Лабораторная работа №10

# Deep Learning, теперь и в OpenCV

Загрузка и получение результатов (предсказаний) с помощью моделей, созданных в любом из трех популярных фреймворков (Caffe, TensorFlow, Torch), быстрая работа на CPU, поддержка основных слоев нейронных сетей и, как всегда, кроссплатформенность, открытость исходного кода и поддержка.

Зачем вообще понадобился Deep Learning в OpenCV? В последние годы во многих областях глубокое обучение (в некоторых источниках глубинное обучение) показывает результаты, значительно превосходящие аналогичные у классических алгоритмов. Это касается и области компьютерного зрения, где масса задач решается с применением нейронных сетей. В свете данного факта кажется логичным дать пользователям OpenCV возможность работы с нейросетями.

## Основные возможности DNN

Основная возможность dnn заключается, конечно же, в загрузке и запуске нейронных сетей (inference). При этом модель может быть создана в любом из трех фреймворков глубокого обучения — Caffe, TensorFlow или Torch; способ ее загрузки и использования сохраняется независимо от того, где она была создана.

Поддерживая сразу три популярных фреймворка, мы можем достаточно просто комбинировать результаты работы загруженных из них моделей без необходимости создавать все заново в одном единственном фреймворке.

При загрузке происходит конвертация моделей во внутреннее представление, близкое к используемому в Caffe. Так произошло в силу исторических причин — поддержка Caffe была добавлена самой первой. Однако взаимно однозначного соответствия между представлениями нет.

Поддерживаются все основные слои: начиная от базовых (Convolution и Fully connected) и заканчивая более специализированными — всего более 30.

(

Список поддерживаемых слоев

AbsVal

AveragePooling

BatchNormalization

Concatenation

Convolution (with dilation)

Crop

DetectionOutput

Dropout

Eltwise

Flatten

FullConvolution

FullyConnected

LRN

LSTM

MaxPooling

MaxUnpooling

MVN

NormalizeBBox

Padding

Permute

Power

PReLU

PriorBox

ReLU

RNN

Scale

Shift

Sigmoid

Slice

Softmax

Split

TanH

)

Кроме поддержки отдельных слоев, важна также и поддержка конкретных архитектур нейронных сетей. Модуль содержит примеры для классификации ([AlexNet](https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf), [GoogLeNet](https://arxiv.org/pdf/1409.4842.pdf), [ResNet](https://arxiv.org/pdf/1512.03385.pdf), [SqueezeNet](https://arxiv.org/pdf/1602.07360.pdf)), сегментации ([FCN](https://arxiv.org/pdf/1411.4038.pdf), [ENet](https://arxiv.org/pdf/1606.02147.pdf)), детектирования объектов ([SSD](https://arxiv.org/pdf/1512.02325.pdf)); многие из указанных моделей проверены на исходных датасетах.

## Примеры использования

По хорошей традиции, каждый модуль OpenCV включает в себя примеры использования. dnn — не исключение, примеры на С++ и Python доступны в поддиректории samples в репозитории с исходным кодом. В примерах присутствуют комментарии, да и в целом все достаточно просто.

Краткий пример, выполняющий классификацию изображений с помощью модели GoogLeNet.

Данный код загружает картинку, проводит небольшую предобработку и получает для изображения выход сети. Предобработка заключается в масштабировании изображения таким образом, чтобы наименьшая из сторон стала равной 224, вырезании центральной части и вычитании среднего значения из элементов каждого канала. Данные операции необходимы, так как модель была натренирована на изображениях заданного размера (224 x 224) с именно такой предобработкой.

