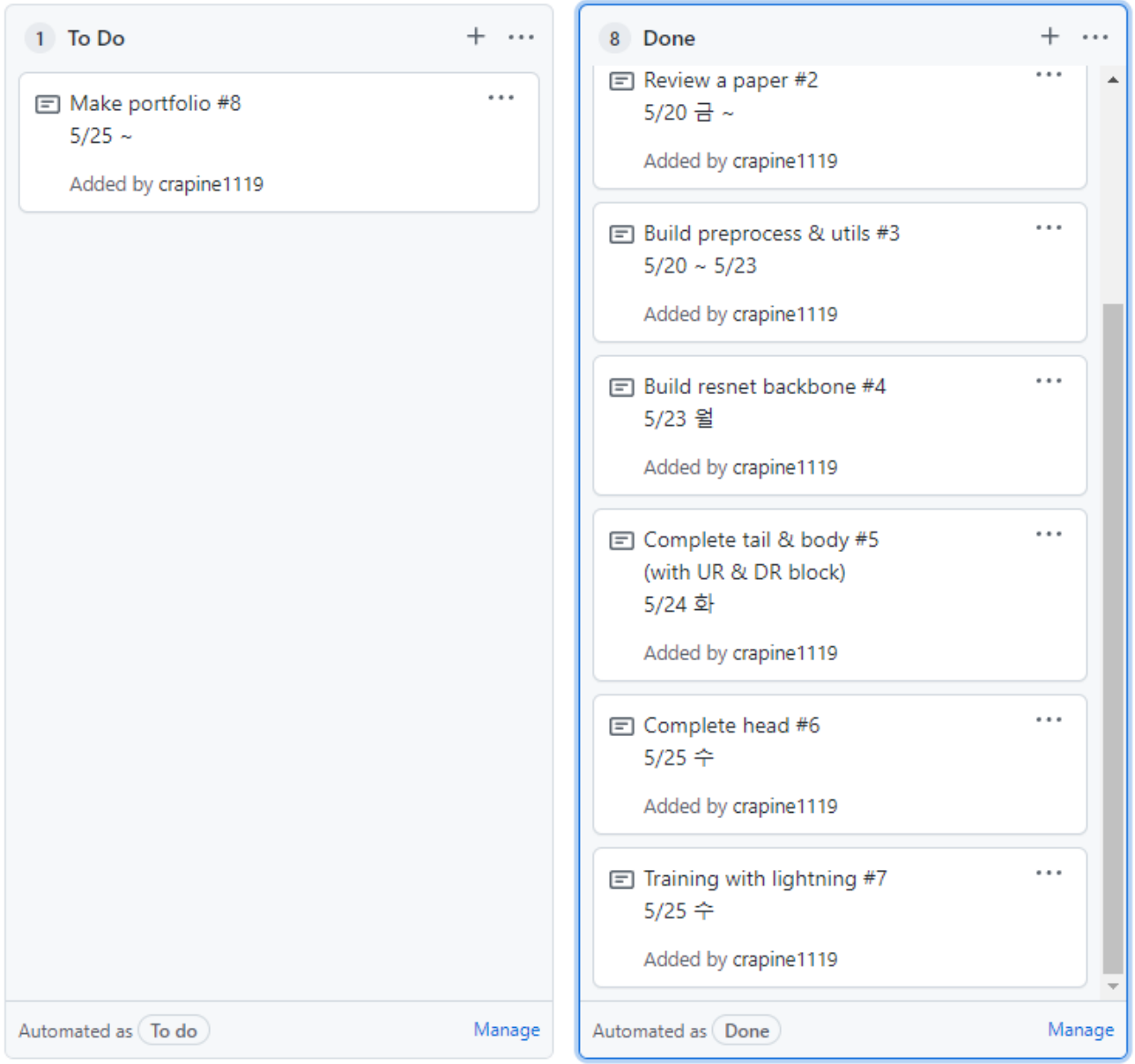
GENESISLAB

S.H.KIM

CONTENTS

작업 순서


1 To Do
Make portfolio #8
5/25 ~
Added by crapine1119
Automated as To do
Manage
8 Done
Review a paper #2
5/20 금 ~
Added by crapine1119
Build preprocess & utils #3
5/20 ~ 5/23
Added by crapine1119
Build resnet backbone #4
5/23 월
Added by crapine1119
Complete tail & body #5
(with UR & DR block)
5/24 화
Added by crapine1119
Complete head #6
5/25 수
Added by crapine1119
Training with lightning #7
5/25 수
Added by crapine1119
Automated as Done
Manage

논문 리뷰 : FishNet

- **요약**

- Object Detection & Segmentation network의 장점을 통합한 backbone 개발 (Escape from Classification backbone!)
- Direct gradient propagation from deep to shallow layers

- **연구 목적**

1. Versatile backbone

- Image(I) level : Classification > 연속적 Downsample > Deep feature but low resolution (Region/Pixel level에 부적합)
- R/P level : Detection/Segmentation > Down/Upsample > Deep feature with high resolution (U-Net, FPN etc.)

➢ **I/R/P level의 장점을 통합**한 Backbone??

