Введение в искусственные нейронные сети

Свёрточные нейронные сети

На этом уроке

1. Познакомимся со свёрточными нейронными сетями и глубоким обучением
2. Рассмотрим их архитектуру
3. Изучим применение сетей на практике

Оглавление

Что такое свёрточные нейронные сети

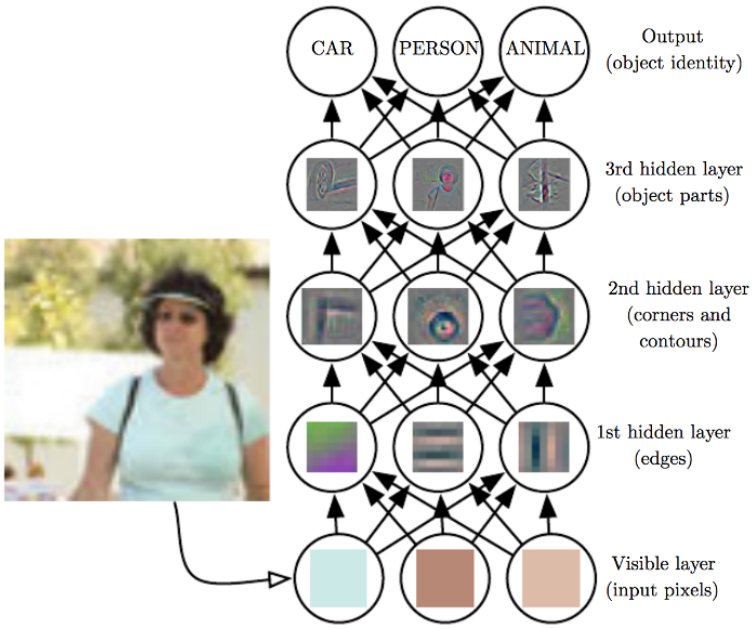
Свёрточные нейронные сети — это нейронные сети, приспособленные в первую очередь для задач распознавания образов. В их основе лежат работы в области изучения зрительной коры головного мозга. Их отличительная черта — добавление свёрточных и пуллинговых слоёв в архитектуру нейронной сети. Подробности архитектуры мы рассмотрим в следующей части методического пособия, а пока взглянем на области применения данного вида нейронных сетей:

- Задачи, связанные с определением того, к какому классу принадлежит объект на фотографии
- Свёрточные нейронные сети в модифицированном виде могут определять не только что именно находится на фотографии, но и где оно находится (этому виду нейронных сетей будет посвящён отдельный урок)
- Распознавание лиц. В 2001 г. появился алгоритм Виолы-Джонса, который предложил технологию, позволяющую технике находить лица на фотографиях и в видеопотоке. На данный момент этот алгоритм превзойдён свёрточными нейронными сетями по эффективности
- Проставление лейблов изображениям. Используется Google, Amazon, Facebook
- Визуальный поиск. Используется Google
- Рекомендательные системы. Amazon, например, использует это для секции «вам также может понравиться» в категории одежды.
- В социальных сетях с их помощью отмечаются люди на фотографиях
- Помощь врачам в анализе медицинских снимков
- Предиктивная аналитика. Помощь в предсказании проблем со здоровьем
- Оценка банками цифр, написанных от руки. Одно из самых ранних применений свёрточных нейронных сетей

Применение свёрточных нейронных сетей не ограничивается областью компьютерного зрения. Они также применяются и в других областях:

- Анализ текстов. Для этого больше подходят рекуррентные нейронные сети, но когда речь заходит о детекции определённых признаков в тексте, например бранной речи, наилучшим вариантом являются свёрточные нейронные сети
- Предиктивный анализ. В частности, предсказание погоды

Глубокое обучение



Глубокое обучение — это обучение глубоких нейронных сетей. Глубокие нейронные сети — это сети с как минимум двумя внутренними слоями.

Прежде чем разбирать глубокое обучение, вкратце опишем свёрточные нейронные сети. Типичная свёрточная нейронная сеть состоит из входного слоя, череды свёрточных и пуллинговых слоёв, обычно следующих друг за другом, и нескольких полносвязных слоёв на выходе.

