リアルタイム検出
物体検出手法, VGG-16, SSD

リアルタイム検出

実機のWebカメラを使用して、リアルタイムであるターゲットを認識させる

必要なもの：

- 対象画像（学習用、評価用）
- 背景画像（学習用、評価用）
- データセットを記述したcsvファイル

かなりの部分はこちらで用意してありますが（料理番組方式）、本当はこれらを全部自分で用意します。ぜひ1度自分でチャレンジしてください

リアルタイム検出の手順

次の順番で行う

1. 対象物を撮影する
2. 背景画像を撮影する
3. 対象物画像を水増しして300枚用意する
4. 背景画像を分割して625枚用意する
5. 対象物画像と背景画像をcsvファイルに記述する
6. 用意したデータで学習を行う
7. 学習されたデータを用いて、リアルタイムで判定させる

対象物の撮影

対象物をカメラアプリで撮影し、128x128のサイズで切り出す
正方形で切り出すため、なるべく縦横比が極端でないものを選ぶ
顔は意外と認識されにくい（頑張って下さい）
手のひらは割と簡単。スマホケース、ペットボトルの模様、社員証など
ファイル名はtarget.jpgとする。（詳しい手順は、この後の対象画像作成手順を参照）

今度は、同じ位置で背景のみで撮影し、ファイル名をother.jpgとする。サイズ等は変更しない

撮影

対象物が入った写真を撮影 →　対象物だけ切り抜いてtarget.jpg
次に対象物がない写真を撮影　→　other.jpg

認識させたい対象物

自分が映り込まないようにして撮影

指定サイズでの切り出し(1)

正方形での切り出しが可能なXnViewを使用する

https://forest.watch.impress.co.jp/library/software/xnview/ などからダウンロード（XnViewで検索すると窓の杜が最初にヒット）

もちろん、オフィシャルサイトからDLしてもよい。
https://www.xnview.com/

XnViewを起動し、カメラアプリで撮影した対象物のファイルを読み込む
もし画像が鏡映しになっていたら、画像 → 反転 → 左右反転

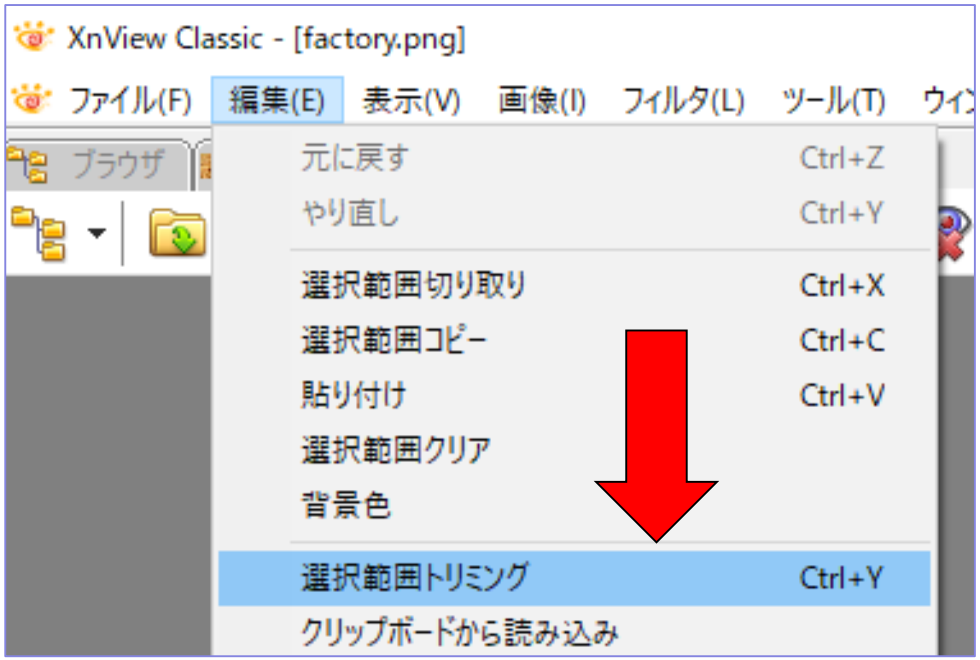
指定サイズでの切り出し(2)

