УНИВЕРСИТЕТ
искусственного
интеллекта

Сверточные нейронные сети

занятие 3

УНИВЕРСИТЕТ
искусственного интеллекта

Сверточные нейронные сети

Занятие № 3

Прежде чем приступить к новой теме, повторим пройденный материал. На предыдущих занятиях мы использовали полносвязные сети, т.е. сети, где все нейроны слоя были связаны со всеми нейронами предыдущего слоя. При работе с Mnist мы “вытягивали” картинку в одномерный вектор и вместо 28x28x1 получили 784, нормировали значение и отдавали эти данные полносвязной нейронной сети.

Хотя правильнее говорить “сети из полносвязных слоев”, т.к. в сложных архитектурах среди сверточных слоев, где-нибудь в середине, вполне может быть полносвязный слой. От его наличия сеть не становится полносвязной.

Так в чем же недостаток полносвязных сетей? Во-первых, в ресурсоемкости таких слоев. Во-вторых, есть такое понятие “локализация признака”. Что это такое?

Вспомните базу с эхолота из прошлого занятия. Мы свободно могли поменять данные с датчиков местами и разницы бы не было. Практически идеальный вариант для полносвязной сети (состоящей из полносвязных слоев). Почему? Потому что данные между собой не связаны и их взаиморасположение не важно. А что будет, если в картинке мы начнем менять местами расположение пикселей? Это будет уже другая картинка того же размера, т.к. в изображении важно взаиморасположение данных.

Сверточные нейронные сети

И еще одна проблема: при использовании полносвязного слоя каждый нейрон связан с каждым нейроном предыдущего слоя, т.е. каждый нейрон анализирует все изображение и может найти ложные зависимости.

Конечно, это не значит, что их использовать нельзя или что они не сработают. Как-то же у нас распознавание цифр работало. Могут быть даже двумерные Dense слои, но есть более эффективные типы слоев для анализа изображений.

В упрощенном варианте, задача нейронной сети - провести разделяющую плоскость

(в данном случае, прямую) между 2-мя классами.

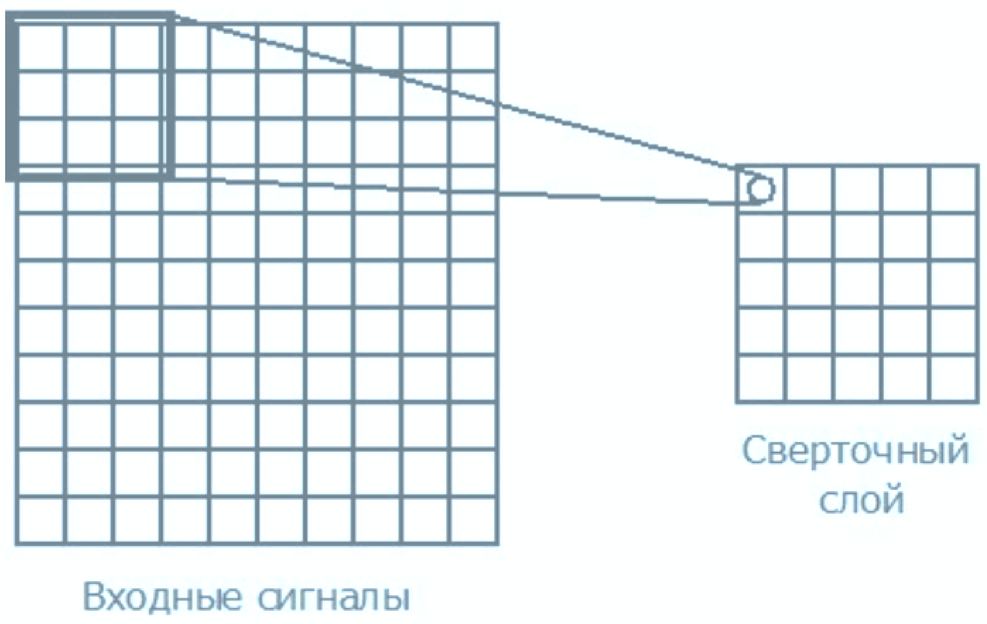
При таком расположении классов прямая проходит через ноль и проблем не возникнет.

