принимает на входе трехмерные тензоры с формой (образцы, время, признаки) и возвращает трехмерные тензоры с той же формой. Окно свертки — это одномерное окно на оси времени: оси с индексом 1 во входном тензоре.
- Создадим простую двухслойную одномерную сверточную сеть и использовать ее для решения уже знакомой нам задачи определения эмоциональной окраски отзывов в наборе данных IMDB.

5.19 Нейросетевая обработка текстовых и последовательностных данных 15.9.2 Обработка с помощью одномерной сверточной нейронной сети

□ Подготовка данных IMDB

```
from keras.datasets import imdb
from keras.preprocessing import sequence

max_features = 10000
max_len = 500

print('Loading data...')
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=
                                                       max_features)
print(len(x_train), 'train sequences')
print(len(x_test), 'test sequences')

print('Pad sequences (samples x time)')
x_train = sequence.pad_sequences(x_train, maxlen=max_len)
x_test = sequence.pad_sequences(x_test, maxlen=max_len)
print('x_train shape:', x_train.shape)
print('x_test shape:', x_test.shape)
```

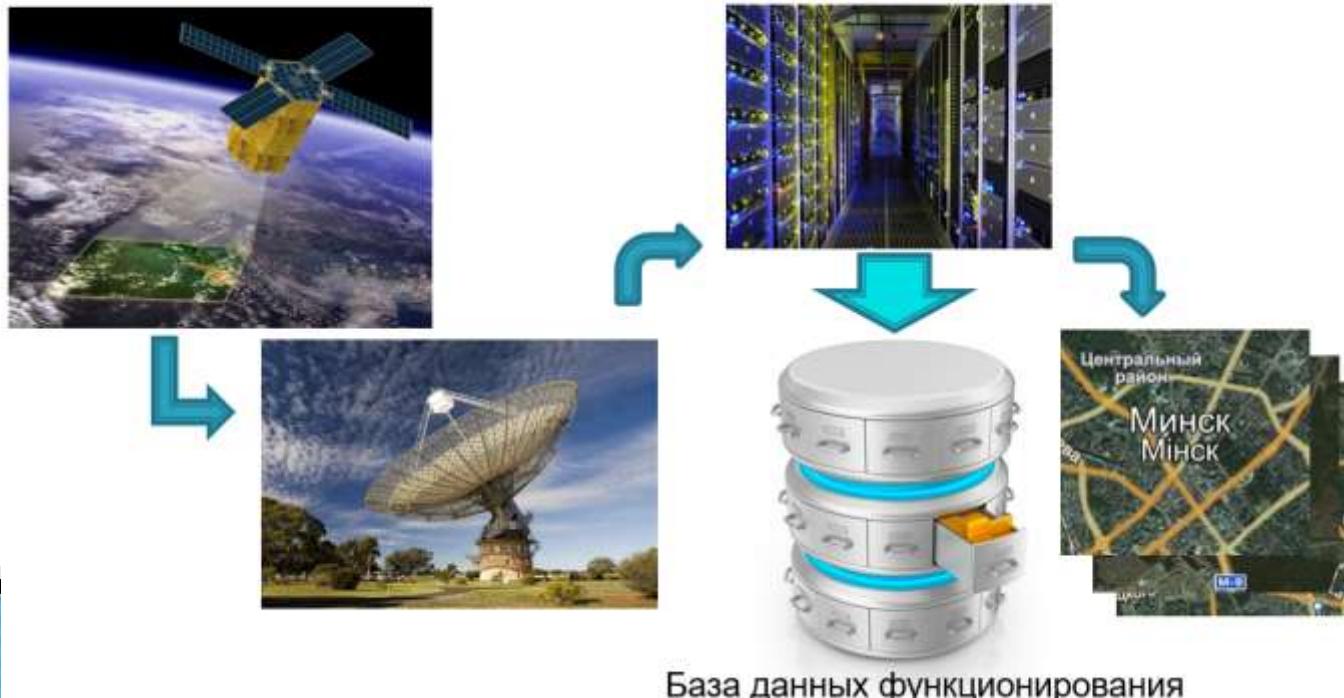
5.19 Нейросетевая обработка текстовых и последовательностных данных 15.9.2 Обработка с помощью одномерной сверточной нейронной сети

- Одномерные сверточные сети устроены так же, как их двумерные сородичи, с которыми мы познакомились ранее: они состоят из последовательности слоев Conv1D, MaxPooling1D или AveragePooling1D завершающейся слоем GlobalMaxPooling1D или Flatten, который преобразует трехмерный вывод в двумерный, что позволяет добавить в модель один или несколько слоев Dense для классификации или регрессии.
- Одно из различий — возможность использовать в одномерных сверточных сетях более крупные окна свертки. Окно свертки 3×3 в двумерном сверточном слое содержит $3 \times 3 = 9$ векторов признаков, а в одномерном сверточном слое окно с размером 3 содержит только 3 вектора признаков. Соответственно, мы легко можем использовать окна свертки с размером 7 или 9.

