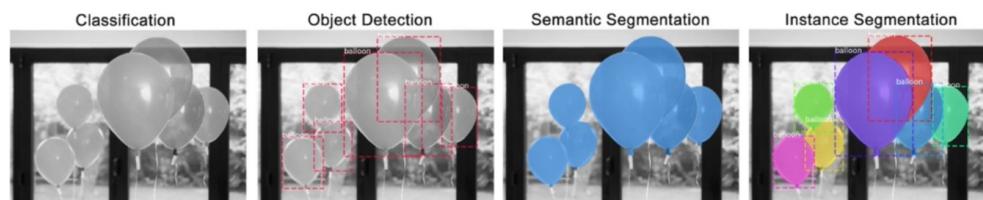


# 1 物体検知と SS 解説

## 1.1 導入 (分類タスク)

- ・物体検知なので基本的には画像認識
- ・ただし、単体の検知ではなく、複数の物体が写った画像に対する個々の検知 (多クラス検知になる)
- ・物体を示す範囲は BoundingBox(bbox/BB と略す場合もある) で囲う  
→ 後述するが BB 自体の精度も物体検知タスクでは精度指標になる
- ・物体の形状をピクセル単位で示す
- ・以下図の様な 4 つの分類タスクに分けられるとのこと



	分類 (Classification)	物体検知 (Object Segmentation)	意味領域分割 (Semantic Segmentation)	個体領域分割 (Instance Segmentation)
出力	画像に対し 单一または複数の クラスラベル	BoundingBox[bbox/BB]	各ピクセルに対し单一のク ラスラベル	各ピクセルに対し单一のク ラスラベル+BoundingBox
粒度は?	物体の位置はどうでもよい	物体の位置は意識する	物体の位置だけでなく形状 も意識する(が同一Classは ひとつで示す)	物体の位置だけでなく形状 も意識する(さらに個体 (Instance)は別々に示す)

難易度

低

高

## 1.2 導入 (データセットについて)

- ・物体検知の公なコンペティション(等)にてデータセットが用意されている
- ・有名なデータセットは名前(ex:VOC12とかILSVRC17とか色々)付きで公開されている
- ・データセットは個人が自由に利用できる(コンペティション以外でも利用可能)
- ・データセット仕様(クラス数、訓練画像枚数、個体(クラス)数/1画像、画像の解像度)は様々  
(以降、個体(クラス)数/1画像は「BOX/画像」と表現する)
  - それらを把握した上で個人利用目的に適合したデータセットを利用する事
  - 例えば日常写真は多クラス(多個体)混在、アイコン的な成型画像は单クラス(単個体)
  - 目的に応じて選択すべき(「クラスが多い=万能」という事はない)
- ・データセットのアノテーション(画像のトリミングやラベル付け)は人力
  - 玉石混在かもしれない
  - より高品質なアノテーションを施したデータセットを選択するかは自己判断にて

## 1.3 導入 (混同行列の復習)

- ・Bounding Box(BB)の精度指標として用いる、等の為復習がされた

		検証用データの結果		正しい予測の個数	間違った予測の個数	
		positive	negative			
モデルの予測結果	positive	真陽性 (True Positive)	偽陰性 (False Positive)			
	negative	偽陽性 (False Negative)	真陰性 (True Negative)			

- ・適合率(Precision):例えば positiveと判定した全体に対する、真の positive割合

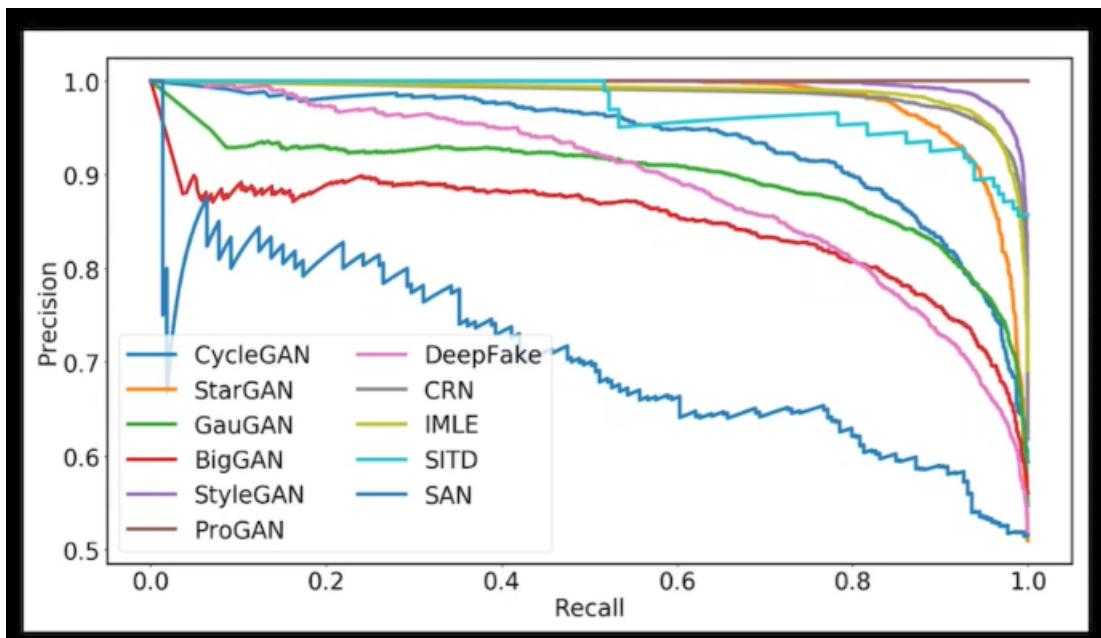
$$Precision = \frac{TP}{TP+FP}$$

- ・再現率(Recall):例えば positiveのうち、真の positiveとして検出した割合

$$Recall = \frac{TP}{TP+FN}$$

## 1.4 導入 (閾値変化に対する振る舞い)

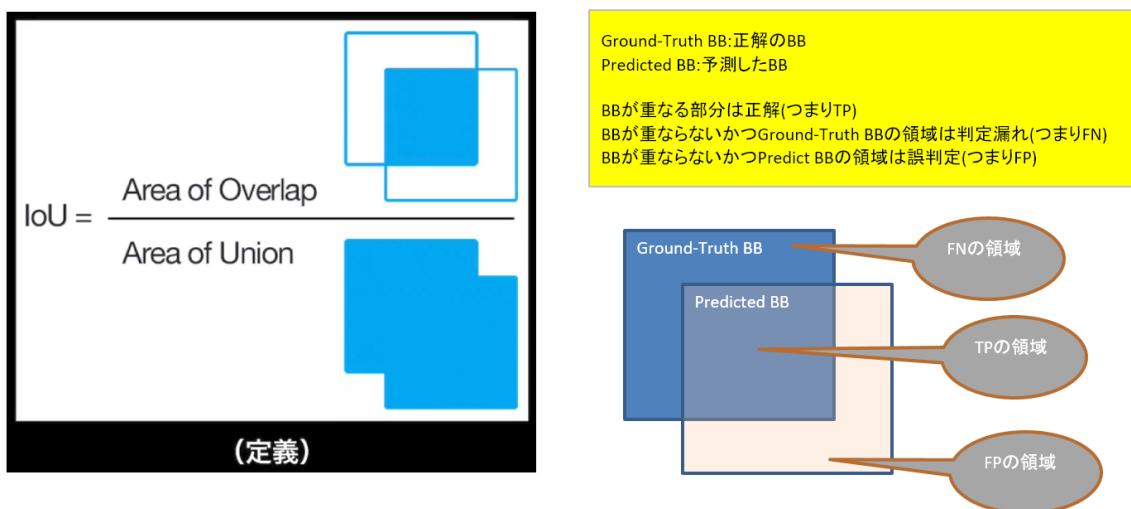
- ・単純な二値分類の閾値 (ex シグモイド関数出力値で positive-negative 判定)
- ・判定閾値を変動して混同行列をグラフ評価する事がある (Precision-Recall 曲線)