Попробуем разобраться в смысле данной архитектуры и её связью с глубоким обучением. Нейронная сеть в один слой может решить любую задачу, но такой подход будет очень грубым решением проблемы. Так, вычислительной мощности современного компьютера не хватит, чтобы нейронная сеть в один слой могла различить классы объектов на фотографии.

Эта дилемма решается через иной научный факт, известный в отношении нейронных сетей, — чем больше слоёв имеет сеть, тем она эффективнее. Для построения многослойной нейронной сети может понадобиться меньше нейронов, чем для однослойной. Связано это с тем, что каждый слой выучивает признаки на определённом уровне абстракции, и следующие за ним слои используют уже имеющиеся признаки, а не выучивают их заново.

Посмотрим, как происходит процесс обучения в глубокой нейронной сети, (такой же процесс в общих чертах будет характерен и для свёрточных нейронных сетей).

Представим работу с изображениями животных. Первые слои выучат признаки животных низкого уровня абстракции, такие как линии под определёнными углами; следующие слои на базе этих признаков выучат более сложные признаки, например геометрические фигуры на базе сочетания этих линий. Последующие слои выучат такие признаки как глаза, уши и т.д., которые будут составлены из

этих геометрических фигур. Подобные высокоабстрактные признаки уже можно использовать для того, чтобы сделать заключение, какое животное изображено на картинке.

Описанная система лучше с точки зрения вычислительных затрат. Однако если мы сделаем несколько полносвязных слоёв, где каждый нейрон связан с каждым нейроном другого слоя, то на обсчитывание этих связей уйдёт меньше вычислительных ресурсов, чем если бы нейронная сеть была в один слой. В не учебных задачах такая нейронная сеть всё равно будет обучаться неприемлемо долго.

Архитектура свёрточных нейронных сетей

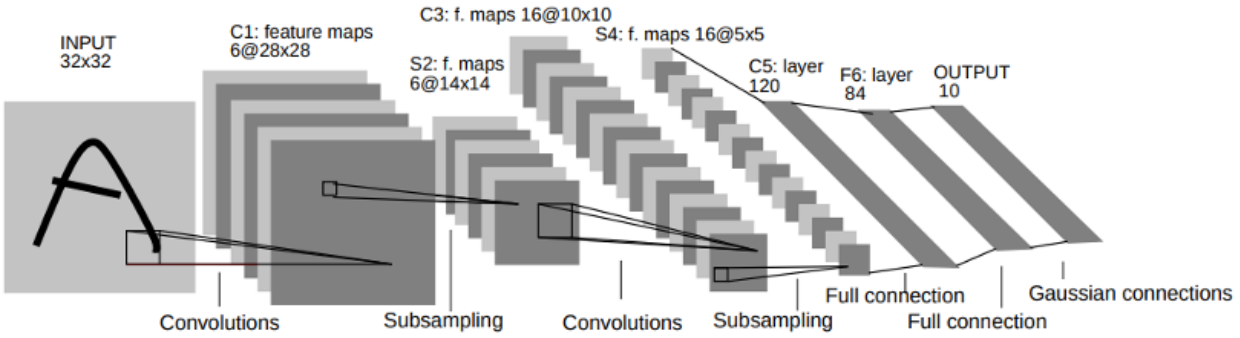


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Свёрточные нейронные сети — это самый природоподобный алгоритм из всех. Современные свёрточные нейронные сети основаны на произвёдшей революцию в компьютерном зрении нейронной сети AlexNet, она базируется на свёрточных нейронных сетях, которые разрабатывал Ян Лекун в 90 гг. Те, в свою очередь, основаны на японском Neocognitron 1980 г., а он — на открытиях в области зрительной коры головного мозга. Конечно, современные архитектуры свёрточных нейронных сетей (такие, как Inception-v4) сильно отличаются от сетей 90-х. Однако у них есть общие черты, которые и делают свёрточные нейронные сети эффективными. Особенности, о которых пойдет далее речь, призваны помочь строить глубокие нейронные сети, имеющие меньшие вычислительные затраты, нежели полносвязные.

