ディープラーニング入門
【画像認識（物体認識とセグメンテーション）】
デモ向け補足資料

ver.1

2019年3月3日

次世代人材開発研究所

目次

==========

【１】Mask RCNN の実装フレームワーク

==========

・matterport(Keras で実装)版：学習、認識
https://github.com/matterport/Mask_RCNN
・Opencv 版：認識のみ
https://github.com/opencv/opencv/blob/master/samples/dnn/mask_rcnn.py
・Pytorch 版：学習、認識
https://github.com/multimodallearning/pytorch-mask-rcnn

※Opencv 版は、まだ Nvidia GPU に未対応（間もなくサポート）。
ただし、インテル GPU および CPU、FPGA 向けの最適化ツール（OpenVINO）をサポート。

==========

【２】物体認識

==========

２．１．Opencv よる静止画の物体検出

python3 mask_rcnn.py --mask-rcnn mask-rcnn-coco --image images/$1.jpg

--mask-rcnn：モデルおよび重みファイルの格納ディレクトリ名
--image　　：画像ファイル名

モデルディレクトリ内のファイル：
colors.txt　：物体のカラーファイル
frozen_inference_graph.pb　：重みファイル
mask_rcnn_inception_v2_coco_2018_01_28.pbtxt　：Mask RCNN のモデルファイル
object_detection_classes_coco.txt　：coco のラベルクラスファイル

２．２．Opencv による動画の物体検出

```
python3 mask_rcnn_video.py --input videos/$1.mp4 ￥
                           --output output/$1.avi ￥
                           --mask-rcnn mask-rcnn-coco
```

--input videos/$1.mp4　　：入力動画ファイル(MP4 形式)
--output output/$1.avi　　：物体検出後の動画ファイル(AVI 形式)
--mask-rcnn mask-rcnn-coco　：モデルおよび重みファイルの格納ディレクトリ名

２．３．matterport Mask-RCNN による静止画の物体検出

環境変数の PYTHONPATH の sys.path.append("./samples/coco")

python demo.py image

重みファイル　　：mask_rcnn_coco.h5
入力画像ディレクトリ：images
出力画像ディレクトリ：output/image

２．４．matterport Mask-RCNN による動画の物体検出

python demo.py image

重みファイル　　：mask_rcnn_coco.h5
入力画像ディレクトリ：images
出力画像ディレクトリ：output/image

==========

【３】背景ぼかし

==========

python video_conference.py --mask-rcnn mask-rcnn-coco --kernel 41

--mask-rcnn ：モデルファイルディレクトリ（mask-rcnn-coco）
--kernel ：ぼかし度合（デフォルト、41）

==========

【４】風船の学習と検出

==========

４．１．学習

転移学習
python3 balloon.py train --dataset=~/dataset/balloon --weights=coco

継続学習
python3 balloon.py train --dataset=/path/to/balloon/dataset --weights=last

４．２．認識

./exec_balloon.sh <image|video> <fileName>

４．３．Mask RCNN のディレクトリ構成

Mask_RCNN
|
├── datasets（学習用データセット））
| └── balloon（風船学習画像））
| ├── train（学習用）
| | ├── 10464445726_6f1e3bbe6a_k.jpg

```
..
│           │      ├── 9330497995_4cf0438cb6_k.jpg
│           │      └── via_region_data.json（学習用アノテーションファイル）
│           └── val（検証用）
│               ├── 14898532020_ba6199dd22_k.jpg
..
│               ├── 8053085540_a72bd21a64_k.jpg
│               └── via_region_data.json（検証アノテーションファイル）
├── demo.py
├── images（デモ静止画ディレクトリ）
│   ├── デモ画像
..
│
├── mask_rcnn_balloon.h5（風船学習済み重み）
├── mask_rcnn_coco.h5（coco 学習済み重み）
├── mrcnn（Mask R-CNN 本体）
│   ├── __init__.py
│   ├── config.py
│   ├── isz_visualize.py
│   ├── model.py
│   ├── parallel_model.py
│   ├── utils.py
│   └── visualize.py
├── output
│   ├── デモの結果ディレクトリ
..
├── samples（サンプル）
│   │   ____________________________________________
│   ├── ******  balloon（風船モデルディレクトリ）*****
│   │   │ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
│   │   ├── README.md
│   │   ├── balloon.py
│   │   ├── inspect_balloon_data.ipynb
│   │   ├── inspect_balloon_model.ipynb
│   │   └── train_balloon.sh（学習認識スクリプト）
│   ├── coco
```

```
│   │   ├── coco.py
│   │   ├── inspect_data.ipynb
│   │   ├── inspect_model.ipynb
│   │   └── inspect_weights.ipynb
│   ├── demo.ipynb
│   ├── demo_v1.py
│   ├── exec_balloon.sh（風船認識実行スクリプト）
│   ├── nucleus
│   │   ├── README.md
│   │   ├── inspect_nucleus_data.ipynb
│   │   ├── inspect_nucleus_model.ipynb
│   │   └── nucleus.py
│   └── shapes
│       ├── shapes.py
│       └── train_shapes.ipynb
├── setenv.sh
├── setup.cfg
├── setup.py
└── videos（デモ動画ファイル）
    ├── example_mv1.mp4
    ├── example_mv2.mp4
    └── test_balloon1.mp4
```

==========

【５】matterport Mask_RCNN の keras モデルを Opencv で使用可能する

==========

５．１．Keras モデル（.h5）を tensorflow（.pb）に変換

tensorflow の saved_model を使うことで、簡単に変換できます。

```
import tensorflow as tf
from tensorflow.python.keras.models import load_model

####### keras モデルファイル名
```

```
input_keras_model = './conv_mnist.h5'

####### tensorflow モデルに変換した結果格納ディレクトリ
export_dir = './conv_mnist_pb'

if __name__ == '__main__':
    old_session = tf.keras.backend.get_session()
    sess = tf.Session()
    sess.run(tf.global_variables_initializer())
    tf.keras.backend.set_session(sess)
    model = load_model(input_keras_model)
    builder = tf.saved_model.builder.SavedModelBuilder(export_dir)
    signature = tf.saved_model.predict_signature_def(inputs={t.name: ¥
                                        t for t in model.inputs}, ¥
                                        outputs={t.name:t for t in model.outputs})
    builder.add_meta_graph_and_variables(sess, ¥
                                        tags=[tf.saved_model.tag_constants.SERVING], ¥
                                         signature_def_map={'predict': signature})
    builder.save(as_text=True)
    sess.close()
    tf.keras.backend.set_session(old_session)

    print('output_node_names:')
    for t in model.inputs:
        print(t.name)

    print('output_node_names:')
    for t in model.outputs:
        print(t.name)
```

５．２．パラメータもフリーズされた pb ファイル(frozen graph)を生成

```
freeze_graph ¥
--input_saved_model_dir=./conv_mnist_pb ¥
--output_graph=conv_mnist.pb ¥
--output_node_names=dense_1/Softmax ¥
--clear_devices
```

５．３．OpenCV で使用必要な tensorflow のファイル

OpenCV の DNN モジュールが Tensorflow net importer として追加されました。
DNN を使用するには opencv_contrib が必要です。必ずインストールしておいてください。

Python での関数の呼び出し形式は次のとおりです。

```
cv2.dnn.readNetFromTensorflow('frozen_inference_graph.pb', 'graph.pbtxt')
```

ご覧のとおり、それを使用するには、2 つのファイルが必要です。

- frozen_inference_graph.pb
- graph.pbtxt

５．４．Tensorflow モデルの .pb と .pbtxt ファイルについて

Tensorflow モデルは通常かなり多数のパラメータを持っています。 フリーズは、必要なもの（グラフ、重みなど）だけを識別して後で使用できる単一のファイルに保存するプロセスです。つまり、モデルを“エクスポート”するための TF の方法です。
フリーズプロセスにより、Protobuf（.pb）ファイルが作成されます。

良いニュースは次のとおりです。Tensorflow の検出モデル zoo リポジトリには、訓練され最適化され広く使用されている多数のモデルがあり、自由に使用できます。
あなたは、それを訓練する必要はありません（よく知られたデータセットで訓練された利用可能なモデルがあなたのニーズに合っているなら）。

さらに、OpenCV は.pb、.pbtxt に基づく追加の設定ファイルを必要とします。
OpenCV Github リポジトリから以下のファイルのいずれかを使用して、独自のモデルをインポートし、独自の.pbtxt ファイルを生成することが可能です。

tf_text_graph_ssd.py
tf_text_graph_common.py
tf_text_graph_faster_rcnn.py
tf_text_graph_mask_rcnn.py　　####### このスクリプトを使用する。

python tf_text_graph_mask_rcnn.py ¥
　　--input <Path to frozen TensorFlow graph>, ¥
　　--output <Path to output text graph>, ¥
　　--config <Path to a *.config file is used for training>

