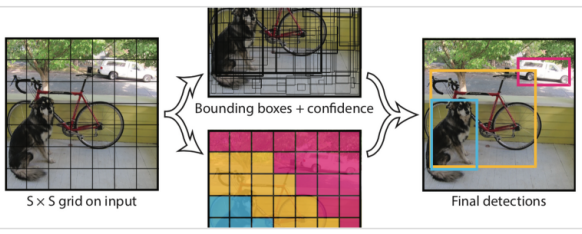
YOLO(2015)

갈아먹는 Object Detection [5] Yolo: You Only Look Once

갈아먹는 Object Detection [1] R-CNN 갈아먹는 Object Detection [2]Spatial Pyramid Pooling Network 갈아먹는 Object Detection [3] Fast R-CNN 갈아먹는 Object Detection [4] Faster R-CNN 오늘 리뷰할 논문은 real time object detection의 혁명을 몰고 온 yolo입니다. 우선 결과부터 보시죠. Yolo는 2015년에 나온 논문으로 Faster R-CNN에 비하여 부려 6배 가량 빠른 속도를 보입니다.

https://yeomko.tistory.com/19?category=888201



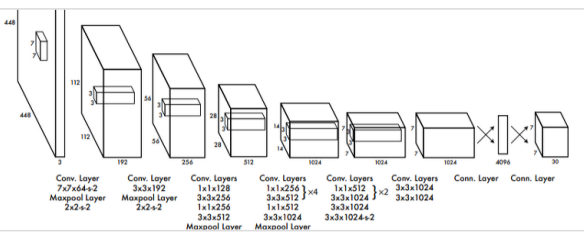
https://arxiv.org/pdf/1506.02640.pdf

https://herbwood.tistory.com/13?category=856250

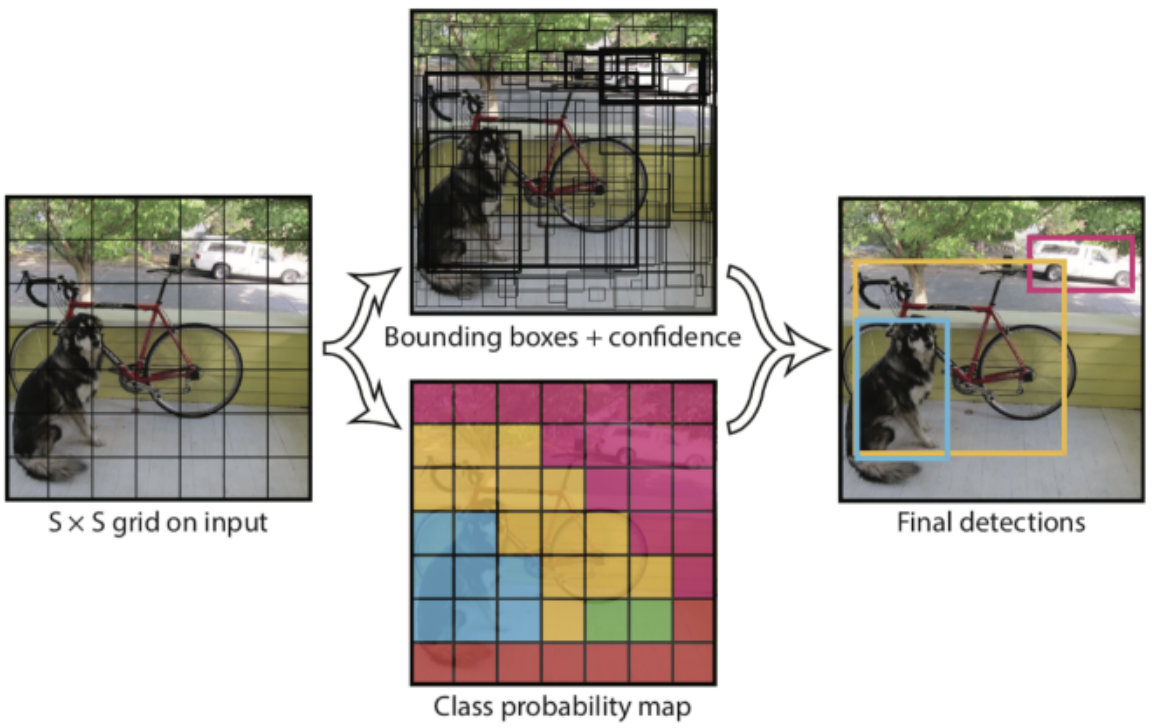
Pytorch로 구현한 YOLO v1 모델

이번 포스팅에서는 pytorch로 구현한 YOLO v1 모델의 코드를 분석해보도록 하겠습니다. 코드 구현체를 찾다가 우연히 Aladdin Persson 님이 올리신 "Pytorch YOLO From Scratch" 영상을 보게 되었는데 설명이친절하고 코드가 깔끔하여 참고하기 좋다는 생각을 하게 되었습니다. aladdinpersson님의 github repository 에 올라온 코드를 보면서 YOLO v1 모델의 전체적인 학습 과정을 살펴보도록 하겠습니다.

