GAN:
Projects

Contents:

Contents:

I. Project I: Colorization

I. Project I: Colorization

A. Basic Structure

- **Straight-line Structure**

```
class Net(nn.Module):
    def __init__(self, input_size=256):
        super(Net, self).__init__()
        # ResNet - First layer accepts grayscale images,
        # and we take only the first few layers of ResNet for this task
        resnet = models.resnet18(num_classes=100)
        resnet.conv1.weight = nn.Parameter(resnet.conv1.weight.sum(dim=1).unsqueeze(1))
        self.midlevel_resnet = nn.Sequential(*list(resnet.children())[0:6])
        RESNET_FEATURE_SIZE = 128
```

```
class Net(nn.Module):
    def __init__(self, input_size=256):
        super(Net, self).__init__()
        # ResNet - First layer accepts grayscale images,
        # and we take only the first few layers of ResNet for this task
        resnet = models.resnet18(num_classes=100)
        resnet.conv1.weight = nn.Parameter(resnet.conv1.weight.sum(dim=1).unsqueeze(1))
        self.midlevel_resnet = nn.Sequential(*list(resnet.children())[0:6])
        RESNET_FEATURE_SIZE = 128
        # Upsampling Network
        self.upsample = nn.Sequential(
          nn.Conv2d(RESNET_FEATURE_SIZE, 128, kernel_size=3, stride=1, padding=1),
                                                                    s=3, stride=1, padding=1),
                                                                    =3, stride=1, padding=1),
                                                                    =3, stride=1, padding=1),
          nn.Conv2d(32, 3, kernel_size=3, stride=1, padding=1),
          nn.Upsample(scale_factor=2)
        )

    def forward(self, input):
        midlevel_features = self.midlevel_resnet(input)
        output = self.upsample(midlevel_features)
        return output
```

I. Project I: Colorization

A. Basic Structure

- **Straight-line Structure**

```
>>> print(hei[0])
Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
>>> print(hei[1])
BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
>>> print(hei[2])
ReLU(inplace=True)
>>> print(hei[3])
MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
```

sum all parameters along 1st dim
[64, 3, 7, 7] → [64, 7, 7]

expand 1 dim at 1st spot
[64, 7, 7] → [64, 1, 7, 7]

```
class Net(nn.Module):
    def __init__(self, input_size=256):
        super(Net, self).__init__()
        # ResNet - First layer accepts grayscale images,
        # and we take only the first few layers of ResNet for this task
        resnet = models.resnet18(num_classes=100)
        resnet.conv1.weight = nn.Parameter(resnet.conv1.weight.sum(dim=1).unsqueeze(1))
        self.midlevel_resnet = nn.Sequential(*list(resnet.children())[0:6])
        RESNET_FEATURE_SIZE = 128
```

Set conv1.weight from 3-dim to 1-dim [gray image]

unpack a list

resnet layers/operations

```
>>> print(hei[4])
Sequential(
  (0): BasicBlock(
    (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (1): BasicBlock(
    (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
)
```

nn.Sequential():
It contains sequential/ordered executions

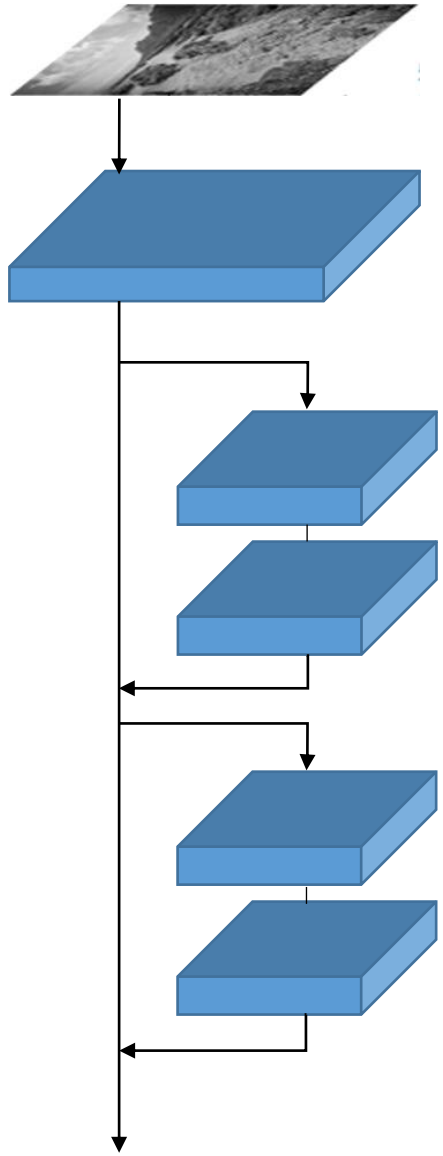
I. Project I: Colorization

A. Basic Structure

- **Straight-line Structure**

```
class Net(nn.Module):
    def __init__(self, input_size=256):
        super(Net, self).__init__()
        # ResNet - First layer accepts grayscale images,
        # and we take only the first few layers of ResNet for this task
        resnet = models.resnet18(num_classes=100)
        resnet.conv1.weight = nn.Parameter(resnet.conv1.weight.sum(dim=1).unsqueeze(1))
        self.midlevel_resnet = nn.Sequential(*list(resnet.children())[0:6])
        RESNET_FEATURE_SIZE = 128
```

```
>>> print(hei[0])
Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
>>> print(hei[1])
BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
>>> print(hei[2])
ReLU(inplace=True)
>>> print(hei[3])
MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
>>> print(hei[4])
Sequential(
  (0): BasicBlock(
    (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (1): BasicBlock(
    (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
)
```