学習済みモデルを使用したいのであれば、に OpenCV コミュニティは既に以下のモデルを用意しています。

Model	Version	
MobileNet-SSD v1	2017_11_17	weights config
MobileNet-SSD v1 PPN	2018_07_03	weights config
MobileNet-SSD v2	2018_03_29	weights config
Inception-SSD v2	2017_11_17	weights config
Faster-RCNN Inception v2	2018_01_28	weights config
Faster-RCNN ResNet-50	2018_01_28	weights config
Mask-RCNN Inception v2	2018_01_28	weights config

５．５．変換した Tensorflow モデルを Opencv で使用する Python コード

インポート/使用手順全体は、5 つのステップに分けられます。

・ダウンロードしたファイルを使ってモデルをロードします。
・画像を読み込みます。
・これらの画像をネットワーク入力として使用します。
・検出されたオブジェクトとともに出力を取得します。

以下の手順を実行するための以下の Python コードを見てください。

```
# How to load a Tensorflow model using OpenCV
# Jean Vitor de Paulo Blog -
#　　https://jeanvitor.com/tensorflow-object-detecion-opencv/
```

```
import cv2

# Load a model imported from Tensorflow
tensorflowNet = cv2.dnn.readNetFromTensorflow('frozen_inference_graph.pb', ¥
    'graph.pbtxt')

# Input image
img = cv2.imread('img.jpg')
rows, cols, channels = img.shape

# Use the given image as input, which needs to be blob(s).
tensorflowNet.setInput(cv2.dnn.blobFromImage(img, size=(300, 300), ¥
    swapRB=True, crop=False))

# Runs a forward pass to compute the net output
networkOutput = tensorflowNet.forward()

# Loop on the outputs
for detection in networkOutput[0,0]:

    score = float(detection[2])
    if score > 0.2:

        left = detection[3] * cols
        top = detection[4] * rows
        right = detection[5] * cols
        bottom = detection[6] * rows

        #draw a red rectangle around detected objects
        cv2.rectangle(img, (int(left), int(top)), (int(right), int(bottom)), ¥
            (0, 0, 255), thickness=2)

# Show the image with a rectagle surrounding the detected objects
cv2.imshow('Image', img)
cv2.waitKey()
cv2.destroyAllWindows()
```

==========

【６】チュートリアル

==========

６．１．OpenCV によう Mask RCNN

この資料は、以下を参照されたい。
https://www.pyimagesearch.com/2018/11/19/mask-r-cnn-with-opencv/
このチュートリアルでは、OpenCV で Mask R-CNN を使う方法を学びます。
Mask R-CNN を使用すると画像内のすべてのオブジェクトに対してピクセル単位のマスクを自動的にセグメント化して構築できます。Mask R-CNN を画像とビデオストリームの両方に適用します。

YOLO オブジェクト検出器を使用して画像内のオブジェクトの存在を検出する方法を学びました。YOLO、Faster R-CNN、Single Shot Detectors(SSD)などのオブジェクト検出器は、画像内のオブジェクトのバウンディングボックスを表す 4 セットの(x,y)座標を生成します。

オブジェクトのバウンディングボックスを取得するのは良いスタートですが、(1)どのピクセルが前景のオブジェクトに属し、どのピクセルが背景に属しているのかについては、バウンディングボックス自体からは何もわかりません。

ここでは、OpenCV で Mask R-CNN を画像とビデオストリームの両方に適用する方法を学ます。

Mask R-CNN with OpenCV

このチュートリアルの最初の部分では、画像分類、オブジェクト検出、インスタンスセグメンテーション、およびセマンティックセグメンテーションの違いについて説明します。

そこから Mask R-CNN のアーキテクチャと Faster R-CNN への接続について簡単に説明します。

それでは、OpenCV で Mask R-CNN を画像とビデオストリームの両方に適用する方法を説明します。始めましょう！

Instance segmentation vs. Semantic segmentation

図 1:画像分類(左上)、オブジェクト検出(右上)、セマンティックセグメンテーション(左下)、およびインスタンスセグメンテーション(右下)。このチュートリアルでは Mask R-CNN を使用してインスタンスセグメンテーションを実行します。

従来の画像分類、オブジェクト検出、セマンティックセグメンテーション、およびインスタンスセグメンテーションの違いの説明は、視覚的に行うのが最善です。

従来の画像分類を実行する際の目標は、入力画像の内容を特徴付けるためにラベルのセットを予測することです(左上)。

オブジェクト検出は画像分類に基づいていますが、今回は画像内の各オブジェクトをローカライズすることができます。画像の特徴は次のとおりです。

1.各オブジェクトのバウンディングボックス(x,y)座標
2.各バウンディングボックスの関連クラスラベル

セマンティックセグメンテーションの例は、左下に見ることができます。セマンティックセグメンテーションアルゴリズムでは、入力画像内のすべてのピクセルをクラスラベル(背景のクラスラベルを含む)に関連付ける必要があります。

セマンティックセグメンテーションの視覚化に細心の注意を払ってください - 各オブジェクトは実際にセグメント化されていますが、各「キューブ」オブジェクトは同じ色を持っています。

セマンティックセグメンテーションアルゴリズムは、画像内のすべてのオブジェクトにラベルを付けることができますが、同じクラスの 2 つのオブジェクトを区別することはできません。

同じクラスの 2 つのオブジェクトが互いに部分的に隠れている場合、この動作は特に問題になります。
2 つの紫色の立方体で示されるように、1 つのオブジェクトの境界がどこで終わり、次のオブジェクトの境界がどこで始まるかはわかりません。

一方、インスタンスセグメンテーションアルゴリズムは、オブジェクトが同じクラスラベ

ル(右下)にある場合でも、画像内のすべてのオブジェクトに対してピクセル単位のマスクを計算します。ここでは、それぞれの立方体がそれぞれ独自の色を持っていることがわかります。これは、このインスタンスセグメンテーションアルゴリズムが各立方体をローカライズするだけでなく、それらの境界も予測したことを意味します。

このチュートリアルで説明する Mask R-CNN アーキテクチャは、インスタンスセグメンテーションアルゴリズムの一例です。

What is Mask R-CNN?

Mask R-CNN アルゴリズムは、He らによって導入されました。彼らの 2017 年の論文では、Mask R-CNN。
Mask R-CNN は、Girshick らによる R-CNN(2013)、Fast R-CNN(2015)、および Faster R-CNN(2015)の以前のオブジェクト検出作業に基づいています。

Mask R-CNN を理解するために、オリジナルの R-CNN から始めて、R-CNN の変種を簡単に見てみましょう。

図 2:オリジナルの R-CNN アーキテクチャ

オリジナルな R-CNN アルゴリズムは以下の 4 ステップの処理から成る。

- Step#1：画像をネットワークに入力する。
- Step#2：セレクティブサーチのようなアルゴリズムを用いてリージョンプロポーザル(すなわち、オブジェクトを潜在的に含む画像のリージョン)を抽出する。
- Step#3：各プロポーザルのための特徴を計算するために学習済み CNN を用いて、転移学習を使用する。
- Step#4：SVM で抽出した特徴を用いて各プロポーザルを分類する。

この手法を使用する理由は、CNN によって学習された判別可能な特徴が安定的であるためである。

しかしながら、R-CNN を使用する問題は非常に遅いことである。さらに、ディープニューラルネットワークを通じてローカライズするために実際的に学習はしない。
より効果的で先進的な HOG + 線形 SVM 検出器を構築する。
オリジナルな R-CNN を改善するために、Girshick らにより Fast R-CNN が提案された。

図 3：Fast R-CNN のアーキテクチャー

オリジナルな R-CNN と同様、Fast R-CNN もリージョンプロポーザルを抽出するために、まだ、Selective Search を使用している。しかしながら、この論文での最新技術は、ROI(Region of Interest)プーリングモジュールである。

ROI プーリングは、最終的なクラスラベルとバウンディングボックスを得るためにこれらの特徴マップから、そしてこれらの特徴を用いて固定サイズのウィンドウを抽出することで行われる。ここでの主な利点は、ネットワークが事実上、エンドツーエンドでトレーニング可能であるということです。

1．画像と想定したグランドトゥルースバウンディングボックスを入力する
2．特徴マップを抽出する。
3．ROI プーリングを適用し、ROI 特徴マップを得る。
4．そして、最後に予測クラスラベルと各プロポーザルのバウンディングボックスの位置を得るために、２つの全結合層を使用する。

ネットワークは end-to-end で学習を可能であるが、推論のパフォーマンスは Selective Search に依存しているので、低下する。

R-CNN をさらに高速にするためにリージョンプロポーザルを R-CNN に直接組ことが必要である。

図 4：Faster R-CNN のアーキテクチャー

Girshick らの Faster R-CNN の論文は、Selective Search アルゴリズムの要求を緩和するために、リージョンプロポーザルをアーキテクチャーに直接組み込む Region Proposal Network(RPN)を導入した。