https://herbwood.tistory.com/14?category=867198

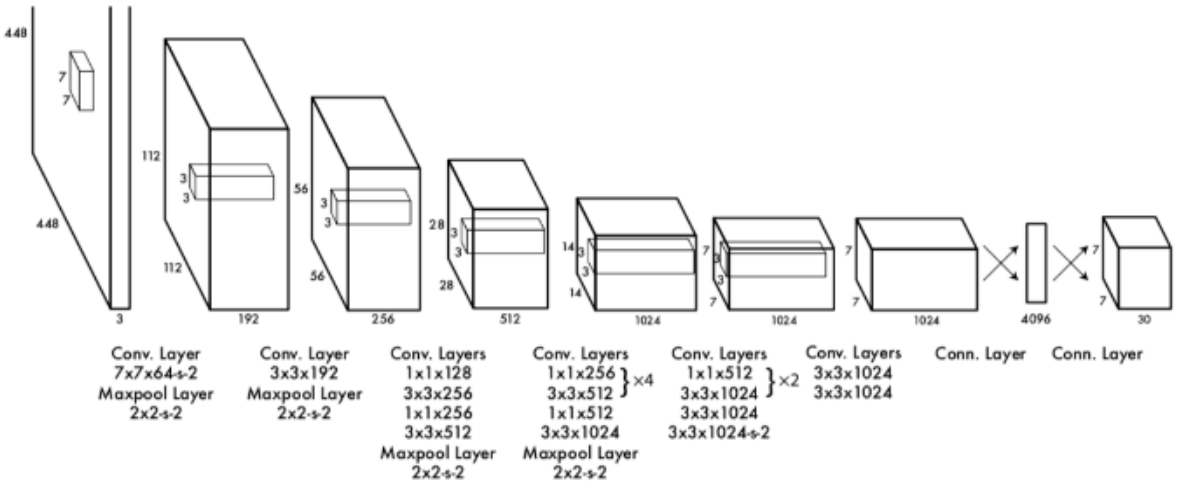


YOLO는 region proposal 단계를 제거하고 한번에 object detection 을 수행하는 구조를 갖는다.

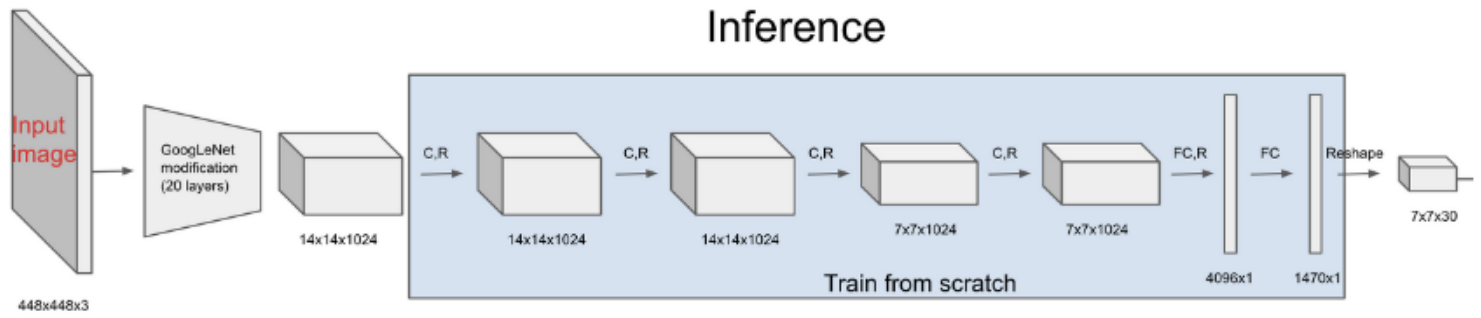


입력 이미지를 s*s 그리드 영역으로 나누고 각 그리드 영역에서 물체가 있을 만한 영역에 해당하는 특정 개수만큼의 bounding box 를 예측한다. bounding box는 (x,y,w,h) 로 나타난다. 다음으로 해당 박스의 신뢰도를 나타내는 confidence 를 계산한다. confidence는 해당 그리드에 물체가 있을 확률과 예측한 박스-ground truth 사이의 IOU 를 곱해주는 연산의 결과이다. confidence 계산이 끝나면 각각의 그리드마다 c 개의 클래스에 대해 해당 클래스일 확률을 계산한다.

YOLO는 입력 이미지를 그리드로 나누고, 각 그리드 별로 bounding box 와 classification 을 동시에 수행한다.