編集 → 選択範囲縦横率設定 → 1:1 (1.00) を選択
これで正方形で範囲選択が可能

指定サイズでの切り出し(3)

任意の範囲を選択後、編集 → 選択範囲トリミング を選択し、切り抜く

切り抜いた後、画像 → リサイズ、でサイズ変更画面が開くので、縦横128ピクセルを指定してサイズ変更し、target.jpg という名前で保存



CSVファイル

こちらで準備済み

- target_or_other_train.csv　：学習（訓練用）ファイルパスとラベル
- target_or_other_test.csv　：検証用ファイルパスとラベル

それぞれのファイル内に、各画像のパスとラベルがカンマ区切りで書かれている

エディタやExcelなどで1度中味をチェックすること

水増しと学習

Colab: 08-Augmentation_Learning.ipynb を開く

VM内にtarget.jpgとother.jpg, csvファイルをコピーし、実行する

先程のcsvファイルと、フォルダtrain_target, test_targetとの対応、実際のファイルの存在をチェックしておく

学習後、重みファイル(detection_weight.h5)をダウンロードし、後は実機で

ネットワークはある程度しか書いていないので、認識できるように、拡張して下さい

重みファイルはネットワークごとに名前を変えておいて、比較するとよい

実機での認識

実機では、Anaconda promptから。前回作成したpython 3.7環境を使用

conda activate py37 など

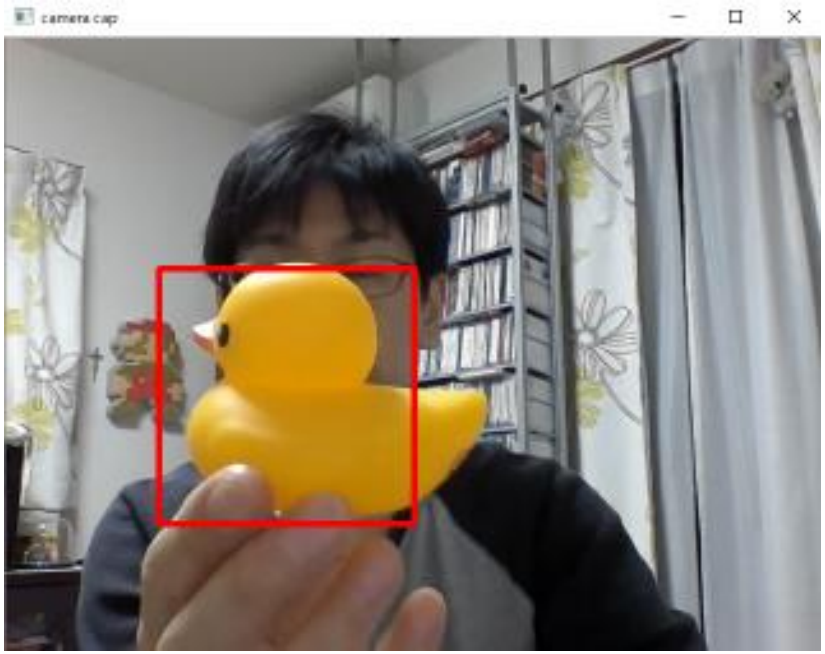
さらに必要なパッケージをまずインストールする

```
pip install opencv-python
pip install pillow
pip install tensorflow
```

その後、`python detection_roi.py` と実行

結果

緑の枠内にターゲットを持っていく
無事に認識されると赤枠になる
どうにもダメなら、ネットワークやエポック数など変えてみる



もし改良するなら

ターゲットの画像サイズをもう少し大きくしてみる (192x192など?)
画像サイズを大きくしたら、検出用ソースも手を入れる
ネットワークの規模を大きくする
そもそも水増し画像をもっと増やしてみる（CSVファイルにも手を入れる）