全体として、Faster R-CNN アーキテクチャーはおよそ 7〜10FPS の実行可能であり、これは、現実的にディープラーニングを用いてリアルタイムの物体検出を行うための大きな一歩である。
Mask R-CNN アルゴリズムは、２つの技術を用いて、Faster R-CNN アーキテクチャ上に構築する。

1．ROI プーリングモジュールをより精度の高い ROI Align モジュールに置き換える。
2．ROI Align モジュールの出力からの付加的な分岐を挿入する。

この付加的な分岐は、ROI Align の出力を受け入れ、その後、それを CONV レイヤの送る。

CONV レイヤの出力は、それ自身マスクである。

以下に Mask R-CNN アーキテクチャーを図示する。

図 5 ：He 等によるマスク R-CNN 処理では ROI Polling モジュールをより高精度な ROI Align モジュールに置き換え、ROI モジュールの出力は 2 つの CONV レイヤに送られる。その CONV レイヤーの出力はマスクそのものです。

ROI Align モジュールから出てくる 2 つの CONV レイヤーの分岐に注目してください
- これがマスクが実際に生成される場所である。

既に知っているように、Faster R-CNN/Mask R-CNN アーキテクチャは、リージョンプロポーザルネットワーク（RPN）を利用して、潜在的にオブジェクトを含む画像の領域を生成します。

これらの領域のそれぞれは、それらの「客観性スコア」(すなわち、得られた領域が潜在的にオブジェクトを含み得る可能性がどれくらいありそうか)に基づいてランク付けされ、
次いで上位N個の最も信頼できる客観性領域が保持される。

He らの論文では N=300 を設定しており、それはここでも使用している値である。

300 の選択された ROI のそれぞれは、ネットワークの 3 つの並列ブランチを通過します。

1．ラベルの予測
2．バウンディングボックスの予測
3．マスクの予測

図 5 にこれらの 3 つの分岐を図示化している。
予測中、300 個の ROI のそれぞれが non-maxima の抑制を受け、上位 100 個の検出ボックスが保持され、100xLx15x15 の 4D テンソルが得られます。ここで、L はデータセット内のクラスラベルの数で、15x15 は L 個のマスクの各々のサイズである。

ここで使用している Mask R-CNN は、L=90 クラスの COCO データセットでトレーニングされたため、Mask R-CNN のマスクモジュールから得られるサイズは 100x90x15x15 です。

Mask R-CNN の処理を視覚化すると以下の図のようになる。

図 6：Mask R-CNN を視覚化して 15x15 のマスクを作成し、マスクを画像の元の寸法にサイズに変更し、最後に元の画像にマスクを重ねた。
（source: Deep Learning for Computer Vision with Python, ImageNet Bundle）

ここでは入力画像から始めてマスク予測を得るために Mask R-CNN ネットワークを通してそれを供給するのを見ることができます。

予測されたマスクはたった 15x15 ピクセルなので、マスクのサイズを元の入力画像のサイズに戻します。

最後に、サイズ変更したマスクを元の入力画像の上に重ねることができます。Mask R-CNN がどのように機能するかについてのより徹底的な議論については、以下を参照されたい：

1．He らによるオリジナルの Mask R-CNN の論文。
2．出版物の「Deep Learning for Computer Vision with Python」
自分のデータを使って自分の Mask R-CNN を最初から訓練する方法など、Mask R-CNN について詳しく説明している。

Project structure

このプロジェクトは、2 つのスクリプトで構成されていますが、他にも重要なファイルがいくつかあります。

プロジェクトを次の方法で整理しました（tree コマンド出力に直接表示される）。

```
$ tree
.
├── mask-rcnn-coco
│   ├── colors.txt
│   ├── frozen_inference_graph.pb
│   ├── mask_rcnn_inception_v2_coco_2018_01_28.pbtxt
```

```
│       └── object_detection_classes_coco.txt
├── images
│       ├── example_01.jpg
│       ├── example_02.jpg
│       └── example_03.jpg
├── videos
│       ├──
├── output
│       ├──
├── mask_rcnn.py
└── mask_rcnn_video.py

4 directories, 9 files
```

このプロジェクトは、4つのディレクトリから構成される：

・mask-rcnn-coco/：Mask R-CNN モデルファイルで以下の4つのファイルがある：
  ・frozen_inference_graph.pb：
    Mask R-CNN モデルの重み。この重みは COCO データセットで学習済みである。
  ・mask_rcnn_inception_v2_coco_2018_01_28.pbtxt：
    Mask R-CNN モデルのコンフィグレーション。
    自分のアノテーションデータで学習したいなら、以下を参照されたい：
    Deep Learning for Computer Vision with Python
  ・object_detection_classes_coco.txt：
    90 クラス全てがこのテキストファイルにリストされている(一行一クラス)。
    モデルを認識できるオブジェクトを確認するにはテキストエディタでそれを開く。
  ・colors.txt：
    このテキストファイルには、画像内で見つかったオブジェクトにランダムに割り当てるための6色が含まれています。
・images/：3つのテスト画像を用意した。テスト用の独自の画像を追加できる。
・videos/：これは空のディレクトリです。実際に YouTube から削り取った大きなビデオでテストしました（クレジットは「概要」セクションのすぐ下にあります)。
  本当に大きな zip ファイルを提供するのではなく、YouTube でダウンロード

してテストするためのビデオをいくつか見つけることをお勧めします。
携帯電話でビデオを撮ってコンピュータに戻って使用することもできます。

・output/：処理されたビデオを保持する別の空のディレクトリ（このディレクトリに出力するようにコマンドライン引数フラグを設定したと仮定します)。

2 つのスクリプトをレビューします。

・mask_rcnn.py：このスクリプトはインスタンスセグメンテーションを実行してイメージにマスクを適用するので、ピクセルまで、R-CNN マスクがオブジェクトであると考える場所を見ることができます。

・mask_rcnn_video.py：このビデオ処理スクリプトは同じ Mask R-CNN を使用し、ビデオファイルのすべてのフレームにモデルを適用します。
その後、スクリプトは出力フレームをディスク上のビデオファイルに書き戻します。

OpenCV and Mask R-CNN in images

Mask R-CNN がどのように機能するのかを見てきたので、Python コードを使って実際にやってみます。

始める前に、Python 環境に OpenCV 3.4.2 / 3.4.3 以降がインストールされていることを確認してください。OpenCV をアップグレード/インストールするには、OpenCV インストールチュートリアルに従うことです。5 分以内に起動して実行したい場合は、pip を使用して OpenCV をインストールすることを検討できます。他にも要件がある場合は、ソースから OpenCV をコンパイルすることをお勧めします。

mask_rcnn.py ファイルを開き、以下のコードを挿入します。

```
# import the necessary packages
import numpy as np
import argparse
import random
import time
import cv2
import os
```

まず、2 行目から 7 行目に必要なパッケージをインポートします。特に NumPy と OpenCV を import しています。他のすべてのものはほとんどの Python インストールに付属しています。

コマンドライン引数を解析します。

```
# 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("-m", "--mask-rcnn", required=True,
	help="base path to mask-rcnn directory")
ap.add_argument("-v", "--visualize", type=int, default=0,
	help="whether or not we are going to visualize each instance")
ap.add_argument("-c", "--confidence", type=float, default=0.5,
	help="minimum probability to filter weak detections")
ap.add_argument("-t", "--threshold", type=float, default=0.3,
	help="minimum threshold for pixel-wise mask segmentation")
args = vars(ap.parse_args())
```

このスクリプトは、コマンドライン引数フラグとパラメータが実行時に端末から渡されることを要求します。以下の最初の 2 つは必須で、残りはオプションです。

- --image : 入力画像へのパス
- --mask-rnn : Mask R-CNN ファイルへ基本パス
- --visualize （オプション): 正の値はマスク領域を画面上で抽出した方法を視覚化したいことを示します。
  どちらの方法でも、最終出力を画面に表示します。
- --confidence （オプション): 弱い検出をフィルタリングするのに役立つ確率値 0.5 を上書きすることができます。
- --threshold （オプション): 画像内の各オブジェクトに対してバイナリマスクを作成します。このしきい値は、弱いマスク予測を除外するのに役立ちます。デフォルト値 0.3 がかなりうまくいくことを発見しました。

コマンドライン引数が args 辞書に格納されたので、ラベルと色をロードしましょう。

```
# load the COCO class labels our Mask R-CNN was trained on
labelsPath = os.path.sep.join([args["mask_rcnn"],
        "object_detection_classes_coco.txt"])
LABELS = open(labelsPath).read().strip().split("¥n")

# load the set of colors that will be used when visualizing a given
# instance segmentation
colorsPath = os.path.sep.join([args["mask_rcnn"], "colors.txt"])
COLORS = open(colorsPath).read().strip().split("¥n")
COLORS = [np.array(c.split(",")).astype("int") for c in COLORS]
COLORS = np.array(COLORS, dtype="uint8")
```

