2020_PD_Challenge 결과 발표

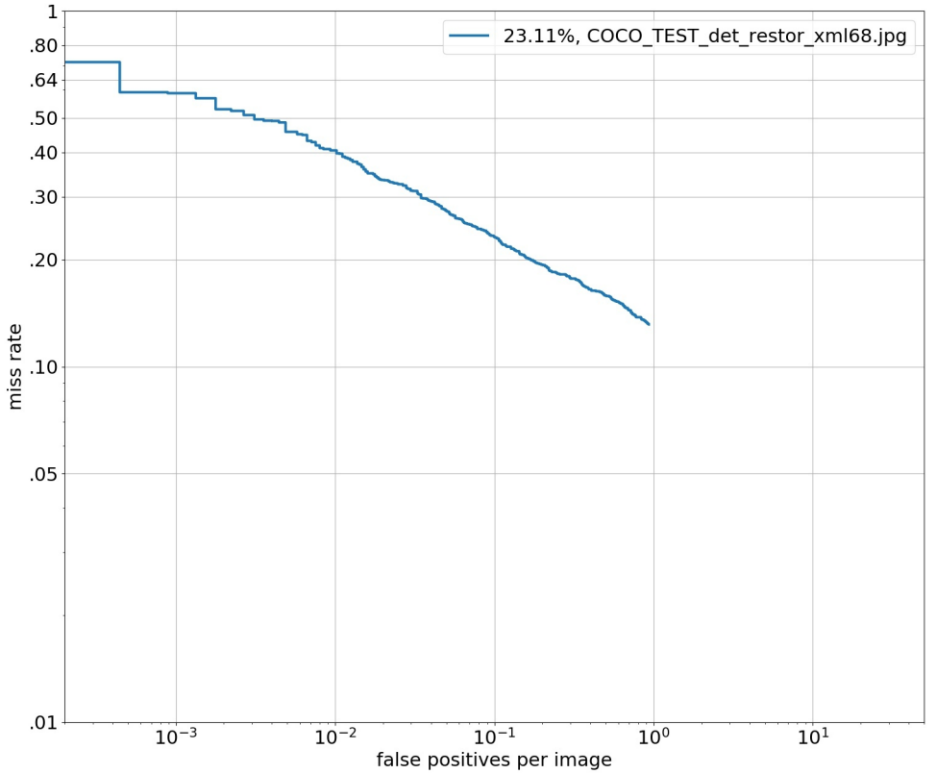
멘토:한대찬

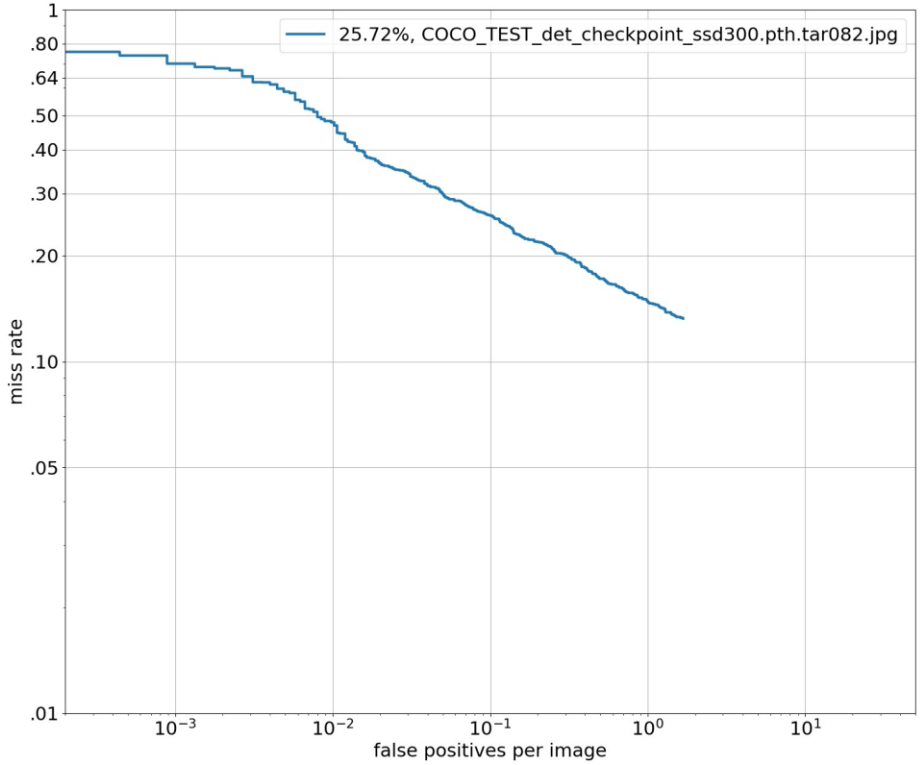
멘티:정찬호

24% demo

목차

1.베이스 라인


23.11%, COCO_TEST_det_restor_xml68.jpg
miss rate
1
.80
.64
.50
.40
.30
.20
.10
.05
.01
10^{-3}
10^{-2}
10^{-1}
10^{0}
10^{1}
false positives per image


25.72%, COCO_TEST_det_checkpoint_ssd300.pth.tar082.jpg
miss rate
1
.80
.64
.50
.40
.30
.20
.10
.05
.01
10^{-3}
10^{-2}
10^{-1}
10^{0}
10^{1}
false positives per image

2.연구방향 및 자료 서베이

- EfficientNet 논문 한줄 요약:

  Width(filter)scaling,depth(layer)scaling,resolution scaling

  ->compound scaling 하는 효율적인 비율을 제시(파이를 늘려가며 모델 사이즈 조정)

$$\begin{aligned} \text{depth: } & d = \alpha^\phi \\ \text{width: } & w = \beta^\phi \\ \text{resolution: } & r = \gamma^\phi \\ \text{s.t. } & \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2 \\ & \alpha \geq 1, \beta \geq 1, \gamma \geq 1 \end{aligned} \tag{3}$$

- VGG-16 논문 한줄 요약:

  3*3 convolution filter를 반복 사용해 16layer를 형성

2.연구방향 및 자료 서베이

- Efficient net을 고른 이유

  imageNet Clasification에서 SOTA를 찍을 만큼 이미지 분류를 잘하기 위해서는 정확한 피쳐맵이 필요할 것이다.즉 efficient net구조는 인풋이미지를 효율적인 학습 파라미터 용량으로 정확하게 특징들을 뽑아낼 것이라는 기대를 했다.