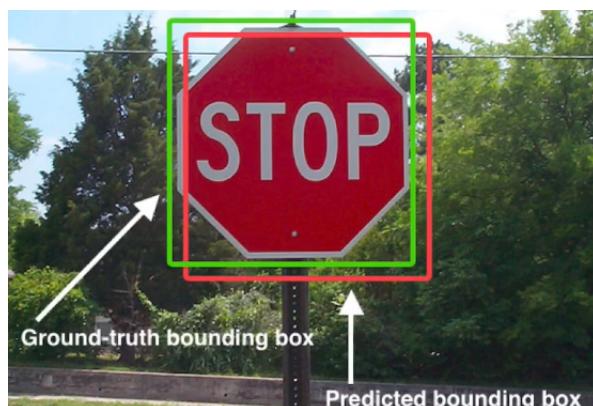
- ・Precision-Recall 曲線のグラフ下側面積 (積分値) を評価指標に用いる (後述にて説明)
- ・物体検知における対象の位置判定
  - IoU(後述にて説明) を用いて評価する
  - 閾値より例えば BB が消える (閾値以下)、閾値を緩くすると BB が増える
  - つまり BB の検出精度により画像認識の母数が変動する
- ・つまり物体検出は評価指標が二重 (クラス分類の精度と物体位置検出精度)

## 1.5 IoU(Intersection over Union)

- ・物体検出ではクラス分類だけでなく物体位置の予測精度も評価必要  
→IoU を用いて評価する
- これは正解の BB に対する予測 BB の正解適合部分の面積割合で決まる
- 混同行列の表現方法で示すと以下

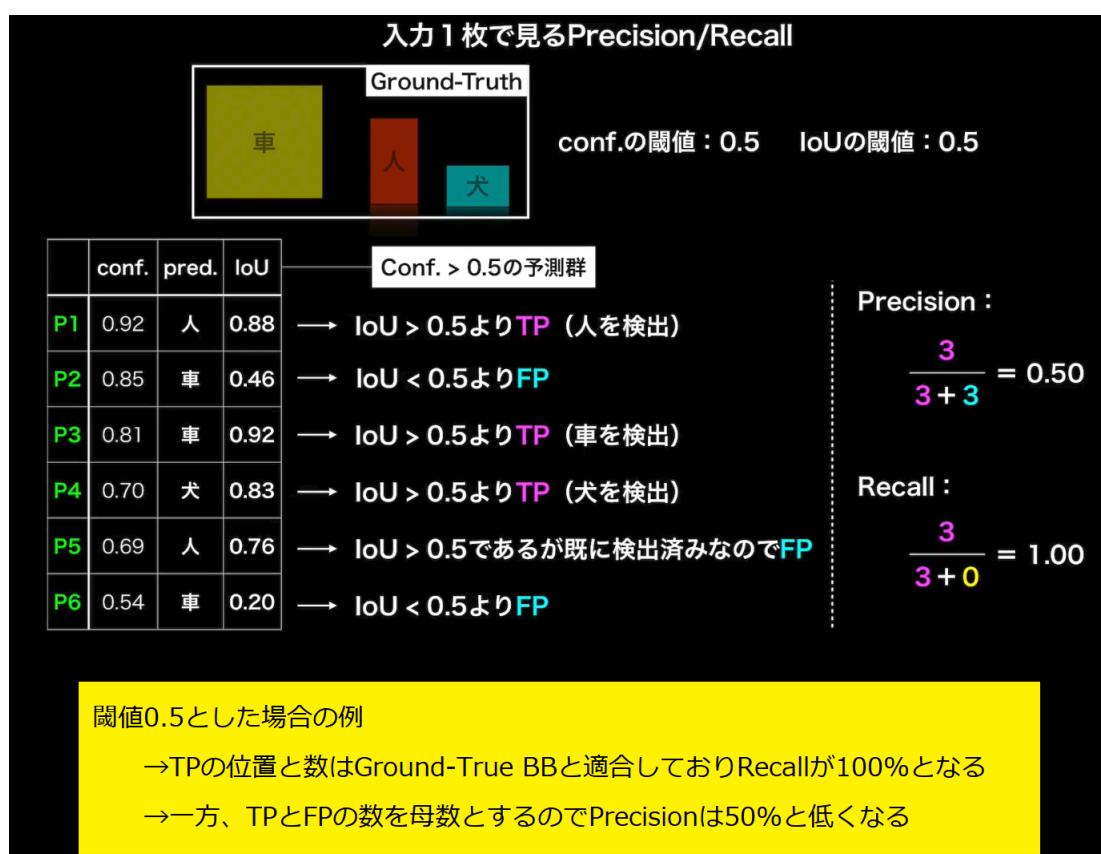


- ・つまり IoU の式は  $\text{IoU} = \frac{TP}{TP+FP+FN}$  (IoU は別名 Jaccard 係数ともいう)
- ・実際の物体検知で示すと以下の様な状態