Принципы сверточных нейросетей

3 основных принципа сверточных сетей:

- локальное восприятие
- разделяемые веса
- уменьшение размерности

Локальное восприятие

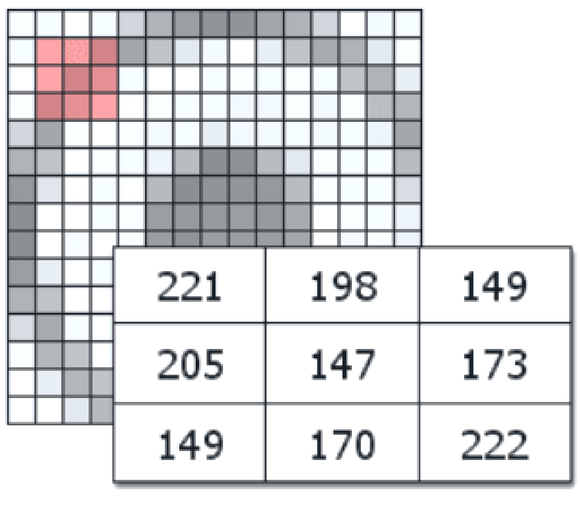


В отличие от полносвязных сетей, которые анализируют всю картинку целиком, сверточные сети анализируют картинку по частям, проходя по ней ядром свертки.

Принципы сверточных нейросетей

Существуют одномерные, двумерные и даже трехмерные сверточные слои. Но на данном уроке будут рассматриваться в основном двумерные.

Что делает ядро свертки? Оно выполняет поэлементное умножение части изображения на маску и складывает полученные значения. Делает шаг и повторяет операцию.



N(x,y) = 221 * (-1) +
198 * 0 +
149 * 1 +
205 * (-2) +
147 * 0 +
173 * 2 +
149 * (-1) +
170 * 0 +
222 * 1 = - 63

Так формируется новая картинка. Каждое ядро\нейрон свертки формирует свою картинку. Не важно, сколько слоев пришло на вход. Каждый нейрон\ядро выдаст только одну картинку. Поэтому независимо от того, подали мы однослойную ч\б картинку или трехслойную RGB, количество картинок, сформированных сверточным слоем, равно количеству нейронов\ядер свертки сверточного слоя.

Размытие

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Выделение границ

0	-1	0
-1	4	-1
0	-1	0

Повышение четкости

0	-1	0
-1	5	-1
0	-1	0

В нейронных сетях ядра свертки определяются автоматически в процессе обучения

У каждого ядра свертки в процессе обучения формируется своя матрица, и это ядро начинает отвечать за поиск определенного признака. В упрощенном варианте: одно ядро ищет горизонтальную прямую, другое вертикальную, третье диагональ и т.п. На следующем слое ищутся углы, потом фигуры и т.д.

Принципы сверточных нейросетей

Размер ядра свертки задается в настройках слоя, также как и количество таких ядер. Самые распространенные ядра двумерных сверточных слоев 3x3, 5x5 и 7x7, а количество ядер, как правило, используют кратное двум (8, 16, 32, 64 и т.д.). Но никто не запрещает использовать ядро 3x7 или 27x27 и кол-во ядер 315.

Например, такие ядра свертки используются в фильтрах photoshop.

Слой Conv2D

Добавляется сверточный слой просто:

```
add(Conv2D(32, (3, 3), padding='same', activation='relu'))
```

Первый параметр(**32**) - определяет кол-во ядер свертки.

Второй параметр(**3,3**) - размер ядра свертки.

Параметр **padding='same'** - нужен для сохранения размера картинки, иначе пиксели по краям “обрезаются”, по умолчанию значение “**valid**”.

Параметр **activation='relu'** обычное указание функции активации.

strides - не обязательный параметр, задает сдвиг, т.е. на сколько должно сдвинуться окно свертки.

https://keras.io/api/layers/convolution_layers/convolution2d/

Документация свёрточных слоёв на русском языке https://ru-keras.com/convolutional-layers/