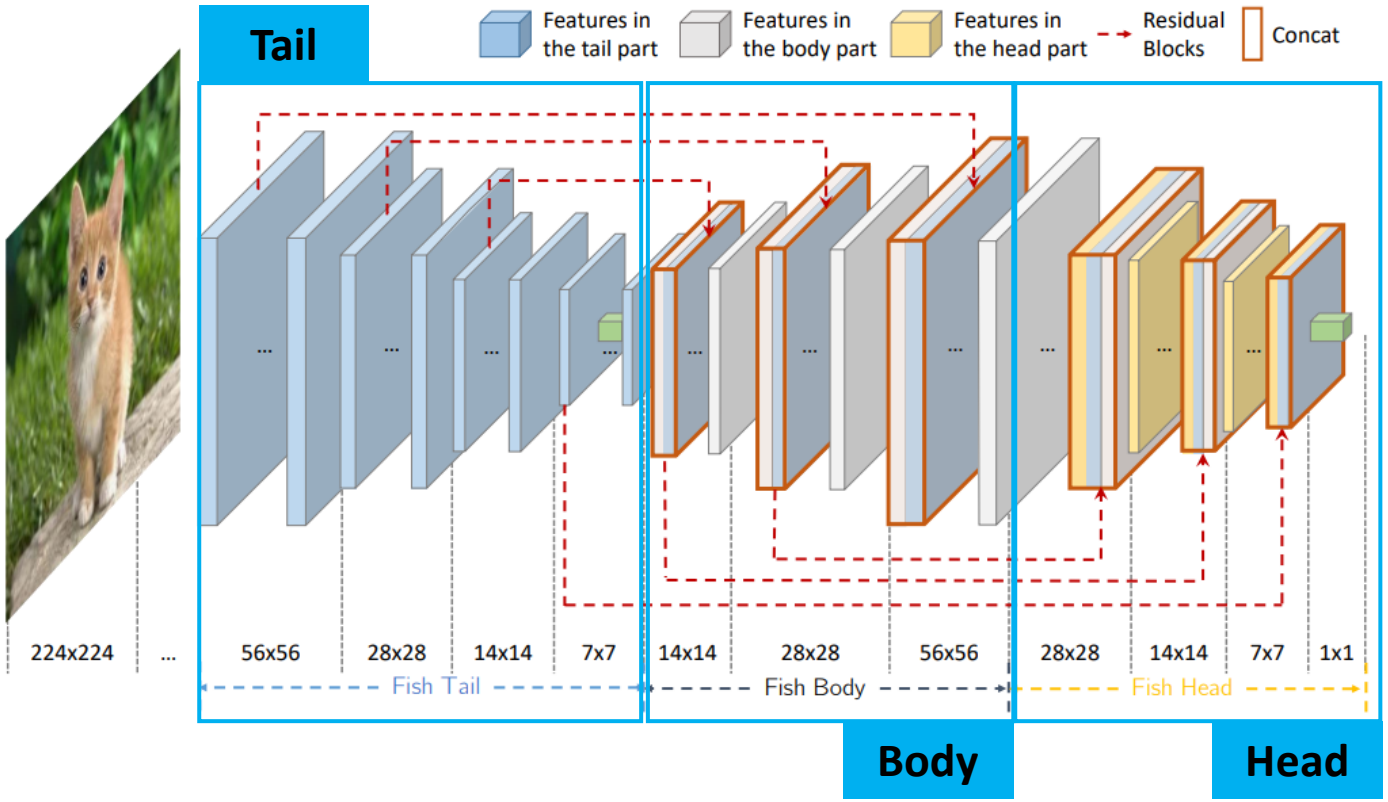
2. Direct gradient propagation

- Identity mapping(residual block or concatenation)의 문제점?

➢ Resolution이 변하면서 convolution (**Isolated-convolution**)을 이용 : Direct gradient propagation를 방해

➢ **Direct gradient propagation**을 위한 네트워크 구조 변경

논문 리뷰 : FishNet



- **FishNet Structure**

- 같은 크기의 feature를 "stage"라고 정의

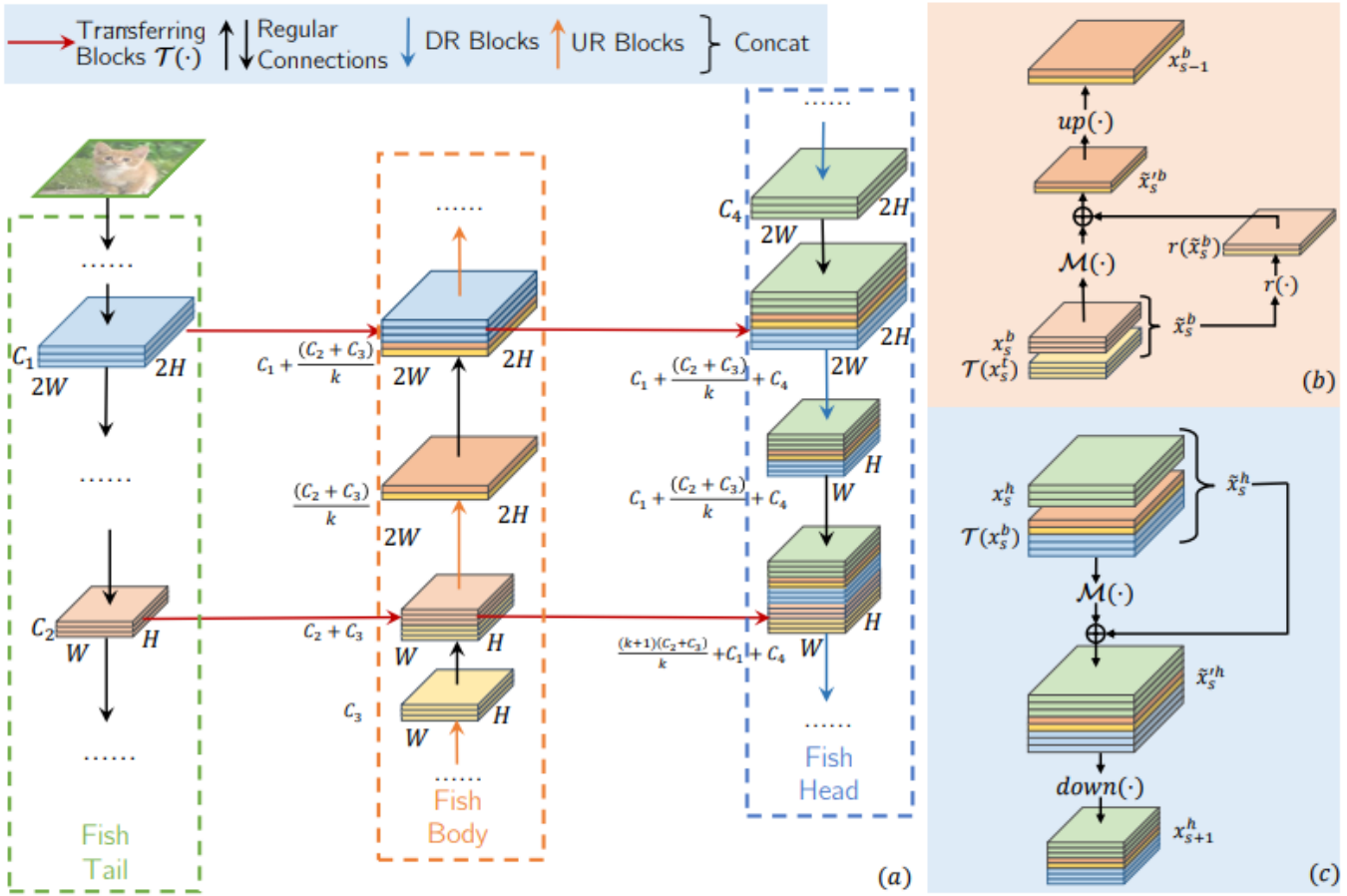
- 같은 Stage의 I/R/P level feature를 모두 concatenation (붉은색 점선)

- Direct propagation을 통해 semantic meaning을 네트워크 전반에 걸쳐 유지

- Feature Refining을 통해 Deep/Shallow feature간 상호작용

- **Tail : Deep (high-level semantic meaning) low-resolution feature**
- **Body : High resolution with high level semantic information**
- **Head : Feature Preserving/Refining/Unifying > Do task**

논문 리뷰 : FishNet



- **Details**

- **검은 선** : Identity mapping

$$x_{L_s,s} = x_{0,s} + \sum_{l=1}^{L_s} \mathcal{F}(x_{l,s}, W_{l,s}), \quad \frac{\partial \mathcal{L}}{\partial x_{0,s}} = \frac{\partial \mathcal{L}}{\partial x_{L_s,s}}\left(1 + \frac{\partial}{\partial x_{0,s}} \sum_{l=1}^{L_s} \mathcal{F}(x_{l,s}, W_{l,s})\right)$$