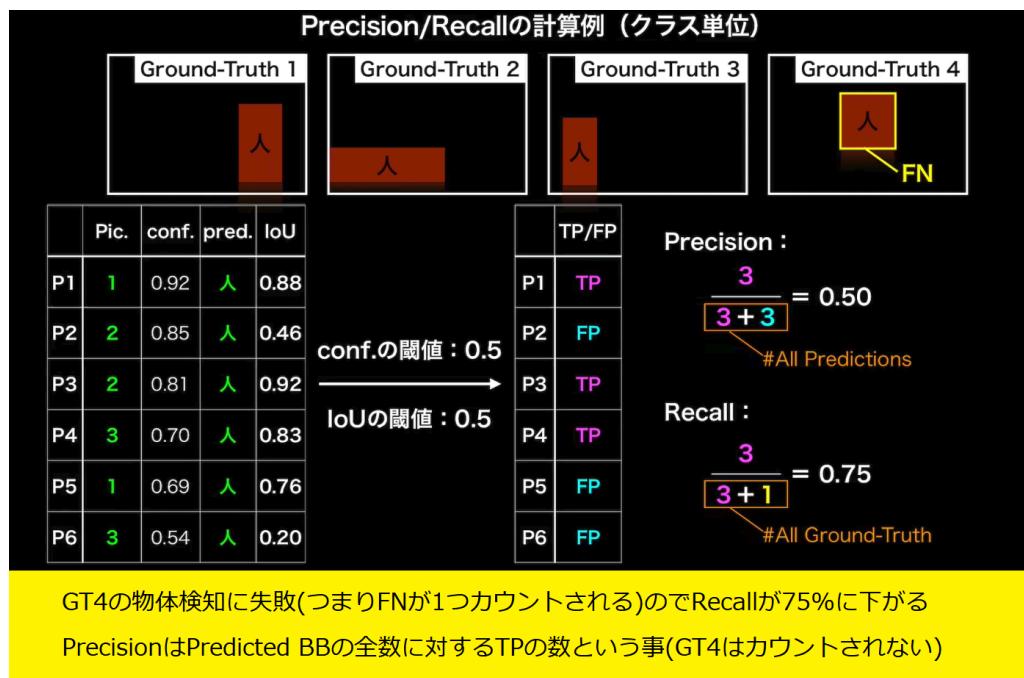


## 1.6 物体検知評価方法 (検出率に着目)

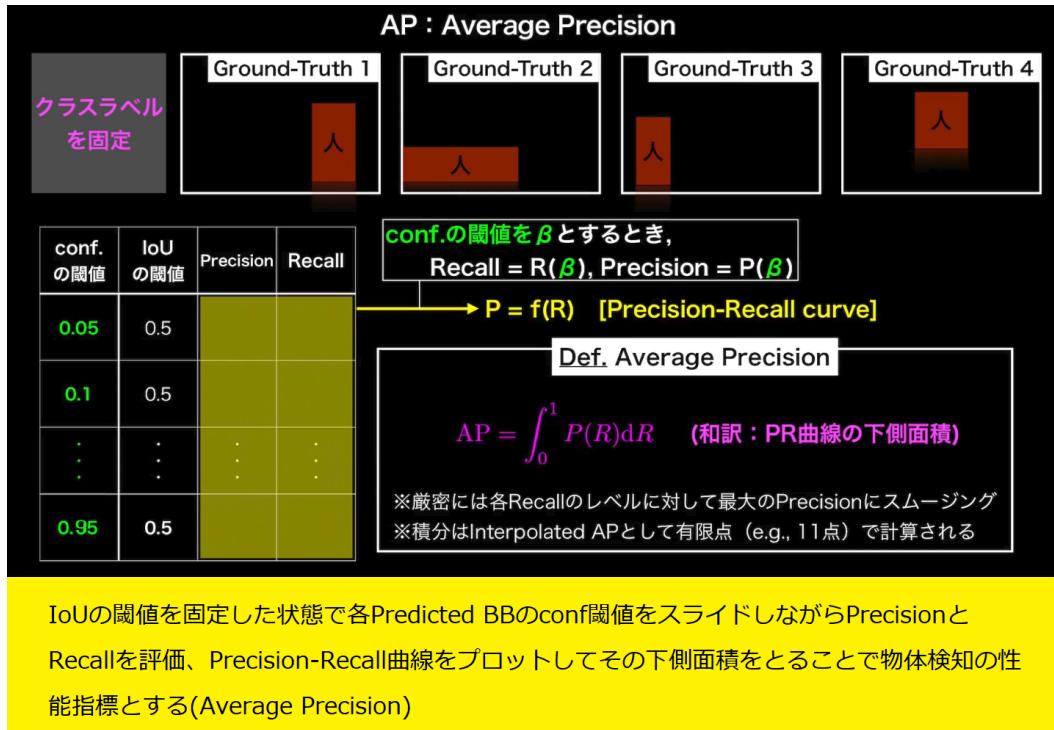
- $P_n$  はそれぞれ検出された Predicted BB を指す (ここでは物体に対して重複している)
- conf は検出 Predicted BB に対する画像識別のスコア
- IoU は前述の通り Ground-Truth BB と Predicted BB の割合を示す指標
- 同一の物体を別々の Predicted BB で捉えた場合はより成績の良い方を TP とし他は FP とする  
→ 物体検知アルゴリズムによっては別の判定かもしれない



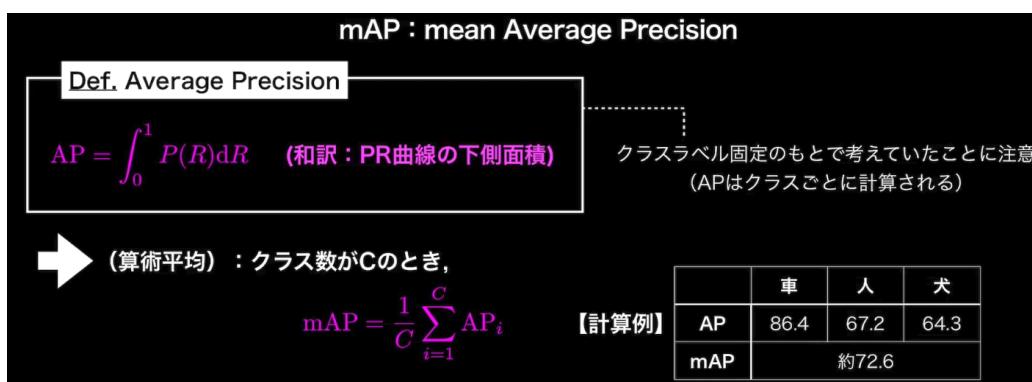
- Predicted BB の取りこぼしが発生した例



- Precision-Recall 曲線を評価指標に用いる (Note: クラス毎に算出のこと)



- Average Precision はクラス (人だったり車だったり) 每に算出との事
- 全クラスの平均をとる事でモデルの指標とする (mAP) との事
- よく、mAP と AP を混同した様な論文もあるとか (気を付けましょうとの事にて)



- MS COCO ではさらに IoU の閾値もスライドして mAP を算出 (mAPcoco)

---

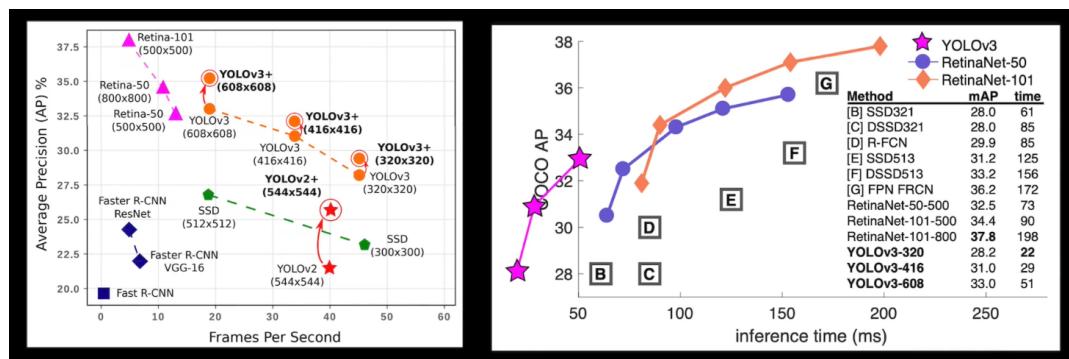
——おまけ：MS COCOで導入された指標——

ここまで、IoU閾値は0.5で固定→0.5から0.95まで0.05刻みでAP&mAPを計算し算術平均を計算

$$mAP_{COCO} = \frac{mAP_{0.5} + mAP_{0.55} + \dots + mAP_{0.95}}{10}$$

## 1.7 物体検知評価方法 (速度に着目)

- ・実用性を考慮する → 検出速度も指標となる
- ・FPS(Frames per Second) を指標とする (事が多い)
- ・FPS は 1 秒で検出できる画像枚数 (よってクラス混在等は見えないはず)
- ・ただし、以下の様に AP(mAP や mAPcoco かも) も含めて評価すると理解