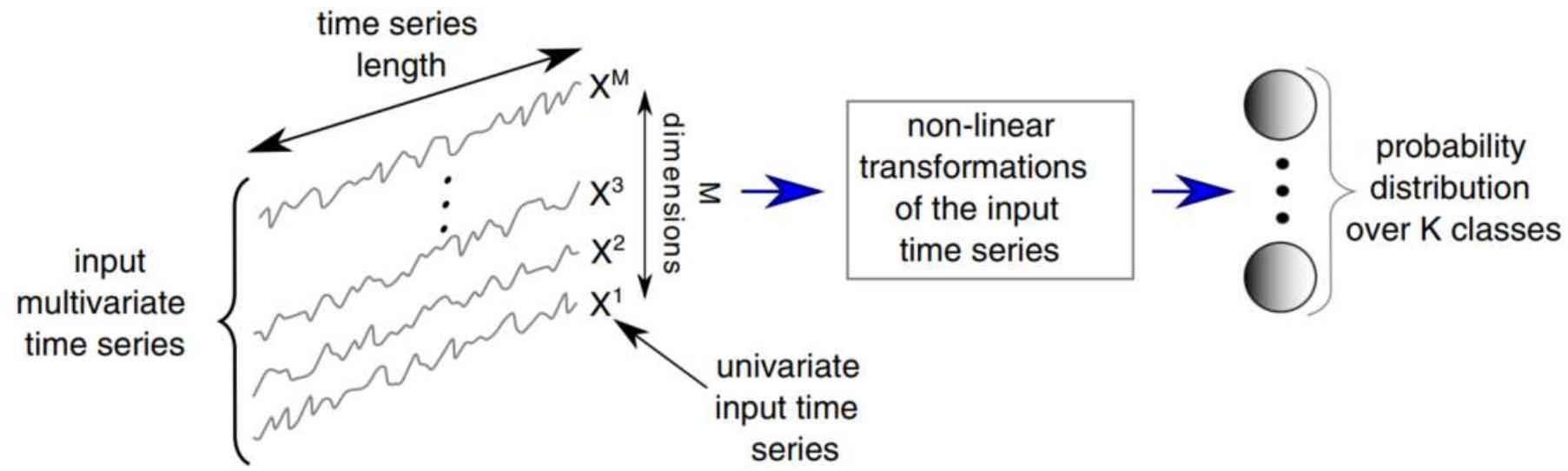
5.19 Нейросетевая обработка текстовых и последовательностных данных 15.9.3 Классификация данных телеметрии МКА с помощью одномерной сверточной нейронной сети

- Данные функционирования высокотехнологичных информационных и инженерных систем, таких как МКА, на сегодняшний день по своим объемам и принципам работы с ними относятся к категории Big Data.
- Для решения задач анализа таких данных эффективным является разработка и применение методов и моделей машинного обучения с целью извлечения из них полезной информации и последующего построения с их использованием кластерно-классификационных и прогностических моделей для определения технического состояния МКА и принятия корректных управляющих и эксплуатационных решений МКА.

Система ДЗЗ



5.19 Нейросетевая обработка текстовых и последовательностных данных 15.9.3 Классификация данных телеметрии МКА с помощью одномерной сверточной нейронной сети



5.19 Нейросетевая обработка текстовых и последовательностных данных 15.9.3 Классификация данных телеметрии МКА с помощью одномерной сверточной нейронной сети