行 24-26 は、COCO オブジェクトクラス LABELS をロードします。Mask R-CNN は、人、車、看板、動物、日用品、スポーツ用品、台所用品、食べ物など、90 のクラスを認識することができます。利用可能なクラスを見るために object_detection_classes_coco.txt を見ることをお勧めします。

パスから COLORS をロードし、いくつかの配列変換操作を実行します（30?33 行目）。

モデルをロードしましょう。

```
# derive the paths to the Mask R-CNN weights and model configuration
weightsPath = os.path.sep.join([args["mask_rcnn"],
        "frozen_inference_graph.pb"])
configPath = os.path.sep.join([args["mask_rcnn"],
        "mask_rcnn_inception_v2_coco_2018_01_28.pbtxt"])

# load our Mask R-CNN trained on the COCO dataset (90 classes)
# from disk
print("[INFO] loading Mask R-CNN from disk...")
net = cv2.dnn.readNetFromTensorflow(weightsPath, configPath)
```

まず、重みパスとコンフィグレーションパスを作成し(36-39 行目)、続いてこれらのパスを介してモデルを読み込みます(44 行目)。

次のブロックでは、Mask R-CNN ニューラルネットに画像をロードして渡します。

```
# load our input image and grab its spatial dimensions
image = cv2.imread(args["image"])
(H, W) = image.shape[:2]

# construct a blob from the input image and then perform a forward
# pass of the Mask R-CNN, giving us (1) the bounding box  coordinates
# of the objects in the image along with (2) the pixel-wise segmentation
# for each specific object
blob = cv2.dnn.blobFromImage(image, swapRB=True, crop=False)
net.setInput(blob)
start = time.time()
(boxes, masks) = net.forward(["detection_out_final", "detection_masks"])
end = time.time()

# show timing information and volume information on Mask R-CNN
print("[INFO] Mask R-CNN took {:.6f} seconds".format(end - start))
print("[INFO] boxes shape: {}".format(boxes.shape))
print("[INFO] masks shape: {}".format(masks.shape))
```

・入力画像をロードし、後で拡大縮小するために寸法を抽出します（47、48 行目）。
・cv2.dnn.blobFromImage（54 行目）で BLOB を作成します。 この機能を使用する理由と使用方法は、前回のチュートリアルで学ぶことができます。
・タイムスタンプを収集しながら、ネットを介してブロブのフォワードパスを実行します（55-58 行目）。 結果は、2 つの重要な変数、ボックスとマスクに含まれています。

画像上で Mask R-CNN の順方向パスを実行したので、結果をフィルタリングして視覚化します。それがまさに次の for ループが達成することです。かなり長いので、ここから始めて 5 つのコードブロックに分割しました。

```
# loop over the number of detected objects
for i in range(0, boxes.shape[2]):
        # extract the class ID of the detection along with the confidence
        # (i.e., probability) associated with the prediction
        classID = int(boxes[0, 0, i, 1])
```

```
        confidence = boxes[0, 0, i, 2]

        # filter out weak predictions by ensuring the detected probability
        # is greater than the minimum probability
        if confidence > args["confidence"]:
                # clone our original image so we can draw on it
                clone = image.copy()

                # scale the bounding box coordinates back relative to the
                # size of the image and then compute the width and the height
                # of the bounding box
                box = boxes[0, 0, i, 3:7] * np.array([W, H, W, H])
                (startX, startY, endX, endY) = box.astype("int")
                boxW = endX - startX
                boxH = endY - startY
```

このブロックでは、フィルタ/視覚化ループを始めます(66 行目)。

検出された特定のオブジェクトの classID と確信度を抽出します(69、70 行目)。

そこから、確信度をコマンドライン引数の確信度値と比較し、確実に超過するようにして、弱い予測を除外します(74 行目)。

その場合は、先に進んで画像のクローンを作成します(76 行目)。この画像は後で必要になります。

次に、オブジェクトのバウンディングボックスを拡大縮小し、ボックスの寸法を計算します(81-84 行目)。

画像セグメンテーションでは、オブジェクトが存在するすべてのピクセルを見つける必要があります。そのため、オブジェクトの上に透明なオーバーレイを配置して、アルゴリズムのパフォーマンスを確認します。そのためには、マスクを計算します。

```
                # extract the pixel-wise segmentation for the object, resize
                # the mask such that it's the same dimensions of the bounding
                # box, and then finally threshold to create a *binary* mask
```

```
mask = masks[i, classID]
mask = cv2.resize(mask, (boxW, boxH),
        interpolation=cv2.INTER_NEAREST)
mask = (mask > args["threshold"])

# extract the ROI of the image
roi = clone[startY:endY, startX:endX]
```

89 行目から 91 行目では、オブジェクトのピクセル単位のセグメンテーションを抽出し、それを元の画像サイズにサイズ変更しています。最後に、マスクがバイナリ配列/イメージになるようにマスクを設定します(92 行目)。

オブジェクトが存在する関心領域も抽出します(95 行目)。

マスクと ROI の両方は、後の図 8 で視覚的に見ることができます。

便宜上、次のブロックでは、--visualize フラグがコマンドライン引数で設定されている場合、マスク、roi、およびセグメント化されたインスタンスの視覚化を行います。

```
# check to see if are going to visualize how to extract the
# masked region itself
if args["visualize"] > 0:
        # convert the mask from a boolean to an integer mask with
        # to values: 0 or 255, then apply the mask
        visMask = (mask * 255).astype("uint8")
        instance = cv2.bitwise_and(roi, roi, mask=visMask)

        # show the extracted ROI, the mask, along with the
        # segmented instance
        cv2.imshow("ROI", roi)
        cv2.imshow("Mask", visMask)
        cv2.imshow("Segmented", instance)
```

このブロックでは、

・ROI、マスク、およびセグメント化されたインスタンスを視覚化する必要があるかどうかを確認します（99 行目)。

・マスクをブール値から整数値に変換します。ここで、値「0」は背景を示し、「255」は前景を示します（102 行目）。
・インスタンス自体だけを視覚化するためにビットごとのマスキングを実行します（103 行目）。
・3 つの画像をすべて表示します（行 107-109）。

繰り返しますが、これらの視覚化画像は、--visualize フラグがオプションのコマンドライン引数で設定されている場合にのみ表示されます(デフォルトでは表示されません)。

それでは、引き続き視覚化を続けましょう。

```
# now, extract *only* the masked region of the ROI by passing
# in the boolean mask array as our slice condition
roi = roi[mask]

# randomly select a color that will be used to visualize this
# particular instance segmentation then create a transparent
# overlay by blending the randomly selected color with the ROI
color = random.choice(COLORS)
blended = ((0.4 * color) + (0.6 * roi)).astype("uint8")

# store the blended ROI in the original image
clone[startY:endY, startX:endX][mask] = blended
```

行 113 は、ブールマスク配列をスライス条件として渡すことで、ROI のマスク領域のみを抽出します。

次に、6 つの COLORS のいずれかをランダムに選択して、透明オーバーレイをオブジェクトに適用します（行 118）。

続いて、マスクされた領域と ROI をブレンドし(119 行目)、このブレンドした領域をクローンイメージに配置します（122 行目）。

最後に、結果を表示するだけでなく、矩形とテキストクラスラベル+確信度を画像に描画します。

```
# draw the bounding box of the instance on the image
color = [int(c) for c in color]
cv2.rectangle(clone, (startX, startY), (endX, endY), color, 2)

# draw the predicted label and associated probability of the
# instance segmentation on the image
text = "{}: {:.4f}".format(LABELS[classID], confidence)
cv2.putText(clone, text, (startX, startY - 5),
        cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)

# show the output image
cv2.imshow("Output", clone)
cv2.waitKey(0)
```

クローズするには、

・オブジェクトの周囲に色付きのバウンディングボックスを描画します(125、126 行目)。
・クラスラベルと確信度のテキストを作成し、バウンディングボックスの上にテキストを描画します（130-132 行目)。
・いずれかのキーが押されるまでイメージを表示します（135 行目と 136 行目)。

Mask R-CNN コードを試してみましょう。

端末を開いて次のコマンドを実行してください。

```
$ python mask_rcnn.py --mask-rcnn mask-rcnn-coco --image images/example_01.jpg
[INFO] loading Mask R-CNN from disk...
[INFO] Mask R-CNN took 0.761193 seconds
[INFO] boxes shape: (1, 1, 3, 7)
[INFO] masks shape: (100, 90, 15, 15)
```

図 7 ：車のシーンに適用されるマスク R-CNN。
Python と OpenCV はマスクを生成するために使用されました

上の画像では、Mask R-CNN が画像内の各自動車をローカライズしているだけでなく、ピクセルごとのマスクも構成しているので、各自動車を画像からセグメント化することがで