- ・物体検知技術は大きく 2 つの方に分かれる

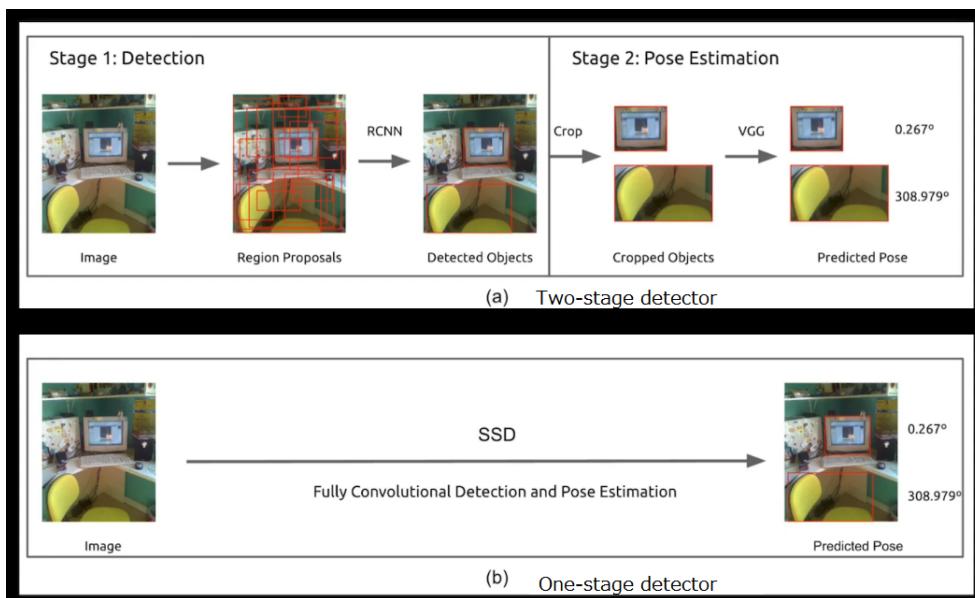
#### 2 段階検出器 (Two-stage detector)

- 候補領域の検出とクラス推定を別々に行う
- 相対的に精度が高い傾向
- 相対的計算量が大きく推論も遅い傾向

#### 1 段階検出器 (One-stage detector)

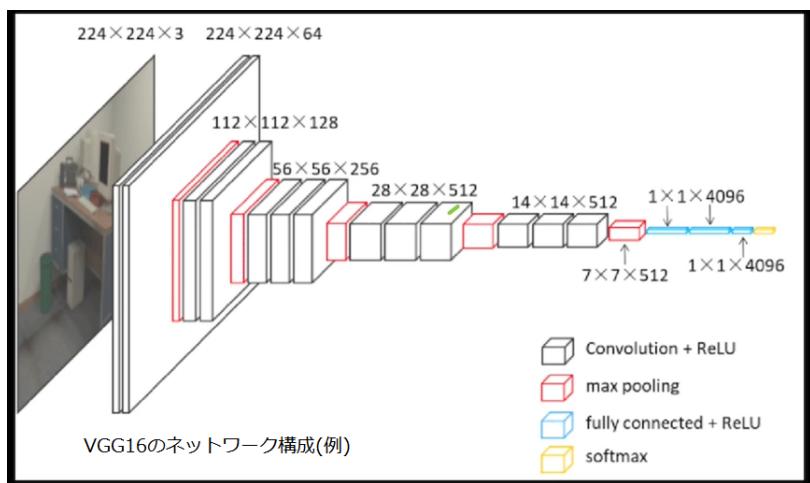
- 候補領域の検出とクラス推定を同時に行う
- 相対的に精度が低い傾向
- 相対的計算量が小さく推論も速い傾向

以下図の様な違い (SSD については後述する)

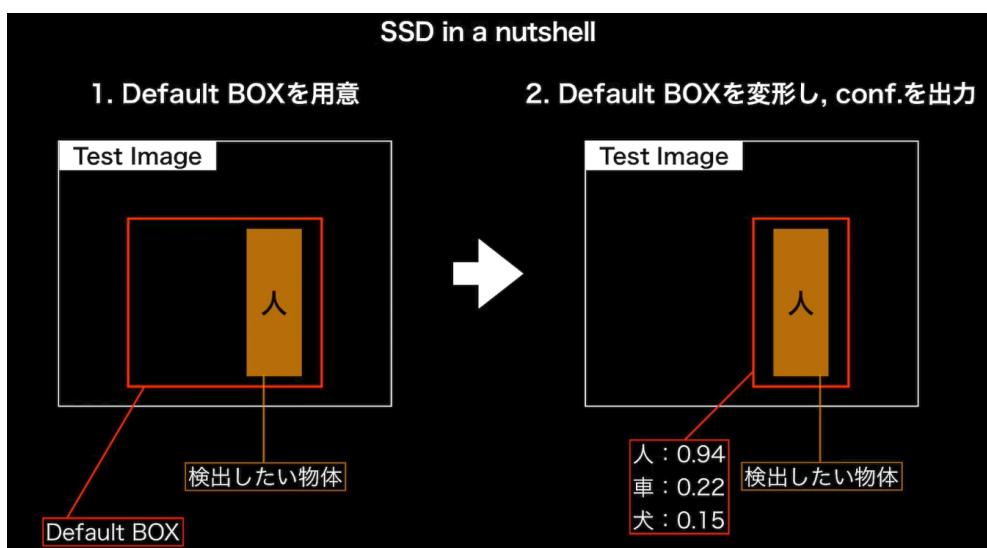


## 1.8 SSD(Single Shot Detector)

- ・1段階検出器である
- ・VGG16 ネットワーク(以下概略)をベースとしたネットワーク

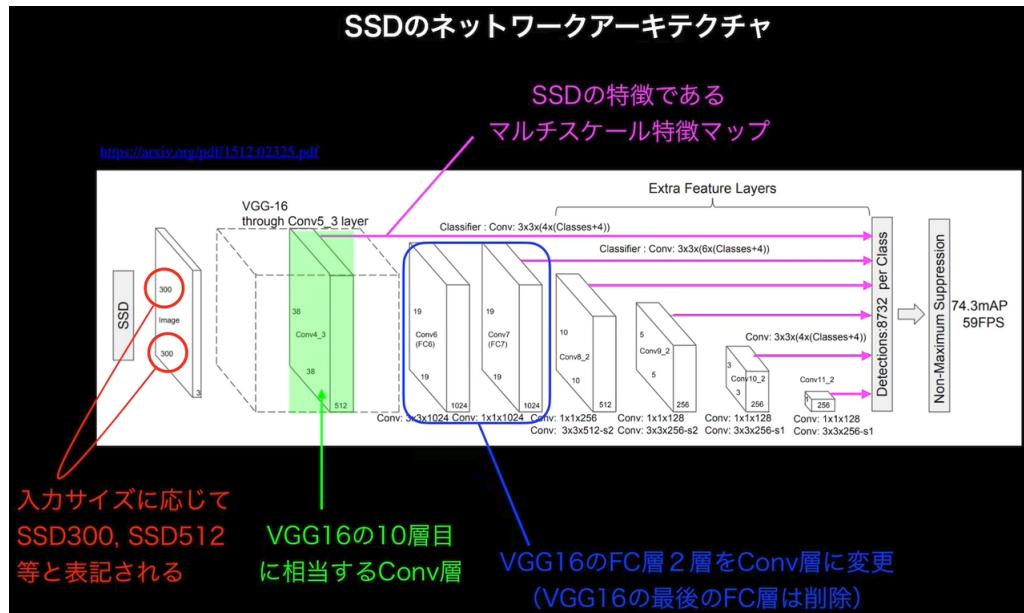


- ・VGG16 ネットワークをベースとしたネットワーク
- ・ザックリとした Default BOX を置き、IoU や pred の精度が高くなる様に学習を行う(以下図)
- ・クラス数は画像クラス数 +1(背景クラスを足す)