- Исходные данные ТМИ являются временным рядом, который можно представить как матрицу $X = (x_{ij})$, где i -я строка X_i является анализируемым вектором показателей ТМИ в i -й момент времени, индекс j соответствует j -му показателю ТМИ в i -м векторе X_i .
- Данные ТМИ являются M -мерным временным рядом $X = (X_1, X_2, \dots, X_M)$, каждый элемент которого X_j является столбцом матрицы данных ТМИ X и в тоже время одномерным временным рядом, описывающим поведение j -го показателя ТМИ на отрезке дискретных моментов времени $[1, T]$.
- Для каждого вектора показателей ТМИ в i -й момент времени X_i в соответствие поставлена метка класса $y_i \in Y$, который характеризует состояние функционирования анализируемого по данным ТМИ МКА или его подсистемы.
- Рассматриваем случаи k -классовой классификации, где k - общее число состояний, определяемое экспертом. Конечной целью является классификация векторов X_i M -мерного временного ряда X ТМИ к 1 штатному и $k-1$ нештатным состояниям. В этом случае значения компонент вектора меток классов $Y \in \{0, 1, \dots, k-1\}$, где 0 обозначает штатное состояние и $1, \dots, k-1$ - нештатные состояния анализируемого МКА или его подсистемы.

5.19 Нейросетевая обработка текстовых и последовательностных данных 15.9.3 Классификация данных телеметрии МКА с помощью одномерной сверточной нейронной сети

- Стоит задача нахождения классификационной модели следующего нелинейного отображения:

$$y: X \rightarrow Y.$$

- Для кодирования меток классов можно использовать исходное скалярное или One Hot кодирование:

вектору X_i M -мерного временного ряда X соответствует не скалярное значение метки класса, а вектор $Y_i = (y_{i0}, y_{i1}, \dots, y_{ik-1})$ размерности k . При этом в векторе Y_i присутствует только одно значение 1, соответствующее метке класса.

	0	1
11488	1	0
11489	1	0
11490	0	1
11491	0	1
11492	0	1
11493	1	0
11494	1	0
11495	1	0

	0	1	2
472	1	0	0
473	0	1	0
474	0	1	0
475	0	1	0
476	0	1	0
477	0	1	0
478	0	1	0
479	0	1	0
480	0	1	0
481	1	0	0
482	1	0	0
483	1	0	0
484	1	0	0

5.19 Нейросетевая обработка текстовых и последовательностных данных 15.9.3 Классификация данных телеметрии МКА с помощью одномерной сверточной нейронной сети

Экспериментальные данные ТМИ МКА

- Для проведения компьютерных экспериментов и анализа точности разработанных нейросетевых классификационных моделей использованы данные ТМИ навигационной подсистемы МКА.
- Для БКА вектор матрицы ТМИ X_i имеет размерность 9 и помечается 0 в случае штатного состояния и 1 в случае нештатного состояния подсистемы. Общая размерность 9-мерного временного ряда X составляет 121690 векторов, из которых 77881 векторов составляет обучающий набор данных, 19471 векторов составляет валидационный набор данных, 24338 вектор составляет тестовый набор данных.

5.19 Нейросетевая обработка текстовых и последовательностных данных 15.9.3 Классификация данных телеметрии МКА с помощью одномерной сверточной нейронной сети

□ Подготовка данных:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import tensorflow as tf
import keras

data = pd.read_csv("E:\\PythonProjects\\Reliability_analysis\\June_2017_1_121691.csv")
data_arr = data.values
size = data_arr.shape
x = data_arr[:,0:size[1]-1]
y = data_arr[:,size[1]-1]
x = x.reshape((x.shape[0], x.shape[1], 1))
y = y.reshape((y.shape[0], 1))

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=.25)
```

5.19 Нейросетевая обработка текстовых и последовательностных данных 15.9.3 Классификация данных телеметрии МКА с помощью одномерной сверточной нейронной сети

□ Создание и компиляция модели нейронной сети:

```
from keras.models import Sequential
from keras import layers
from keras.optimizers import RMSprop, Adam
model = Sequential()
model.add(layers.Conv1D(filters=256, kernel_size=4, padding='same', activation='relu',
input_shape=(x.shape[1], 1)))
#model.add(layers.Conv1D(32, 7, activation='relu'))
model.add(layers.MaxPooling1D(2))
model.add(layers.Conv1D(filters=128, kernel_size=2, padding='same', activation='relu'))
model.add(layers.GlobalMaxPooling1D())
model.add(layers.Dense(1, activation='sigmoid'))
model.summary()
|
model.compile(optimizer='adam',
loss='binary_crossentropy',
metrics=['acc'])
m=model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=500, batch_size=128)
```

```
Epoch 4500/4500
91267/91267 [=====] - 4s 38us/step - loss: 0.1561 - acc: 0.9341 -
val_loss: 0.2037 - val_acc: 0.9190
```