Главная отличительная черта свёрточных нейронных сетей — наличие свёрточных слоёв и пуллинг слоёв. Подобные слои как раз и были обнаружены в зрительной коре головного мозга, но названы они по-другому и работают более сложным образом. В искусственной нейронной сети свёрточный слой состоит из фрагментов, которые связаны только с определённой частью изображения, что позволяет не связывать каждый нейрон с каждым пикселем и уменьшить вычислительные затраты. Конечная цель свёрточного слоя — получить определённые признаки от изображения и передать их в следующий слой, точно так же, как и в случае с обычной полносвязной нейронной сетью. Свёрточный

слой делает это более эффективно. Сама операция свёртки представляет собой процесс преобразования большего набора чисел в меньший набор чисел, их репрезентующий. Пуллинг слои следуют за свёрточными и призваны очистить эти признаки от лишней информации и убрать у них локальную привязку. Сама операция пуллинга в целом представляет собой процесс отбрасывания менее значимых сигналов, представленных в виде чисел. Пулинг слои — важная составляющая нейронных сетей, однако из-за них свёрточной нейронной сети всё равно, где, например, располагаются глаза у кота: над носом или под носом, главное — сочетание этих признаков.

Свёрточная нейронная сеть строится по принципу пирамиды — первые слои содержат больше нейронов, а последующие — всё меньше и меньше. Связано это с тем, что низкоабстрактных признаков больше, чем высокоабстрактных.

Как правило, на конце нейронной сети располагаются несколько полносвязных слоёв. Эти слои учатся на немногочисленных высокоабстрактных признаках, не требующих много слоёв, что считается приемлемым с точки зрения вычислительных затрат. Получается, что свёрточную нейронную сеть можно условно поделить на две части — одна извлекает признаки, а другая, полносвязная, на них обучается.

Стоит отметить, что современные свёрточные нейронные сети в целях оптимизации работы снабжаются дополнительными архитектурными решениями, такими как возможность иметь в одном слое разные конфигурации свёртки, пропускать при необходимости сигнал обратного распространения ошибки сквозь слои, использовать нескольких слоёв свёртки подряд или не использовать полносвязные слои на конце нейронных сетей. Обычно все эти нововведения направлены на то, чтобы сделать нейронные сети более глубокими, что, в свою очередь, улучшает точность работы нейронных сетей.

Пример создания свёрточных и пуллинг слоёв в Keras

```
from numpy import asarray
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import GlobalMaxPooling2D


# определение входных данных(8 массивов с 8 элементами)
data = [[0, 0, 0, 1, 1, 0, 0, 0],
            [0, 0, 0, 1, 1, 0, 0, 0],
            [0, 0, 0, 1, 1, 0, 0, 0],
```

```
            [0, 0, 0, 1, 1, 0, 0, 0],
            [0, 0, 0, 1, 1, 0, 0, 0],
            [0, 0, 0, 1, 1, 0, 0, 0],
            [0, 0, 0, 1, 1, 0, 0, 0],
            [0, 0, 0, 1, 1, 0, 0, 0]]

data = asarray(data)
data = data.reshape(1, 8, 8, 1)

# создание модели
model = Sequential()
model.add(Conv2D(1, (3,3), activation='relu', input_shape=(8, 8, 1)))
model.add(GlobalMaxPooling2D())

# вывод описания созданной модели
model.summary()

# определение дектора вертикальной линии
detector = [[[[0]],[[1]],[[0]]],
            [[[0]],[[1]],[[0]]],
            [[[0]],[[1]],[[0]]]]
weights = [asarray(detector), asarray([0.0])]

# сохранение весов в модель
model.set_weights(weights)

# применение фильтра к входным данным
yhat = model.predict(data)

print(yhat)
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_9 (Conv2D)            (None, 6, 6, 1)           10
_________________________________________________________________
global_max_pooling2d_2 (Glob (None, 1)                 0
=================================================================
Total params: 10
```

```
Trainable params: 10

Non-trainable params: 0

_________________________________________________________________

[[3.]]
```