  따라서 우리의 SSD model object detection의 재료가 되는 feature map들을 우리가 기대한 만큼 사람의 특징들을 잘 뽑아 detection의 정확도가 올라갈 것이라고 예상함

2.연구방향 및 자료 서베이 -efficientnet-b4 pytorch 구조

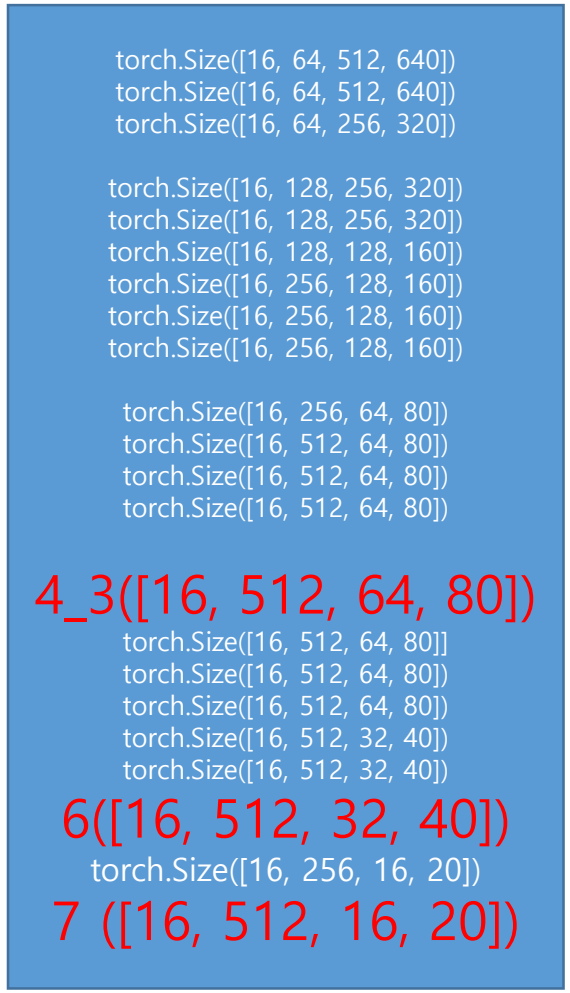
```
  (31): MBConvBlock(
    (_expand_conv): Conv2dStaticSamePadding(
      448, 2688, kernel_size=(1, 1), stride=(1, 1), bias=False
      (static_padding): Identity()
    )
    (_bn0): BatchNorm2d(2688, eps=0.001, momentum=0.010000000000000009, affine=True, track
_running_stats=True)
    (_depthwise_conv): Conv2dStaticSamePadding(
      2688, 2688, kernel_size=(3, 3), stride=(1, 1), groups=2688, bias=False
      (static_padding): ZeroPad2d(padding=(1, 1, 1, 1), value=0.0)
    )
    (_bn1): BatchNorm2d(2688, eps=0.001, momentum=0.010000000000000009, affine=True, track
_running_stats=True)
    (_se_reduce): Conv2dStaticSamePadding(
      2688, 112, kernel_size=(1, 1), stride=(1, 1)
      (static_padding): Identity()
    )
    (_se_expand): Conv2dStaticSamePadding(
      112, 2688, kernel_size=(1, 1), stride=(1, 1)
      (static_padding): Identity()
    )
    (_project_conv): Conv2dStaticSamePadding(
      2688, 448, kernel_size=(1, 1), stride=(1, 1), bias=False
      (static_padding): Identity()
    )
    (_bn2): BatchNorm2d(448, eps=0.001, momentum=0.010000000000000009, affine=True, track_
running_stats=True)
    (_swish): MemoryEfficientSwish()
  )
```

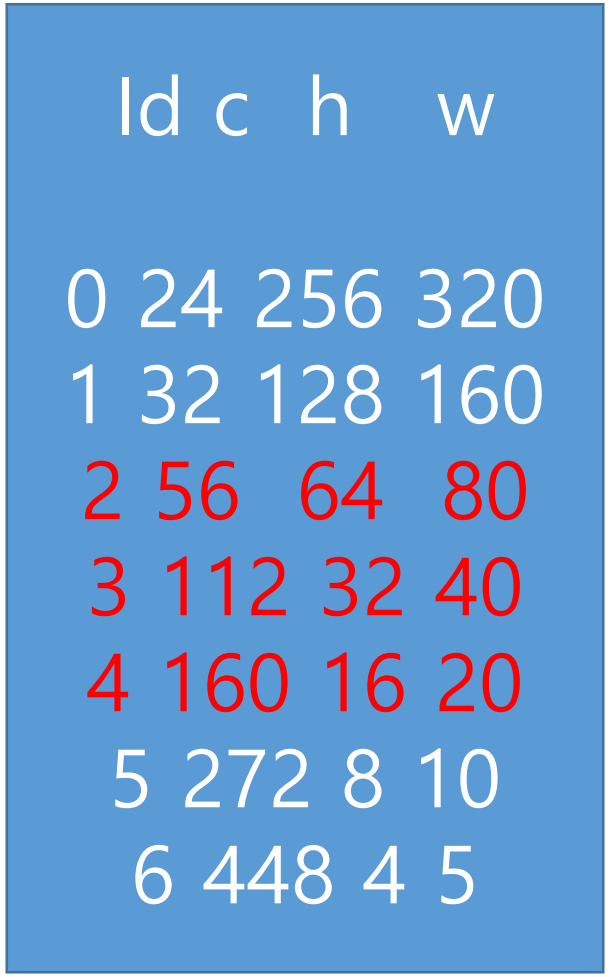
Convblock으로 구성된것들을 정해진 설정대로 파라미터 값을 넣어주고 정해진 depth만큼 반복해서 진행

```
DEPTHS: [2, 4, 4, 6, 6, 8, 2]
WIDTHS: [24, 32, 56, 112, 160, 272, 448]
```