Yolo Network



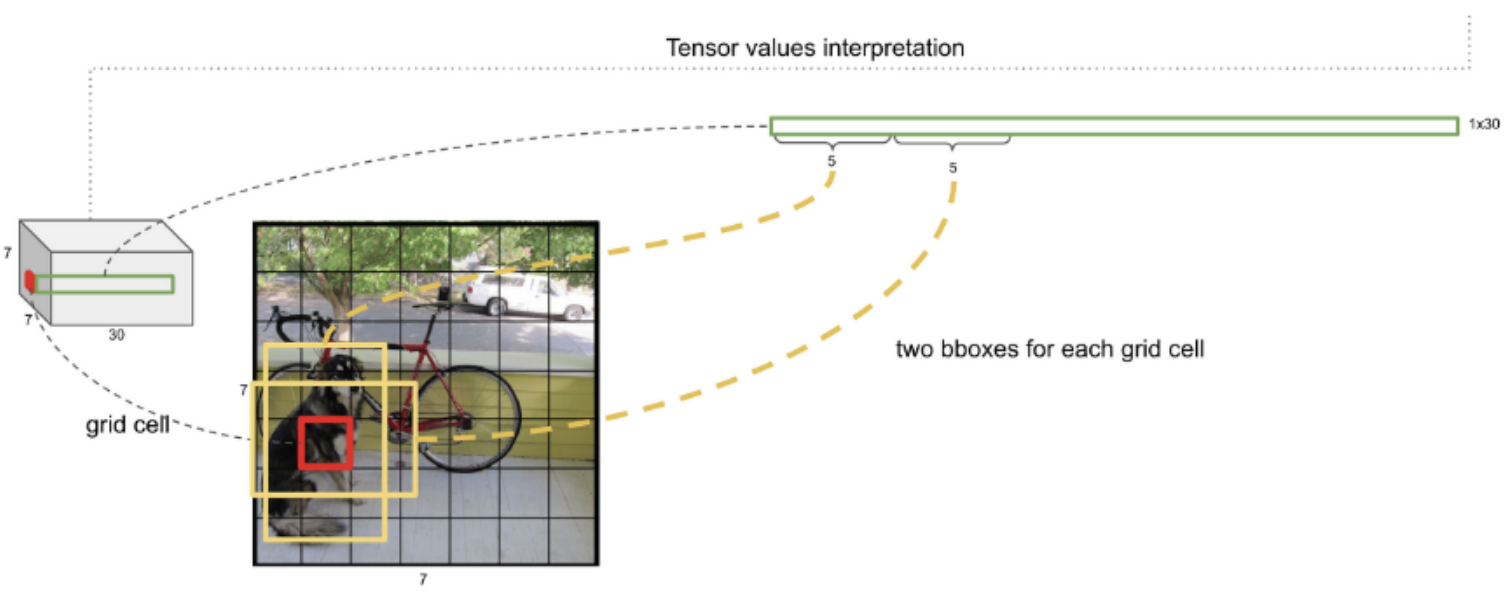
Yolo Network

저자가 GoogleNet 의 구조에서 영감을 받았고, inception block 대신 단순한 convolution 으로 네트워크를 구성했다고 한다. YOLO 네트워크에 관해 더 직관적인 이미지는 아래와 같다.

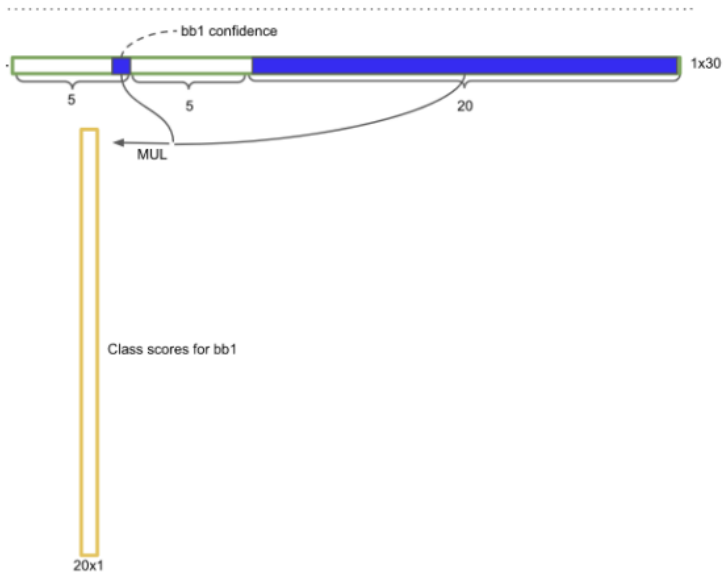
네트워크의 출력에 관해

YOLO 네트워크의 출력은 7*7*30 feature map 이다. 여기 안에는 그리드 별 바운딩 박스와 ㅅ니뢰도 지수, 각 클래스 별 예측값이 담겨져 있다.

먼저 7*7 은 그리드를 의미하며, 각각의 인덱스는 총 30차원의 벡터 값을 가진다.