Слои MaxPooling2D\AveragePooling2D

https://keras.io/api/layers/pooling_layers/max_pooling2d/

https://keras.io/api/layers/pooling_layers/average_pooling2d/

Документация Pooling-слоёв на русском языке https://ru-keras.com/pooling-layers/

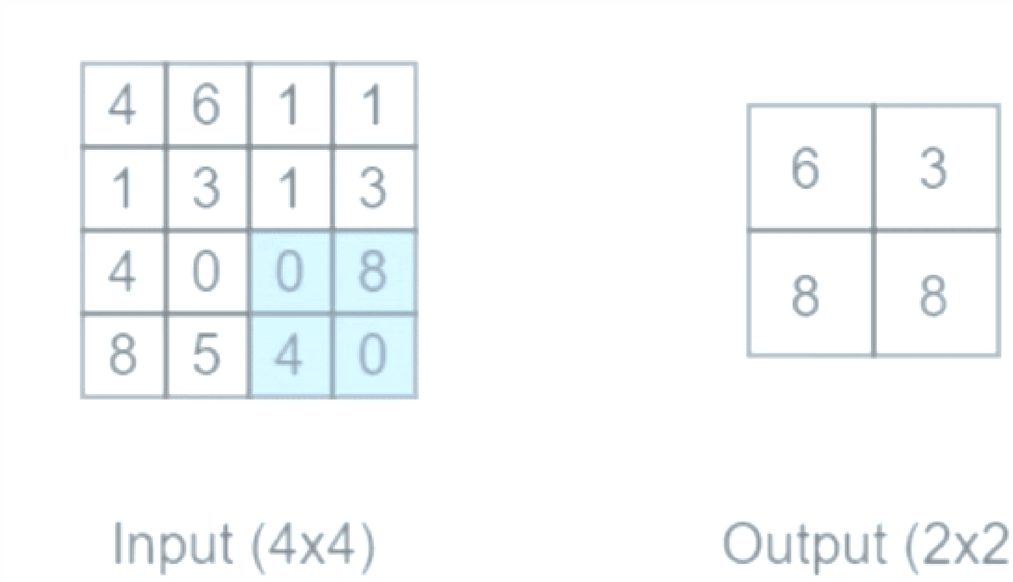
- Распознавание объектов вне зависимости от масштаба.
- Факт наличия признака важнее знания места его точного положения на изображении.

Принципы сверточных нейросетей

Слои подвыборки (subsampling):

- Усреднение.
- Выбор максимального значения.

Этот слой немного похож на сверточный, у него также есть ядро свертки. Но в отличие от сверточного слоя он уменьшает размер изображения (обычно в 2 раза), выбирая максимальное (MaxPooling) или среднее (AveragePooling) значение из окна свертки. При смещении окна свертки не пересекаются. В некотором роде слой делает информацию более сконцентрированной



Лучше стараться использовать такие разрешения изображений, чтобы обе размерности делились на 2 без остатка. Зачем? Существуют слои, которые выполняют обратную операцию. Но если вы возьмете изображение 15x15, примените MaxPooling, то на выходе у вас будет изображение 7x7. И если потребуется применить Conv2DTranspose (один из “разжимающих“ слоев), то получите изображение 14x14, что не будет совпадать с исходными размерами. С такими проблемами вы еще столкнетесь в следующих темах.

Слой Flatten

https://keras.io/api/layers/reshaping_layers/flatten/

Документация слоя Flatten на русском языке https://ru-keras.com/core-layers/

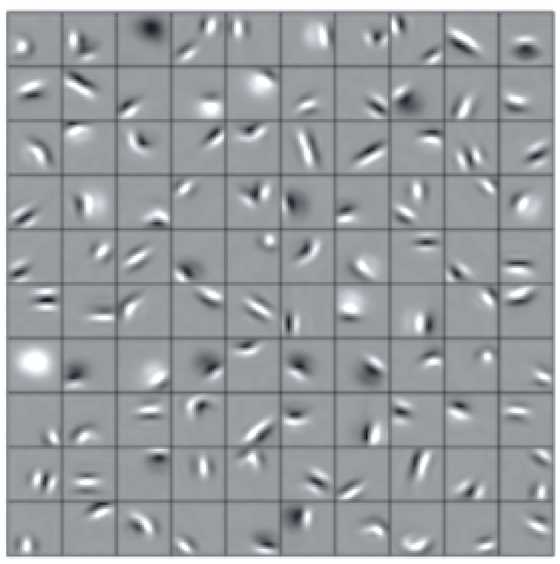
Это достаточно простой слой, он вытягивает данные в строку, делает их плоскими.
Аналогичное действие мы выполняли при переводе с Mnist, когда 28x28x1 превращали в 784. Слой не требует параметров.

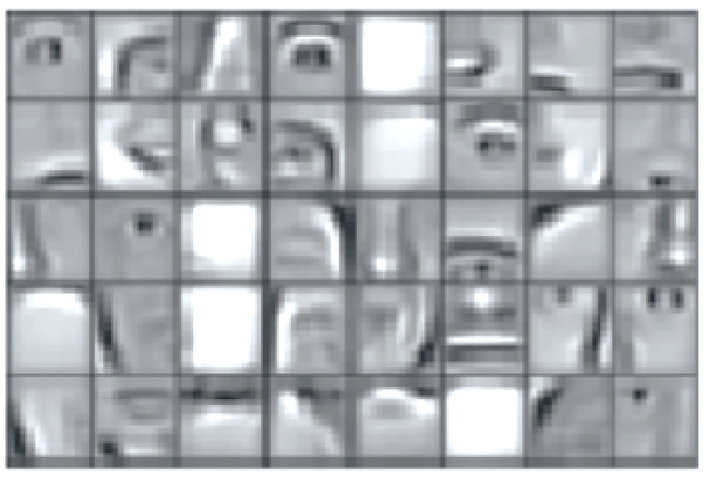
```
add(Flatten())
```

Принципы сверточных нейросетей

Это же действие можно выполнить с помощью слоя Reshape, но нужно будет считать новые размерности. Со слоем Flatten ничего высчитывать не нужно.