그래서 총 7 level의 output fmap을 뽑아낼 수 있음

2.연구방향 및 자료 서베이
-efficientnet b4<->vgg16

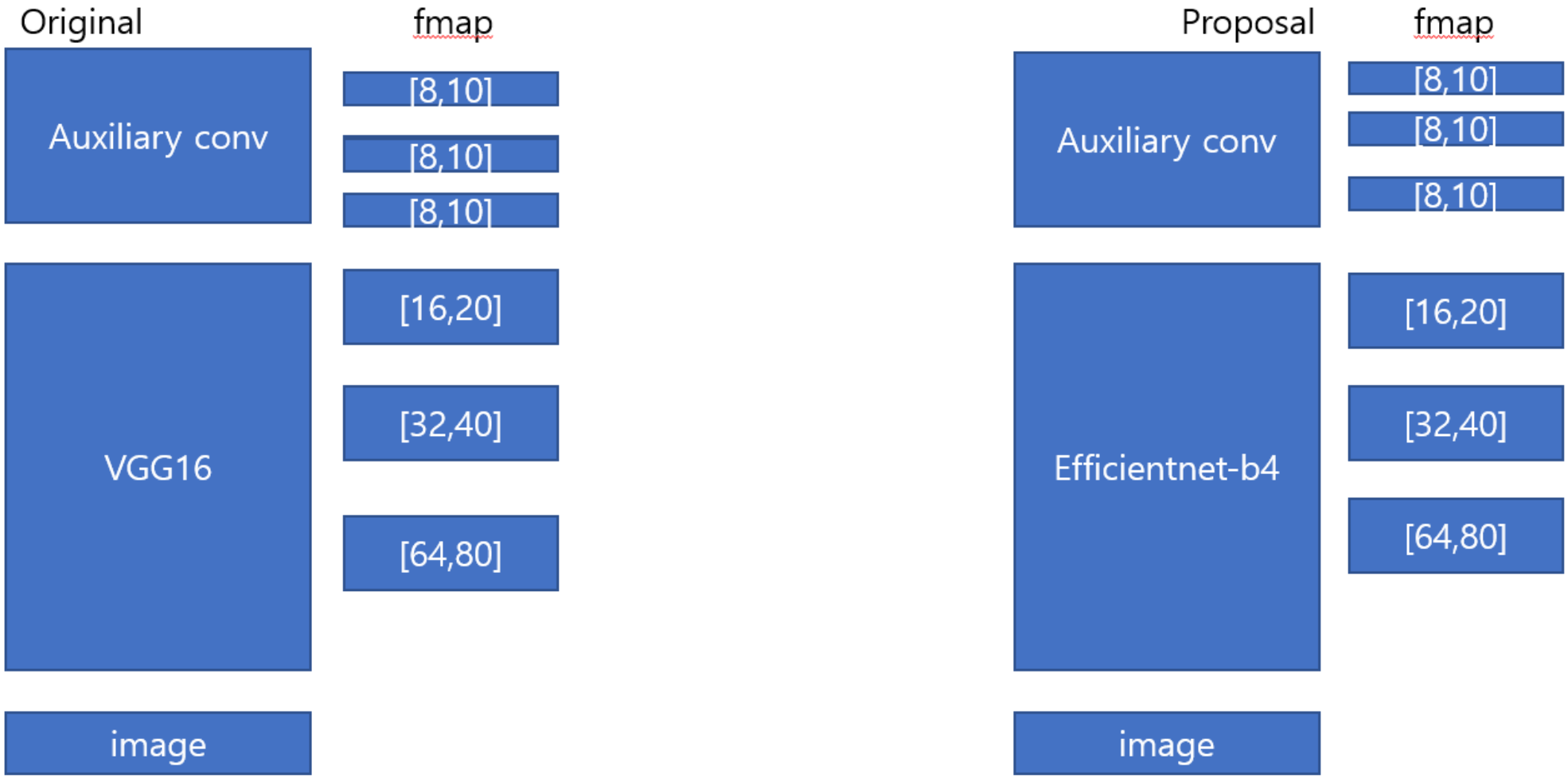




정확히 backbone의 network만 건들이고 나머지는 건들이지 않아서 backbone change에 따른 성능을 보려고 했습니다.

따라서 기존 output feature map size에 맞는것을 선택적으로 뽑아냈습니다.

2.연구방향 및 자료 서베이



2.연구방향 및 자료 서베이

Backbone을 바꾸려다 보니 다음과 같은 issue가 있었습니다.

Issue:

기존 모델에 fine tuning된 optimizer(SGD),scheduler(mutistepLR) 들을 그대로 적용하니 학습진행이 안됨.

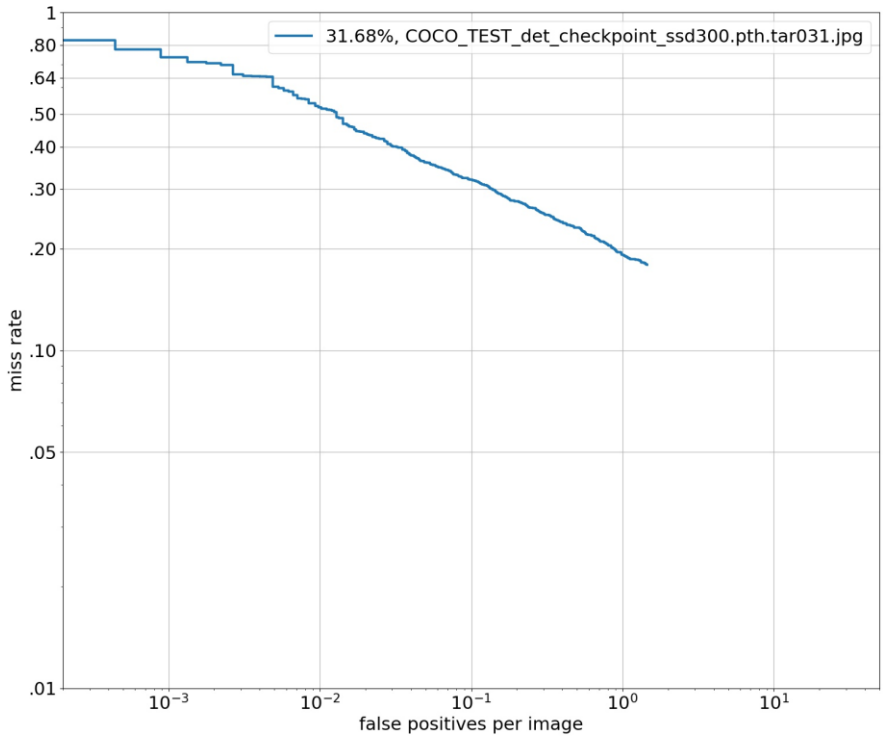
2.연구방향 및 자료 서베이

Solution:
SGD->Rmsprop
MultistepLR->ReduceLR

여기서 ReduceLR에 적용하기 위한 validation set으로 Set02/V01/00121~00189 설정하였습니다.설정 이유는

아래와 같이 모든 이미지에 사람이 있고 많은 사람들이 포진해 있어 일반적인 trainset에 비해 특수성이 있어 일반적인 것을 학습시키고 특수한 것을 맞추게 끔 학습하고자 했습니다.

연구결과 및 분석
1. efficient backbone



- 다음과 같이 납득하기 힘든 결과가 나왔고 문제점을 분석하기 위해 정성적으로 평가해보았는데
- 그 과정에서 dataset을 normalize 하는 과정이 넣었다고 생각만 했지 실제로는 작동하지않고 있었습니다.
- 또한 feature map size에 맞게 뽑아내다 보니 channel수가 상대적으로 현저히 작았습니다.기존 512,512,512 <->effnet 56,112,160

연구결과 및 분석
2. normalize&width scaling

```
self.conv1=nn.Conv2d(56,512,kernel_size=3,padding=1,bias=True)
self.conv1_bn= nn.BatchNorm2d(512, affine=True)
```