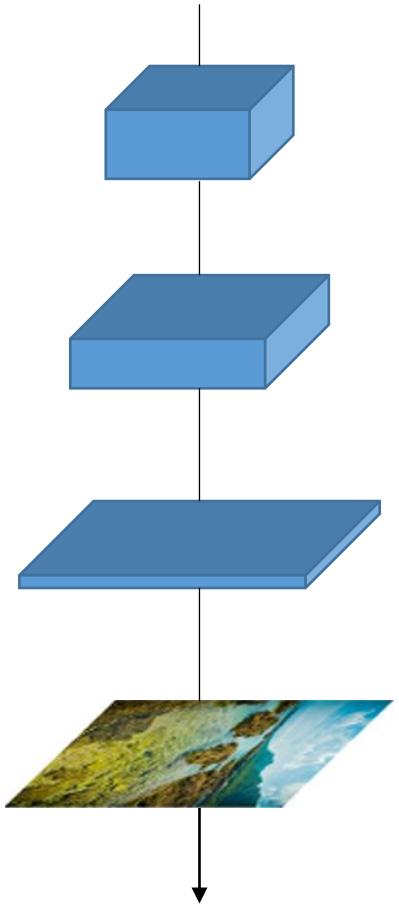


I. Project I: Colorization

A. Basic Structure

- **Straight-line Structure**

```
class Net(nn.Module):
    def __init__(self, input_size=256):
        ......
        RESNET_FEATURE_SIZE = 128
        # Upsampling Network
        self.upsample = nn.Sequential(
          nn.Conv2d(RESNET_FEATURE_SIZE, 128, kernel_size=3, stride=1, padding=1),
          nn.BatchNorm2d(128),
          nn.ReLU(),
          nn.Upsample(scale_factor=2),
          nn.Conv2d(128, 64, kernel_size=3, stride=1, padding=1),
          nn.BatchNorm2d(64),
          nn.ReLU(),
          nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1),
          nn.BatchNorm2d(64),
          nn.ReLU(),
          nn.Upsample(scale_factor=2),
          nn.Conv2d(64, 32, kernel_size=3, stride=1, padding=1),
          nn.BatchNorm2d(32),
          nn.ReLU(),
          nn.Conv2d(32, 3, kernel_size=3, stride=1, padding=1),
          nn.Upsample(scale_factor=2)
        )

    def forward(self, input):
        midlevel_features = self.midlevel_resnet(input)
        output = self.upsample(midlevel_features)
        return output
```



I. Project I: Colorization

B. Updated Version

- **UNet Structure**

```
class Net(nn.Module):
    def __init__(self, input_size=256):
        ......
        RESNET_FEATURE_SIZE = 128
        # Upsampling Network
        self.upsample = nn.Sequential(
          nn.Conv2d(RESNET_FEATURE_SIZE, 128, kernel_size=3, stride=1, padding=1),
          nn.BatchNorm2d(128),
          nn.ReLU(),
          nn.Upsample(scale_factor=2),
          nn.Conv2d(128, 64, kernel_size=3, stride=1, padding=1),
          nn.BatchNorm2d(64),
          nn.ReLU(),
          nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1),
          nn.BatchNorm2d(64),
          nn.ReLU(),
          nn.Upsample(scale_factor=2),
          nn.Conv2d(64, 32, kernel_size=3, stride=1, padding=1),
          nn.BatchNorm2d(32),
          nn.ReLU(),
          nn.Conv2d(32, 3, kernel_size=3, stride=1, padding=1),
          nn.Upsample(scale_factor=2)
        )

    def forward(self, input):
        midlevel_features = self.midlevel_resnet(input)
        output = self.upsample(midlevel_features)
        return output
```