- **붉은 선** : Transferring block (residual block)

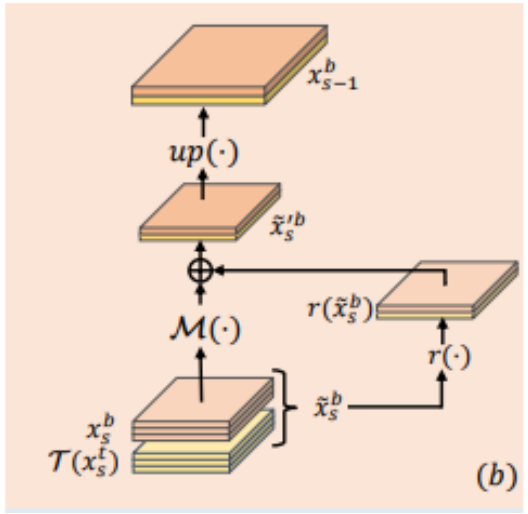
- **주황 선** : Upsample & Refinement

  (Concat > Refinement > Nearest upsample)

- **파랑 선** : Downsample & Refinement

  (Concat > Refinement > Maxpool 2x2)

논문 리뷰 : FishNet



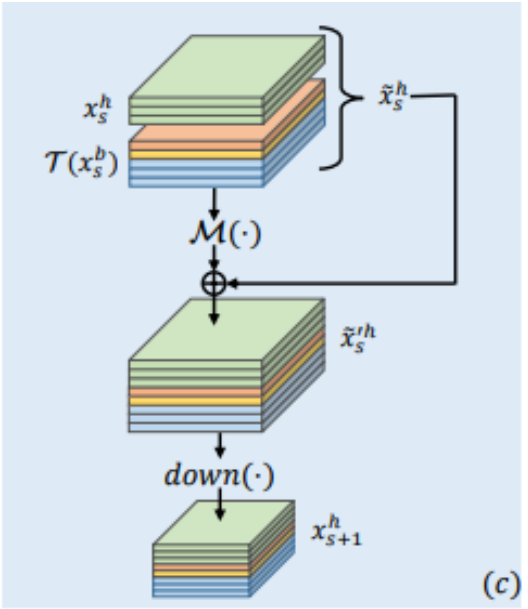
(b) UR block

$$x_{s-1}^b = UR(x_s^b, \mathcal{T}(x_s^t)) = up(\tilde{x}_s'^b)$$

- M(x) : BottleNeck residual unit (conv1x1 > conv3x3 > conv1x1)

- r(x) : Element wise summation (adjacent k)

- up(•) : Nearest upsample (scale 2.0)
  (Upsampled feature는 **dilated convolution으로 해결**)



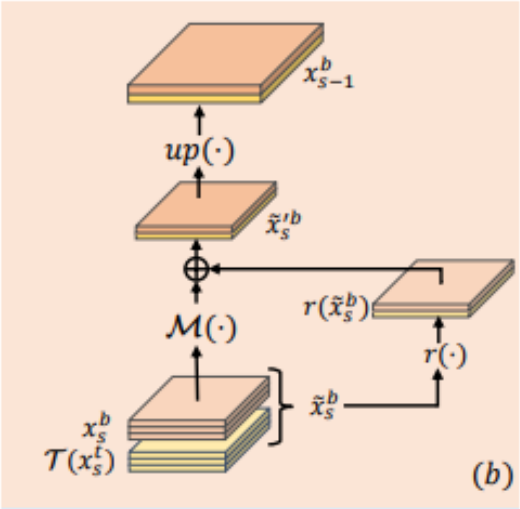
(c) DR block

$$x_{s+1}^h = down(\tilde{x}_s'^h),$$

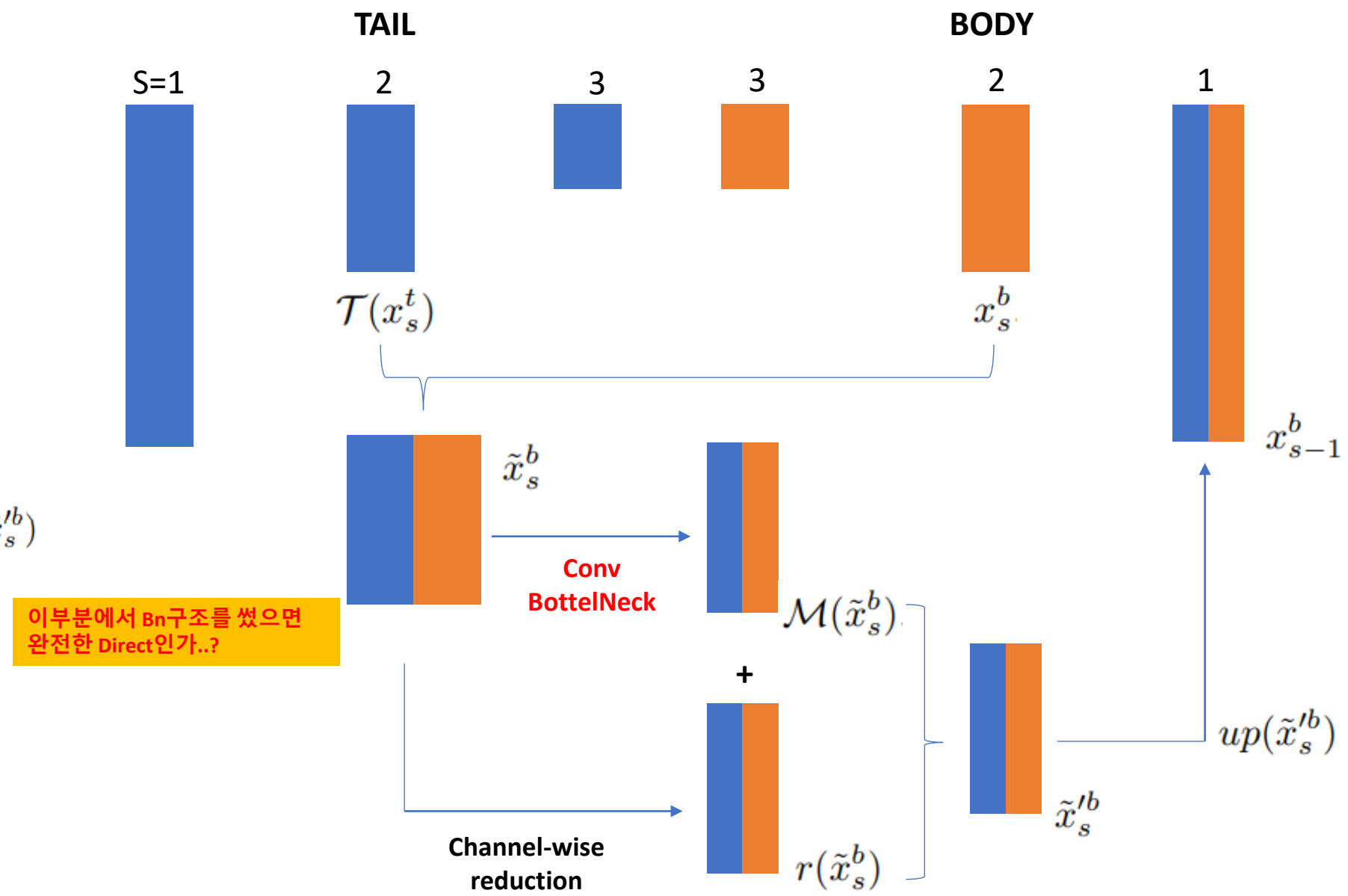
- down (•) : Maxpool 2x2 (stride 2)