きます。

同じコマンドを実行し、今度は--visualize フラグを指定すると、ROI、マスク、インスタンスも視覚化できます。

図８：--visualize フラグを使用すると、Python と OpenCV で構築された Mask R-CNN パイプラインの ROI、マスク、およびセグメンテーションの中間ステップを表示できます。

別の画像例を試してみましょう。

```
$ python mask_rcnn.py --mask-rcnn mask-rcnn-coco --image images/example_02.jpg ¥
        --confidence 0.6
[INFO] loading Mask R-CNN from disk...
[INFO] Mask R-CNN took 0.676008 seconds
[INFO] boxes shape: (1, 1, 8, 7)
[INFO] masks shape: (100, 90, 15, 15)
```

図９：Python と OpenCV を使用して、Mask R-CNN を使用してインスタンスセグメンテーションを実行できます。

Mask R-CNN は、画像から、犬、馬、トラックの両方の人物を正しく検出してセグメント化しました。

動画での Mask R-CNN の使用に進む前の最後の例です。

```
$ python mask_rcnn.py --mask-rcnn mask-rcnn-coco --image images/example_03.jpg
[INFO] loading Mask R-CNN from disk...
[INFO] Mask R-CNN took 0.680739 seconds
[INFO] boxes shape: (1, 1, 3, 7)
[INFO] masks shape: (100, 90, 15, 15)
```

図１０：ここで家族のビーグル犬、Jemma に餌を与えているのを見ることができる。識別された各オブジェクトのピクセル毎のマップはマスクされ、オブジェクト上に透過的に重ねられる。この画像は、訓練済みの Mask R-CNN モデルを使用して

OpenCV と Python で生成されました。

この画像では、ビーグル犬 Jemma の写真を見ることができます。

Mask R-CNN は、男性、Jemma、そして椅子を高い信頼度で検出し、位置特定することができます。

OpenCV and Mask R-CNN in video streams

Mask R-CNN を画像に適用する方法を見てきたので、動画にも適用する方法を検討しましょう。

mask_rcnn_video.py ファイルを開き、以下のコードを挿入します。

```
# import the necessary packages
import numpy as np
import argparse
import imutils
import time
import cv2
import os

# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--input", required=True,
	help="path to input video file")
ap.add_argument("-o", "--output", required=True,
	help="path to output video file")
ap.add_argument("-m", "--mask-rcnn", required=True,
	help="base path to mask-rcnn directory")
ap.add_argument("-c", "--confidence", type=float, default=0.5,
	help="minimum probability to filter weak detections")
ap.add_argument("-t", "--threshold", type=float, default=0.3,
	help="minimum threshold for pixel-wise mask segmentation")
args = vars(ap.parse_args())
```

まず必要なパッケージをインポートし、コマンドライン引数を解析します。

2 つの新しいコマンドライン引数(前のスクリプトの--image を置き換えるもの)があります。
・--input：入力ビデオへのパス。
・--output：出力ビデオへのパス（結果をビデオファイルのディスクに書き込むため)。

それでは、クラス LABELS、COLORS、および Mask R-CNN ニューラルネットをロードしましょう。

```
# load the COCO class labels our Mask R-CNN was trained on
labelsPath = os.path.sep.join([args["mask_rcnn"],
        "object_detection_classes_coco.txt"])
LABELS = open(labelsPath).read().strip().split("¥n")

# initialize a list of colors to represent each possible class label
np.random.seed(42)
COLORS = np.random.randint(0, 255, size=(len(LABELS), 3),
        dtype="uint8")

# derive the paths to the Mask R-CNN weights and model configuration
weightsPath = os.path.sep.join([args["mask_rcnn"],
        "frozen_inference_graph.pb"])
configPath = os.path.sep.join([args["mask_rcnn"],
        "mask_rcnn_inception_v2_coco_2018_01_28.pbtxt"])

# load our Mask R-CNN trained on the COCO dataset (90 classes)
# from disk
print("[INFO] loading Mask R-CNN from disk...")
net = cv2.dnn.readNetFromTensorflow(weightsPath, configPath)
```

LABELS と COLORS は 24-31 行目にロードされています。

そこから、Mask R-CNN ニューラルネットを読み込む前に、自分の weightPath と configPath を定義します(34-42 行目)。

それでは、ビデオストリームとビデオライターを初期化しましょう。

```
# initialize the video stream and pointer to output video file
vs = cv2.VideoCapture(args["input"])
writer = None

# try to determine the total number of frames in the video file
try:
        prop = cv2.cv.CV_CAP_PROP_FRAME_COUNT if imutils.is_cv2() ¥
                else cv2.CAP_PROP_FRAME_COUNT
        total = int(vs.get(prop))
        print("[INFO] {} total frames in video".format(total))

# an error occurred while trying to determine the total
# number of frames in the video file
except:
        print("[INFO] could not determine # of frames in video")
        total = -1
```

ビデオストリーム(vs)とビデオライターは、45 行目と 46 行目で初期化されます。

ビデオファイルのフレーム数を決定して合計を表示しようとします（49-53 行目)。
失敗した場合は、例外を捕捉してステータスメッセージを印刷し、合計を-1 に設定します(57-59 行目)。この値を使用して、動画ファイル全体の処理にかかる時間を概算します。

フレーム処理ループを始めましょう。

```
# loop over frames from the video file stream
while True:
        # read the next frame from the file
        (grabbed, frame) = vs.read()

        # if the frame was not grabbed, then we have reached the end
        # of the stream
        if not grabbed:
                break
```

```
# construct a blob from the input frame and then perform a
# forward pass of the Mask R-CNN, giving us (1) the bounding box
# coordinates of the objects in the image along with (2) the
# pixel-wise segmentation for each specific object
blob = cv2.dnn.blobFromImage(frame, swapRB=True, crop=False)
net.setInput(blob)
start = time.time()
(boxes, masks) = net.forward(["detection_out_final",
        "detection_masks"])
end = time.time()
```

無限の while ループを定義し、最初のフレームをキャプチャすることで、フレームのループを始めます(62-64 行目)。ループは、完了するまでビデオを処理します。
これは 68 行目と 69 行目の終了条件によって処理されます。

次に、フレームからブロブを作成し、経過時間を把握しながらそれをニューラルネットに渡します。これにより、完了までの推定時間を後で計算できます(75-80 行目)。
結果はボックスとマスクの両方に含まれています。

それでは、検出されたオブジェクトのループを始めましょう。

```
# loop over the number of detected objects
for i in range(0, boxes.shape[2]):
        # extract the class ID of the detection along with the
        # confidence (i.e., probability) associated with the
        # prediction
        classID = int(boxes[0, 0, i, 1])
        confidence = boxes[0, 0, i, 2]

        # filter out weak predictions by ensuring the detected
        # probability is greater than the minimum probability
        if confidence > args["confidence"]:
                # scale the bounding box coordinates back relative to the
                # size of the frame and then compute the width and the
                # height of the bounding box
                (H, W) = frame.shape[:2]
```

```
box = boxes[0, 0, i, 3:7] * np.array([W, H, W, H])
(startX, startY, endX, endY) = box.astype("int")
boxW = endX - startX
boxH = endY - startY

# extract the pixel-wise segmentation for the object,
# resize the mask such that it's the same dimensions of
# the bounding box, and then finally threshold to create
# a *binary* mask
mask = masks[i, classID]
mask = cv2.resize(mask, (boxW, boxH),
        interpolation=cv2.INTER_NEAREST)
mask = (mask > args["threshold"])

# extract the ROI of the image but *only* extracted the
# masked region of the ROI
roi = frame[startY:endY, startX:endX][mask]
```

まず、低い確信度で弱い検出を除外します。次にバウンディングボックスの座標を決定し、マスクと ROI を取得します。

それではオブジェクトの透明なオーバーレイ、四角形の境界、ラベル+確信度を描画しましょう。

```
# grab the color used to visualize this particular class,
# then create a transparent overlay by blending the color
# with the ROI
color = COLORS[classID]
blended = ((0.4 * color) + (0.6 * roi)).astype("uint8")

# store the blended ROI in the original frame
frame[startY:endY, startX:endX][mask] = blended

# draw the bounding box of the instance on the frame
color = [int(c) for c in color]
cv2.rectangle(frame, (startX, startY), (endX, endY),
```

```
                    color, 2)

                # draw the predicted label and associated probability of
                # the instance segmentation on the frame
                text = "{}: {:.4f}".format(LABELS[classID], confidence)
                cv2.putText(frame, text, (startX, startY - 5),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)
```