На языке Python наш пример будет выглядеть следующим образом:

**import numpy as np**

**import cv2 as cv**

**# read names of classes**

**with open('synset\_words.txt') as f:**

**classes = [x[x.find(' ') + 1:] for x in f]**

**image = cv.imread('IMG\_20190720\_140452.jpg')**

**# create tensor with 224x224 spatial size and subtract mean values (104, 117, 123)**

**# from corresponding channels (R, G, B)**

**input = cv.dnn.blobFromImage(image, 1, (224, 224), (104, 117, 123))**

**# load model from caffe**

**net = cv.dnn.readNetFromCaffe('bvlc\_googlenet.prototxt', 'bvlc\_googlenet.caffemodel')**

**# feed input tensor to the model**

**net.setInput(input)**

**# perform inference and get output**

**out = net.forward()**

**# get indices with the highest probability**

**indexes = np.argsort(out[0])[-5:]**

**for i in reversed(indexes):**

**print('class:', classes[i], ' probability:', out[0][i])**

Выходной тензор интерпретируется как вектор вероятностей принадлежности изображения к тому или иному классу и имена для 5 классов с наибольшими вероятностями выводятся в консоль.

# Обнаружение объектов с помощью deep learning и OpenCV

*от Adrian Rosebrock 11 сентября 2017 года*

Пару недель назад мы научились классифицировать изображения с помощью модуля deep learning и OpenCV 3.3 deep neural network (dnn).

Хотя этот оригинальный пост в блоге продемонстрировал, как мы можем классифицировать изображение в одну из 1000 отдельных меток классов ImageNet, он не мог сказать нам, где находится объект в изображении.

Чтобы получить ограничивающую рамку (x, y)-координаты объекта на изображении, нам нужно вместо этого применить обнаружение объекта.

Обнаружение объекта может не только сказать нам, что находится на изображении, но и где находится объект.

В оставшейся части сегодняшнего сообщения в блоге мы обсудим, как применять обнаружение объектов с помощью глубокого обучения и OpenCV.

## Обнаружение объектов с помощью глубокого обучения и OpenCV

В первой части сегодняшнего поста об обнаружении объектов с помощью глубокого обучения мы обсудим Single Shot Detectors and MobileNets.

В сочетании эти методы могут быть использованы для сверхбыстрого обнаружения объектов в реальном времени на устройствах с ограниченными ресурсами (включая Raspberry Pi, смартфоны и т. д.)

Оттуда мы узнаем, как использовать модуль dnn OpenCV для загрузки предварительно обученной сети обнаружения объектов.

Это позволит нам пропускать входные изображения через сеть и получать выходную ограничивающую рамку (x, y)-координаты каждого объекта на изображении.

Наконец, мы рассмотрим результаты применения Single Shot Detectors и MobileNet к примерам входных изображений.

В будущем блоге мы расширим наш скрипт для работы с видеопотоками в реальном времени.

## Single Shot Detectors

Когда дело доходит до обнаружения объектов на основе глубокого обучения, есть три основных метода обнаружения объектов, с которыми вы, вероятно, столкнетесь:

* [Faster R-CNNs](https://arxiv.org/abs/1506.01497) (Girshick et al., 2015)
* [You Only Look Once (YOLO)](https://arxiv.org/abs/1506.02640) (Redmon and Farhadi, 2015)
* [Single Shot Detectors (SSDs)](https://arxiv.org/abs/1512.02325) (Liu et al., 2015)

Faster R-CNNs, вероятно, являются наиболее “известным” методом обнаружения объектов с использованием глубокого обучения; однако эта техника может быть трудной для понимания (особенно для начинающих в глубоком обучении), трудной для реализации и сложной для обучения.

Кроме того, даже при “более быстрой” реализации R-CNNs (где “R” означает “предложение региона”) алгоритм может быть довольно медленным, порядка 7 кадров в секунду.

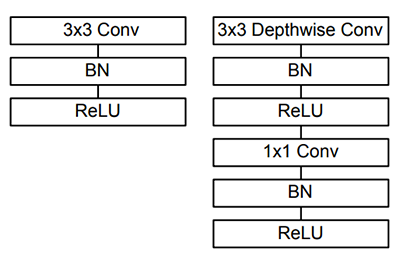
Если мы ищем чистую скорость, то мы склонны использовать YOLO, так как этот алгоритм намного быстрее, способный обрабатывать 40-90 кадров в секунду на графическом процессоре Titan X. Супер быстрый вариант YOLO может даже достигать 155 кадров в секунду.