논문 리뷰 : FishNet

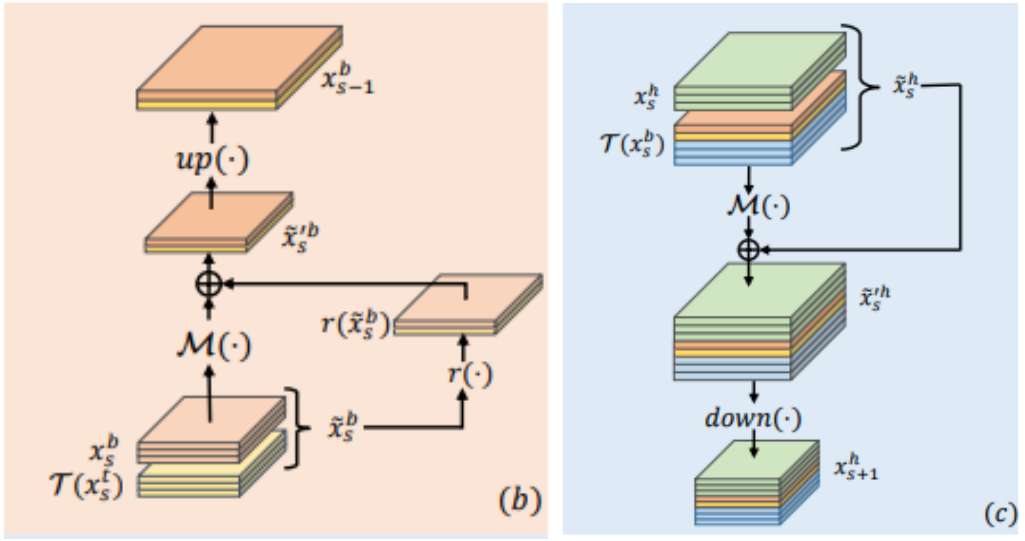
- 이해하기 위한 과정

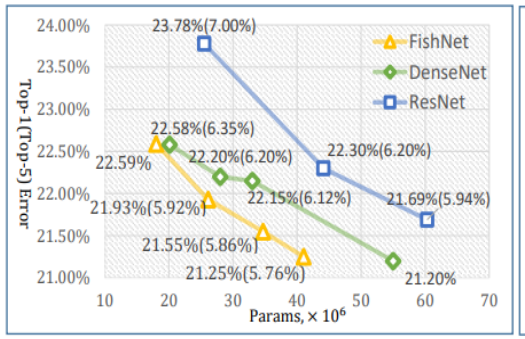


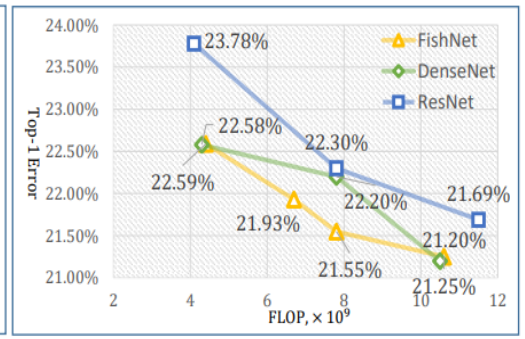
(4) $x_{s-1}^b = UR(x_s^b, \mathcal{T}(x_s^t)) = up(\tilde{x}_s'^b)$



논문 리뷰 : FishNet







- **이러한 구조의 장점**
- 더 적은 파라미터로도 좋은 accuracy
  - ➢ 보존/정제된 Feature의 상호작용이 Width/Depth를 늘리는 것보다 효과적이기 때문

- Body/Head에서 I-Conv를 이용하지 않음
  - ➢ Direct BP를 통해 성능을 향상

- I/R/P level의 feature를 모두 고려
  - ➢ High resolution & Deep feature가 반영된 backbone network
  - ➢ Detection/Segmentation 성능 향상

(전체 논문 내용 리뷰)

https://github.com/crapine1119/fishnet/blob/main/%EB%85%BC%EB%AC%B8%20%26%20PPT/REVIEW.md

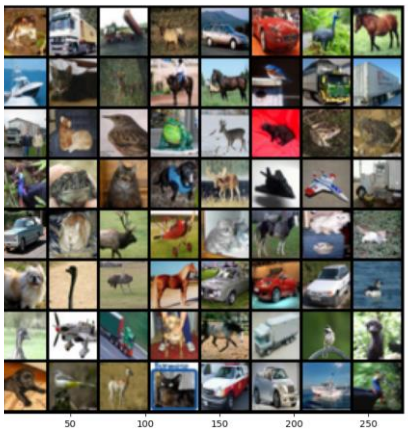
코드 구현

1. Data Read & Check

https://github.com/crapine1119/fishnet

데이터를 읽고, make_grid 함수를 이용해서 이미지를 확인

```
# http://www.cs.toronto.edu/~kriz/cifar.html
Def unpickle(file):
    with open(file, 'rb') as fo:
        dict = pickle.load(fo, encoding='bytes')
    fo.close()
    return dict


# torchvision.utils.make_grid
def check_img(rdir):
    files = glob('%s/*data_batch*'%rdir)
    assert len(files)>0, 'No file error'
    d = unpickle(files[0])
    imgset_flat = d[b'data']
    imgset = imgset_flat.reshape(-1, 3, 32, 32)
    batch = torch.LongTensor(imgset[:64])
    plt.figure(figsize=[18,8])
    plt.imshow(make_grid(batch, padding=2).permute(1, 2, 0).numpy())


def load_cifar(rdir):
    trn_files = glob('%s/*data_batch*'%rdir)
    assert len(trn_files)>0, 'No file error'
    tst_file = '%s/test_batch'%rdir
    # train set
    trnx,trny = [],[]
    for fnm in trn_files:
        batch = unpickle(fnm)
        trnx.append(batch[b'data'])
        trny.extend(batch[b'labels'])
    trnx = np.concatenate(trnx).reshape(-1,3,32,32)
    trny = np.array(trny)

    # test set
    batch = unpickle(tst_file)
    tstx = batch[b'data'].reshape(-1,3,32,32)
    tsty = np.array(batch[b'labels'])
    return trnx,tstx,trny,tsty
```