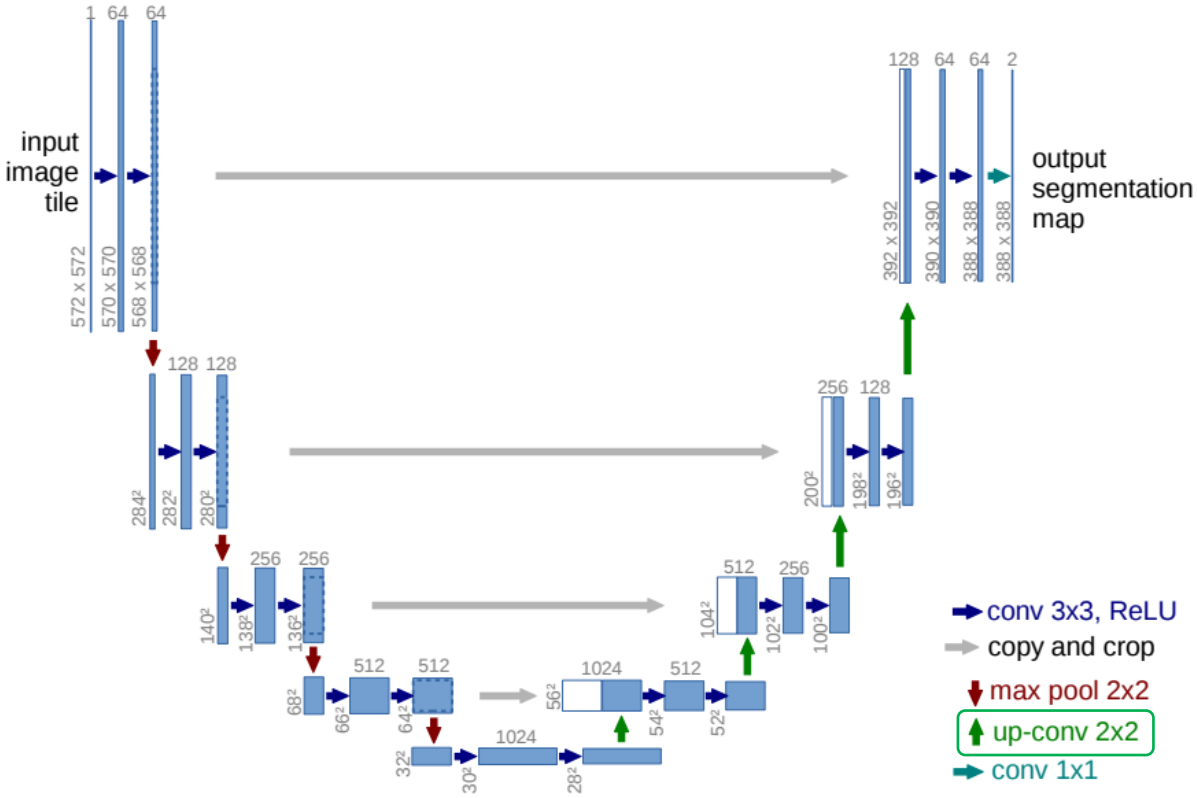
Encoder

Decoder

I. Project I: Colorization

B. Updated Version

- **UNet Structure [2015, Olaf Ronneberger]**



I. Project I: Colorization

B. Updated Version

- **Deconv / Transpose Conv / Strided Conv, Upsample**

  Normal convolution

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \otimes \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} 1a+2b+3d+4e & 1b+2c+3e+4f \\ 1d+2e+3g+4h & 1e+2f+3h+4i \end{pmatrix} \equiv \begin{pmatrix} A & B \\ C & D \end{pmatrix}$$

I. Project I: Colorization

B. Updated Version

- **Deconv / Transpose Conv / Strided Conv, Upsample**

Normal convolution

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \otimes \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} 1a+2b+3d+4e & 1b+2c+3e+4f \\ 1d+2e+3g+4h & 1e+2f+3h+4i \end{pmatrix} \equiv \begin{pmatrix} A & B \\ C & D \end{pmatrix}$$

$\mathbf{3 \times 3 \rightarrow 2 \times 2}$

变小了

Normal convolution in matrix multiplication

$$\begin{pmatrix} A \\ B \\ C \\ D \end{pmatrix} = \begin{pmatrix} 1 & 2 & 0 & 3 & 4 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 & 3 & 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 2 & 0 & 3 & 4 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 & 0 & 3 & 4 \end{pmatrix} * \begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \\ i \end{pmatrix}$$

$\mathbf{9 \rightarrow 4}$

I. Project I: Colorization

B. Updated Version

- **Deconv / Transpose Conv / Strided Conv, Upsample**

Normal convolution in matrix multiplication

$$\begin{pmatrix} A \\ B \\ C \\ D \end{pmatrix} = \begin{pmatrix} 1 & 2 & 0 & 3 & 4 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 & 3 & 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 2 & 0 & 3 & 4 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 & 0 & 3 & 4 \end{pmatrix} * \begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \\ i \end{pmatrix}$$

变小了

$\mathbf{9 \rightarrow 4}$

Normal transpose convolution in matrix multiplication

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 0 & 3 & 4 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 & 3 & 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 2 & 0 & 3 & 4 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 & 0 & 3 & 4 \end{pmatrix}^{T} * \begin{pmatrix} A' \\ B' \\ C' \\ D' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 3 & 0 & 1 & 0 \\ 4 & 3 & 2 & 1 \\ 0 & 4 & 0 & 2 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 4 & 3 \\ 0 & 0 & 0 & 4 \end{pmatrix} * \begin{pmatrix} A' \\ B' \\ C' \\ D' \end{pmatrix}$$

变大了

$\mathbf{4 \rightarrow 9}$

I. Project I: Colorization

B. Updated Version

- **Deconv / Transpose Conv / Strided Conv, Upsample**

Normal transpose convolution in matrix multiplication

变大了

$\mathbf{4 \rightarrow 9}$

$$\begin{pmatrix} x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 \\ x_7 & x_8 & x_9 \end{pmatrix} \equiv \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 0 & 3 & 4 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 & 3 & 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 2 & 0 & 3 & 4 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 & 0 & 3 & 4 \end{pmatrix}^{\boldsymbol{T}} * \begin{pmatrix} A' \\ B' \\ C' \\ D' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 3 & 0 & 1 & 0 \\ 4 & 3 & 2 & 1 \\ 0 & 4 & 0 & 2 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 4 & 3 \\ 0 & 0 & 0 & 4 \end{pmatrix} * \begin{pmatrix} A' \\ B' \\ C' \\ D' \end{pmatrix} \equiv \begin{pmatrix} A' & B' \\ C' & D' \end{pmatrix}$$

By transposing the kernel matrix,
we can upscale the output vector comparing to the size of the input vector

I. Project I: Colorization

B. Updated Version

- **Deconv / Transpose Conv / Strided Conv, Upsample**

Normal transpose convolution in matrix multiplication

$$\begin{pmatrix} x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 \\ x_7 & x_8 & x_9 \end{pmatrix} \equiv \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 0 & 3 & 4 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 & 3 & 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 2 & 0 & 3 & 4 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 & 0 & 3 & 4 \end{pmatrix}^T * \begin{pmatrix} A' \\ B' \\ C' \\ D' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 3 & 0 & 1 & 0 \\ 4 & 3 & 2 & 1 \\ 0 & 4 & 0 & 2 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 4 & 3 \\ 0 & 0 & 0 & 4 \end{pmatrix} * \begin{pmatrix} A' \\ B' \\ C' \\ D' \end{pmatrix} = \begin{pmatrix} A' \\ 2A' + B' \\ 2B' \\ 3A' + C' \\ 4A' + 3B' + 2B' + C' \\ 4B' + 2D' \\ 3C' \\ 4C' + 3D' \\ 4D' \end{pmatrix}$$