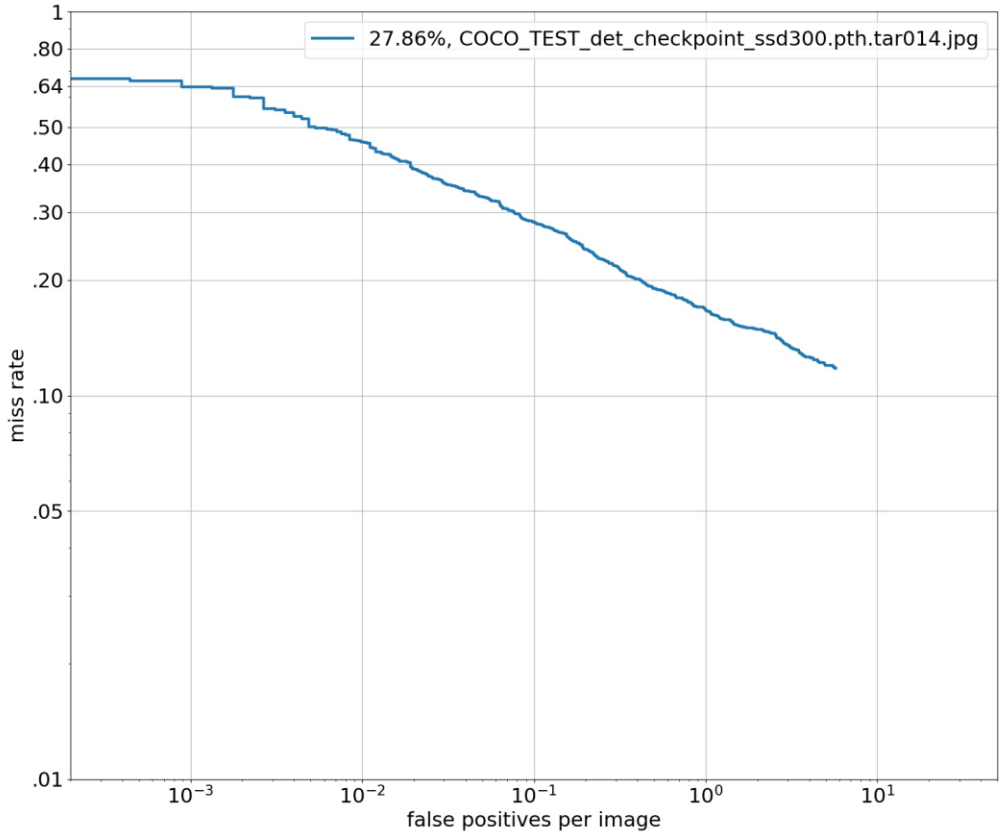
```
self.conv2=nn.Conv2d(112,512,kernel_size=3,padding=1,bias=True)
self.conv2_bn= nn.BatchNorm2d(512, affine=True)
```

```
self.conv3=nn.Conv2d(160,512,kernel_size=3,padding=1,bias=True)
self.conv3_bn= nn.BatchNorm2d(512, affine=True)
```

따라서 다음 시도로 채널별 평균과 표준편차를 기준으로 nomalize하였습니다.

그리고 다음과 같이 channel을 늘려주는 과정을 통해 모델이 얻을 수 있는 정보량을 늘려 성능 개선을 시도했습니다

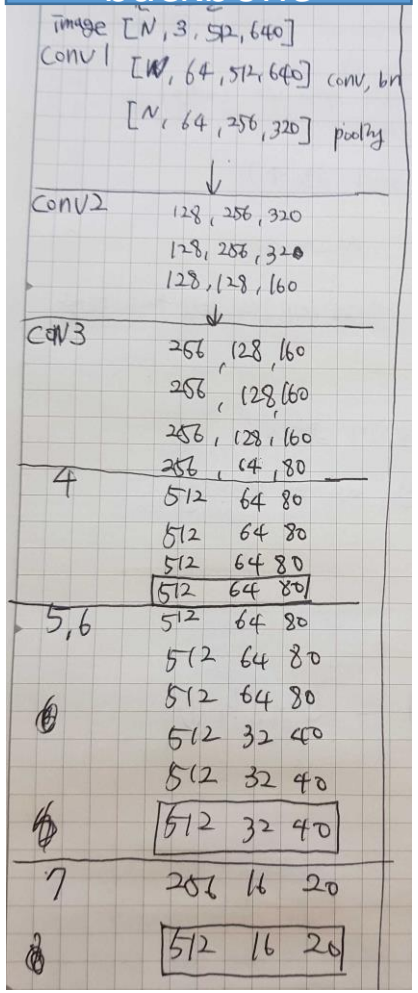
연구결과 및 분석
2. normalize&width scaling



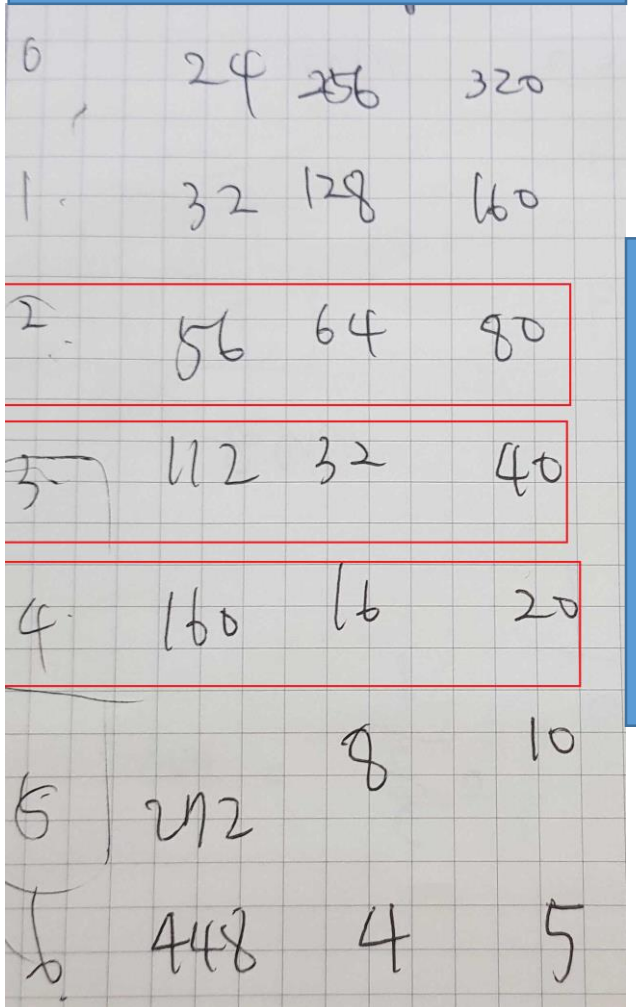
- 기존 모델에 비해 학습속도가 비교적 빠르다는 점이 인상깊었습니다.(10~30에포크 성적 기준)
- 하지만 기존에 vgg16 base인 25%보단 좋지 않은 성능이기 때문에 분석이 필요했습니다.

연구결과 및 분석
2. normalize&width scaling

Original backbone



Efficientnet-b4



Depths
4
4
6

- Efficientnet을 쓰는 이유는 더 정확하게 우리가 원하는 featur를 뽑아줄것같아서였습니다.
- 하지만 fmapsize에 맞게 뽑다보니 vgg-16에 비해 deep하지 않은 정보를 사용했고
- 이를 이용해 detection을 하였으니 성능이 안나오는 것으로 판단했습니다.