Проблема с YOLO заключается в том, что он оставляет желать лучшего.

SSDs, первоначально разработанные Google, представляют собой баланс между ними. Этот алгоритм более прост (и я бы сказал, что он лучше объяснен в оригинальной основополагающей статье), Чем более быстрые R-CNN.

Мы также можем наслаждаться гораздо более высокой пропускной способностью кадров в секунду, чем Girshick et al. при 22-46 кадрах в секунду, в зависимости от того, какой вариант сети мы используем. SSD также имеет тенденцию быть более точными, чем YOLO. Чтобы узнать больше о твердотельных накопителях, пожалуйста, обратитесь к Liu et al.

## MobileNets: эффективные (глубокие) нейронные сети



(Слева) стандартный сверточный слой с пакетной нормализацией и ReLU. ((Право) Глубинная сепарабельная свертка с глубинными и точечными слоями с последующей пакетной нормализацией и ReLU (рисунок и подпись от Liu et al.).

При построении сетей обнаружения объектов мы обычно используем существующую сетевую архитектуру, такую как VGG или ResNet, а затем используем ее внутри конвейера обнаружения объектов. Проблема в том, что эти сетевые архитектуры могут быть очень большими-порядка 200-500 МБ.

Сетевые архитектуры, подобные этим, непригодны для устройств с ограниченными ресурсами из-за их огромного размера и результирующего количества вычислений.

Вместо этого мы можем использовать мобильные сети (Howard et al., 2017), еще одну статью исследователей Google. Мы называем эти сети “мобильными сетями”, потому что они предназначены для устройств с ограниченными ресурсами, таких как ваш смартфон. Мобильные сети отличаются от традиционных CNN использованием разделяемой по глубине свертки (Рис. 2 выше).

Общая идея разделяемой свертки по глубине состоит в том, чтобы разделить свертку на два этапа:

1. Глубинная свертка 3×3.
2. Затем следует точечная свертка 1×1.

Это позволяет нам фактически уменьшить количество параметров в нашей сети.

Проблема в том, что мы жертвуем точностью — мобильные сети обычно не так точны, как их более крупные старшие братья…

...но они гораздо более ресурсоэффективны.

## Сочетание MobileNets и Single Shot Detectors для быстрого и эффективного обнаружения объектов на основе глубокого обучения

Если мы объединим как мобильную сетевую архитектуру, так и фреймворк Single Shot Detector (SSD), то получим быстрый и эффективный метод обнаружения объектов на основе глубокого обучения.

The model we’ll be using in this blog post is a Caffe version of the [original TensorFlow implementation](https://github.com/Zehaos/MobileNet) by Howard et al. and was trained by chuanqi305 ([see GitHub](https://github.com/chuanqi305/MobileNet-SSD)).

SSDs MobileNet сначала был обучен набору данных COCO ([COCO dataset](http://cocodataset.org/) ), а затем был точно настроен на PASCAL VOC, достигнув 72,7% mAP (средняя средняя точность).

Таким образом, мы можем обнаружить 20 объектов на изображениях (+1 для фонового класса), включая самолеты, велосипеды, птиц, лодки, бутылки, автобусы, автомобили, кошек, стулья, коров, обеденные столы, собак, лошадей, мотоциклы, людей, растения в горшках, овец, диваны, поезда и телевизионные мониторы.

## Глубокое изучение на основе распознавания объектов с использованием OpenCV

В этом разделе мы будем использовать модуль Mobile Net SSD + deep neural network (dnn ) в OpenCV для построения нашего детектора объектов.

Я бы предложил использовать код “загрузки” в нижней части этого сообщения в блоге, чтобы загрузить исходный код + обученную сеть + примеры изображений, чтобы вы могли протестировать их на своей машине.