I. Project I: Colorization

B. Updated Version

- **Deconv / Transpose Conv / Strided Conv, Upsample**

$$\begin{pmatrix} A' & B' \\ C' & D' \end{pmatrix} \otimes \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} A' & B' & 0 \\ C' & D' & 0 \\ 0 & 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 2A' & 2B' \\ 0 & 2C' & 2D' \\ 0 & 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 \\ 3A' & 3B' & 0 \\ 3C' & 3D' & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 \\ 0 & 4A' & 4B' \\ 0 & 4C' & 4D' \end{pmatrix}$$

$$= \begin{pmatrix} A' & 2A' + B' & 2B' \\ 3A' + C' & 4A' + 3B' + 2B' + C' & 4B' + 2D' \\ 3C' & 4C' + 3D' & 4D' \end{pmatrix} \equiv \begin{pmatrix} A' \\ 2A' + B' \\ 2B' \\ 3A' + C' \\ 4A' + 3B' + 2B' + C' \\ 4B' + 2D' \\ 3C' \\ 4C' + 3D' \\ 4D' \end{pmatrix}$$

变大了

$\mathbf{2 \times 2 \rightarrow 3 \times 3}$

I. Project I: Colorization

B. Updated Version

- **Deconv / Transpose Conv / Strided Conv, Upsample**

$$\begin{pmatrix}1 & 2\\ 3 & 4\end{pmatrix}\otimes\begin{pmatrix}A' & B'\\ C' & D'\end{pmatrix}=\begin{pmatrix}A' & 2A' & 0\\ 3A' & 4A' & 0\\ 0 & 0 & 0\end{pmatrix}+\begin{pmatrix}0 & B' & 2B'\\ 0 & 3B' & 4B'\\ 0 & 0 & 0\end{pmatrix}+\begin{pmatrix}0 & 0 & 0\\ C' & 2C' & 0\\ 3C' & 4C' & 0\end{pmatrix}+\begin{pmatrix}0 & 0 & 0\\ 0 & D' & 2D'\\ 0 & 3D' & 4D'\end{pmatrix}$$

$$=\begin{pmatrix}A' & 2A'+B' & 2B'\\ 3A'+C' & 4A'+3B'+2B'+C' & 4B'+2D'\\ 3C' & 4C'+3D' & 4D'\end{pmatrix}\equiv\begin{pmatrix}A'\\ 2A'+B'\\ 2B'\\ 3A'+C'\\ 4A'+3B'+2B'+C'\\ 4B'+2D'\\ 3C'\\ 4C'+3D'\\ 4D'\end{pmatrix}$$

变大了

$\mathbf{2\times 2\rightarrow 3\times 3}$

I. Project I: Colorization

B. Updated Version

- **Deconv / Transpose Conv / Strided Conv, Upsample**

```
>>> input = torch.arange(1, 5, dtype=torch.float32).view(1, 1, 2, 2)
>>> input
tensor([[[[ 1.,  2.],
          [ 3.,  4.]]]])

>>> m = nn.Upsample(scale_factor=2, mode='nearest')
>>> m(input)
tensor([[[[ 1.,  1.,  2.,  2.],
          [ 1.,  1.,  2.,  2.],
          [ 3.,  3.,  4.,  4.],
          [ 3.,  3.,  4.,  4.]]]])
```

```
>>> m = nn.Upsample(scale_factor=2, mode='bilinear')
>>> m(input)
tensor([[[[ 1.0000,  1.2500,  1.7500,  2.0000],
          [ 1.5000,  1.7500,  2.2500,  2.5000],
          [ 2.5000,  2.7500,  3.2500,  3.5000],
          [ 3.0000,  3.2500,  3.7500,  4.0000]]]])
```

I. Project I: Colorization

B. Updated Version

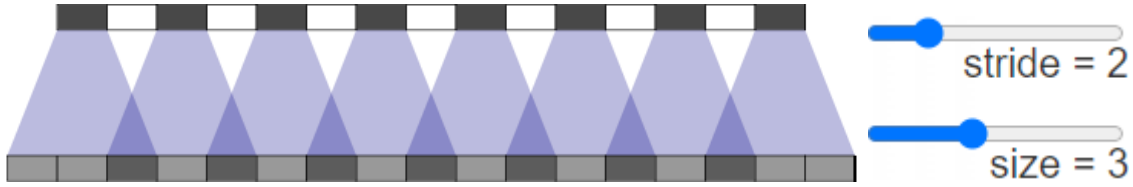
- **Transpose Conv / Strided Conv bad effect – chessboard pattern**

I. Project I: Colorization

B. Updated Version

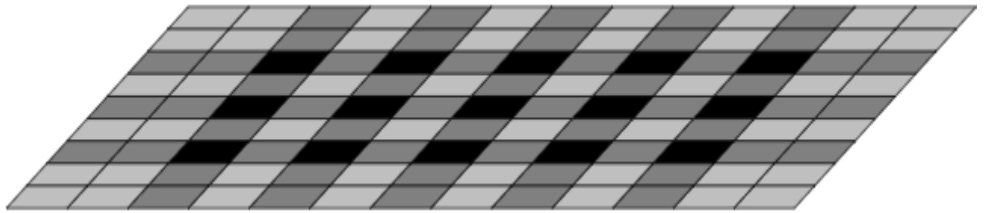
- **Transpose Conv / Strided Conv bad effect – chessboard pattern**

Reason: great illustration



How to solve:

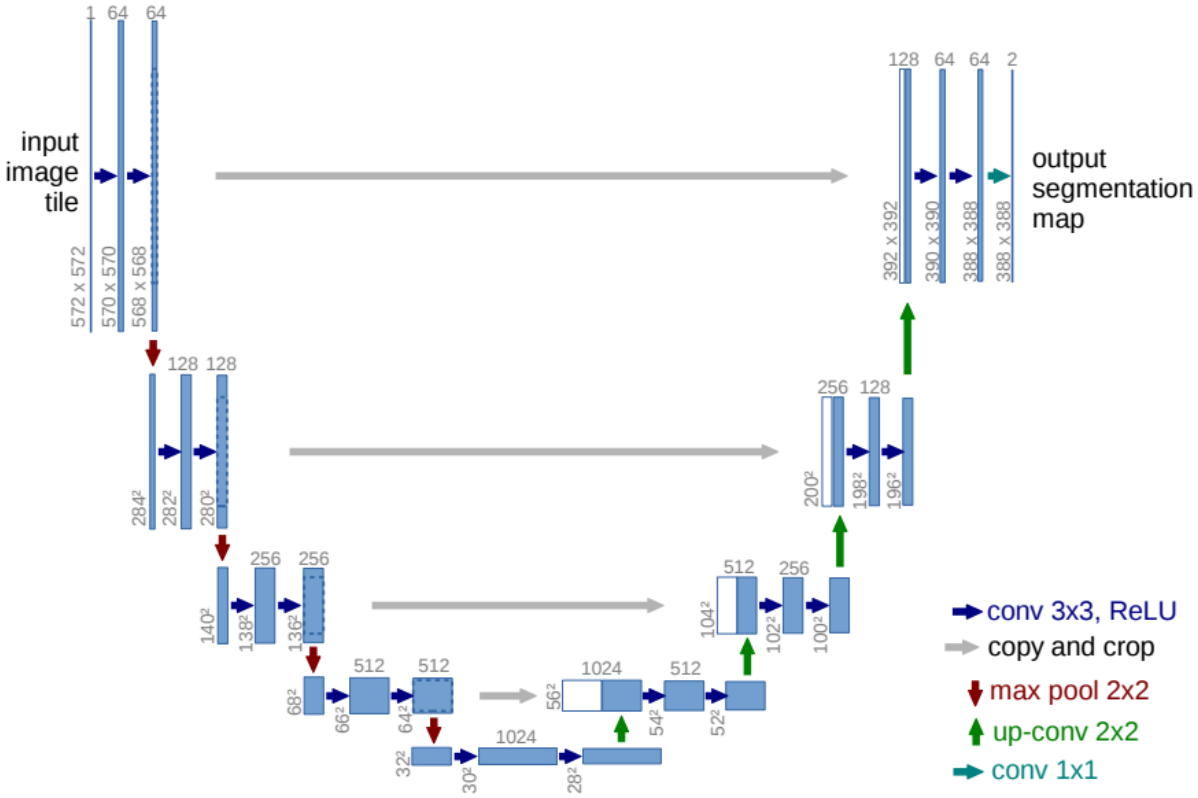
1. Ticks provided in the link
   [sophisticated design] → deprecated
2. Several training strategies,
   we'll talk later