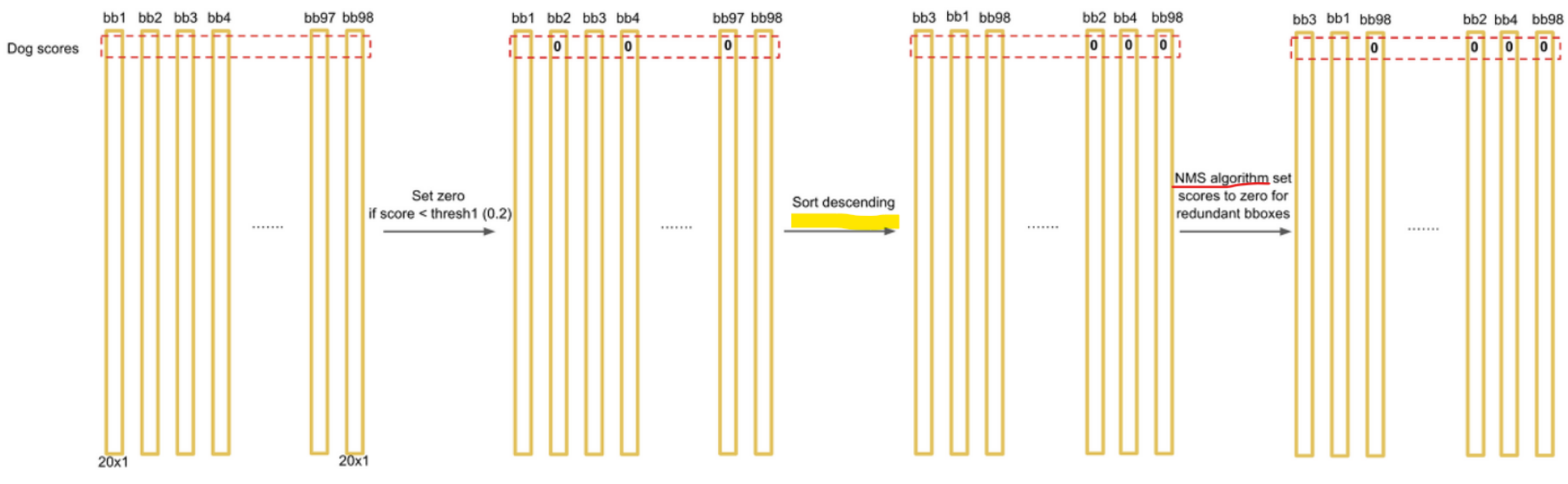


7*7 그리드 가운데 붉은 색 박스는 하나의 인덱스를 의미한다. 그림에서는 하나의 인덱스에서 2개의 바운딩 박스를 추측했다. 30차원 가운데 앞의 10개의 수는 바운딩 박스 2개에 대한 정보 값이다. 각각의 바운딩 박스 당 5개의 정보가 있는데 (중심점 x, 중심점 y, 너비, 높이, 신뢰도 지수) 으로 구성된다.



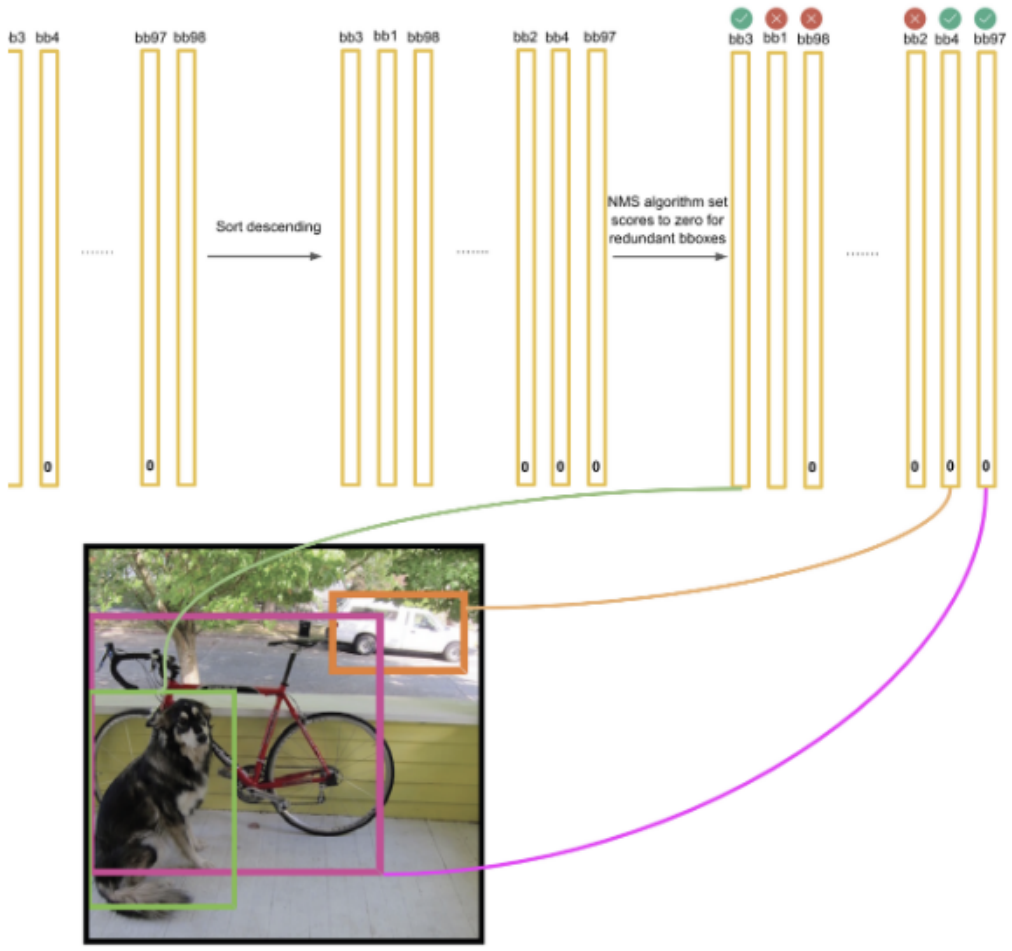
30차원 중 앞의 10 차원이 바운딩 박스에 관한 것이었다면, 뒷단의 20차원 벡터는 해당 인덱스가 특정 클래스일 확률 값들이며, 여기서는 클래스가 20인 데이터 셋을 사용하였기 때문에 20차원 벡터로 표현된다.

우리는 박스의 신뢰도를 Pr(obj) * IoU 로 구했고, 각 클래스별 확률 값을 구할 때는 Pr(class i | obj) 로 구했다. 따라서 이 둘을 곱하면 Pr(class i ∩ obj) * IoU 가 되고, 이는 곧 해당 박스가 특정 클래스일 확률 값이 된다. 이제 이 작업을 인덱스 i 의 모든 바운딩 박스에 적용하고 이를 다시 모든 인덱스에 적용하면 다음과 같은 결과를 얻는다.



이렇게 구한 벡터들을 모두 모든 뒤 일렬로 정렬하면, 가장 위 차원부터 각 클래스별로 전체 바운딩 박스에서의 확률 값을 구할 수 있다. 여기에는 동일한 물체에 중복되어 지정된 박스들도 많을 것이다. 이를 방지하기 위해 NMS 라는 작업을 거치고 최종 결과를 이미지 위에 그려주면 된다.