Давайте продолжим и начнем строить наш детектор объектов глубокого обучения с использованием OpenCV.

Откройте новый файл, назовите его deep\_learning\_object\_detection.py , и вставьте следующий код:

# import the necessary packages

import numpy as np

import argparse

import cv2

# construct the argument parse and parse the arguments

ap = argparse.ArgumentParser()

ap.add\_argument("-i", "--image", required=True,

help="path to input image")

ap.add\_argument("-p", "--prototxt", required=True,

help="path to Caffe 'deploy' prototxt file")

ap.add\_argument("-m", "--model", required=True,

help="path to Caffe pre-trained model")

ap.add\_argument("-c", "--confidence", type=float, default=0.2,

help="minimum probability to filter weak detections")

args = vars(ap.parse\_args())

В строках 2-4 мы импортируем пакеты , необходимые для этого скрипта — модуль dnn включен в cv2, опять же, делая предположение hte, что вы используете OpenCV 3.3.

Затем мы анализируем наши аргументы командной строки (строки 7-16):

-- image: путь к входному изображению.

--prototxt : путь к файлу Caffe prototxt.

-- model: путь к предварительно подготовленной модели.

--уверенность : минимальный порог вероятности для фильтрации слабых обнаружений. Значение по умолчанию-20%.

Опять же, примеры файлов для первых трех аргументов включены в раздел “Загрузки” этого сообщения в блоге. Я настоятельно призываю вас начать с этого, а также предоставить некоторые собственные изображения запросов.

Далее, Давайте инициализируем метки классов и цвета ограничительных рамок:

# initialize the list of class labels MobileNet SSD was trained to

# detect, then generate a set of bounding box colors for each class

CLASSES = ["background", "aeroplane", "bicycle", "bird", "boat",

"bottle", "bus", "car", "cat", "chair", "cow", "diningtable",

"dog", "horse", "motorbike", "person", "pottedplant", "sheep",

"sofa", "train", "tvmonitor"]

COLORS = np.random.uniform(0, 255, size=(len(CLASSES), 3))

В строках 20-23 строится список классов, содержащий наши метки. Далее следует список цветов, содержащий соответствующие случайные цвета для ограничивающих прямоугольников (строка 24).

Теперь нам нужно загрузить нашу модель:

# load our serialized model from disk

print("[INFO] loading model...")

net = cv2.dnn.readNetFromCaffe(args["prototxt"], args["model"])

Приведенные выше строки не требуют объяснений, мы просто печатаем сообщение и загружаем нашу модель (строки 27 и 28).

Затем мы загрузим изображение вашего запроса и подготовим наш blob-объект , который мы будем передавать через сеть:

# load the input image and construct an input blob for the image

# by resizing to a fixed 300x300 pixels and then normalizing it

# (note: normalization is done via the authors of the MobileNet SSD

# implementation)

image = cv2.imread(args["image"])

(h, w) = image.shape[:2]

blob = cv2.dnn.blobFromImage(cv2.resize(image, (300, 300)), 0.007843,

(300, 300), 127.5)

Принимая к сведению комментарий в этом блоке, мы загружаем наше изображение (строка 34), извлекаем высоту и ширину (строка 35) и вычисляем большой двоичный объект размером 300 на 300 пикселей из нашего изображения (строка 36).

Теперь мы готовы сделать тяжелую работу — мы пропустим эту каплю через нейронную сеть:

# pass the blob through the network and obtain the detections and

# predictions

print("[INFO] computing object detections...")

net.setInput(blob)

detections = net.forward()

В строках 41 и 42 мы устанавливаем вход в сеть и вычисляем прямой проход для входа, сохраняя результат в виде обнаружений. Вычисление прямого прохода и связанных с ним обнаружений может занять некоторое время в зависимости от вашей модели и размера входных данных, но в данном примере это будет относительно быстро на большинстве процессоров.