物体検出

物体検出手法の概要

画像のどの位置に何があるか、を判断するのが物体検出

検出：物体と思われる領域を見つける処理

識別：領域内の物体について識別する処理

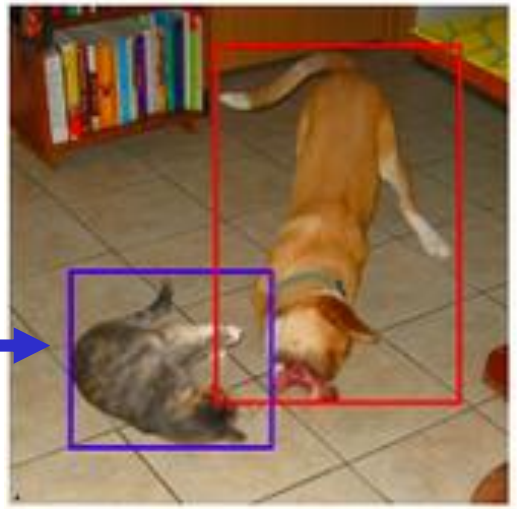
この2つから成り立っている

単なるCNNでは画像全体から判断していたが、物体検出ではいかにして領域を見つけるかが鍵になる

自動運転には欠かせない

画像中からこの枠で囲うのが検出

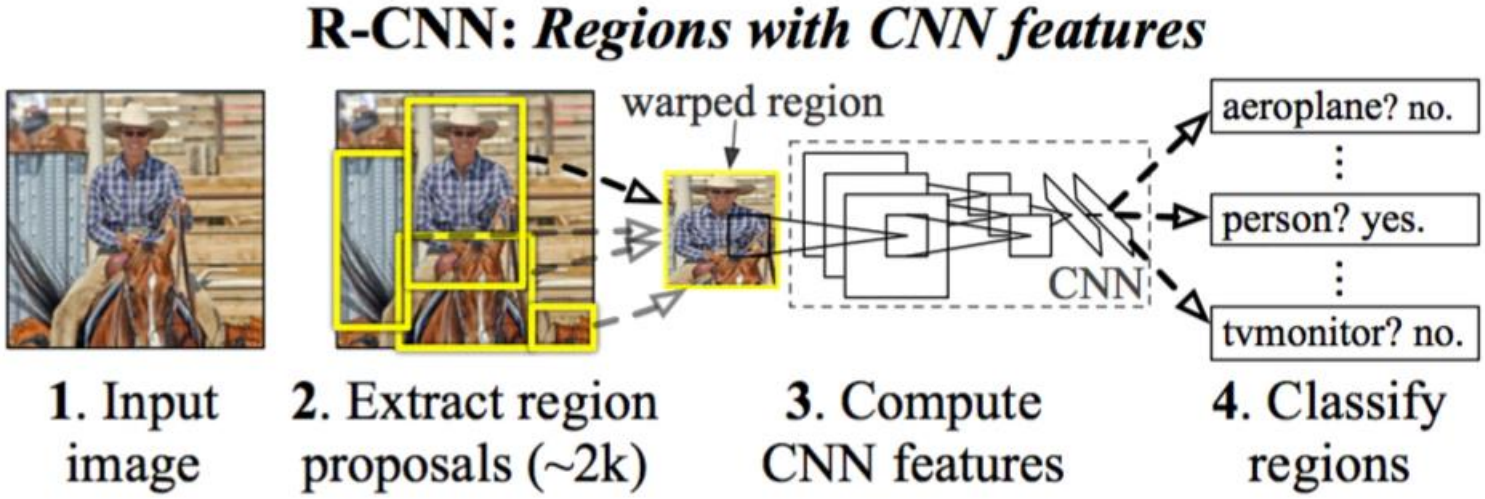
枠の中が猫と出力するのが識別



物体検出手法の概要(R-CNN)