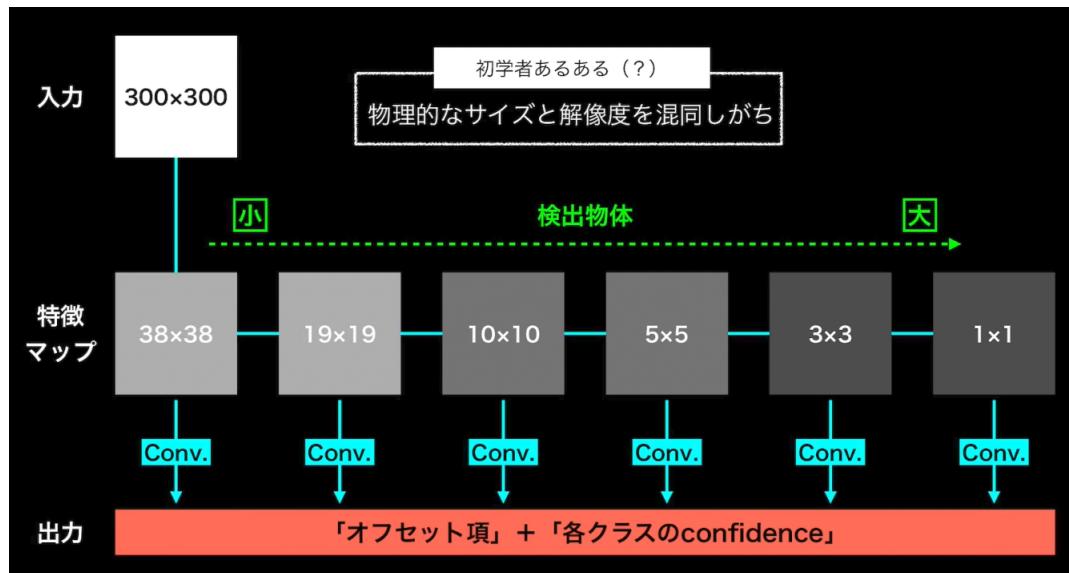


- SSD のネットワーク構成は以下 (SSD300 の場合)

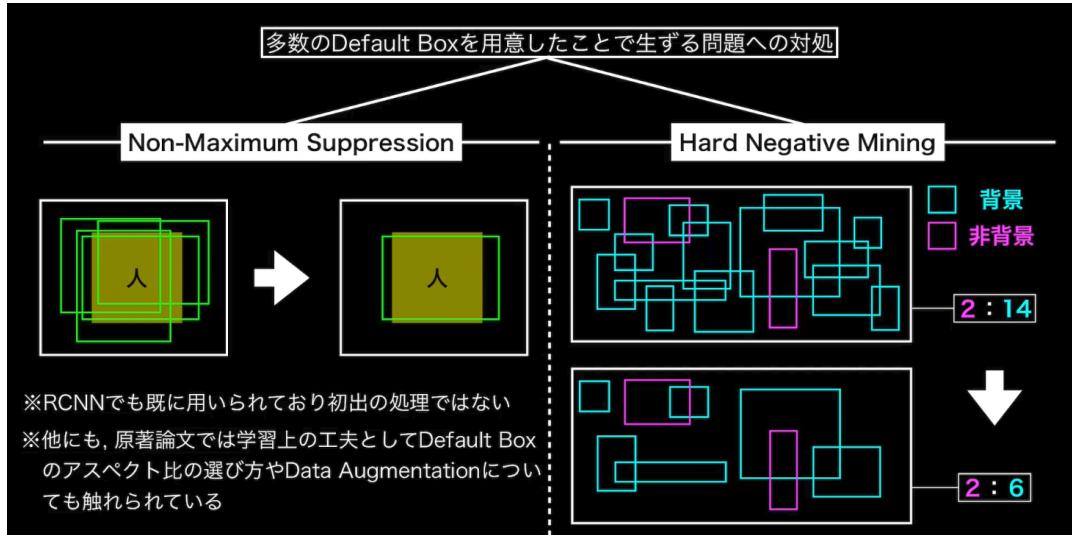
→ 解像度の異なる特徴マップを結合



→ 解像度の異なる特徴マップの=画像の中の物体の大きさに依存と説明



- 重なる Predicted BB への対処
  - IoU の値や pred. 値を評価して選択 (NonMaximumSuppression)
- ノイズ的に作用する Predicted BB への対処
  - 物体に重ならない (背景にのみ重なる)BB を排除 (HardNegativeMining)



- 損失関数の考え方 → 位置検出と conf の両方の Loss を考慮して定義

損失関数

$$L(x, c, l, g) = \frac{1}{N} (L_{conf}(x, c) + \alpha L_{loc}(x, l, g))$$

Faster RCNNでも用いられる Smooth L1 loss

検出位置に対する損失

$$L_{loc}(x, l, g) = \sum_{i \in Pos}^N \sum_{m \in \{cx, cy, w, h\}} x_{ij}^k \text{smooth}_{L1}(l_i^m - \hat{g}_j^m)$$

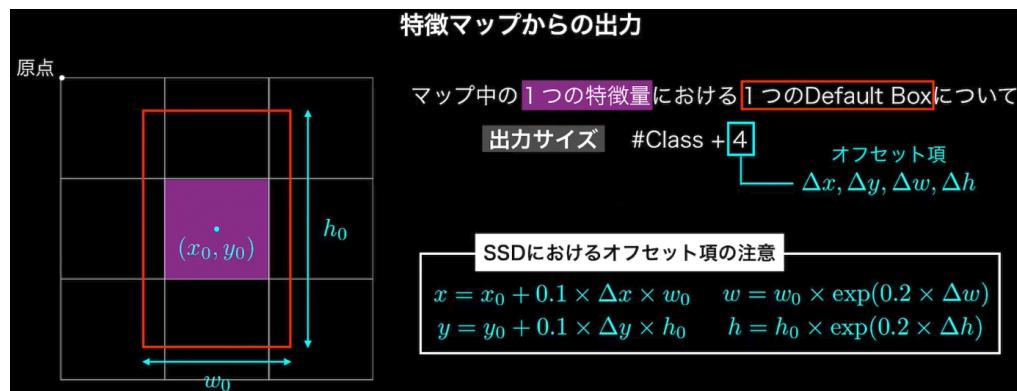
$$\hat{g}_j^{cx} = (g_j^{cx} - d_i^{cx})/d_i^w \quad \hat{g}_j^{cy} = (g_j^{cy} - d_i^{cy})/d_i^h$$

$$\hat{g}_j^w = \log \left( \frac{g_j^w}{d_i^w} \right) \quad \hat{g}_j^h = \log \left( \frac{g_j^h}{d_i^h} \right)$$

confidenceに対する損失

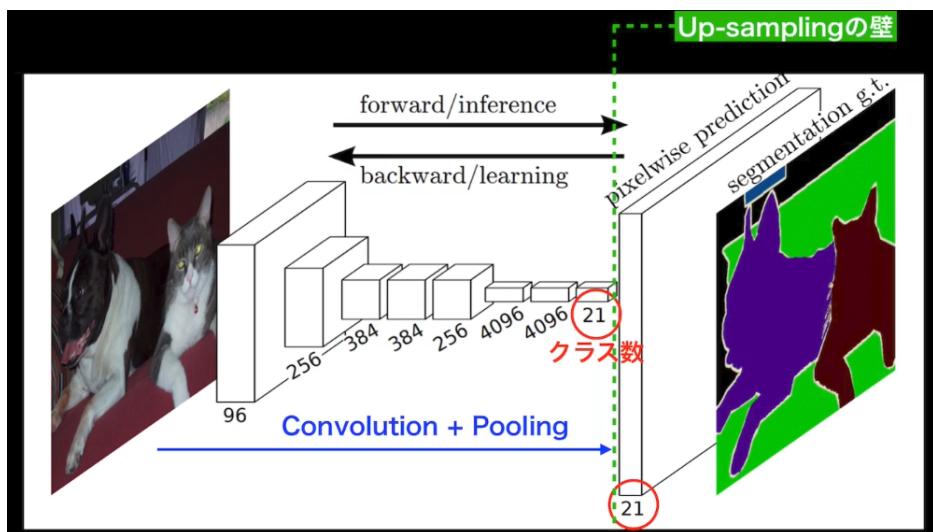
$$L_{conf}(x, c) = - \sum_{i \in Pos}^N x_{ij}^p \log(\hat{c}_i^p) - \sum_{i \in Neg} \log(\hat{c}_i^0) \quad \text{where} \quad \hat{c}_i^p = \frac{\exp(c_i^p)}{\sum_p \exp(c_i^p)}$$