NMS 를 거치게 되면 벡터들의 대부분의 값들은 0이 된다.



Loss function

YOLO 의 저자들은 아주 세심한 Loss function 을 고안했다.

Loss function 의 앞단은 다음과 같다.

$$\lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+ \lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$

Yolo Loss Front Half

1 obj ij 라는 기호는 object 가 등장하는 i 인덱스의 j 번째 바운딩 박스가 최종 예측값을 산출한 것을 의미한다. 앞서 우리는 NMS 를 거쳐서 0이 되지 않은 일부 바운딩 박스만 최종 예측값에 포함시켰다. 따라서 Loss function ㅇ르 구할 때도 살아남은 이 박스들을 찾아서 loss 를 구하려는 것이다.

위에서 제시된 loss function 의 앞 단이 의미하는 것은 다음과 같다.

최종 예측에 포함된 바운딩 박스를 찾아 내어 x,y 좌표, w,h 값, 신뢰도 c 값이, 예측값과 ground truth 값의 차이를 구해 모두 더해 준다. 이 때, x,y,c 값은 단순 차를 구했고 w,h 는 비율 값이기 때문에 루트를 씌워 차이를 구해준 점이 다르다. 앞에 붙는 람다는 물체가 있을 때의 오차와 없을 때의 오 차 간의 비율을 맞춰주기 위한 값이다. (논문에서는 모두 5로 설정)

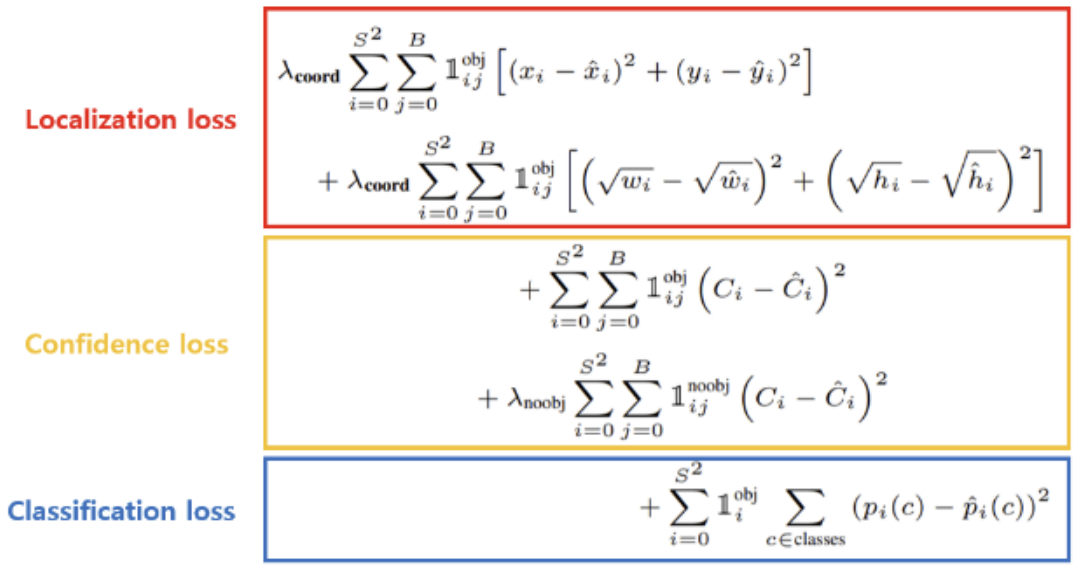
$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i\right)^2$$

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c)\right)^2$$

Loss function 의 나머지 뒷단은 위와 같다. 못 찾아낸 물체들에 대한 penalty 를 매기는 것이 목적이다. 1 noobj ij 라는 것은 물체가 없다고 판단된 i 인덱스의 j 번째 바운딩 박스가 사실은 가장 ground truth 와 IoU 가 가장 높은 인덱스를 말한다. 즉, 물체로 찾아냈어야 하는데 못찾아낸 인덱스이다. 이에 대해선 찾아냈어야 하므로 신뢰도 C 값의 차이를 구해 LOSS 에 더해준다. 마지막으로 모든 물체가 있다고 판단된 인덱스 I 들에 대해서 모든 클래스들에 대한 예측값과 실제값의 차를 구해 더해준다.

Loss function 에 대해 좀 더 구체적으로

기존 R-CNN 계열이 classification, localization task 에 맞게 서로 다른 loss function 을 사용했던 것과 달리 YOLO v1 모델은 regression 시 주로 사용되는 SSE(sum of squared error)를 사용한다.



$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2\right]$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i}\right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i}\right)^2\right]$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i\right)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i\right)^2$$

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c)\right)^2$$

YOLO v1 loss function

아래에서 나오는 구체적인 설정값은 논문에서의 설정을 따른다.