코드 구현

2. Implement details & Custom dataset

- Image size = 32x32
- Augmentation : Random resize crop (32), Horizontal flip, Fancy PCA
- Optimizer : Adam
- SGD의 학습 속도가 너무 느려서 변경

- Learning rate / weight decay = 1e-2 / 1e-4
- 30 epochs마다 1/10로 감소

- Batch size : 256
- Normalize [0,1]
- Subtracting mean and divide by std

- Loss function = CrossEntropy

```
class custom(Dataset):
    def __init__(self, imgs, labels, trans):
        super().__init__()
        self.imgs =imgs
        self.labels = labels
        self.trans = trans

    def __len__(self):
        return len(self.labels)

    def __getitem__(self, item):
        img = self.imgs[item]
        img = np.transpose(img, (1,2,0))
        label = self.labels[item]
        mat = {}
        img = self.trans(image=img)['image']
        mat['img'] = torch.FloatTensor(img)
        mat['label'] = torch.LongTensor([label])
        return mat
```

코드 구현

3. Make ResNet backbone for Tail

Bottleneck block : Tail/Body/Head에 모두 이용, pre-activation on/off 가능

SE block : Bottleneck 구현이 끝난 후, 쉽게 추가할 수 있도록 구성

```
class BN_block(nn.Module):
    def __init__(self, in_c, out_c, stride, bottleneck, dilation, se, preact):
        super().__init__()
        self.bottleneck = bottleneck
        self.bn_c = int(out_c/bottleneck)
        self.se = se
        self.preact = preact

        if preact:
            self.conv1 = (…)
        else:
            self.conv1 = (…)

        if se:
            self.bridge = SE_block(out_c, out_c)

        self.relu = nn.ReLU(inplace=True)
        self.shortcut = nn.Sequential()

        if (stride != 1) | (in_c != out_c):
            self.shortcut = nn.Conv2d(in_c, out_c, kernel_size=1, stride=stride, bias=False)
```

pre-activation 유무에 따라 Conv, Batch, ReLU 순서 변경

Tail과 Body를 연결하기 위한 SE block

```
    def forward(self, x):
        if self.se:
            feature_map = self.conv(x)
            x = self.shortcut(x) + feature_map*self.bridge(feature_map)
        else:
            x = self.shortcut(x) + self.conv(x)
        if not self.preact:
            x = self.relu(x)
        return x
```

Identity mapping with pre-activation & SE block

```
class SE_block(nn.Module):
    def __init__(self, in_c, out_c, reduction_rate=16):
        super().__init__()
        bn_c = int(in_c / reduction_rate)
        self.se_net = nn.Sequential(nn.AdaptiveAvgPool2d(1), # squeeze
                                    nn.Conv2d(in_c, bn_c, kernel_size=1, stride=1, bias=False),
                                    nn.ReLU(),
                                    nn.Conv2d(bn_c, out_c, kernel_size=1, stride=1, bias=False),
                                    nn.Sigmoid()) # excitation

    def forward(self,x):
        return self.se_net(x)
```

Squeeze > FC > Unsqueeze 대신 Conv2d로 쉽게 변환

코드 구현

3. Make ResNet backbone for Tail

Layer stacking (make_layer)함수를 정의하고 Tail/ module을 완성

Upsample은 Stage가 바뀔 경우 한번만 수행되기 때문에
첫 블록에만 dilated convolution을 할 수 있도록 만들었습니다.

```
def make_layer(block, in_c, out_c, repeat, dilation=1, last=None, preact=True):
    stacked = []
    for i in range(repeat):
        if i==0:
            stacked.append(block(in_c, out_c, dilation=dilation, stride=1, preact=preact))
        else:
            stacked.append(block(out_c, out_c, dilation=1, stride=1 , preact=preact))
    if last is not None:
        stacked.append(last)
    return nn.Sequential(*stacked)
```

2번의 Bottleneck unit은 BN_block.conv를 재활용하였습니다.

```
class tail(nn.Module):
    def __init__(self, in_c, out_c, model, Ls='3,4,6,3', num_c='64,128,256,512', preact=True):
        super().__init__()
        (…)
        if preact:
            self.conv1 = nn.Sequential(nn.Conv2d(in_c, self.num_c[0], kernel_size=3,
                                       stride=1, padding=1, bias=False),
                                  BN_block(self.num_c[0],self.num_c[0],preact=preact).conv,
                                  BN_block(self.num_c[0],self.num_c[0],preact=preact).conv)

        else:
          self.conv1 = nn.Sequential(BN_block(in_c,          self.num_c[0], preact=preact).conv,
                                     BN_block(self.num_c[0],self.num_c[0], preact=preact).conv)
```

Stage 구분

```
self.t1 = make_layer(BN_block, self.num_c[0], self.num_c[1], repeat=self.Ls[0],
                     preact=preact, last=nn.MaxPool2d(kernel_size=2,stride=2,padding=0))
self.t2 = make_layer(BN_block, self.num_c[1], self.num_c[2], repeat=self.Ls[1],
                     preact=preact, last=nn.MaxPool2d(kernel_size=2,stride=2,padding=0))
self.t3 = make_layer(BN_block, self.num_c[2], self.num_c[3], repeat=self.Ls[2],
                     preact=preact, last=nn.MaxPool2d(kernel_size=2,stride=2,padding=0))
```

모델 이름을 지정하여 Tail/ResNet 자유롭게 전환 가능

```
if model=='fishnet':
    self.t4 = make_layer(BN_block, self.num_c[3], self.num_c[3], repeat=self.Ls[3],
     preact=preact, last=BN_block(self.num_c[3], self.num_c[3], se=True, preact=preact))

else: # resnet
    self.t4 = make_layer(BN_block, self.num_c[3], self.num_c[3], repeat=self.Ls[3],
     preact=preact, last=nn.Sequential(nn.AdaptiveAvgPool2d(1),
                                       squeeze(),
                                       nn.Linear(self.num_c[3], out_c)))
```

코드 구현

3. Make ResNet backbone for Tail

완성한 ResNet/Tail 구조

```
resnet(
(conv1):
    (0~1)Residual_block

  (t1): Sequential(
    (0~2): BN_block
    (3): MaxPool2d)

  (t2): Sequential(
    (0~3): BN_block
    (4): MaxPool2d)

  (t3): Sequential(
    (0~5): BN_block
    (6): MaxPool2d)

  (t4): Sequential(

    (0~2): BN_block
    (3): Sequential(
      (0): AdaptiveAvgPool2d
      (1): squeeze()
      (2): Linear(in_features=512, out_features=10, bias=True)))
```

Stage에 따라 3,4,6,3개의 block을 stack

기존 ResNet과 다르게 Stride Convolution 대신 Maxpool을 이용

Tail은 마지막 Stage의 마지막 block만 SE_block으로 변경

```
(bridge): SE_block(
      (se_net): Sequential(
        (0): AdaptiveAvgPool2d(output_size=1)
        (1): Conv2d(512, 32, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (2): ReLU()
        (3): Conv2d(32, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (4): Sigmoid()
```