- 位置検出の出力は結局は DefaultBox の誤差をオフセットを調整して埋めるという考え方



## 1.9 Semantic Segmentation 概略

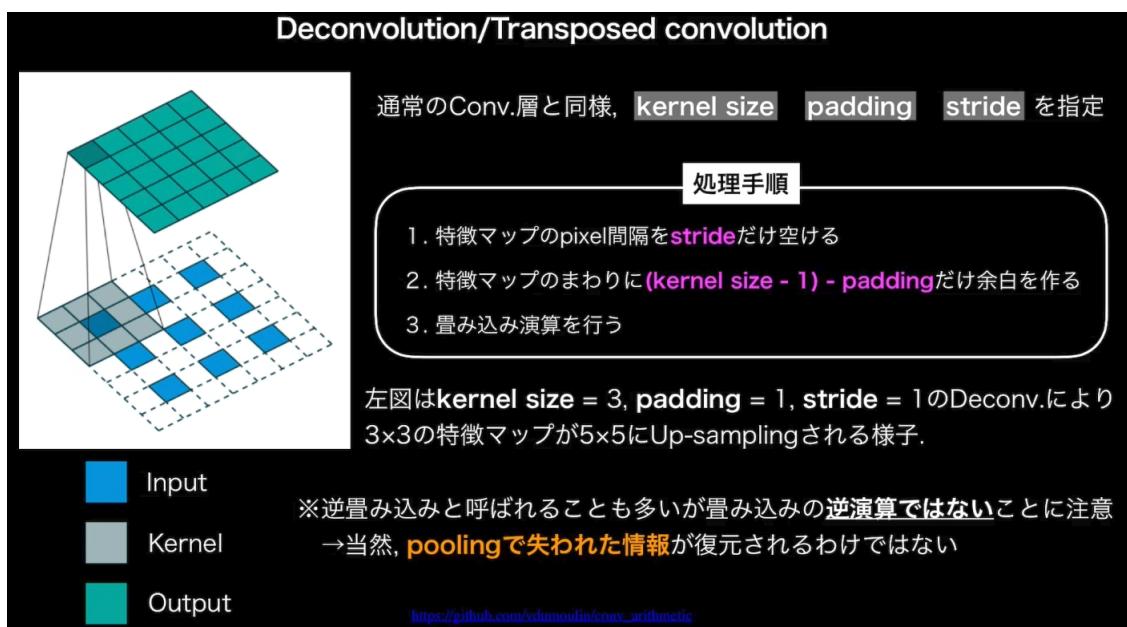
- Semantic Segmentation: 画像のピクセルがどのクラスに属しているかを分類する手法



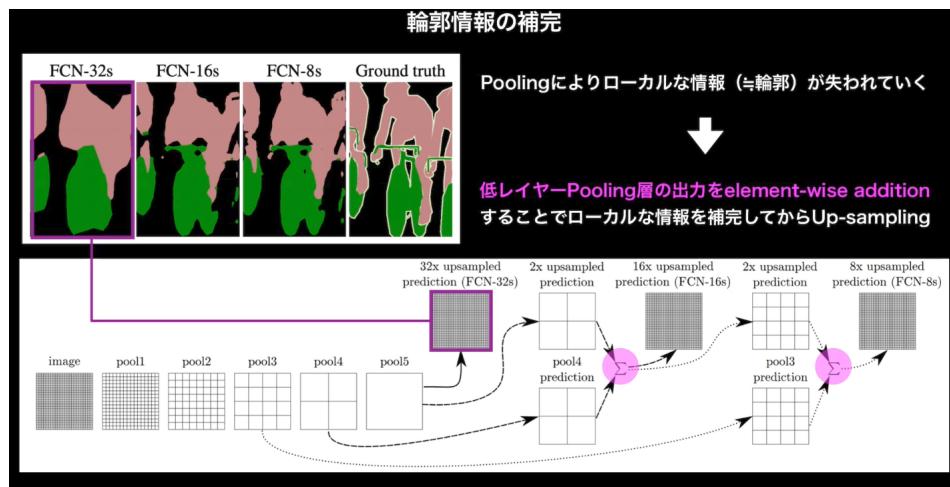
- 講義では「画像の解像度が落ちてしまう問題」を挙げている
  - 畳み込み + pooling を重ねると解像度が落ちている（意図的に落としている）
  - しかし元画像の輪郭は再現したい、という事
- Semantic Segmentation を行いつつ画像の解像度を元に戻す（近づける）方法の紹介
  - UpSampling 手法を紹介する

- Deconvolution/Transposed convolution

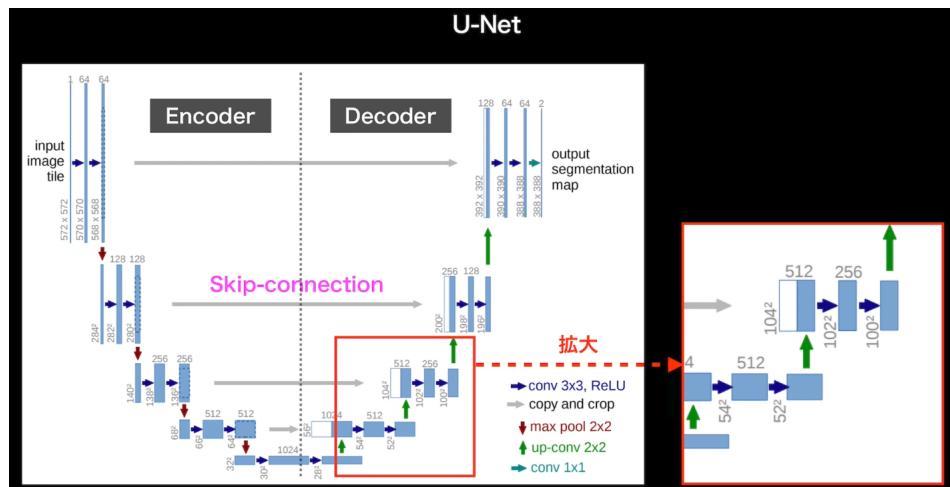
- 特徴マップのピクセル間にストライド幅分の隙間を開ける
- 特徴マップの見た目上の解像度を上げる
- 隙間を開けた特徴マップを畳み込み演算する
- 結果的に特徴マップサイズの解像度を上げる事になる (が情報が復元するわけではない)



- 輪郭情報の補完 (Fully Convolutional Network(FCN))
  - deconvolution で解像度を戻した特徴マップに対して輪郭を加算する
  - 加算する特徴マップは畳み込み前の特徴マップ (同じ解像度)
  - 下図の様に段階的に戻し、Ground truth の状態を復元する