入力画像に対して、物体がある候補領域(region proposal)を2000個まで抽出し、各領域に対して画像をリサイズしてCNNから特徴マップを生成

Region proposalは色や濃淡勾配などから切り分けている



Ross Girshick, Jeff Donahue, et al.,"Rich feature hierarchies for accurate object detection and semantic segmentation", (2013), https://arxiv.org/pdf/1311.2524

SSD: Single Shot Multibox Detector

現在最もよく使用されている物体判別器

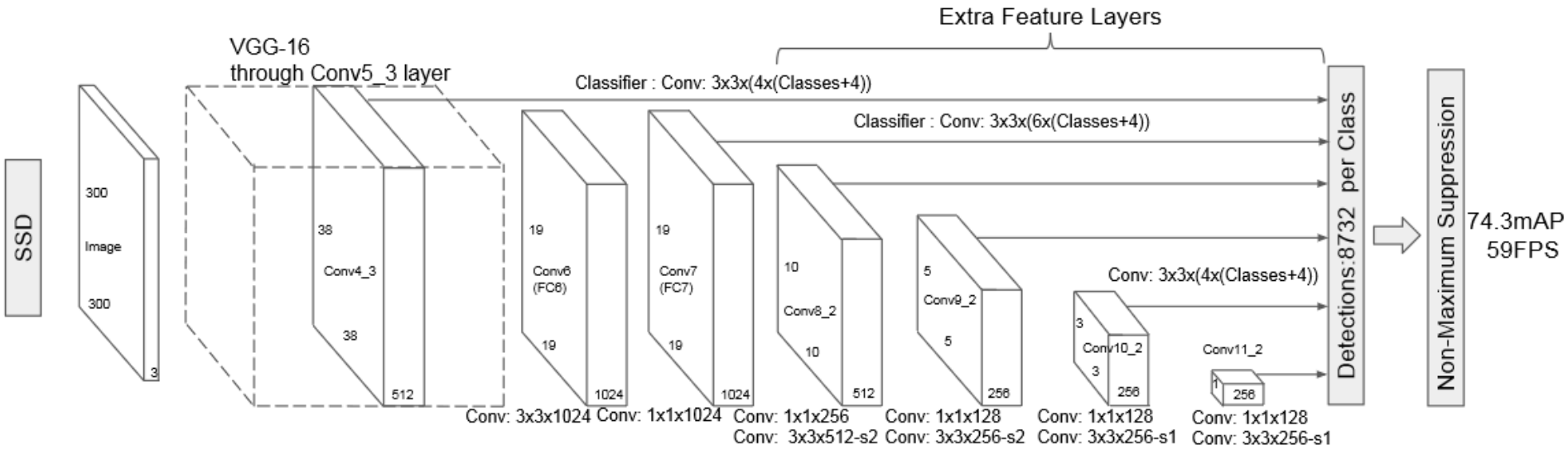
CNNを使用して、画像内のどの位置にどのような物体が存在するかを判別

物体判別器には他にYOLOやFaster R-CNNなどのモデルもあるが、元論文ではSSDの方がより優れた結果を挙げている

Wei Liu, Dragomir Anguelov, Dumitru Erhan, et al., **SSD: Single Shot MultiBox Detector**, https://arxiv.org/abs/1512.02325, (2015)