Localization loss

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$
$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

Localization loss

- λ_{coord} : 많은 grid cell은 객체를 포함하지 않아 confidence score가 0이 되어 객체를 포함하는 grid cell의 gradient를 압도하여, 모델이 불안정해질 수 있습니다. λ_{coord}는 이러한 문제를 해결하기 위해 객체를 포함하는 cell에 가중치를 두는 파라미터입니다. 논문에서는 $\lambda_{coord} = 5$로 설정합니다.
- S^2 : grid cell의 수(=7x7=49)
- B : grid cell별 bounding box의 수(=2)
- $1_{i,j}^{obj}$: i번째 grid cell의 j번째 bounding box가 객체를 예측하도록 할당(responsible for)받았을 때 1, 그렇지 않을 경우 0인 index parameter입니다. 앞서 설명했듯이 grid cell에서는 B개의 bounding box를 예측하지만 그 중 confidence score가 높은 오직 1개의 bounding box만을 학습에 사용합니다.
- x_i, y_i, w_i, h_i : ground truth box의 x, y 좌표와 width, height. 여기서 크기가 큰 bounding box의 작은 오류가 크기가 작은 bounding box의 오류보다 덜 중요하다는 것을 반영하기 위해 w_i, h_i 값에 루트를 씌어주게 됩니다.
- $\hat{x}_i, \hat{y}_i, \hat{w}_i, \hat{h}_i$: 예측 bounding box의 x, y 좌표, width, height

Confidence loss

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$
$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

Confidence loss

- λ_{noobj} : 앞서 언급한 객체를 포함하지 않는 grid cell에 곱해주는 가중치 파라미터입니다. 논문에서는 $\lambda_{noobj} = 0.5$로 설정했습니다. $\lambda_{obj} = 5$로 설정한 것에 비해 상당히 작게 설정하여 객체를 포함하지 않은 grid cell의 영향력을 줄였습니다.
- $1_{i,j}^{noobj}$: i번째 grid cell의 j번째 bounding box가 객체를 예측하도록 할당(responsible)받지 않았을 때 1, 그렇지 않을 경우 0인 index parameter입니다.
- C_i : 객체가 포함되어 있을 경우 1, 그렇지 않을 경우 0
- $\hat{C}_i$: 예측한 bounding box의 confidence score

Classification loss

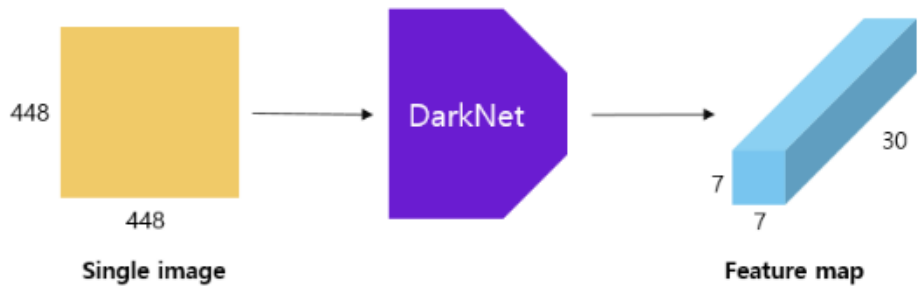
$$+\sum_{i=0}^{S^2}\mathbb{1}_i^{\text{obj}}\sum_{c\in\text{classes}}(p_i(c)-\hat{p}_i(c))^2$$

Classification loss

- $p_i(c)$: 실제 class probabilities
- $\hat{p}_i(c)$: 예측한 class probabilities

Training YOLO v1

448*448 이미지를 darknet 에 통과시켜 7*7*30 feature map 을 얻는 것으로 학습 과정은 단순하다.



Summary

- sliding window 방식이나 region proposal 기반의 모델과는 달리 YOLO v1 모델은 전체 이미지를 인지하여 contextual information 을 학습한다고 한다. 이를 통해 배경 영역을 객체로 인식하는 False Positive 오류를 Fast R-CNN 모델보다 상대적으로 줄일 수 있다고 한다.

- BASE network 의 경우 45fps 의 속도를, 경량화한 버전은 150fps 의 속도를 낸다고 한다. 당시의 R-CNN 모델과 비교했을 때 객체 탐지에 상당히 빠른 속도였다.

- 정확도 측면에서 당시 SOTA 보다는 부족한 성능. 특히 작은 객체를 제대로 탐지하지 못하는 단점이 있다.

파이토치 코드 구현

Machine-Learning-Collection/ML/Pytorch/object_detection/YOLO at master · aladdinpersson/Machine-Learning-Collection

A resource for learning about ML, DL, PyTorch and TensorFlow. Feedback always appreciated :) - Machine-Learning-Collection/ML/Pytorch/object_detection/YOLO at master · aladdinpersson/Machine-Learni...

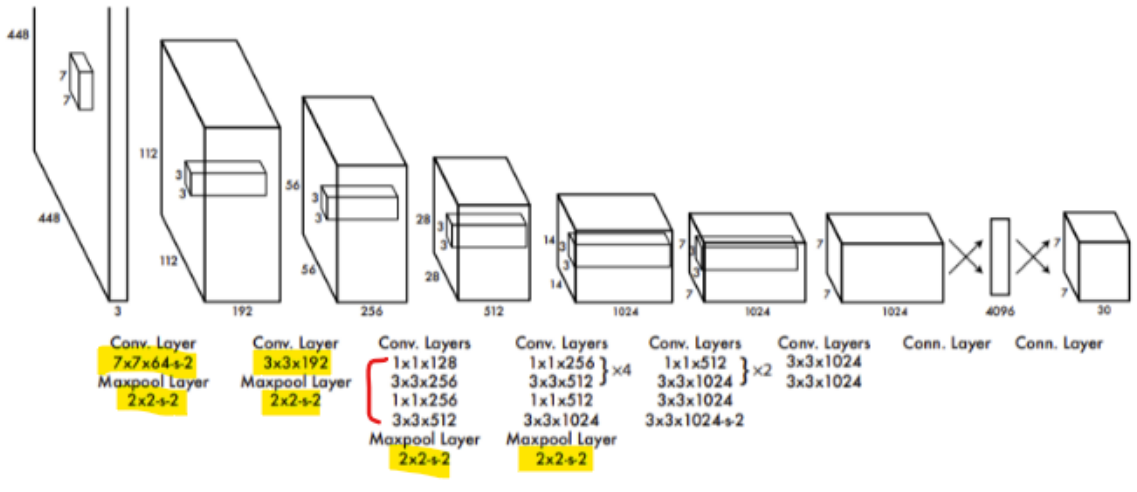
https://github.com/aladdinpersson/Machine-Learning-Collection/tree/master/ML/Pytorch/object_detection/YOLO

aladdinpersson/**Machine-Learning-Collection**

A resource for learning about ML, DL, PyTorch and TensorFlow. Feedback always appreciated :)