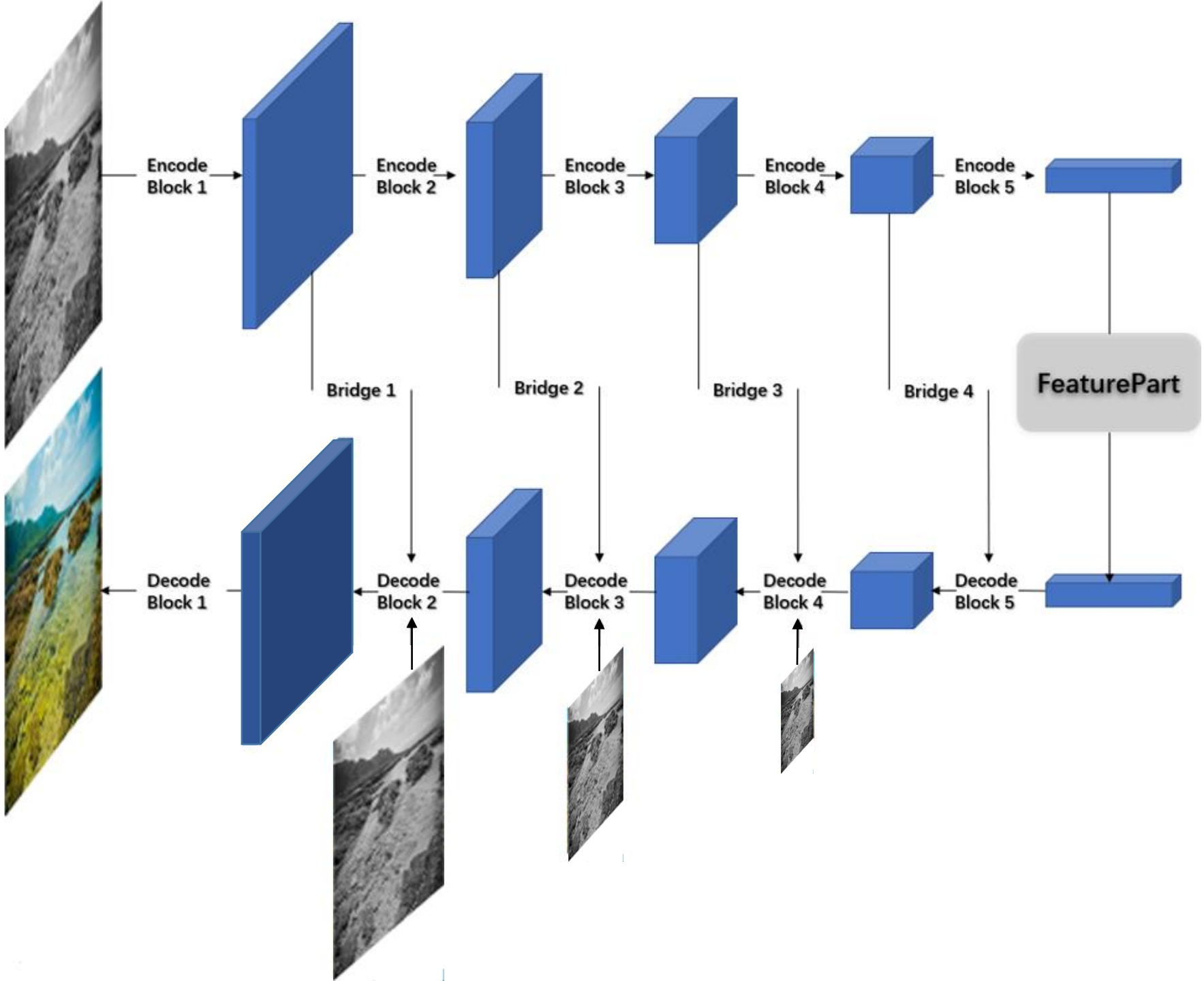
I. Project I: Colorization

B. Updated Version

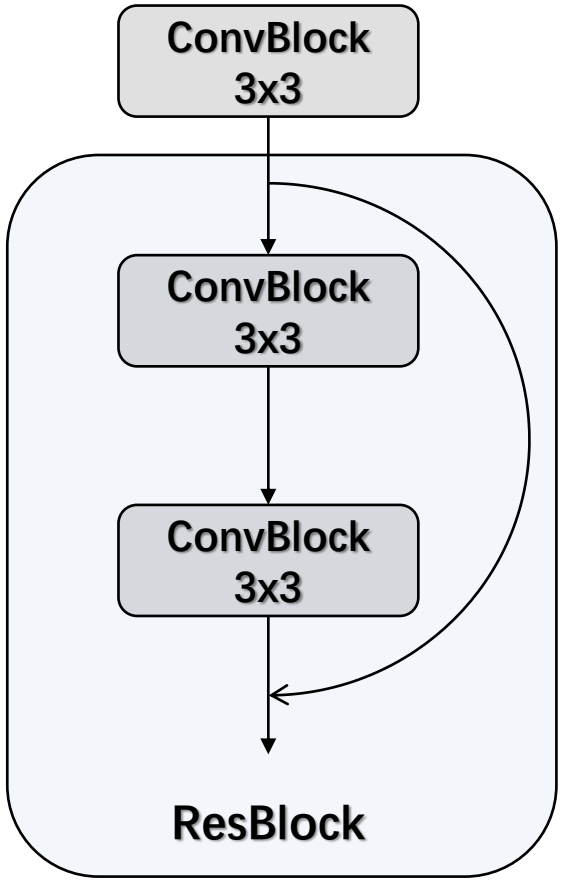
- **UNet Structure [2015, Olaf Ronneberger]**



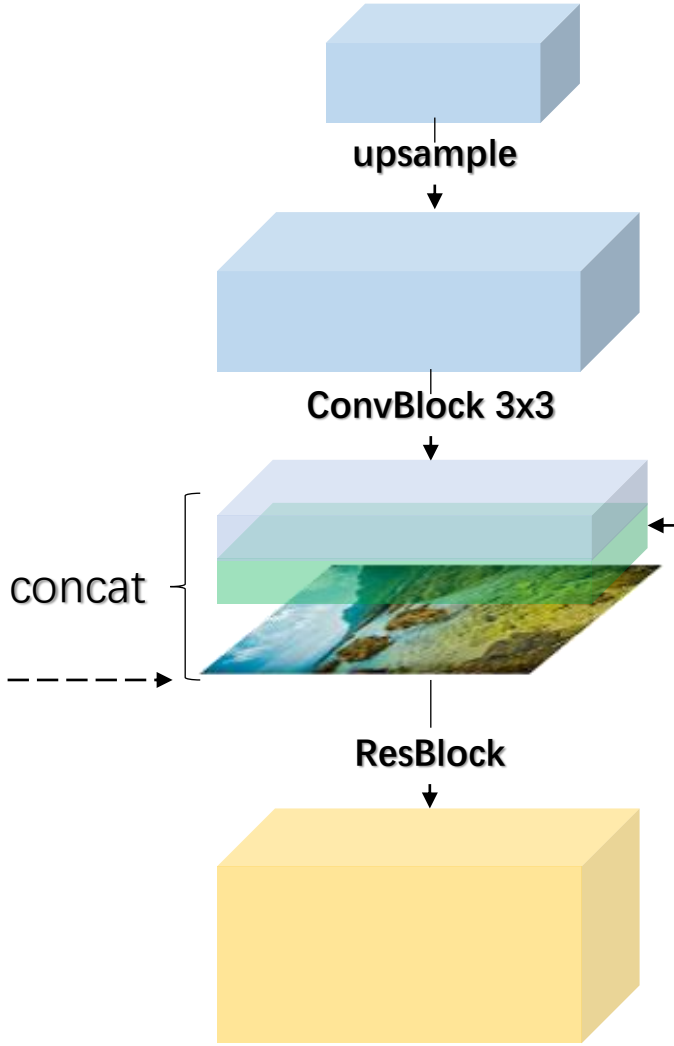
Any ideas to make it better?