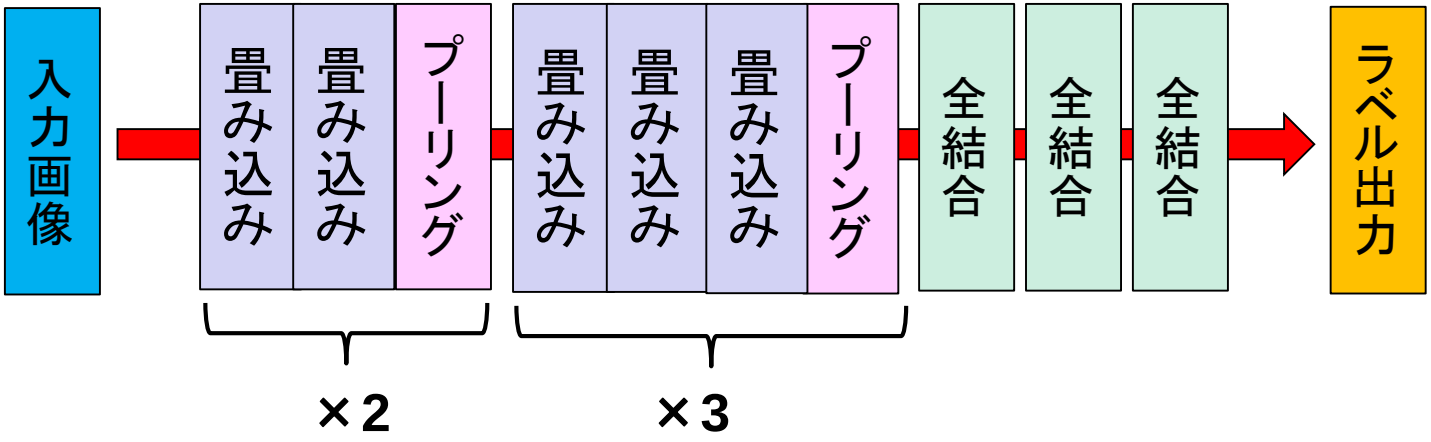
SSDのネットワーク

VGG-16をベースとして、特徴マップを拡張する
ではVGG-16をまず……



VGG-16

ImageNetで学習された16層のモデル。ILSVRC2014で2位を獲得した
224×224のカラー入力画像に対し、以下のネットワークで学習
※プーリング層以外をすべて足すと16層



Karen Simonyan, Andrew Zisserman, “**Very Deep Convolutional Networks for Large-Scale Image Recognition**”, https://arxiv.org/abs/1409.1556, (2014)

VGG-16の特徴

入力画像は224x224、これを112x112, 56x56, … と7x7まで小さくする

最終出力は1000

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              102764544
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
_________________________________________________________________
predictions (Dense)          (None, 1000)              4097000
=================================================================
```

VGG-16演習

Colab: 08-VGG16.ipynb を実行

実機：vgg16.py, vgg16_camera.py

実機では、Anaconda promptから。前回作成したpython 3.7環境を使用

conda activate py37 など

実行前に必要なパッケージをインストールする