Нейронная сеть Lenet5

Lenet5 — одна из первых свёрточных нейронных сетей, она отражает характерные для них набор элементов: свёрточные слои, пуллинг слои и полносвязные слои на конце нейронной сети. Данная архитектура послужила основой для многих современных архитектур свёрточных нейронных сетей.

```
from keras.datasets import mnist

from keras.utils import np_utils


# загрузка тренировочных и тестовых данных

(x_train, y_train), (x_test, y_test) = mnist.load_data()


# конвертация чисел из uint8 в float32

x_train = x_train.astype('float32')

x_test = x_test.astype('float32')


# нормализация данных [0, 1]

x_train /= 255

x_test /= 255


# трансформация лейблов в one-hot encoding

y_train = np_utils.to_categorical(y_train, 10)

y_test = np_utils.to_categorical(y_test, 10)


# изменение размерности массива в 4D массив

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

x_test = x_test.reshape(x_test.shape[0], 28,28,1)


from keras.models import Sequential

from keras import models, layers

import keras
```

```
# инициализация пустой модели

model = Sequential()



# первый сверточный слой

model.add(layers.Conv2D(6, kernel_size=(5, 5), strides=(1, 1), activation='tanh', input_shape=(28,28,1),
padding="same"))



# второй пуллинговый слой

model.add(layers.AveragePooling2D(pool_size=(2, 2), strides=(1, 1), padding='valid'))



# третий сверточный слой

model.add(layers.Conv2D(16, kernel_size=(5, 5), strides=(1, 1), activation='tanh', padding='valid'))



# четвертый пуллинговый слой

model.add(layers.AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))



# пятый полносвязный слой

model.add(layers.Conv2D(120, kernel_size=(5, 5), strides=(1, 1), activation='tanh', padding='valid'))



# сглаживание CNN выхода чтобы можно было его присоединить к полносвязногому слою

model.add(layers.Flatten())



# шестой полносвязный слой

model.add(layers.Dense(84, activation='tanh'))



# выходной слой с функцией активации softmax

model.add(layers.Dense(10, activation='softmax'))




# компилияция модели

model.compile(loss=keras.losses.categorical_crossentropy, optimizer='SGD', metrics=["accuracy"])




hist = model.fit(x=x_train,y=y_train, epochs=1, batch_size=128, validation_data=(x_test, y_test), verbose=1)



test_score = model.evaluate(x_test, y_test)

print("Test loss {:.4f}, accuracy {:.2f}%".format(test_score[0], test_score[1] * 100))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/1
60000/60000 [==============================] - 11s 184us/step - loss: 0.7097 - acc: 0.8193 - val_loss: 0.3449 -
val_acc: 0.9069
10000/10000 [==============================] - 1s 68us/step
Test loss 0.3449, accuracy 90.69%
```