코드 구현

4. Make Pipeline by pytorch-lightning

Layer stacking (make_layer)함수를 정의하고 Tail module을 완성한 뒤
Pytorch lightning을 이용하여 전체적인 학습 파이프라인을 구성

```
class fishnet(nn.Module):
    def __init__(self, hparams):
        super().__init__()

            self.tail = tail(in_c, block, Ls_tail)
            # self.body = body
            # self.head = head

    def forward(self,x1):

        x2 = self.tail(x1)

        # x3 = self.body(x2)
        # out = self.head(x1,x2,x3)
        # out = self.classification(out)
     return out
```

처음 생각한 모델 구조는
Input x1와 Tail/Body의 output x2,x3를
Head로 전달하는 방식

```
def make_fish(hparams):
    return fishnet(hparams)
```

Argparse를 통해 모델 생성

```
class net(LightningModule):
    def __init__(self, hparams):
        super().__init__()
        self.save_hyperparameters(hparams)

        self.fish = make_fish(hparams)

        self.init_weights()

    def forward(self,x):
        out = self.fish(x)
        return out

    def loss_f(self, modely, targety):
        f = nn.CrossEntropyLoss()
        return f(modely, targety)

    def init_weights(self):

    def configure_optimizers(self):
        return

    def step(self, x):
        return
```

생성한 FishNet을 통해 학습 진행

코드 구현

5. Make Body & Head

Body와 Head에 이용하기 위한 Up/Down sample(UDR) block을 구현

```
class UDR_block(nn.Module):
    def __init__(self, in_c=1024,k=2, phase='up', preact=True):
        super().__init__()
        self.phase=phase

    if phase=='up':
        self.M = BN_block(in_c, int(in_c/k),
                          preact=preact).conv
    else:
        self.M = BN_block(in_c, in_c,
                           preact=preact).conv

        self.upsample = nn.Upsample(scale_factor=2, mode='nearest')
        self.dwsample = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)
        self.avgpool = nn.Sequential(nn.AdaptiveAvgPool2d(1),
                                     squeeze())

    def forward(self,x):

        if self.phase=='up':
            x_new = self.M(x) + self.r_func(x)
            x_new = self.upsample(x_new)

        elif self.phase=='down':
            x_new = x + self.M(x)
            x_new = self.dwsample(x_new)

        elif self.phase=='last':
            x_new = x + self.M(x)
            x_new = self.avgpool(x_new)

        return x_new
```

BottleNeck unit을 통해
Feature Refinement 수행
(Upsample은 reduction rate 적용)

채널을 합칠 때 파라미터 사이즈를
줄이기 위한 channel-wise
(AvgPool을 통해 간단히 구현)

```
    def r_func(self, x, k=2):
        """
        :param x: tensor {n,c,h,w}
        :param k: int
        :return: x with reduced dimension
        """
        _, c, h, w = x.size()
        x_ = x.contiguous().view(-1, c, h * w)

        x_r_ = nn.AvgPool1d(kernel_size=k, stride=k)(x_.permute(0, 2, 1))

        x_r = x_r_.permute(0, 2, 1).contiguous().view(-1, int(c / k), h, w)
        return x_r * k
```

코드 구현

5. Make Body & Head

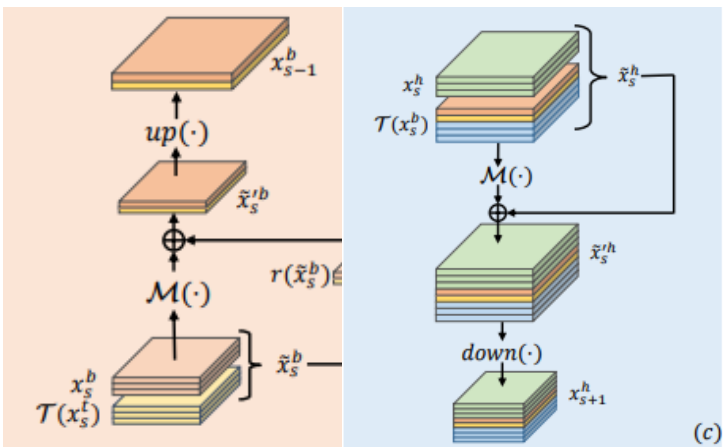
Transferring, Regular Connections, Concatenation을 수행하는 모듈 구성

```
class CAT_block(nn.Module):
    def __init__(self, in_c, add_c, dilation=1, repeat=1, k=2, phase='up', preact=True):
        super().__init__()
        #
        self.phase=phase

        self.regular = make_layer(BN_block, in_c, in_c, repeat=repeat, dilation=dilation,
                                  preact=preact)

        self.transfer = block(add_c, add_c).conv

        self.sample = UDR_block(in_c+add_c, BN_block, k=k, phase=phase)

    def forward(self,x,x_add):
        """
        :param x:
        :param x_add:
        :return: tuple (concatenated, sampled)
        """
        x = self.regular(x) # input

        x_add = self.transfer(x_add)

        x_new = concat(x,x_add)

        out = self.sample(x_new)

        if self.phase=='up':
            return x_new, out
        else:
            return out
```

Body/Head의 Regular connection (Identity mapping)

같은 Stage의 feature를 전달

아래의 그림을 구현하고
Resolution이 바뀐 feature를
다음 Stage로 전달하는 과정



코드 구현

5. Make Body & Head

Body와 Head를 따로 구현하려다 보니 Forward 부분이 불필요하게 복잡해졌고, 이를 해결하기 위해 두 모듈을 동시에 구현

Number of stacked layers in Body & Head

Number of channels in Tail

```
class body_head(nn.Module):
    def __init__(self, k, preact, Ls_body='1,1,1', Ls_head='1,1,1,1', num_c = '64,128,256,512'):
        super().__init__()
        self.Ls_body = [*map(int,Ls_body.split(','))]
        self.Ls_head = [*map(int,Ls_head.split(','))]

        self.in_c, self.added_c = auto_calcul(num_c, k)

        # body
        self.b4 = CAT_block(self.in_c[0], self.added_c[0], repeat=self.Ls_body[0], phase='up',
                            k=k, preact=preact)
        self.b3 = CAT_block(self.in_c[1], self.added_c[1], repeat=self.Ls_body[1], phase='up',
                            k=k, preact=preact, dilation=2)
        self.b2 = CAT_block(self.in_c[2], self.added_c[2], repeat=self.Ls_body[2], phase='up',
                            k=k, preact=preact, dilation=4)

        # head
        self.h1 = CAT_block(self.in_c[3], self.added_c[3], repeat=self.Ls_head[0],phase='down',
                            k=k, preact=preact)
        self.h2 = CAT_block(self.in_c[4], self.added_c[4], repeat=self.Ls_head[1],phase='down',
                            k=k, preact=preact)
        self.h3 = CAT_block(self.in_c[5], self.added_c[5], repeat=self.Ls_head[2],phase='down',
                            k=k, preact=preact)
        self.h4 = CAT_block(self.in_c[6], self.added_c[6], repeat=self.Ls_head[3],phase='last',
                            k=k, preact=preact)

    def forward(self, x):
        _ , x3_ = self.b4(x['bridge'], x['4'])
        x3, x2_ = self.b3(x3_, x['3']) # cated, sampled
        x2, out =self.b2(x2_, x['2'])
        #
        out = self.h1(out,  x['1'])
        out = self.h2(out, x2)
        out = self.h3(out, x3)
        out = self.h4(out, x['4'])
        return out
```