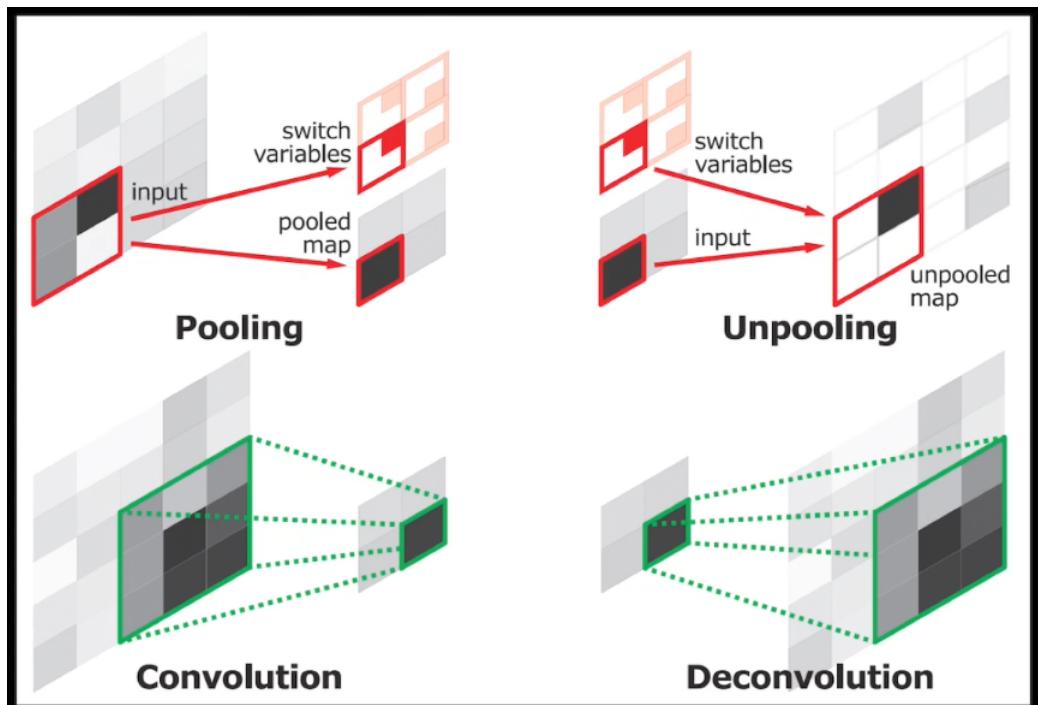


- U-Net
  - FCN と似ているが UpSampling 時に加算と畠み込みを実施
  - Decoder-Encoder モデルでもある



- Unpooling

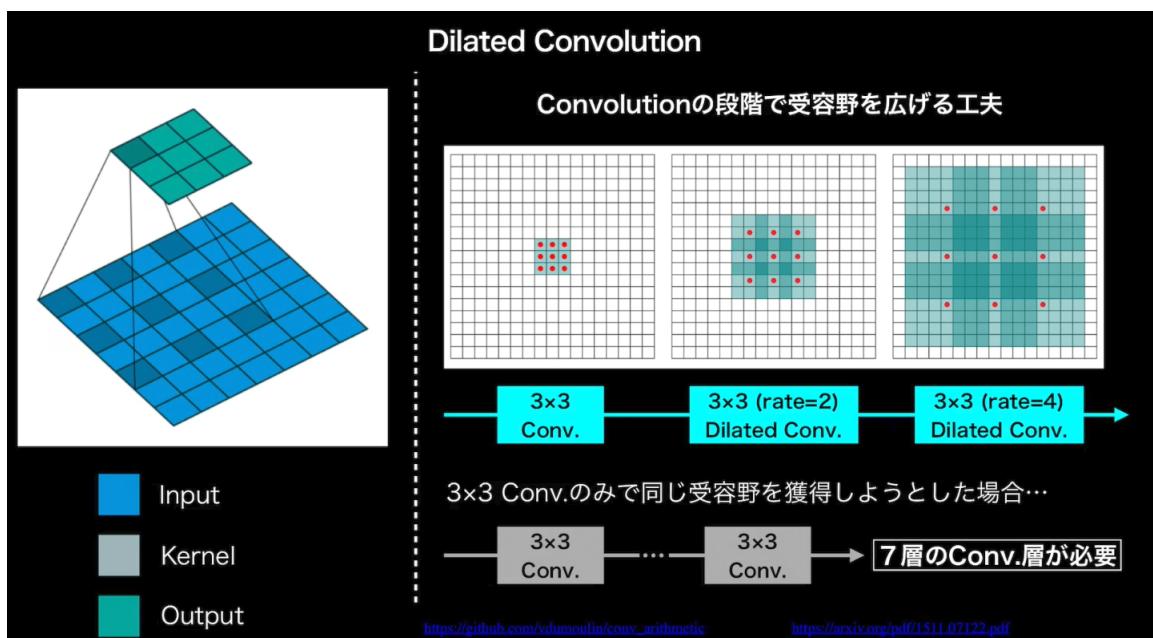
→ 特徴マップ解像度を戻す手法だが位置情報の復元も行う



- Dilated Convolution

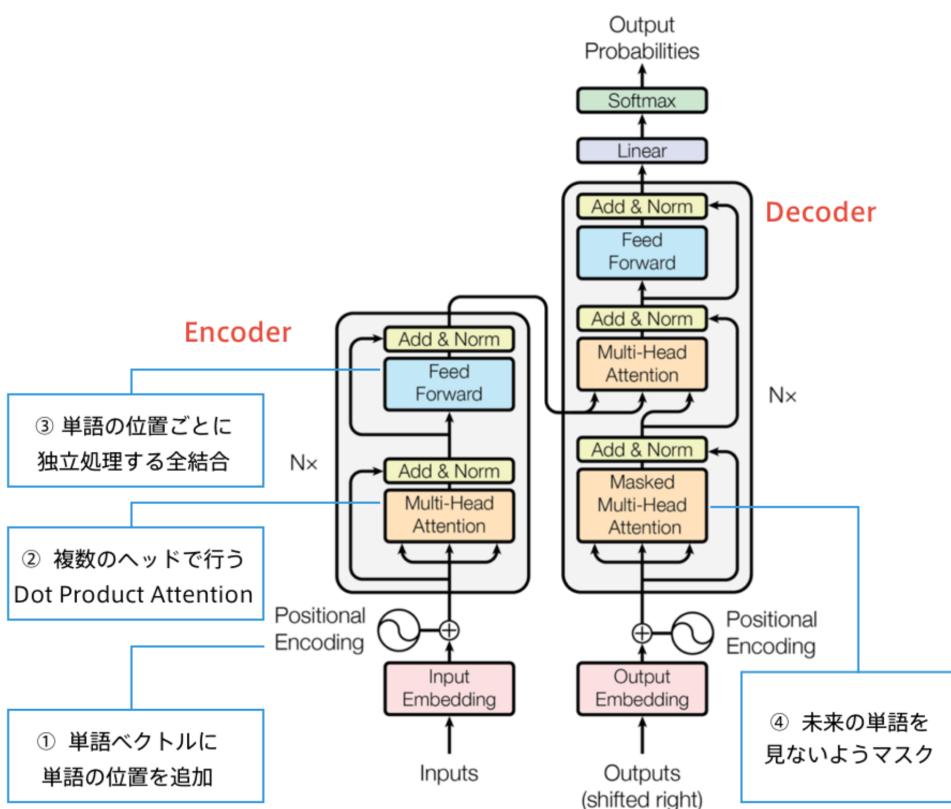
→ Pooling を行わずに解像度を上げる方法

→ 処理層を減らす効果も期待できる



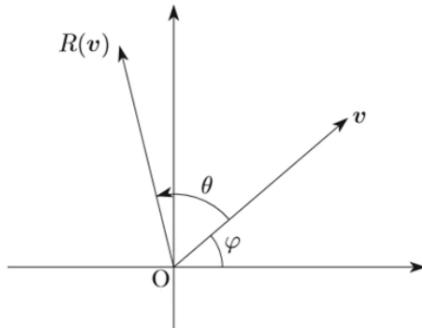
## 2 Transformer

- Transformer は Encoder-Decoder モデルに Attention 機構を導入したもの
- この機構は Google の自然言語処理モデルの BERT に採用され、その当時の SOTA を更新した
- BERT は RNN ではなく純粋に Transformer だけで構成されているとの事