10 Contributors 55 Issues 4k Stars 2k Forks

1. 네트워크 - DarkNet



YOLO v1 architecture

각 layer 를 구성하는 configuration 들이 다소 복잡한 경향이 있다. (인셉션 모듈을 차용한 conv layer의 경우 더욱.) tuple 이나 dict 자료형을 통해서 configuration 관리를 해준다.

```
"""
Implementation of Yolo (v1) architecture
with slight modification with added BatchNorm.
"""

import torch
import torch.nn as nn


architecture_config = [
    # Tuple : (kernel_size, num_filters, stride, padding)
    (7, 64, 2, 3),
    "M",
    (3, 192, 1, 1),
    "M",
    (1, 128, 1, 0),
    (3, 256, 1, 1),
    (1, 256, 1, 0),
    (3, 512, 1, 1),
    "M",

    # List : tuples and then last integer represents number of repeats
    [(1, 256, 1, 0), (3, 512, 1, 1), 4],
    (1, 512, 1, 0),
    (3, 1024, 1, 1),
    "M",
    [(1, 512, 1, 0), (3, 1024, 1, 1), 2],
    (3, 1024, 1, 1),
    (3, 1024, 2, 1),
    (3, 1024, 1, 1),
    (3, 1024, 1, 1),
]
```

config 리스트의 요소 중에 “M” 문자열은 max pooling 을 의미한다. 리스트 요소는 마지막 정수값 만큼 layer를 반복함을 의미한다.

해당 코드에서는 architecture_config 리스트 요소의 type 에 따라 조건문으로 서로 다른 layer 를 추가함으로써 모델을 설계한다. 가령 리스트의 요소가 튜플일 경우 해당 하이퍼파라미터에 맞는 conv layer를, 문자열일 경우 max pooling 을, 리스트일 경우 마지막 정수값만큼 layer 를 반복하여 전체적인 모델을 구성한다.

```
class CNNBlock(nn.Module):
    def __init__(self, in_channels, out_channels, **kwargs):
        super(CNNBlock, self).__init__()
        self.conv = nn.Conv2d(in_channels, out_channels, bias=False, **kwargs)
        self.batchnorm = nn.BatchNorm2d(out_channels)
        self.leakyrelu = nn.LeakyReLU(0.1)

    def forward(self, x):
        return self.leakyrelu(self.batchnorm(self.conv(x)))

class Yolov1(nn.Module):
    def __init__(self, in_channels=3, **kwargs):
        super(Yolov1, self).__init__()
        self.architecture = architecture_config
        self.in_channels = in_channels
        self.darknet = self._create_conv_layers(self.architecture)
        self.fcs = self._create_fcs(**kwargs)


    def forward(self, x):
```

```
        x = self.darknet(x)
        return self.fcs(torch.flatten(x, start_dim=1))

    def _create_conv_layers(self, architecture):
        layers = []
        in_channels = self.in_channels

        for x in architecture:
            if type(x) == tuple:
                layers += [
                    CNNBlock(
                        in_channels, x[1], kernel_size=x[0], stride=x[2], padding=x[3],
                    )
                ]

            elif type(x) == str:
                layers += [nn.MaxPool2d(kernel_size=(2,2), stride=(2,2))]

            elif type(x) == list:
                conv1 = x[0]
                conv2 = x[1]
                num_repeats = x[2]

                for _ im range(num_repeats):
                    layers += [
                        CNNBlock(
                            in_channels,
                            conv1[1],
                            kernel_size=conv1[0],
                            stride=conv1[2],
                            padding=conv1[3],
                        )
                    ]
                    layers += [
                        CNNBlock(
                            conv1[1],
                            conv2[1],
                            kernel_size=conv2[0],
                            stride=conv2[2],
                            padding=conv2[3],
                        )
                    ]
                    in_channels = conv2[1]

        return nn.Sequential(*layers)

    def _create_fcs(self, split_size, num_boxes, num_classes):
        S, B, C= split_size, num_boxes, num_classes

        return nn.Sequential(
        nn.Flatten(),
        nn.Linear(1024 * S * S, 496),
        nn.Dropout(0.0),
        nn.LeakyReLU(0.1),
        nn.Linear(496, S * S *(C + B * 5)),
        )
```

2. Loss function

코드 분석에서 중점적으로 살펴봐야할 부분은 loss function 을 구현한 코드. 구현하는 부분에서 최종 feature map에 대하여 처리해줘야 할 과정들이 몇 가지 있기 때문에 반드시 짚고 넘어가야 한다.

- grid의 크기 : S
- grid cell 별 예측 bounding box의 수 : B
- 예측하는 class 의 수 : C
- 가중치 파라미터: lambda_noobj, lambda_coord

그 다음 forward pass 시 처리할 과정을 정의한다.