Encode Block 1
Encode Block 2
Encode Block 3
Encode Block 4
Encode Block 5
FeaturePart
Bridge 1
Bridge 2
Bridge 3
Bridge 4
Decode Block 1
Decode Block 2
Decode Block 3
Decode Block 4
Decode Block 5

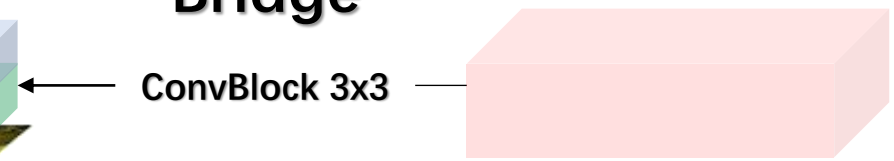
Encode Block



Decode Block



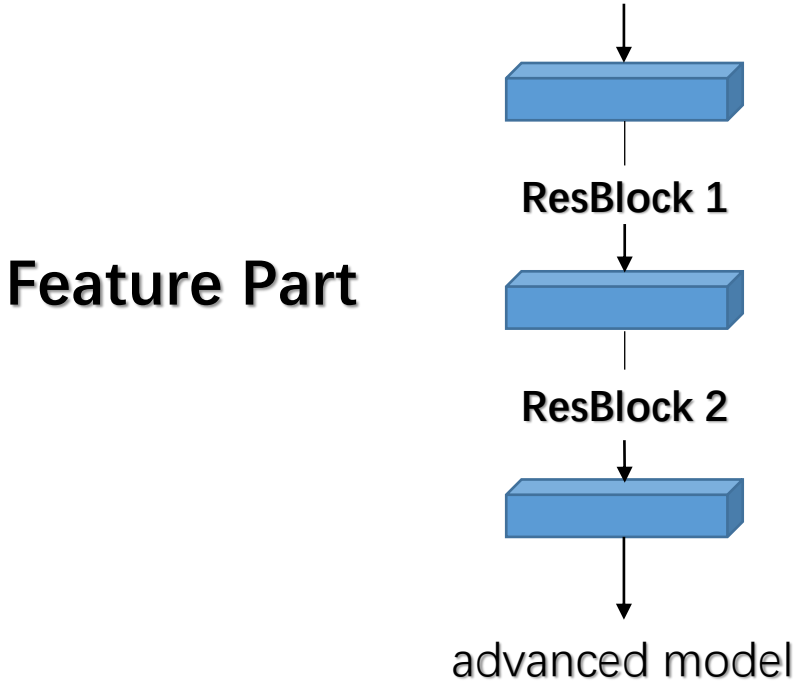
Bridge



I. Project I: Colorization

B. Updated Version

- **UNet Structure [2015, Olaf Ronneberger]**



I. Project I: Colorization

C. GAN based method

- **G-Net**

  We could just use modified UNet

- **D-Net**

  At current stage, D is simpler

```
self.conv0 = ConvBR(c_in=3,
                    kernel_
                    weight_
self.conv1 = ConvBR(c_in=di
                    kernel_
                    weight_
self.conv2 = ConvBR(c_in=di
                    kernel_
                    weight_
self.conv3 = ResBasicBlock(


self.conv4 = ResBasicBlock(


self.conv5 = ConvBR(c_in=di
                    kernel_
                    weight_
```

```
def forward(self, x):
    # 256 -> 128
    x = self.conv0(x)
    # 128 -> 64
    x = self.conv1(x)
    # 64 -> 32
    x = self.conv2(x)
    # 32 -> 32
    x = self.conv3(x)
    # 32 -> 32
    x = self.conv4(x)
    # 32 -> 32
    x = self.conv5(x)
    return x
```

I. Project I: Colorization

C. GAN based method

- **G-Net**

  We could just use modified UNet

- **D-Net**

  At current stage, D is simpler

Question:
Why the output size is 32x32 after D?
Why it's not a single number
[no matter whether it's normal GAN or WGAN]

we'll talk later

```
self.conv0 = ConvBR(c_in=3,
                    kernel_
                    weight_
self.conv1 = ConvBR(c_in=di
                    kernel_
                    weight_
self.conv2 = ConvBR(c_in=di
                    kernel_
                    weight_
self.conv3 = ResBasicBlock(


self.conv4 = ResBasicBlock(


self.conv5 = ConvBR(c_in=di
                    kernel_
                    weight_
```

```
def forward(self, x):
    # 256 -> 128
    x = self.conv0(x)
    # 128 -> 64
    x = self.conv1(x)
    # 64 -> 32
    x = self.conv2(x)
    # 32 -> 32
    x = self.conv3(x)
    # 32 -> 32
    x = self.conv4(x)
    # 32 -> 32
    x = self.conv5(x)
    return x
```

I. Project I: Colorization

C. GAN based method

- **Loss(es)**

D-Net

WGAN-GP
- adversarial loss [WGAN]
- gradient penalty [-GP]