```
pip install keras
```

※独立パッケージをKerasを使用

VGG-16演習: 実機

vgg16.py: Colabでの演習を実機に移したもの
画像表示を行わない分高速です（意外とColabは画像表示が遅い）

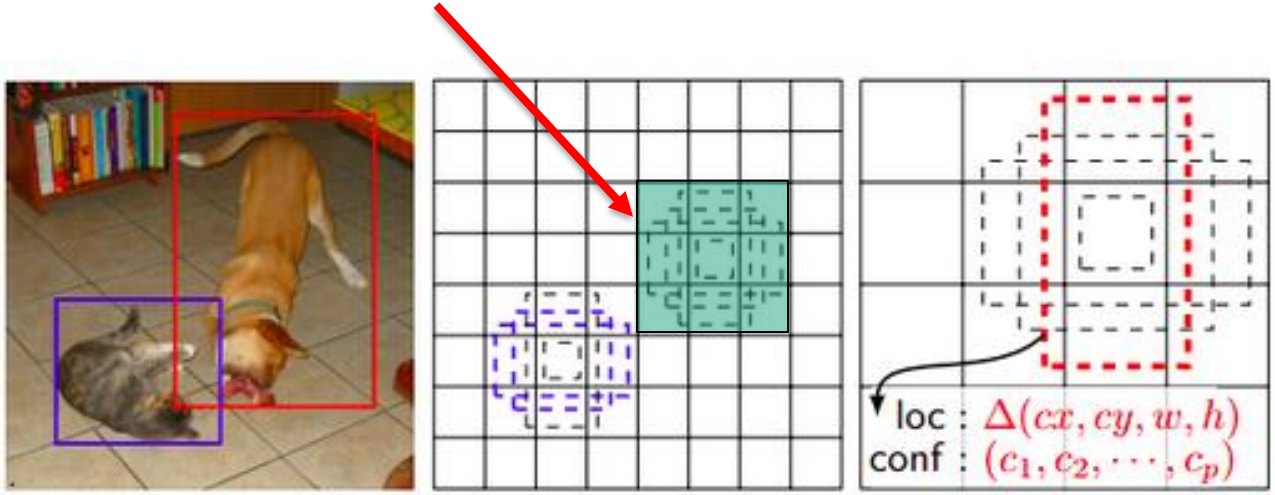
vgg16_camera.py: WebカメラからVGG-16で判別
起動するとWebカメラの映像が映るので、ESCキーを押すと、キャプチャがphoto.jpgで保存され、その内容がVGG-16で判別される
色々な物を持ったり、（最小限の接触で）近くの人に協力してもらって実行して下さい

改めてSSD

図(a)の実線の青い矩形(教師データ)と、特徴マップである図(b)の点線の青い矩形(推定値)の位置・サイズが一致し、また物体クラスが猫となるようにネットワークの重みを更新していく

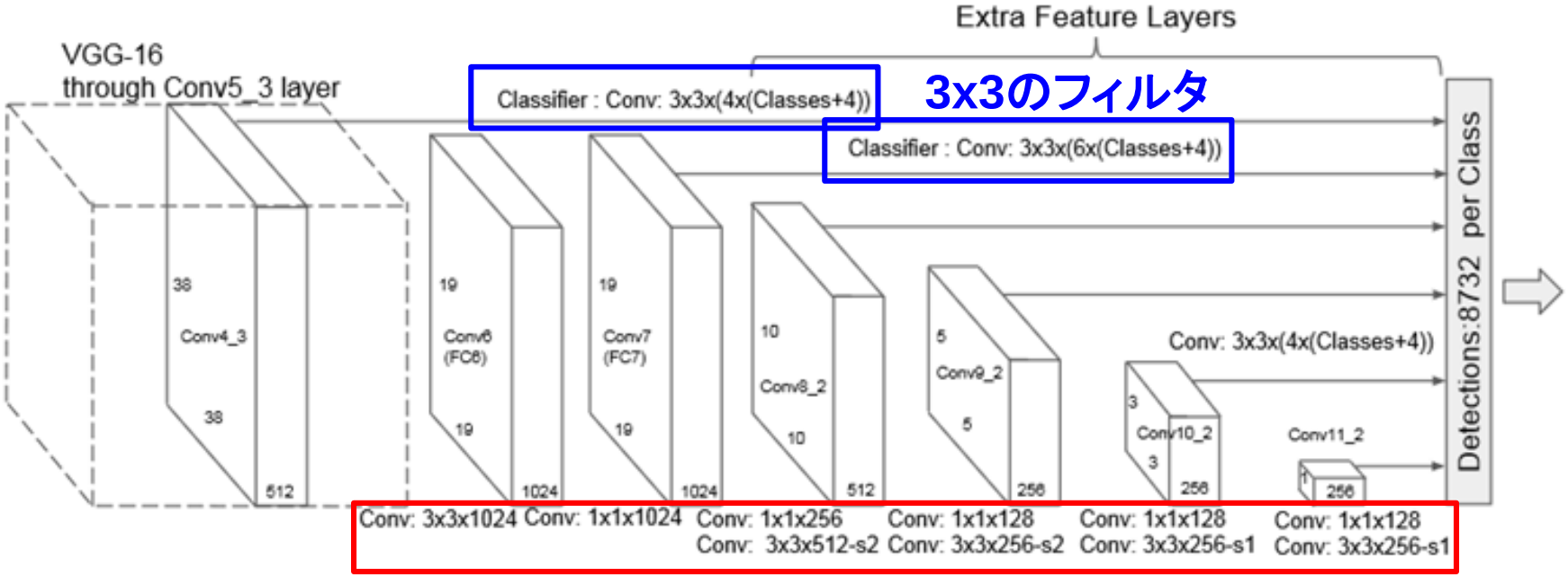
同じように、図(a)の実線の赤い矩形(教師データ)と、図(c)の点線の赤い矩形(推定値)の位置・サイズを一致させ、犬と推定できるように学習を行う

3x3のフィルタをかけるのが特徴



(a) Image with GT boxes (b) 8 × 8 feature map (c) 4 × 4 feature map

ネットワーク構造