Пример на Keras более сложной свёрточной нейронной сети

Попробуем сделать несколько усложнённый вариант разобранной ранее нейронной сети. В ней будет на несколько слоёв больше, а также будет использоваться data augumentation — процедура, позволяющая увеличить количество тренировочных данных за счёт искажений изображений (а как мы знаем, чем больше тренировочных данных, тем лучше будет работать нейросеть). Для обучения нейросети будем использовать датасет cifar-10. В нём 10 категорий объектов, например: лошадь, лягушка, корабль. Для нейронных сетей он уже сложнее, чем mnist, но намного проще датасетов типа imagenet, где используются сотни классов (архитектуры нейронных сетей для подобных датасетов также понадобятся более сложные).

```
from __future__ import print_function

import keras # расскоментируйте эту строку, чтобы начать обучение

from keras.datasets import cifar10

from keras.preprocessing.image import ImageDataGenerator

from keras.models import Sequential

from keras.layers import Dense, Dropout, Activation, Flatten

from keras.layers import Conv2D, MaxPooling2D

import os


# установка параметров нейросети

batch_size = 32

num_classes = 10

epochs = 1

data_augmentation = True

num_predictions = 20

save_dir = os.path.join(os.getcwd(), 'saved_models')

model_name = 'keras_cifar10_trained_model.h5'


# разделение тренировочной и тестовой выборки

(x_train, y_train), (x_test, y_test) = cifar10.load_data()

print('x_train shape:', x_train.shape)
```

```
print(x_train.shape[0], 'тренировочные примеры')
print(x_test.shape[0], 'тестовые примеры')

# преобразование матрицы чисел 0-9 в бинарную матрицу чисел 0-1
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

# конфигурирование слоев нейросети
model = Sequential()

# слои нейросети ответственные за свертку и max-pooling
model.add(Conv2D(32, (3, 3), padding='same',
                 input_shape=x_train.shape[1:]))
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

# полносвязные слои нейронной сети
model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))

# инициализация RMSprop optimizer
opt = keras.optimizers.RMSprop(lr=0.0001, decay=1e-6)

# компиляция модели
```

```
model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])


x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255


if not data_augmentation:
    print('Не используется data augmentation')
    model.fit(x_train, y_train,
              batch_size=batch_size,
              epochs=epochs,
              validation_data=(x_test, y_test),
              shuffle=True)
else:
    print('Использование data augmentation в реальном времени')
    # Препроцессинг и data augmentation в реальном времени:
    datagen = ImageDataGenerator(
        featurewise_center=False,
        samplewise_center=False,
        featurewise_std_normalization=False,
        samplewise_std_normalization=False,
        zca_whitening=False,
        zca_epsilon=1e-06,
        rotation_range=0,
        width_shift_range=0.1,
        height_shift_range=0.1,
        shear_range=0.,
        zoom_range=0.,
        channel_shift_range=0.,
        fill_mode='nearest',
        cval=0.,
        horizontal_flip=True,
        vertical_flip=False,
        rescale=None,
        preprocessing_function=None,
```

```
        data_format=None,
        validation_split=0.0)

    # запуск data augmentation через fit
    #datagen.fit(x_train)

    # запуск data augmentation через fit_generator
    model.fit_generator(datagen.flow(x_train, y_train,
                                     batch_size=batch_size),
                        epochs=epochs,
                        validation_data=(x_test, y_test),
                        workers=4)

# сохранение модели и весов
if not os.path.isdir(save_dir):
    os.makedirs(save_dir)
model_path = os.path.join(save_dir, model_name)
model.save(model_path)
print('сохранить обученную модель как %s ' % model_path)

# проверка работы обученной модели
scores = model.evaluate(x_test, y_test, verbose=1)
print('Test loss:', scores[0])
print('Test accuracy:', scores[1])
```

```
x_train shape: (50000, 32, 32, 3)
50000 тренировочные примеры
10000 тестовые примеры
Использование data augmentation в реальном времени
WARNING:tensorflow:Variable *= will be deprecated. Use `var.assign(var * other)` if you want assignment to the variable value or `x = x * y` if you want a new python Tensor object.
Epoch 1/1
1563/1563 [==============================] - 39s 25ms/step - loss: 1.8751 - acc: 0.3107 - val_loss: 1.5636 - val_acc: 0.4304
сохранить обученную модель как /home/honeyguide/Desktop/Labour/webinars/neuron_seti_vvedenie/4lesson_cnn/saved_models/keras_cifar10_trained_model.h5
10000/10000 [==============================] - 1s 134us/step
Test loss: 1.5635708665847778
Test accuracy: 0.4304
```

Используемые источники

1. https://keras.io
2. Шакла Н. — Машинное обучение и TensorFlow 2019
3. Николенко Сергей Игоревич, Кадурин А. А. — Глубокое обучение. Погружение в мир нейронных сетей 2018
4. Francois Chollet — Deep Learning with Python 2018
5. Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton — ImageNet Classification with Deep Convolutional Neural Networks
6. Karen Simonyan, Andrew Zisserman — Very Deep Convolutional Networks for Large-Scale Image Recognition Википедия