## 2.1 位置情報埋め込みについて

- Transformer では単語の位置情報埋め込みはベクトルの角度重畠で行う  
→ 理解の為理論の元となる原理をまとめてみた



二次元ベクトルvの角度をθだけ移動する場合=>R(v)に移動

まず、二次元ベクトル  $v$  を  $\cos$  と  $\sin$  で座標表現  $v = (s, t) = (|v|\cos\varphi, |v|\sin\varphi)$   
ベクトル  $v$  に対して角度  $\theta$  だけ回転させると  $R(v) = (|v|\cos(\varphi + \theta), |v|\sin(\varphi + \theta))$

上記に対して三角関数の加法定理より変形を行うと

$$R(v) = |v|(\cos(\varphi)\cos(\theta) - \sin(\varphi)\sin(\theta), \sin(\varphi)\cos(\theta) + \cos(\varphi)\sin(\theta))$$

これを行列表現にして変形を行うと

$$\begin{aligned} R(v) &= |v| \begin{pmatrix} \cos(\varphi)\cos(\theta) - \sin(\varphi)\sin(\theta) \\ \sin(\varphi)\cos(\theta) + \cos(\varphi)\sin(\theta) \end{pmatrix} = |v| \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} \begin{pmatrix} \cos(\varphi) \\ \sin(\varphi) \end{pmatrix} \\ &= \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} \begin{pmatrix} |v|\cos(\varphi) \\ |v|\sin(\varphi) \end{pmatrix} = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} \begin{pmatrix} s \\ t \end{pmatrix} \end{aligned}$$

つまり  $R(v)$  は元のベクトル  $v$  に対して

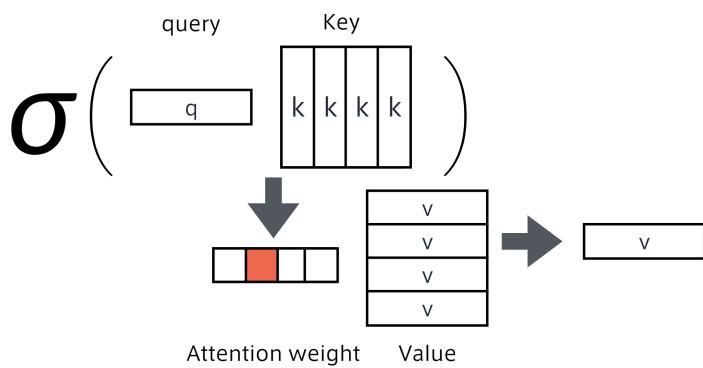
$$\begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix}$$

を積算したものとなる

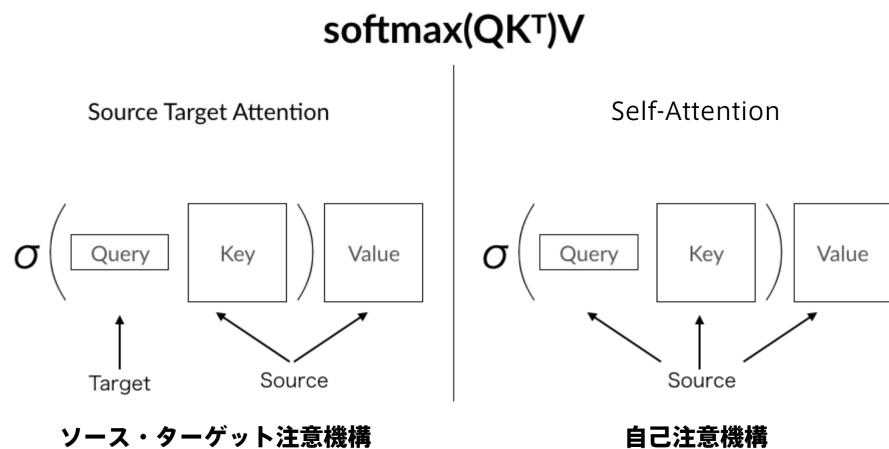
Transformer では  $\theta = pos/10000^{\frac{2i}{512}}$  と定義

## 2.2 Attention(注意)

- 辞書構造 (Query より Key を検索しさらに QV 類似度 (重み) に応じた Value を得る)

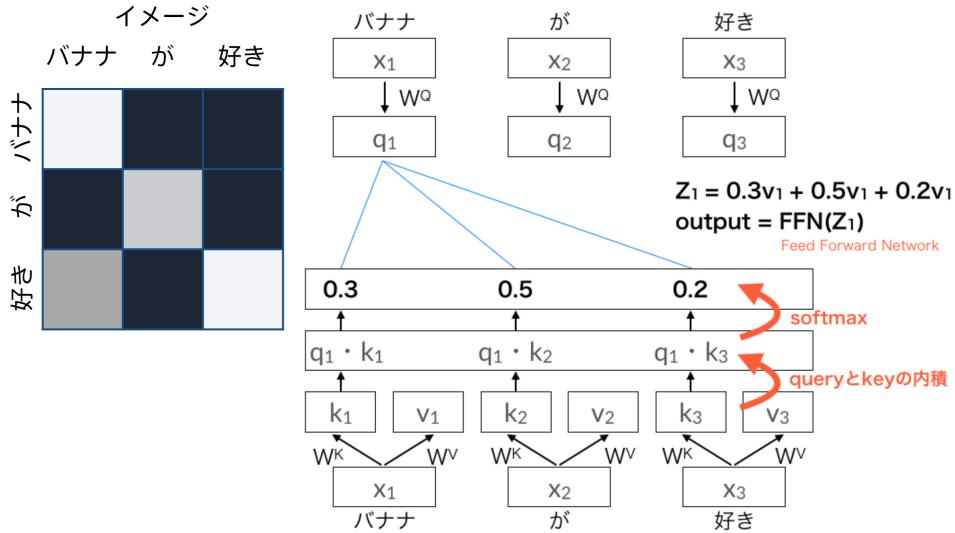


- Query を得る先に応じて Attention は二種類存在する
- Transformer では Self-Attention と Source Target Attention で構成される



- 文章を構成する (例えば) 単語毎に処理を行うが、単語の重み (softmax され合計 1) をかける
  - この措置により長い文章の処理を行える (=意味が薄い単語を無視)
  - RNN の欠点 (固定長単語ベクトルの長文章表現力問題) を克服

- Self-Attention は入力の文章のみより (結果的に) その文章を再現する方法  
→Transformer の主要な Attention である



- その文章の意味より出力すべき単語を判定する様な機構となる (以下、個人的解釈)
  - 結果的に同じ文章が確率的に再現されると理解
  - 多分、未学習の単語を含む似た文章を入力した際に似た意味の出力に誘導する措置と理解
  - 位置重畠も「同じ様な位置にある単語を含んだ同じ様な意味の文章」に寄せる為と理解

- Source Target Attention の入力は Query のみ他からの入力
  - Transformer の Encoder から Decoder への入力パス
  - 翻訳やチャットボット等「入力に強く関連する別の出力を得る」と理解
- Multi-Head Attention 化する事で様々な表現を得る事ができる
  - CNN の畳み込み処理にてフィルタを適応的に作り様々な特徴マップを得るのと同じ思想
  - Transformer では 8 つを重ねて構成

### 3 実習

- 4\_1\_tensorflow\_codes.ipynb
- 4\_3\_keras\_codes.ipynb
  - try 項目の追加と簡単な考察を実施
- lecture\_chap1\_exercise\_public.ipynb
- lecture\_chap2\_exercise\_public.ipynb
  - 実行 (ただし chap1 は動作環境マッチングの為改変した)
- 本レポートと同じレポジトリにそれぞれの実装コードを置く