- 우선 각 grid cell(=인덱스) 마다 2개의 bounding box 를 예측.
- 그 중 confidence score (=신뢰도 점수) 가 높은 1 개의 bounding box 를 학습에 사용
- predictions 파라미터는 네트워크가 최종적으로 산출하는 7*7*30 의 feature map 을 flatten 한 결과이다. 이를 7*7*30 으로 reshape 시켜 준다.
- **predictions[…, 21:25] 는 첫 번째 bounding box 의 좌표값**을 , **prediction[…, 26:30] 은 두 번째 bounding box 의 좌표값**을 의미한다. 이를 정답에 해당하는 target 의 좌표갑소가 비교하여 각각 IoU 를 계산한다.
- best_box 변수에는 두 bounding box 중 IoU 가 더 큰 box 의 index 가 저장된다. 이후 target[…, 20] 을 통해 해당 grid cell에 ground truth box 의 중심이 존재하는지 여부를 확인한다. 만약 존재한다면 exists_box = 1, 존재하지 않는다면 =0 이 될 것이다.

```
import torch
import torch.nn as nn
from utils import intersection_over_union

class YoloLoss(nn.Module):
    """
    Calculate the loss for yolo (v1) model
    """

    def __init__(self, S=7, B=2, C=20):
        super(YoloLoss, self).__init__()
        self.mse = nn.MSELoss(reduction="sum")

        self.S = S
        self.B = B
        self.C = C

        self.lambda_noobj = 0.5
        self.lambda_coord = 5

    def forward(self, predictions, target):
        # predictions are shaped (BATCH_SIZE, S, S, C+B*5)
        predictions = predictions.reshape(-1, self.S, self.S, self.C + self.B * 5)

        # calcualte IoU for the two predicted bounding boxes with target bbox
        iou_b1 = intersection_over_union(predictions[..., 21:25], target[..., 21:25])
        iou_b2 = intersection_over_union(predictions[..., 26:30], target[..., 21:25])
        ious = torch.cat([iou_b1.unsqueeze(0), iou_b2.unsqueeze(0)], dim=0)

        # Take the box with highest IoU out of the two prediction
        # Note that bestbox will be indices of 0, 1 for which bbox was best

        iou_maxes, bestbox = torch.max(ious, dim=0)
        exists_box = target[..., 20].unsqueeze(3)

        ### Localization loss
        '''
        best_box 변수를 활용해 bbox 예측 중 IoU 값이 더 큰 box 를 최종 예측에 사용.
        그리고 width, height 값에는 루트를 씌운다.
        이후 bounding box 좌표값에 대하여 mean squared error loss 를 계산.
        '''
          box_predictions = exists_box * (
              (
                best_box * predictions[..., 26:30]
                + (1 - bestbox) * predcitions[..., 21:25]
              )
          )

        box_targets = exists_box * target[..., 21:25]

        # Take sqrt of width, height of boxes to ensure that
        box_predictions[..., 2:4] = torch.sign(box_predictions[..., 2:4]) * torch.sqrt(
            torch.abs(box_predictions[..., 2:4] + 1e-6)
        )
        box_targets[..., 2:4] = torch.sqrt(box_targets[..., 2:4])

        box_loss = self.mse(
            torch.flatten(box_predictions, end_dim=-2),
            torch.flatten(box_targets, end_dim=-2),
        )

        # Confidence loss
        '''
        predictions[..., 25:26] 은 첫 번째 box 의 confidence score 를,
        predictions[..., 20:21] 은 두 번째 box 의 confidence score 를 의미한다.
        아래 코드를 보면 exists_box 변수를 통해 grid cell에 할당된 ground truch box의
        중심이 있는 경우에만 loss 를 구한다는 것을 확인할 수 있다.

        object 가 없을 경우의 confidence loss 를 구하는 과정을 보면, 두 bounding box를
        모두 학습에 참여시킨다는 것을 확인할 수 있다.
        '''

        # pred_box is the confidence score for the bbox with highest IoU
        pred_box = (
            bestbox * predictions[..., 25:26] + (1 - bestbox) * predictions[..., 20:21]
        )

        object_loss = self.mse(
            torch.flatten(exists_box * pred_box),
            torch.flatten(exists_box * target[..., 20:21]),
        )

        no_object_loss = self.mse(
            torch.flatten((1 - exists_box) * predictions[..., 20:21], start_dim=1),
            torch.flatten((1 - exists_box) * target[..., 20:21], start_dim=1),
        )

        no_object_loss += self.mse(
            torch.flatten((1 - exists_box) * predictions[..., 25:26], start_dim=1),
            torch.flatten((1 - exists_box) * target[..., 20:21], start_dim=1)
        )


        # Class loss
        '''
        predictions[..., :20] 에 해당하는, 즉 20개의 class의 score 를 target과
```

```
        비교하여 mse loss 를 구한다.

        이후 YoloLoss 생성자에서 정의 가중치 파라미터 lambda_coord 를 localization loss에
        곱해주고, lambda_noobj 를 no object confidence loss 에 곱해준다.

        이후 locaslization loss, confidence loss, class loss 를 모두 더해 최종 loss를 구한다.
        '''
        class_loss = self.mse(
            torch.flatten(exists_box * predictions[..., :20], end_dim=-2,),
            torch.flatten(exists_box * target[..., :20], end_dim=-2,),
        )

        loss = (
            self.lambda_coord * box_loss  # first two rows in paper
            + object_loss  # third row in paper
            + self.lambda_noobj * no_object_loss  # forth row
            + class_loss  # fifth row
        )

        return loss
```