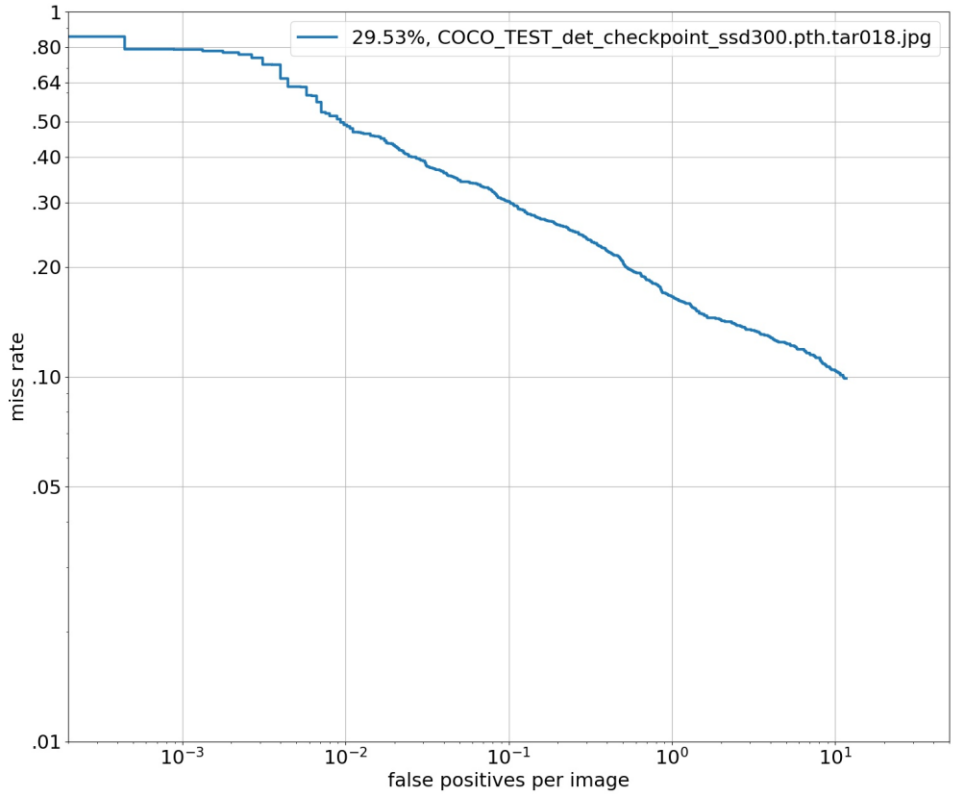
연구결과 및 분석
2. normalize&width scaling

그렇다면 여기서 어떻게 해야 fmapsize도 맞추면서 efficientnet의 마지막 단계까지 혹은 조금 더 딥한 feature를 이용할 수 있을지 고민해 봤습니다.그리고 다음과 같은 결론을 얻었습니다.

1.Resolution up:512,640을 1024,1280으로 높이게 되면 더 아래로 내려간 fmap을 이용할 수 있게 될 수 있을거라 생각했습니다.

2.FPN:bottom->top을 쭉 거친후 top->bottom을 거치고 skip connection을 이용하면 더 아래로 내려간 fmap을 이용할 수 있을거라 생각했습니다.

연구결과 및 분석
3.resolution scaling



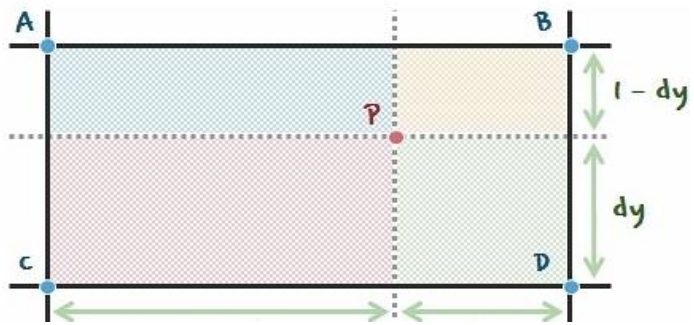
- 실망스러운 성능이였습니다.
- 이렇게 실망스러운 성능을 얻은 이유로는 resolution up과정에서 있다고 생각했습니다.
- Bilinear interpolation을 사용했는데 이런 보간법으로는 적절하게 2배라는 resolution up을 수행하지 못해서 결국 이미지의 위치관계 등이 깨진 것 이 아닌가 라는 생각입니다.

연구결과 및 분석
-interpolation

Linear interpolation (선형 보간법) 은,
값을 아는 두 점 P, Q 사이의 모르는 값 R을 유추할 때, 세 점 P, R, Q 의 값들이 선형적인 관계를 갖는다고 가정하며, 두 점 P, Q 의 값과 거리비를 이용하여 R 의 값을 구한다. (P와 Q 사이의 거리를 모두 1로 가정함)

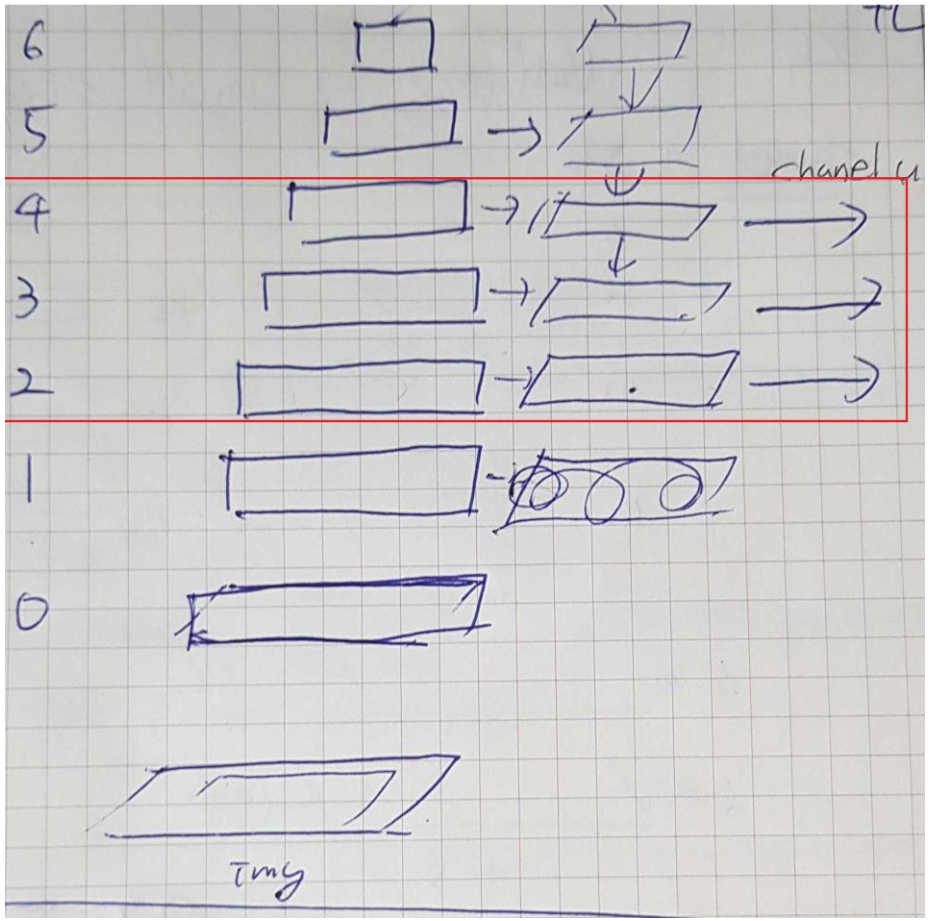
$$R(val) = P(val) \times (dx) + Q(val) \times (1 - dx)$$

Bilinear interpolation (양선형 보간법) 이란,
linear interpolation 을 x축과 y축으로 두 번 적용하여 값을 유추하는 방법이다.
아래 설명에서 그 식을 유도하겠지만, 결과적으로는 네 개의 인접한 점들의 값과 그에 따른 면적을 가중치(weight) 로 하여 값을 구하게 된다.