Давайте пройдемся по нашим обнаружениям и определим, что и где находятся объекты на изображении:

# loop over the detections

for i in np.arange(0, detections.shape[2]):

# extract the confidence (i.e., probability) associated with the

# prediction

confidence = detections[0, 0, i, 2]

# filter out weak detections by ensuring the `confidence` is

# greater than the minimum confidence

if confidence > args["confidence"]:

# extract the index of the class label from the `detections`,

# then compute the (x, y)-coordinates of the bounding box for

# the object

idx = int(detections[0, 0, i, 1])

box = detections[0, 0, i, 3:7] \* np.array([w, h, w, h])

(startX, startY, endX, endY) = box.astype("int")

# display the prediction

label = "{}: {:.2f}%".format(CLASSES[idx], confidence \* 100)

print("[INFO] {}".format(label))

cv2.rectangle(image, (startX, startY), (endX, endY),

COLORS[idx], 2)

y = startY - 15 if startY - 15 > 15 else startY + 15

cv2.putText(image, label, (startX, y),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, COLORS[idx], 2)

Мы начинаем с циклического перебора обнаруженных объектов, имея в виду, что на одном изображении можно обнаружить несколько объектов. Мы также применяем проверку достоверности (т. е. вероятности), связанной с каждым обнаружением. Если достоверность достаточно высока (то есть выше порога), то мы отобразим предсказание в терминале, а также нарисуем предсказание на изображении с текстом и цветной ограничительной рамкой. Давайте разберем его по строчкам:

Циклически просматривая наши обнаружения , сначала мы извлекаем значение уверенности (строка 48).

Если достоверность превышает наш минимальный порог (строка 52), мы извлекаем индекс метки класса (строка 56) и вычисляем ограничивающую рамку вокруг обнаруженного объекта (строка 57).

Затем мы извлекаем (x, y)-координаты прямоугольника (строка 58), который вскоре будем использовать для рисования прямоугольника и отображения текста.

Затем мы создаем текстовую метку, содержащую имя класса и уверенность (строка 61).

Используя метку, мы печатаем ее на терминале (строка 62), а затем рисуем цветной прямоугольник вокруг объекта, используя наши ранее извлеченные (x, y)-координаты (строки 63 и 64).

В общем, мы хотим, чтобы надпись отображалась над прямоугольником, но если там нет места, мы будем отображать ее чуть ниже верхней части прямоугольника (строка 65).

Наконец, мы накладываем цветной текст на изображение, используя только что вычисленное значение y (строки 66 и 67).

# show the output image

cv2.imshow("Output", image)

cv2.waitKey(0)

….

В сегодняшнем блоге мы узнали, как выполнять обнаружение объектов с помощью глубокого обучения и OpenCV.

В частности, мы использовали оба детектора MobileNets + Single Shot вместе с совершенно новым (полностью переработанным) модулем dnn OpenCV 3.3 для обнаружения объектов на изображениях.

Как сообщество компьютерного зрения и глубокого обучения, мы во многом обязаны вкладу Александра Рыбникова, главного автора модуля dnn для обеспечения доступности глубокого обучения из библиотеки OpenCV. Вы можете найти оригинальный пример скрипта OpenCV Александра здесь — я изменил его для целей этого поста в блоге.

В следующем посте в блоге я продемонстрирую, как мы можем изменить сегодняшний учебник для работы с видеопотоками в реальном времени, что позволит нам выполнять обнаружение объектов на основе глубокого обучения в видео. Мы обязательно будем использовать эффективный кадровый ввод-вывод, чтобы увеличить FPS по всему нашему конвейеру.

Вопросы к лабораторной работе

1. привести подробное описание основных функций, рассмотренных в работе - функция, что делает, аргументы, параметры, возвращаемые результаты, типы данных и тп.
2. выполнить примеры, описанные в лабораторной - предоставить комментированный код, поэтапные результаты выполнения действий (исходные фото-, видео- данные для примеров берутся собственные, или из открытых источников или из базового набора библиотеки OpenCV)