ここでは、roi と色を混ぜ合わせて元のフレームに保存し、効果的に色付きの透明なオーバーレイを作成しています(118-122 行目)。

次にオブジェクトの周囲に長方形を描き、そのすぐ上にクラスラベル+確信度を表示します(125-133 行目)。

最後に、ビデオファイルに書き込んでクリーンアップしましょう。

```
        # check if the video writer is None
        if writer is None:
            # initialize our video writer
            fourcc = cv2.VideoWriter_fourcc(*"MJPG")
            writer = cv2.VideoWriter(args["output"], fourcc, 30,
                (frame.shape[1], frame.shape[0]), True)

            # some information on processing single frame
            if total > 0:
                elap = (end - start)
                print("[INFO] single frame took {:.4f} seconds".format(elap))
                print("[INFO] estimated total time to finish: {:.4f}".format(
                    elap * total))

        # write the output frame to disk
        writer.write(frame)

# release the file pointers
print("[INFO] cleaning up...")
writer.release()
```

vs.release()

ループの最初の繰り返しで、ビデオライターは初期化されます。

処理にかかる時間の見積もりは、行 143 から 147 で端末に印刷されます。

ループの最後の操作は、ライターオブジェクトを介してフレームをディスクに
書き込むことです(行 150)。

各フレームを画面に表示していないことに気付くでしょう。表示操作には時間がかかり、スクリプトの処理が終了したときには、どのメディアプレーヤーでも出力ビデオを見る
ことができます。

Note：さらに OpenCV はその dnn モジュールのために NVIDIA GPU をサポートしません。現時点では、限られた数の GPU、主に Intel GPU しかサポートされていません。
NVIDIA GPU のサポートは間もなく開始されますが、当面は OpenCV の dnn モジュールで GPU を簡単に使用することはできません。

最後に、ビデオの入出力ファイルポインタを解放します（154 行目と 155 行目)。

ビデオストリーム用の Mask R-CNN+OpenCV スクリプトをコーディングしたので、
試してみましょう。

このチュートリアルの「ダウンロード」セクションを使用して、ソースコードと
Mask R-CNN モデルをダウンロードしてください。

スマートフォンや他の録画機器を使って自分のビデオを収集する必要があります。
あるいは、YouTube から動画をダウンロードすることもできます。
Note：このダウンロードに含まれるビデオは、かなり大きい(400MB 以上)ため、意図的に含めていません。同じビデオを使用することを選択した場合はクレジットとリンクがこのセクションの下部にあります。

端末を開き、次のコマンドを実行します。

```
$ python mask_rcnn_video.py --input videos/cats_and_dogs.mp4 ¥
        --output output/cats_and_dogs_output.avi --mask-rcnn mask-rcnn-coco
```

[INFO] loading Mask R-CNN from disk...
[INFO] 19312 total frames in video
[INFO] single frame took 0.8585 seconds
[INFO] estimated total time to finish: 16579.2047

図 11：Python と OpenCV を使用してビデオに適用される Mask R-CNN。

上のビデオでは、犬や猫のマスク R-CNN が適用された面白いビデオクリップを見つけることができます。

```
$ python mask_rcnn_video.py --input videos/slip_and_slide.mp4 ¥
        --output output/slip_and_slide_output.avi --mask-rcnn mask-rcnn-coco
[INFO] loading Mask R-CNN from disk...
[INFO] 17421 total frames in video
[INFO] single frame took 0.9341 seconds
[INFO] estimated total time to finish: 16272.9920
```

これは 2 番目の例です。これは OpenCV と Mask R-CNN を車のビデオクリップに適用したものです。

図 12：Mask R-CNN オブジェクト検出は、Python と OpenCV を使用して車のビデオシーンに適用されます。

Mask R-CNN が交通量の多い道路に適用され、渋滞、自動車事故、または緊急の手助けと注意を必要とする旅行者をチェックすることを想定することができます。

ビデオとオーディオのクレジットは次のとおりです。

- Cats and Dogs
- Slip and Slide

How do I train my own Mask R-CNN models?

図 13：『Deep Learning for Computer Vision with Python』の中で、トレーニングデータに

アノテーションをつける方法、カスタム Mask R-CNN を訓練する方法、画像に適用する方法を学ぶ。(1)皮膚病変/癌のセグメンテーションと(2)処方ピルのセグメンテーション、2 つのケーススタディを提供します。

このチュートリアルで使用した Mask R-CNN モデルは COCO データセットで事前にトレーニングされています・・・
・・・しかし、あなた自身のカスタムデータセットで Mask R-CNN を訓練したいとしたら？

「Deep Learning for Computer Vision with Python」では、

1．癌性皮膚病変を自動的に検出してセグメント化するための Mask R-CNN のトレーニング方法を教えます。これは、自動癌危険因子分類システムを構築するための最初のステップです。
2．お気に入りの画像アノテーションファイルツールを提供して、入力画像用のマスクを作成できるようにします。
3．カスタムデータセットで Mask R-CNN をトレーニングする方法を説明します。
4．マスク R-CNN を訓練するときこのベストプラクティス、助言、して提案を提供してください。

すべての Mask R-CNN の章には、アルゴリズムとコードの両方の詳細な説明が含まれているので、自分の Mask R-CNN を正しく訓練することができます。

Summary

このチュートリアルでは、OpenCV と Python で Mask R-CNN アーキテクチャを適用して、画像やビデオストリームからオブジェクトをセグメント化する方法を学びました。

YOLO、SSD、Faster R-CNN などのオブジェクト検出器は、画像内のオブジェクトのバウンディングボックス座標を生成することしかできません。オブジェクト自体の実際の形状については何もわかりません。

Mask R-CNN を使用すると、画像内の各オブジェクトに対してピクセル単位のマスクを生成することができます。これにより前景のオブジェクトを背景からセグメント化することができます。
さらに Mask R-CNN を使用すると、従来のコンピュータビジョンアルゴリズムでは不可能だった複雑なオブジェクトや形状を画像からセグメント化することができます。

==========

６．２．OpenCV によるビデオ背景ぼかし

==========

この資料は、以下を参照されたい。
https://www.pyimagesearch.com/2018/11/26/instance-segmentation-with-opencv/

このチュートリアルでは、OpenCV、Python、および Deep Learning を使用してインスタンスセグメンテーションを実行する方法を学びます。

最近、Microsoft が Office 365 プラットフォームに素晴らしい機能をリリースしたのを見ました。

-- ビデオ会議通話に参加したり、背景をぼかしたり、同僚に自分だけを見させることができます(自分の背後にあるものは何もしません)。

この記事の冒頭にある GIF 画像は、このチュートリアルの目的のために私が実装した同様の機能を示しています。

ホテルの部屋から電話をかけている場合でも、実に醜い事務所ビルで働いている場合でも、単にデスクの周りを掃除したくない場合でも、電話会議のぼかし機能は会議の出席者を集中させ続けることができます(そしてバックグラウンドが乱雑ではありません)。

このような機能は在宅勤務で家族のプライバシーを保護したい人にとって特に役立ちます。

ワークステーションがあなたの台所をはっきりと見えていることを想像してみてください
- 同僚が夕食を食べたり、作業をしているのを見ているのではないでしょうか！
これには、ぼかし機能をクリックするだけですべて完了です。

このような機能を構築するために、マイクロソフトはコンピュータビジョンにディープラーニングによるインスタンスセグメンテーションを利用しました。

ここでも、Mask R-CNN の実装を使用して、Microsoft Office 365 風のビデオぼかし機能を構築するつもりです。

１．OpenCV によるインスタンスセグメンテーション

このチュートリアルは、
(1)Microsoft の Office 365 ビデオ通話のぼかし機能と
(2)PyImageSearch の読者 Zubair Ahmed の両方からヒントを得たものです。
Zubair は、Google の DeepLab を使って同様のぼかし機能を実装しました(以下のブログでその実装があります)。
http://zubairahmed.net/2018/07/17/background-blurring-with-semantic-image-segmentation-using-deeplabv3/

このチュートリアルの最初の部分ではインスタンスのセグメンテーションについて簡単に説明します。そこからインスタンスセグメンテーションと OpenCV を使用して、

１．ビデオストリームからユーザーを検出してセグメント化する
２．背景をぼかす
３．その後、ユーザーをストリーム自体に追加し直します。

いくつかの制限と欠点を含めて OpenCV インスタンスセグメンテーションアルゴリズムの結果を調べます。

２．インスタンスセグメンテーションとは？

図１：オブジェクト検出とインスタンスセグメンテーションの違い
オブジェクト検出(左)の場合、個々のオブジェクトの周囲にボックスが描画されます。
インスタンスセグメンテーション(右)の場合、どのピクセルが各オブジェクトに属しているかを判断しようとします。

インスタンスセグメンテーションの説明は視覚的な例を使用して行うのが最善です。上の図１を参照してください。左側にオブジェクト検出、右側にインスタンスセグメンテーションの例があります。