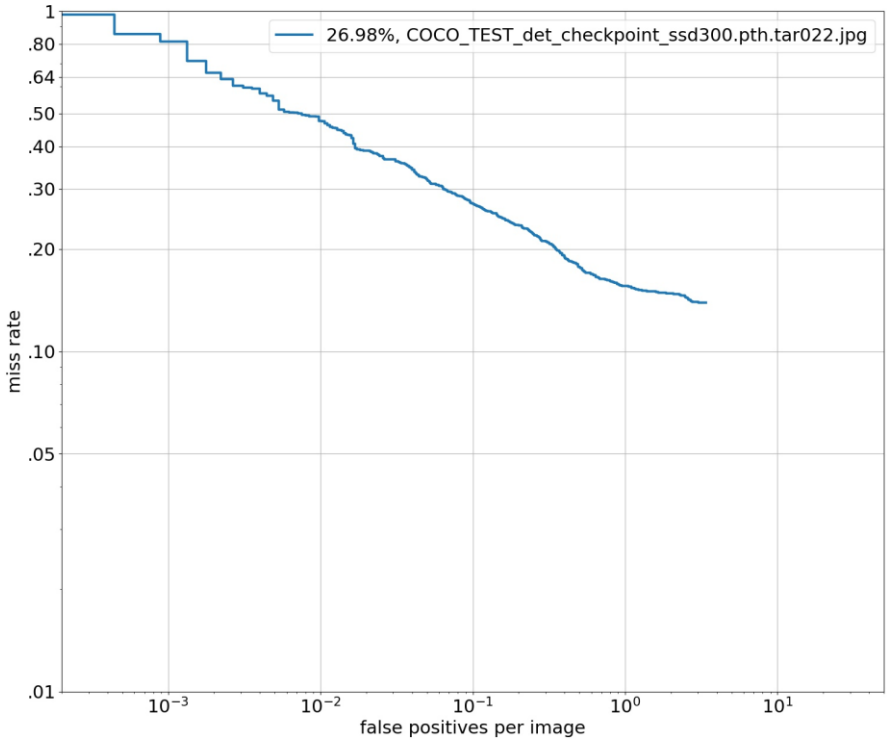
연구결과 및 분석
4.FPN

Proposed FPN structure



- Efficientnet level6까지 갔다가 다시 2까지 내려오는 구조이며 이때 쓰는 것은 2,3,4level입니다.
- Fmapsize를 키울땐 bilinear interpolation을 이용했고 기존에 동일 scale과 단순히 add해서 widthscaling하였습니다

연구결과 및 분석
4.FPN



- 성능이 올랐습니다.따라서 제가 했던 구조가 모델 학습에 도움이 된다고 판단했습니다.
- 하지만 sizeup하는 과정이 단순 interpolation을 통해 이루어지다 보니 정보 손실이 있을 것이란 분석을 했습니다.
- 따라서 deconvolution 연산을 통해 sizeup하는 parameter를 또한 학습을 해야겠다고 생각했습니다.

연구결과 및 분석
5.deconvolution

```
self.deconv6=nn.ConvTranspose2d(448,272,kernel_size=2,stride=2)
self.deconv6_bn= nn.BatchNorm2d(272, affine=True)
self.deconv5=nn.ConvTranspose2d(272,160,kernel_size=2,stride=2)
self.deconv5_bn= nn.BatchNorm2d(160, affine=True)
self.deconv4=nn.ConvTranspose2d(160,112,kernel_size=2,stride=2)
self.deconv4_bn= nn.BatchNorm2d(112, affine=True)
self.deconv3=nn.ConvTranspose2d(112,56,kernel_size=2,stride=2)
self.deconv3_bn= nn.BatchNorm2d(56, affine=True)
```

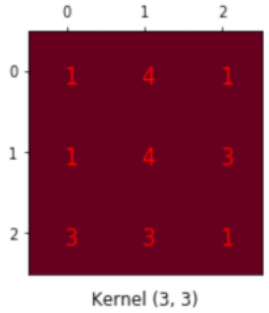
$$Hout = (Hin-1)\times stride[0]-2\times padding[0]+dilation[0]\times(kernel\_size[0]-1)+output\_padding[0]+1$$

- Deconvolution을 pytorch로는 convtranspose2d로 구현하였습니다.
- Convolution과 역함수 개념이므로 손계산을 통해 kernel_size=2,stride=2로 설정하여 input의 2배 outpu을 내도록 설계했습니다.

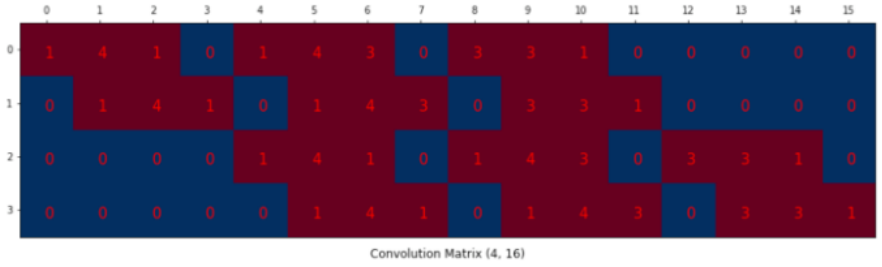
연구결과 및 분석
5.deconvolution<->transposed convolution