VGG-16で38x38x512の特徴マップを作成。

その後、19x19x1024, 10×10×512, 5×5×256, 3×3×256, 1×1×256, と特徴マップを作成して入力としている

最終的に、1つの入力画像に対して分類クラスあたり8732個の物体領域候補が出力される

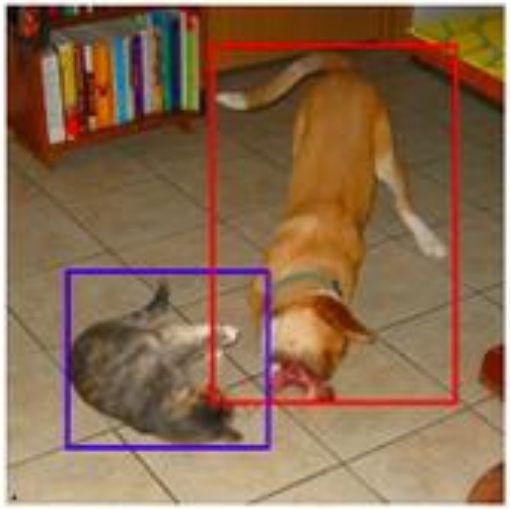
デフォルトボックスの対応

物体のアスペクト比は当然物体ごとに異なる

各畳み込み層で4種類のアスペクト比、4種類の位置をパラメータを持ち、すなわち4×4=16通り、さらに最初のVGG-16からの出力なら画素は38x38なので、38x38x16 = 23104の位置特徴量を持つ

（各レイヤごとにパラメータは若干異なる）

これによって、猫と犬のアスペクト比の違いを学習する



SSD演習(1)

Google ColabでSSDを試行
あらかじめ構築されたモデルを使用するので学習は必要ない
モデルは2つ用意されており、

- **FasterRCNN+InceptionResNet V2**: high accuracy,
- **ssd+mobilenet V2**: small and fast.

の記述通り、前者がより正確、後者はモデルがコンパクトで速い。両者での結果を比較してみるとよい（実際かなり異なる）

SSD演習(2)

ネットから適当に探す、あるいは自分で撮影、手持ちの画像をSSDにかけてください