Следующая иллюстрация может дать представление о том, что происходит в нейронах сверточной сети.





Первый слой, условно, ищет геометрические фигуры, далее следует уменьшение размерности. Второй слой из фигур предыдущего ищет уже отдельные элементы лица, опять уменьшение размерности. Третий слой уже из элементов лиц составляет лицо.

В этом уроке будут использованы уже известный вам набор картинок Mnist, Cifar10, Cifar-100 и база картинок с автомобилями.

Набор данных Cifar-10

Набор состоит из цветных картинок 32x32, 50000 картинок для обучения и 10000 для теста. На каждом изображении только один объект для классификации. Всего в наборе 10 классов. Классы изображений: самолет, легковой автомобиль, птица, кот, олень, собака, лягушка, лошадь, корабль и грузовик.

Разбор ноутбука

Mnist

Первое отличие от полносвязных сетей. Нам нужно преобразовать данные, добавив одну размерность, а не убрать, как на прошлом занятии. Изначально размерность обучающей выборки 60000x28x28, но двумерные сверточные сети требуют на вход 3х мерные данные картинки. На самом деле, даже четырехмерные, т.к. еще размер батча, но он проставляется автоматически, в итоге на входе размерность (None,28,28,1).

Поэтому нам нужно сделать **Reshape** данных, добавив 1 размерность для картинки.

```
x_train.reshape(x_train.shape[0], 28, 28, 1)
```

Это же делаем для тестовой выборки.

Создаем нейронную сеть.

Обратите внимание на первый слой,

```
model.add(BatchNormalization(input_shape=(28, 28, 1)))
```

вместо **input_dim**, который использовался на прошлом уроке, нужно использовать **input_shape,** т.к. данные уже не одномерные.

Дальше идут комбинации слоев, описанных выше: Conv2D, MaxPooling, Flatten и т.д. Выход сети такой же как и на прошлом занятии.

Cifar 10

Практически все то же самое, только картинка цветная и чуть больше. Размерность будет 32x32x3, последняя 3 - это количество каналов цвета: красный, зеленый, синий. Тут Reshape делать не нужно, т.к. при чтении цветной картинки она загружается уже нужных размерностей.

Нейронная сеть используется чуть мощнее. С ее помощью вы сможете получить точность порядка 85%.

Получить лучшие результаты уже проблематично, но возможно. Для этого можно использовать аугментацию, т.е. изменение изображений.

Разбор ноутбука

Аугментация

Это один из способов “расширить” базу для обучения, изменяя имеющиеся картинки.

Для изменения изображений используются отражения по вертикали и горизонтали, повороты, сдвиги, приближения и т.п.

```
datagen = ImageDataGenerator(
    rescale=1. / 255, #Значения цвета меняем на дробные показания
    rotation_range=10, #Поворачиваем изображения при генерации выборки
    width_shift_range=0.1, #Двигаем изображения по ширине при генерации выборки
    height_shift_range=0.1, #Двигаем изображения по высоте при генерации выборки
    zoom_range=0.1, #Зумируем изображения при генерации выборки
    horizontal_flip=True, #Отключаем отзеркаливание изображений
    fill_mode='nearest', #Заполнение пикселей вне границ ввода
    validation_split=0.1 #Указываем разделение изображений на обучающую и тестовую выборку
)
```

Не стоит использовать все изменения сразу. Например, для спутниковых снимков отражение по вертикали пойдет в плюс, а учить нейронку на людях, стоящих на потолке, не стоит. В конце ноутбука есть пример использования ImageGenerator. С его помощью можно сделать аугментацию и обучить сеть до лучшей точности.

Не забудьте, что в случае использования генератора для обучения сети вместо **fit** нужно использовать fit_generator.

```
model.fit_generator(
 train_generator,
 steps_per_epoch = train_generator.samples // batch_size,
```

Разбор ноутбука

```
 validation_data = validation_generator,
 validation_steps = validation_generator.samples // batch_
size,
 epochs=45,
 verbose=1
)
```

Попробуйте добиться точности 90+% на базе cifar10, на данный момент рекорд наших студентов по этой базе - 94%.

Для Cifar 100 все то же самое только уже для 100 классов. Точность будет пониже, но это далеко не самая простая база.

База автомобилей.

На этой базе как раз показывается использование ImageGenerator. Изображения читаются из папки, в отличие от загружаемых датасетов.