Convolution matrix

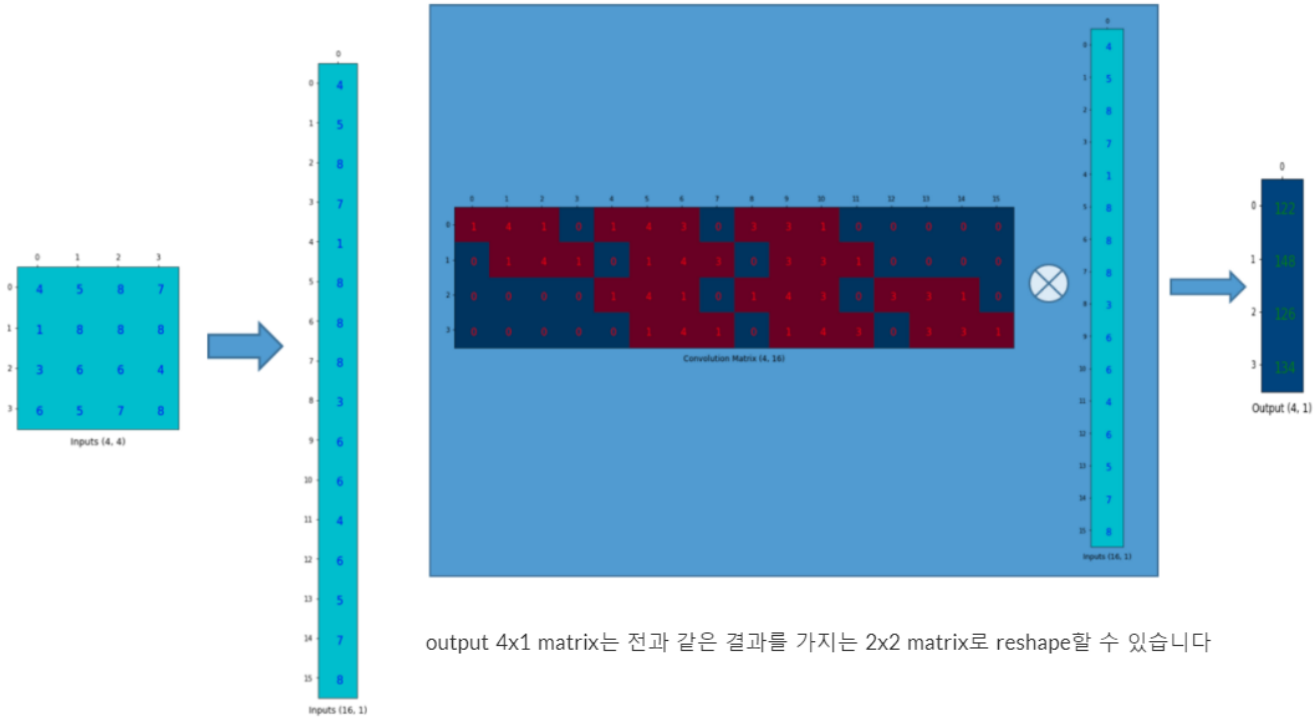
Matrix를 사용해 convolution 연산을 표현할 수 있습니다. convolution 연산을 수행하고 matrix 곱을 위해 kernel matrix를 재배치합니다



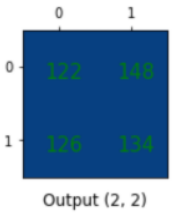
3x3 kernel을 4x16 matrix로 재배치하겠습니다:



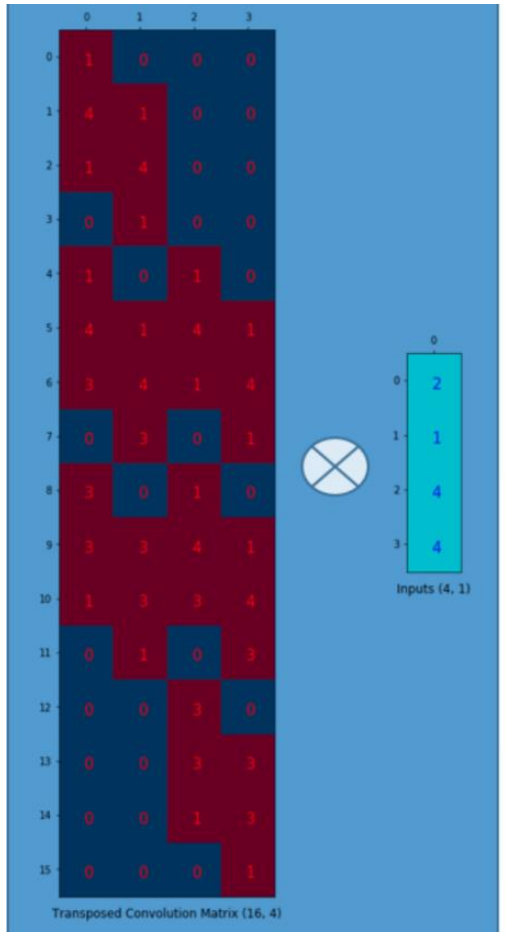
4x16 convolution matrix와 1x16 input matrix를 곱할 수 있습니다 (16 차원의 column vector)