Dilated convolution 구현

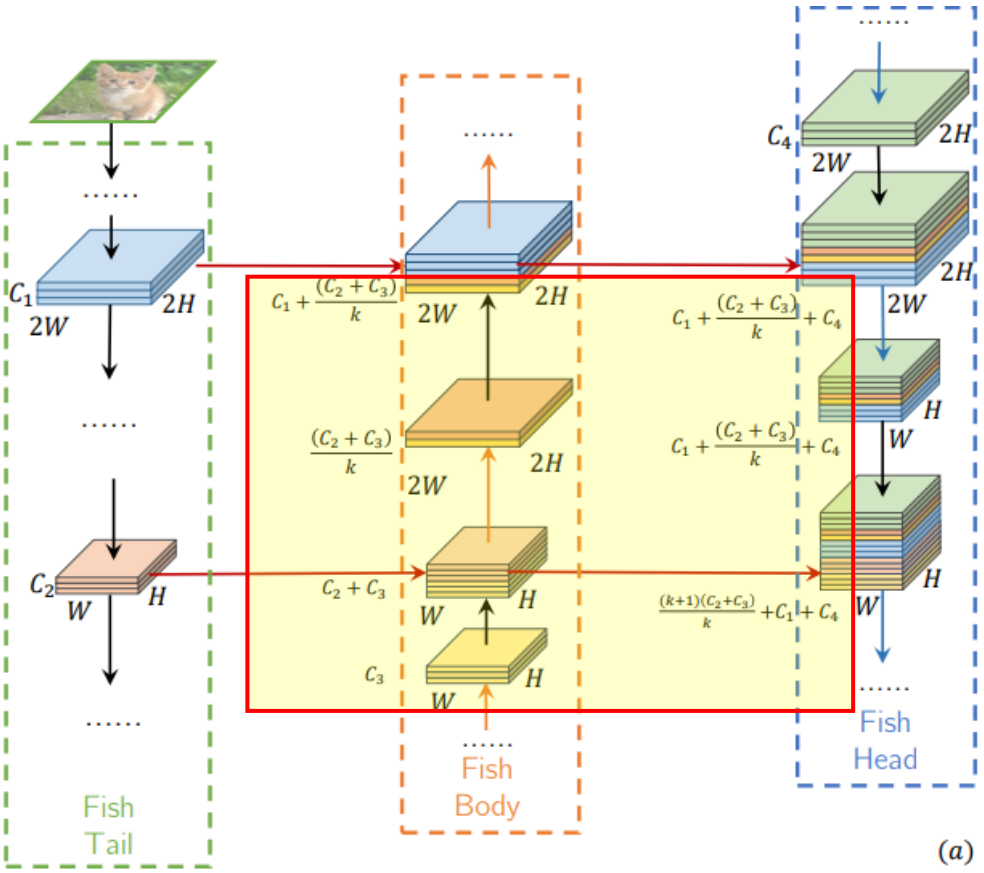
Channel 수를 조절할 때마다

직접 계산해줘야 하는 문제가 발생

코드 구현

6. Channel calculation

아래의 그림을 보고, Tail의 채널만 정해주면 자동으로 계산하는 함수 구현



Regular Connection에 의한 채널 수(검은 선)

```
def auto_calcul(num_c = '64,128,256,512', k=2):
    num_c = [*map(int,num_c.split(','))]

    in_added = [num_c[-1]]

    for i in range(3):
        in_added.append(int((in_added[i] + num_c[-i - 1]) / k))
    in_added.append(int((in_added[3] + num_c[0])))

    num_c.append(in_added[2] + num_c[1])
    num_c.append(in_added[1] + num_c[2])
    num_c.append(num_c[3])
    for i in range(4, 7):
        in_added.append(in_added[i] + num_c[i])

    return in_added, num_c[3::-1]+num_c[4:]
```

Tail 또는 Body에서 전달되는 Feature의 채널 수(붉은 선)