これら２つの例を見ると、両者の違いがはっきりとわかります。

物体検知を行う場合：

1．各オブジェクトのバウンディングボックス（x、y）座標の計算
2．そして、クラスラベルを各バウンディングボックスにも関連付けます。

問題はオブジェクト検出がオブジェクトの形状に関して何も伝えないということです
- これが持っているのはバウンディングボックス座標のセットだけです。
  一方、インスタンスセグメンテーションは画像内の各オブジェクトについてピクセル単位のマスクを計算が必要です。

上の図の 2 匹の犬のようにオブジェクトが同じクラスラベルのものであっても、インスタンス分割アルゴリズムでは合計 2 匹の犬と 1 匹の猫の合計 3 個の固有のオブジェクトが報告されます。

インスタンスセグメンテーションを使用すると、画像内のオブジェクトをより詳細に理解できます。オブジェクトがどの(x,y)座標に存在するかが明確にわかります。

さらに、インスタンスセグメンテーションを使用することで、前景のオブジェクトを背景から簡単にセグメント化できます。

このチュートリアルでは、インスタンスセグメンテーションに Mask R-CNN を使用します。

画像分類、オブジェクト検出、セマンティックセグメンテーション、インスタンスセグメンテーションの比較など、インスタンスセグメンテーションの詳細については、別紙を参照してください。

３．プロジェクトの構成

アーカイブを展開してナビゲートしたら、tree コマンドを利用して端末にディレクトリ構造を表示します。

```
$ tree --dirsfirst
.
├── mask-rcnn-coco
│   ├── frozen_inference_graph.pb
│   ├── mask_rcnn_inception_v2_coco_2018_01_28.pbtxt
│   └── object_detection_classes_coco.txt
└── instance_segmentation.py

1 directory, 4 files
```

このプロジェクトには、1 つのディレクトリ（3 つのファイルからなる）と 1 つの Python スクリプトが含まれています。

・mask-rcnn-coco/：Mask R-CNN モデルディレクトリには 3 つのファイルがあります。
　・frozen_inference_graph.pb：
　　Mask R-CNN モデルの重み。重みは COCO データセットで事前トレーニングされている。
　・mask_rcnn_inception_v2_coco_2018_01_28.pbtxt：
　　Mask R-CNN モデル構成
　・object_detection_classes_coco.txt：
　　このテキストファイルには、90 クラスすべてが 1 行に 1 つずつリストされています。モデルを認識できるオブジェクトを確認するには、テキストエディタでそれを開きます。
・instance_segmentation.py：今日はこの背景ぼかしスクリプトをレビューします。
　　その後、結果を使用して評価するためにそれを置きます。

４．OpenCV によるインスタンスセグメンテーションの実装

instance_segmentation.py ファイルを開き、以下のコードを挿入します。

```
# import the necessary packages
from imutils.video import VideoStream
import numpy as np
import argparse
```

```
import imutils
import time
import cv2
import os
```

必要なパッケージをインポートして、スクリプトから始めます。
あなたの環境には以下のものがインストールされている必要があります(仮想環境が強く推奨されます)。

・OpenCV 3.4.2+ — OpenCV がインストールされていない場合は、私のインストールチュートリアルページに進んでください。
ほとんどのシステムにインストールするための最速の方法は、この記事の執筆時点で OpenCV 3.4.3 をインストールする pip を使用することです。
・imutils — これは私のパーソナルコンピュータビジョン便利機能のパッケージです。
imutils は、pip install --upgrade imutils を使ってインストールできます。

繰り返しますが、他のプロジェクトでは異なるバージョンに対応する必要がある可能性があるため、このソフトウェアを独立した仮想環境に配置することを強くお勧めします。

コマンドライン引数を解析しましょう。

```
# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-m", "--mask-rcnn", required=True,
	help="base path to mask-rcnn directory")
ap.add_argument("-c", "--confidence", type=float, default=0.5,
	help="minimum probability to filter weak detections")
ap.add_argument("-t", "--threshold", type=float, default=0.3,
	help="minimum threshold for pixel-wise mask segmentation")
ap.add_argument("-k", "--kernel", type=int, default=41,
	help="size of gaussian blur kernel")
args = vars(ap.parse_args())
```

各コマンドライン引数の説明は以下にあります。

・--mask-rcnn：Mask R-CNN ディレクトリへのベースパス。 上記の「プロジェクト構造」のセクションで、このディレクトリ内の 3 つのファイルを確認しました。
・--confidence：弱い検出を除外するための最小確率。 この値をデフォルトの 0.5 に設定しましたが、コマンドラインからさまざまな値を簡単に渡すことができます。
・--threshold：ピクセル単位のマスクセグメンテーションに対する私たちの最小しきい値。デフォルトは 0.3 に設定されています。
・--kernel：ガウスぼかしカーネルのサイズ。41x41 カーネルはかなり見栄えが良いことがわかったのでデフォルトの 41 が設定されています。

ラベルと OpenCV インスタンスセグメンテーションモデルをロードしましょう。

```
# load the COCO class labels our Mask R-CNN was trained on
labelsPath = os.path.sep.join([args["mask_rcnn"],
	"object_detection_classes_coco.txt"])
LABELS = open(labelsPath).read().strip().split("¥n")

# derive the paths to the Mask R-CNN weights and model configuration
weightsPath = os.path.sep.join([args["mask_rcnn"],
	"frozen_inference_graph.pb"])
configPath = os.path.sep.join([args["mask_rcnn"],
	"mask_rcnn_inception_v2_coco_2018_01_28.pbtxt"])

# load our Mask R-CNN trained on the COCO dataset (90 classes)
# from disk
print("[INFO] loading Mask R-CNN from disk...")
net = cv2.dnn.readNetFromTensorflow(weightsPath, configPath)
```

当社のラベルファイルは、mask-rcnn-coco/ディレクトリに配置する必要があります。
- コマンドライン引数で指定されたディレクトリ
23 行目と 24 行目で labelsPath が構築され、25 行目で LABELS がリストに読み込まれます。

28 行目から 31 行目に作成されている、weightPath と configPath についても同じことが言えます。

これら 2 つのパスを使用して、ニューラルネットを初期化するために dnn モジュールを利用します(行 36)。 この呼び出しはフレームの処理を開始する前に Mask R-CNN をメモリにロードします(ロードする必要があるのは 1 回だけです)。

ぼかしカーネルを構築し、Web カメラのビデオストリームを開始しましょう。

```
# construct the kernel for the Gaussian blur and initialize whether
# or not we are in "privacy mode"
K = (args["kernel"], args["kernel"])
privacy = False

# initialize the video stream, then allow the camera sensor to warm up
print("[INFO] starting video stream...")
vs = VideoStream(src=0).start()
time.sleep(2.0)
```

ぼかしカーネルタプルは 40 行目で定義されています。

このプロジェクトには「通常モード」と「プライバシーモード」の 2 つのモードがあります。したがってプライバシー論理値がモードロジックに使用されます。41 行目で False に初期化されています。

Web カメラのビデオストリームは 45 行目で開始され、そこでセンサーがウォームアップできるように 2 秒間停止します(46 行目)。

すべての変数とオブジェクトが初期化されたので Web カメラからのフレームの処理を始めましょう。

```
# loop over frames from the video file stream
while True:
	# grab the frame from the threaded video stream
	frame = vs.read()
```

```
	# resize the frame to have a width of 600 pixels (while
	# maintaining the aspect ratio), and then grab the image
	# dimensions
	frame = imutils.resize(frame, width=600)
	(H, W) = frame.shape[:2]

	# construct a blob from the input image and then perform a
	# forward pass of the Mask R-CNN, giving us (1) the bounding
	# box coordinates of the objects in the image along with (2)
	# the pixel-wise segmentation for each specific object
	blob = cv2.dnn.blobFromImage(frame, swapRB=True, crop=False)
	net.setInput(blob)
	(boxes, masks) = net.forward(["detection_out_final",
		"detection_masks"])
```

フレーム処理ループは 49 行目から始まります。

繰り返しごとに、縦横比を維持しながら、フレームをつかみ(51 行目)、既知の幅にサイズ変更します(56 行目)。

後でスケーリングするために、先に進んでフレームのサイズを抽出します(57 行目)。

それからブロブを構築し、ネットワークを通過する順方向パスを完成させます(63-66 行目)。このプロセスがどのように機能するかについての詳細はこの前回のブログ投稿で読むことができます。

結果はボックスとマスクの両方です。マスクを利用するつもりですが、ボックスに含まれるデータも使用する必要があります。

インデックスを並べ替えて変数を初期化しましょう。

```
	# sort the indexes of the bounding boxes in by their corresponding
	# prediction probability (in descending order)
	idxs = np.argsort(boxes[0, 0, :, 2])[::-1]

	# initialize the mask, ROI, and coordinates of the person for the
```

```
        # current frame
        mask = None
        roi = None
        coords = None
```

行 70 は、対応する予測確率によってバウンディングボックスのインデックスを並べ替えます。対応する検出確率が最大の人が私たちのユーザーであると仮定します。
それから、マスク、ROI、バウンディングボックスのコードを初期化します (74-76 行目)。