その結果をTeamsにアップロードし、比較してみましょう

どのような画像がうまくいくか、あるいはうまくいかないか

使用できるネットワーク

今回、VGG-16やInceptionResNet, MobileNetなどを使用したが、KerasやTensorflowではその他にも様々なネットワークが使用できる

詳細はGitHubを参照

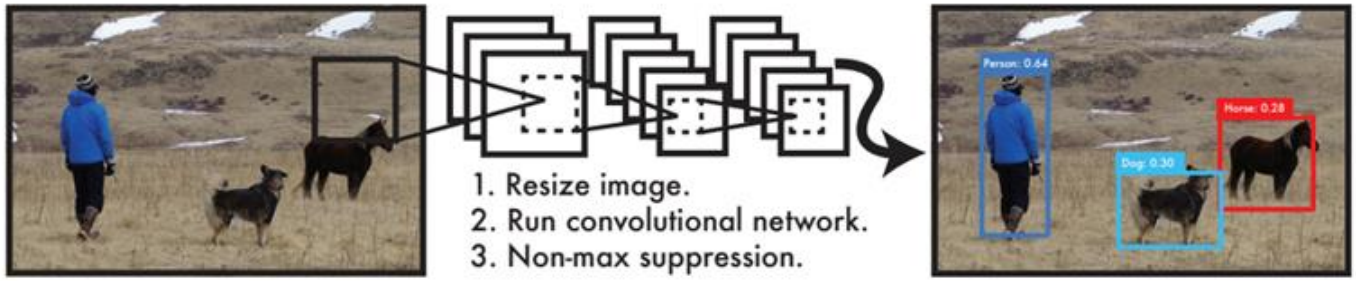
https://github.com/keras-team/keras-applications

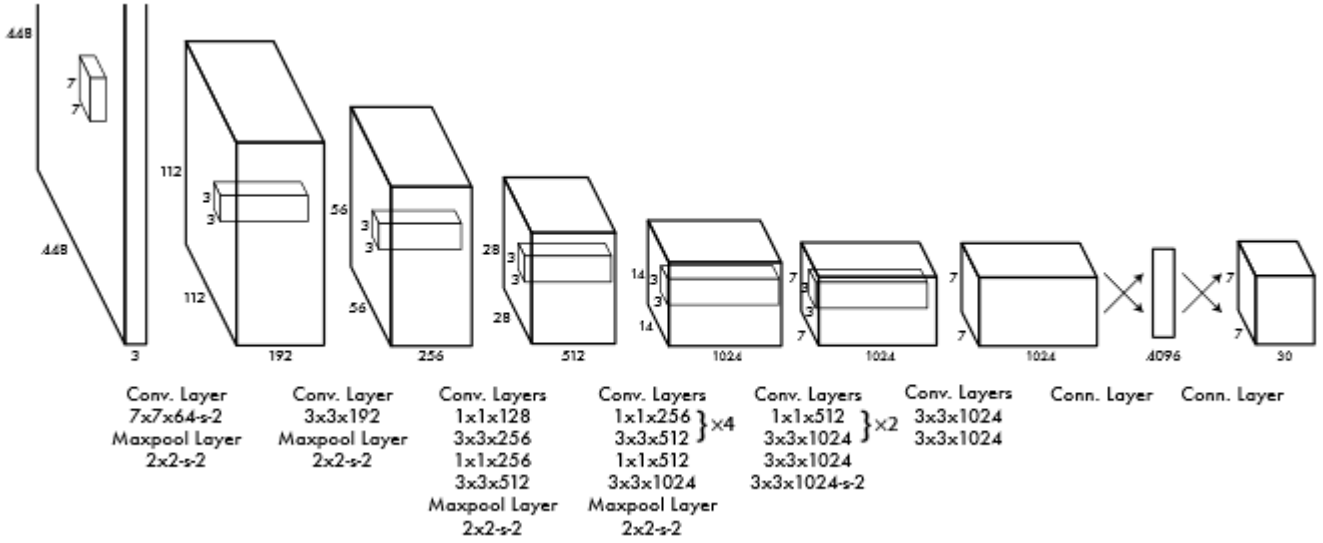
他に有名なネットワークとしてYOLOv3など

https://pjreddie.com/darknet/yolo/

YOLOの手法(1)

画像を正方形(448x448など)にリサイズし、畳み込みへの入力とする





24層の畳み込み,
2層の全結合

Paper: https://pjreddie.com/media/files/papers/yolo_1.pdf

YOLOの手法(2)

画像全体をSxSに分割（論文ではS=7）
分割された1つのセルをB個に分ける（同じくB=2）
それぞれのセル内にてクラス分類を行い、信頼度の高いものだけを採用
それぞれの信頼度の高い部分を太く囲う
1つのセル内に複数の物体があると弱い