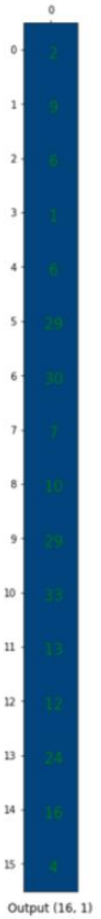


output 4x1 matrix는 전과 같은 결과를 가지는 2x2 matrix로 reshape할 수 있습니다

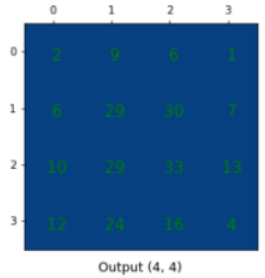


연구결과 및 분석
5.deconvolution<->transposed convolution

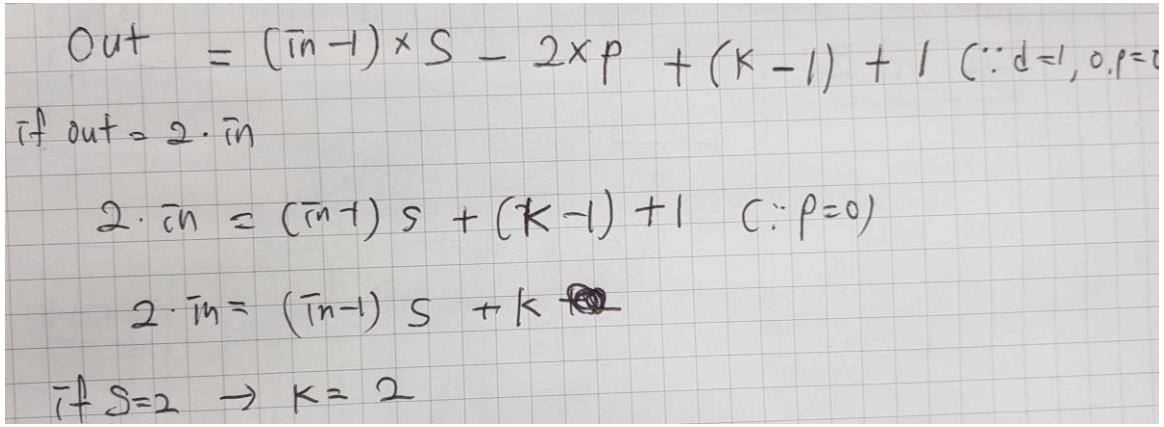




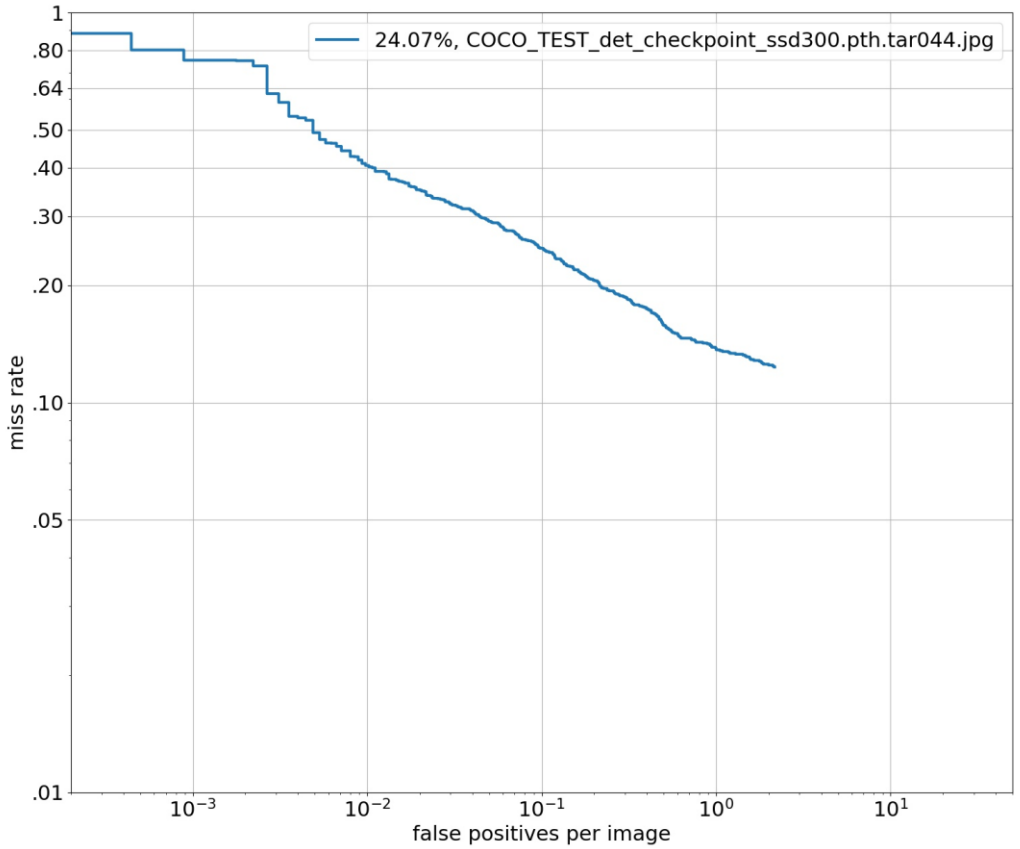
output은 4x4로 reshape할 수 있습니다



작은 matrix(2x2)를 더 큰 matrix(4x4)로 up-sampling했습니다. Transposed Convolution은 가중치를 배치하는 방식 때문에 1-9 관계를 유지합니다



연구결과 및 분석
5.deconvolution



- 결과가 처음 제가 냈던 baseline보단 올랐지만 23.22는 못넘어 아쉽습니다.
- 하지만 대체로 10-20에포크에서 이런 24%대성능이 나오는것은 효과적으로 학습했음을 의미합니다.
- 예상대로 sizeup을 convolution연산을 통해 하니 좀더 위치관계등을 보존하여 성능이 향상되었습니다.

마무리 시도들..

- Skipconnection에서 그냥 더하는 것이 걸려

Scalar weigh와 엡실론을 이용해 가중합을 시도 해보고 있습니다.

$$P_6^{td} = Conv\left(\frac{w_1 \cdot P_6^{in} + w_2 \cdot Resize(P_7^{in})}{w_1 + w_2 + \epsilon}\right)$$

Bifpn이라는 구조를 유사하게라도 시도해보고자

기존 fpn구조의 반복을 통해 구현해 보았다.

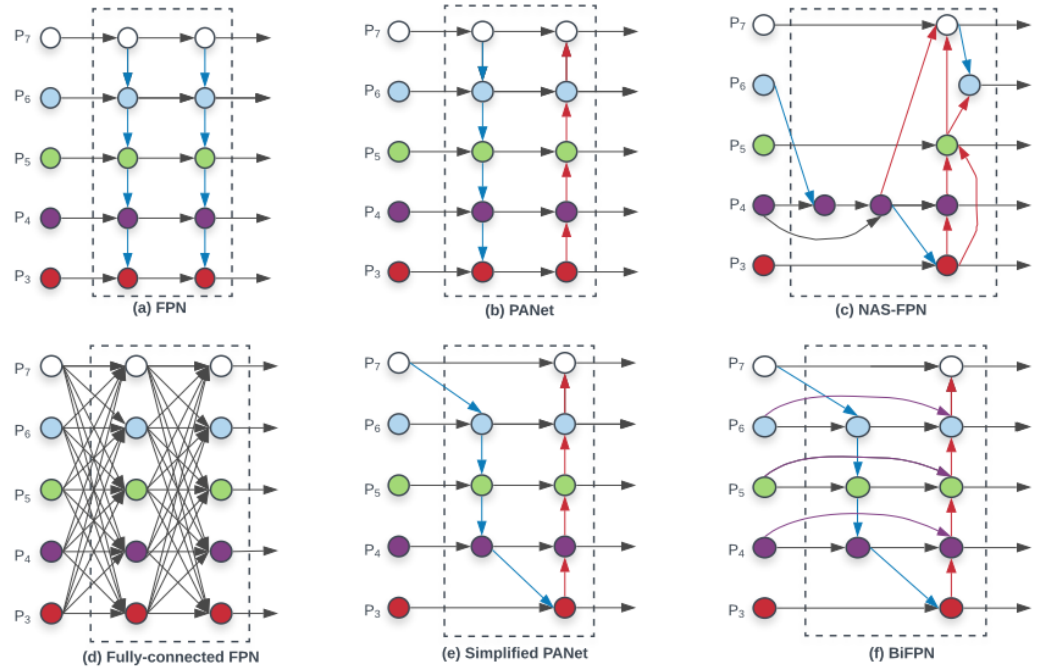


Figure 2: **Feature network design –** (a) FPN [16] introduces a top-down pathway to fuse multi-scale features from level 3 to 7 (P_3 - P_7); (b) PANet [19] adds an additional bottom-up pathway on top of FPN; (c) NAS-FPN [5] use neural architecture search to find an irregular feature network topology; (d)-(f) are three alternatives studied in this paper. (d) adds expensive connections from all input feature to output features; (e) simplifies PANet by removing nodes if they only have one input edge; (f) is our BiFPN with better accuracy and efficiency trade-offs.

마무리

- 교수님의 지적이 있기 전까진 즉 1주일 전까진 논리적으로 판단해서 이게 왜 이렇게 나오는 성능인지에 대해 생각하지 않았습니다.공부하지 않고 무작정 시도만 하니 굉장히 삽질을 많이 했습니다.
- 하지만 교수님의 조언을 듣고 책을 피고 강의를 들으며 왜 ssd에 이 function 필요한지 이게 어떤 효과를 가져왔을 지 공부하고 이를 토대로 분석해보니 문제가 효과적으로 풀렸고,실패하더라도 그 실패 분석을 기반으로 다음 연구에 다른 시도를 하니 좀 더 효과적인 도움을 얻었습니다.
- 이번 첼린지를 통해 나아가기 위한 실패와 분석을 조금이나마 깨달은것 같아 정말 기쁩니다.