インデックスをループして結果をフィルタリングしましょう。

```
        # loop over the indexes
        for i in idxs:
                # extract the class ID of the detection along with the
                # confidence (i.e., probability) associated with the
                # prediction
                classID = int(boxes[0, 0, i, 1])
                confidence = boxes[0, 0, i, 2]

                # if the detection is not the 'person' class, ignore it
                if LABELS[classID] != "person":
                        continue

                # filter out weak predictions by ensuring the detected
                # probability is greater than the minimum probability
                if confidence > args["confidence"]:
                        # scale the bounding box coordinates back relative to the
                        # size of the image and then compute the width and the
                        # height of the bounding box
                        box = boxes[0, 0, i, 3:7] * np.array([W, H, W, H])
                        (startX, startY, endX, endY) = box.astype("int")
                        coords = (startX, startY, endX, endY)
                        boxW = endX - startX
                        boxH = endY - startY
```

79 行目の idx をループし始めます。

次に、box と現在のインデックスを使用して classID と確信度を抽出します(83、84 行目)。

その後、最初のフィルタを実行します。「person」クラスについてのみ注意します。
他のオブジェクトクラスが見つかった場合は、次のインデックスに進みます(87、88 行目)。

次のフィルタは、予測の確信度がコマンドライン引数で設定されたしきい値を超えることを保証します(92 行目)。

このテストに合格すると、バウンディングボックスの座標は画像の相対的な大きさに戻ります(96 行目)。 次に座標とオブジェクトの幅/高さを抽出します(97-100 行目)。

マスクを計算して ROI を抽出しましょう：

```
# extract the pixel-wise segmentation for the object,
# resize the mask such that it's the same dimensions of
# the bounding box, and then finally threshold to create
# a *binary* mask
mask = masks[i, classID]
mask = cv2.resize(mask, (boxW, boxH),
        interpolation=cv2.INTER_NEAREST)
mask = (mask > args["threshold"])

# extract the ROI and break from the loop (since we make
# the assumption there is only *one* person in the frame
# who is also the person with the highest prediction
# confidence)
roi = frame[startY:endY, startX:endX][mask]
break
```

行 106-109 ではマスクを抽出してサイズを変更し、しきい値を適用してバイナリマスク自体を作成します。 マスクの例を図 2 に示します。

図２：バイナリマスクは OpenCV とインスタンスセグメンテーションを使用して私のウェブカメラの前で私のインスタンスセグメンテーションを介して計算しました。
マスクの計算は、プライバシーフィルタのパイプラインの一部です。

図２では、全ての白いピクセルは人（すなわち前景）であると仮定され、一方全ての黒いピクセルは背景である。

このマスクを使用して、NumPy 配列のスライスを介して roi（115 行目）も計算します。

次に、116 行目のループから抜けます（最も確率の高い「person」が見つかったため)。

「privacy mode」になっている場合は、出力フレームを初期化してぼかしを計算しましょう。

```
# initialize our output frame
output = frame.copy()

# if the mask is not None *and* we are in privacy mode, then we
# know we can apply the mask and ROI to the output image
if mask is not None and privacy:
        # blur the output frame
        output = cv2.GaussianBlur(output, K, 0)

        # add the ROI to the output frame for only the masked region
        (startX, startY, endX, endY) = coords
        output[startY:endY, startX:endX][mask] = roi
```

出力フレームは、元のフレームのコピーです(119 行目)。

二人の場合：

１．空ではないマスク
２．「privacy　mode」にいます…

…それから背景をぼかし(カーネルを使って)、マスクを出力フレームに適用します
(123-129 行目)。

それでは、出力画像を表示してキー入力を処理しましょう。

```
        # show the output frame
        cv2.imshow("Video Call", output)
        key = cv2.waitKey(1) & 0xFF

        # if the `p` key was pressed, toggle privacy mode
        if key == ord("p"):
                privacy = not privacy

        # if the `q` key was pressed, break from the loop
        elif key == ord("q"):
                break

# do a bit of cleanup
cv2.destroyAllWindows()
vs.stop()
```

出力フレームは 132 行目に表示されます。

キープレスがキャプチャされます(133 行目)。2 つのキーが異なる振る舞いをします(136-141 行目)。

・"p"：このキーを押すと「プライバシーモード」がオンまたはオフに切り替わります。
・"q"：このキーが押されるとループから抜けてスクリプトを「終了」します。

終了するたびに、144 行目と 145 行目は開いているウィンドウを閉じて、ビデオストリームを停止します。

５．インスタンスセグメンテーションの結果

OpenCV インスタンスセグメンテーションアルゴリズムを実装したので、実際に実行してみましょう。

そこから端末を開き、次のコマンドを実行します。

```
$ python instance_segmentation.py --mask-rcnn mask-rcnn-coco --kernel 41
```

[INFO] loading Mask R-CNN from disk...
[INFO] starting video stream...

図３：Web チャット用の「プライバシーフィルタ」のデモ。 私は OpenCV と Python を使ってインスタンスセグメンテーションを実行して著名な人物（私）を見つけ、それから背景にぼかしをかけました。

ここでは、インスタンスセグメンテーションパイプラインをデモしている短い GIF を見ることができます。

「privacy mode」を有効にすると、

1．OpenCV インスタンスセグメンテーションを使用して、対応する可能性が最も高い人の検出を見つけます（最も可能性が高いのは、カメラに最も近い人です)。
2．ビデオストリームの背景をぼかします。
3．セグメント化された、ぼやけていない人物をビデオストリームに重ねます。

この解説を含むビデオデモを以下に掲載しました。

ただし、リアルタイムのパフォーマンスが得られていないことにすぐに気付くでしょう。
ただし、1 秒間に数フレームしか処理されていません。 どうしてこれなの？

OpenCV インスタンスのセグメンテーションパイプラインがいかに速くなったのでしょうか。
これらの質問に答えるには、必ず以下のセクションを参照してください。

６．制限、欠点、今後の改善

最初の制限は最も明白な制限です - 私たちの OpenCV インスタンスセグメンテーション実装は遅すぎてリアルタイムで実行することができません。

Intel Xeon W では、毎秒数フレームしか処理していません。

真のリアルタイムインスタンスセグメンテーションパフォーマンスを得るためには、GPU を活用する必要があります。
しかし、そこに問題があります。

dnn モジュールに対する OpenCV の GPU サポートはかなり制限されています。
現在、主に Intel GPU をサポートしています。

NVIDIA CUDA GPU サポートは開発中ですが、現在利用できません。

OpenCV が dnn モジュール用に NVIDIA GPU を正式にサポートするようになれば、リアルタイム（さらにはスーパーリアルタイム）のディープラーニングアプリケーションを簡単に構築できるようになります。

しかし今のところ、この OpenCV インスタンスセグメンテーションチュートリアルは以下の教育用デモとして役立ちます。

マイクロソフトの実装とマイクロソフトの Office 365 のビデオぼかし機能を比較すると、マイクロソフトのほうがはるかに「スムーズ」であることがわかります。

ちょっとアルファブレンディングを利用することによってこの特徴をまねることができます。

このインスタンスセグメンテーションパイプラインへの単純だが効果的な更新は潜在的に以下のようになるでしょう。

1．マスクのサイズを大きくするために形態学的操作を使用する
2．マスク自体に少量のガウスぼかしを適用して、マスクを滑らかにします。
3．マスク値を[0、1]の範囲に拡大縮小する
4．拡大縮小したマスクを使用してアルファレイヤーを作成する
5．背景をぼかした写真の上にスムーズマスク+人物の ROI を重ねる

あるいは、マスク自体の輪郭を計算してから輪郭近似を適用して、より滑らかなマスクを作成することもできます。

７．まとめ

このチュートリアルでは、OpenCV、Deep Learning、および Python を使用してインスタンスのセグメンテーションを実行する方法を学びました。

インスタンスセグメンテーションは次の処理から成ります：

１．画像内の各オブジェクトを検出する
２．各オブジェクトに対するピクセル単位のマスクの計算

オブジェクトが同じクラスのものであっても、インスタンスセグメンテーションは各オブジェクトに対して一意のマスクを返すべきです。

OpenCV でインスタンスセグメンテーションを適用するために、Mask R-CNN 実装を使いました。

その後、こは Mask R-CNN モデルを使用して、Microsoft が夏に Office 365 向けにリリースした機能と同様の「ビデオ会議通話のぼかし機能」を構築しました。

インスタンスのセグメンテーション結果は、マイクロソフトの機能と似ていました。
しかし、dnn モジュールに対する OpenCV の GPU サポートは現在非常に限られているので、本当のリアルタイム性能を得ることができませんでした。

したがって、このチュートリアルはデモとして役立ち、OpenCV が GPU サポートしたときに実用になることでしょう。