Body / Head에서 Stage가 역전됨을 고려

코드 구현

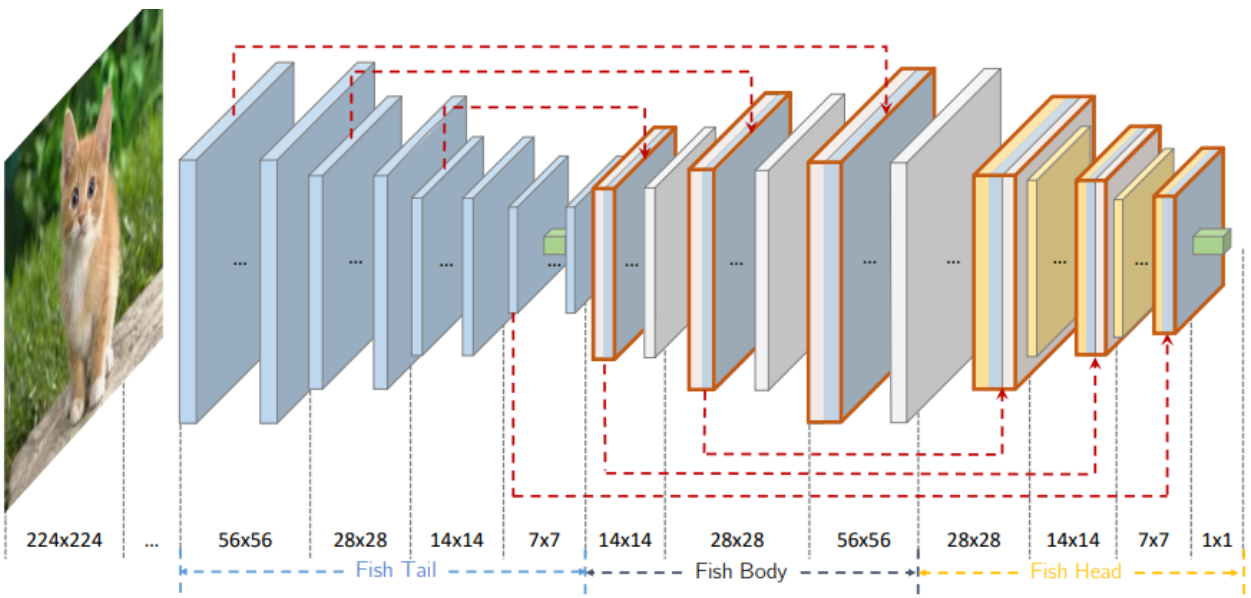
7. FishNet

Tail/Body/Head 모듈을 받아 FishNet99를 구성하고 Classification을 수행

```
class fishnet(nn.Module):
    def __init__(self, hparams):
        super().__init__()
        self.hparams = hparams
        final_c,_ = auto_calcul(self.hparams.num_c, self.hparams.k)

        self.tail = tail(self.hparams.in_c, Ls = self.hparams.Ls_tail,
                         num_c = self.hparams.num_c, preact=self.hparams.preact,
                         model=self.hparams.model)

        if self.hparams.model=='fishnet':
            self.body_head = body_head(k=self.hparams.k, Ls_body=self.hparams.Ls_body,
                                       Ls_head=self.hparams.Ls_head, num_c=self.hparams.num_c,
                                       preact=self.hparams.preact)

            self.cls = nn.Linear(final_c[-1], self.hparams.out_c, bias=False)
```

* FishNet99

	First Layer	T1	T2	T3	T4	B4	B3	B2	H1	H2	H3	H4
Output Size	32	32	16	8	4	8	16	32	16	8	4	1
Blocks	2	3	4	6	3	1	1	1	1	1	1	1
Layers	7	9	12	18	11	6	6	6	6	6	6	6



코드 구현

8. The number parameters & FLOP

https://github.com/sovrasov/flops-counter.pytorch를 이용하여,
구현한 ResNet50과 FishNet99를 확인

```
from ptflops import get_model_complexity_info
# https://github.com/sovrasov/flops-counter.pytorch
def get_params(model, input_size):
    with torch.cuda.device(0):
      macs, params = get_model_complexity_info(model, input_size,
                                               as_strings=False,
                                               print_per_layer_stat=False,
                                               verbose=True)

    print('FLOP : %2.2f G'%(2*macs*1e-9))
    print('Params : %2.2f M'%(params*1e-6))
```

* The number parameters & FLOP

	FLOP (1e+9)	Params (1e+6)
ResNet50	0.56	3.00
FishNet99	2.53	17.92

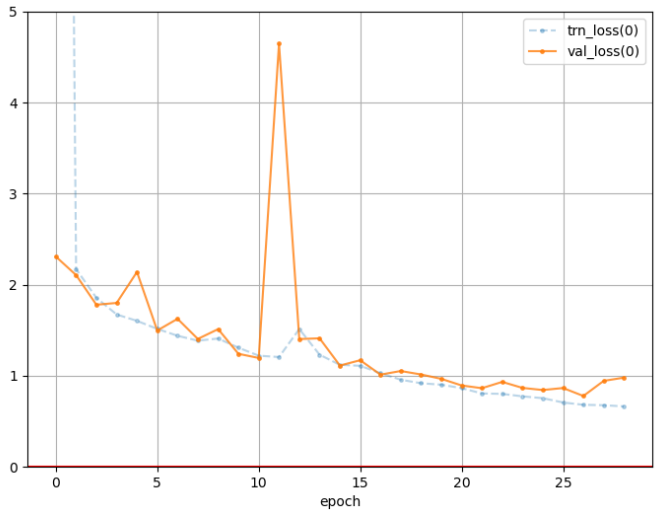
코드 구현

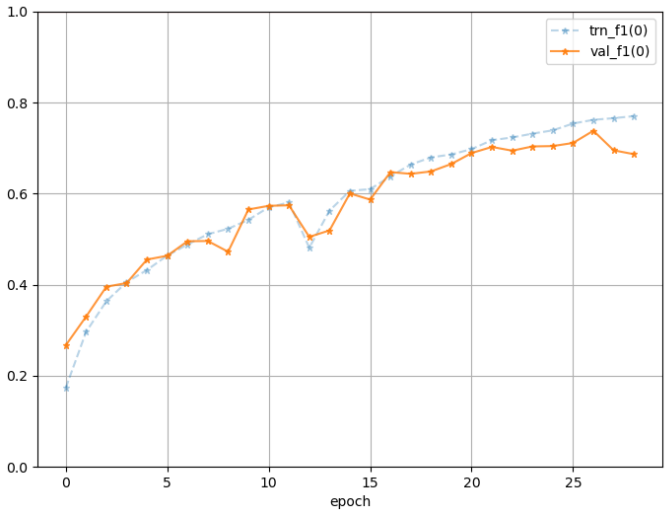
9. Test

Top 1 & 5 error 확인 : 약 30 epoch 기준으로 27.0 (1.7)%를 기록

```
def get_err(model, hparams, ckp, tst_loader):
    model = model.load_from_checkpoint(checkpoint_path=ckp, hparams=hparams)
    model.cuda()
    model.eval()
    num1,numk=0,0
    n_dset = 0
    for i in tqdm(tst_loader):
        with torch.no_grad():
            ans = model(i['img'].cuda())
        _,top1 = ans.topk(1,dim=-1)
        _,top5 = ans.topk(5,dim=-1)
        num1+=(top1.cpu()==i['label']).squeeze().sum().item()
        numk+=(top5.cpu()==i['label']).any(dim=1).sum().item()
        n_dset+=len(top1)
    return {'top1':num1/n_dset,
            'top5':numk/n_dset}
```

* Loss plot





개선 사항

- r(x) : Element wise summation

$$r(x) = \hat{x} = [\hat{x}(1), \hat{x}(2), \ldots, \hat{x}(c_{out})], \quad \hat{x}(n) = \sum_{j=0}^{k} x(k \cdot n + j), n \in \{0, 1, ..., c_{out}\}, \qquad (8)$$

  - 간단한 식이지만 전개할 때 이상함을 느꼈습니다.
  - k개의 element-wise summation을 나타내려면 우측 식의 붉은 박스에서 k를 k-1로 수정해야 합니다.
  - 좌측 식과 우측 식의 n의 수가 일치하지 않습니다.

개선 사항

- Backbone for Object detection & Segmentation?

  - R/P level을 고려하기 위한 네트워크지만 Classification backbone 구조를 차용한 점이 모순적입니다.
    이를 해결하기 위한 새로운 Tail 구조가 필요하다고 생각합니다.

  - R/P level에 집중하기 위해 Tail에 남아있는 I-conv 대신, Channel에 upsample을 적용하는 방법을 제안합니다.

  - 코드를 구현하면서 Tail과 Body를 연결하는 부분의 Identity mapping이 적절치 않다는 생각이 들었습니다.
    SE_block을 이용하지 않고 마지막 Stage를 보존하는 것이 direct gradient propagation에 효과적일 것 같습니다.

- 느낀점

  - Direct propagation을 통해 R/P level meaning을 더 효과적으로 포착할 수 있다는 점이 놀랍습니다.
  - 구조의 변경함으로써 성능을 개선할 뿐 아니라, 그것의 의미를 찾고자 한 의미있는 논문이었습니다.