G-Net

WGAN: adversarial loss

pixel level loss: L1 (texture sensitive) / L2 (color sensitive) [both not enough]

perceptual loss

tv loss

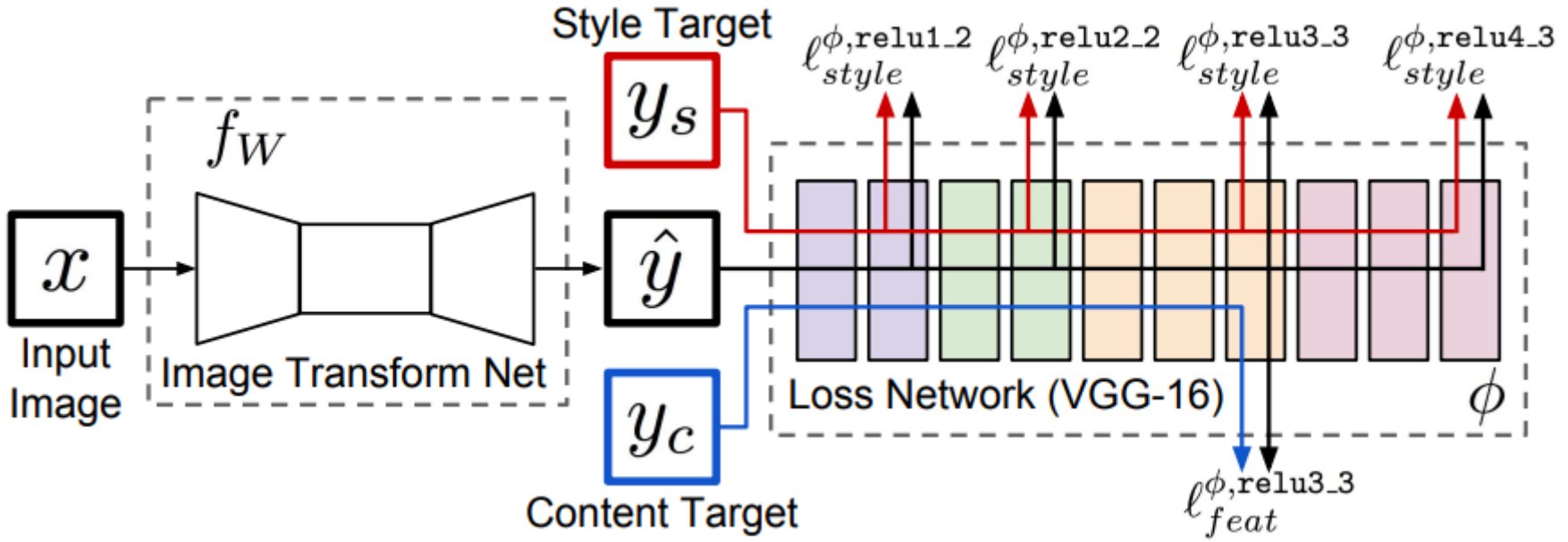
} which can also help to solve the problem of chessboard pattern

I. Project I: Colorization

D. Tricky Losses: Perceptual Loss

Perceptual Losses for Real-Time Style Transfer and Super-Resolution

[Justin Johnson, 2016, link]



I. Project I: Colorization

D. Perceptual Loss

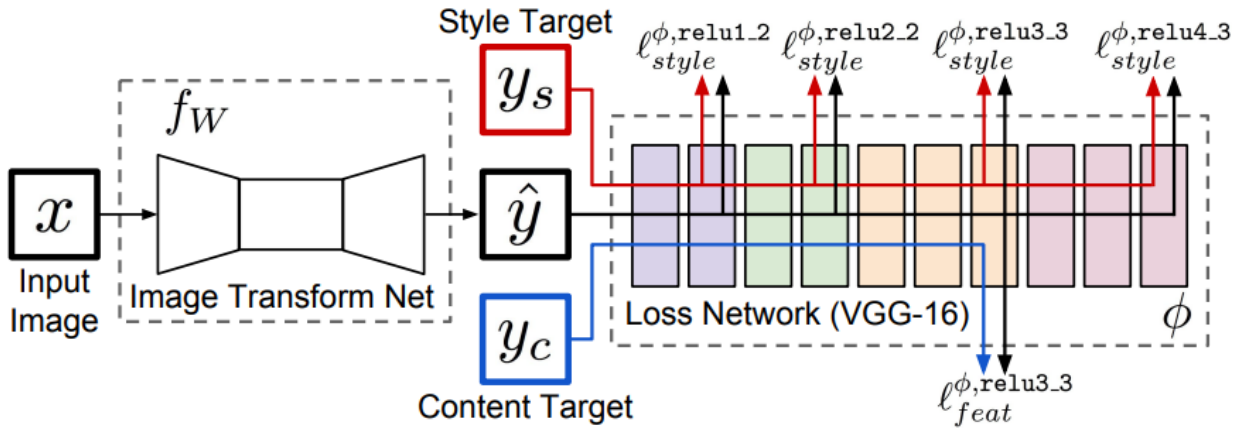
- **Feature loss**

$$\ell_{feat}^{\phi,j}(\hat{y},y) = \frac{1}{C_j H_j W_j}\|\phi_j(\hat{y}) - \phi_j(y)\|_2^2$$

- **Style reconstruction loss**

$$G_j^{\phi}(x)_{c,c'} = \frac{1}{C_j H_j W_j}\sum_{h=1}^{H_j}\sum_{w=1}^{W_j}\phi_j(x)_{h,w,c}\phi_j(x)_{h,w,c'}$$

$$\ell_{style}^{\phi,j}(\hat{y},y) = \|G_j^{\phi}(\hat{y}) - G_j^{\phi}(y)\|_F^2$$



Gram matrix

$$\boldsymbol{G = X^T X} = \begin{pmatrix} \boldsymbol{x_1^T} \\ \vdots \\ \boldsymbol{x_m^T} \end{pmatrix} (\boldsymbol{x_1} \quad \cdots \quad \boldsymbol{x_m}), \qquad \boldsymbol{G_{ij} = x_i^T x_j}$$

I. Project I: Colorization

D. Perceptual Loss

- Feature loss

$$\ell_{feat}^{\phi,j}(\hat{y},y) = \frac{1}{C_j H_j W_j}\|\phi_j(\hat{y}) - \phi_j(y)\|_2^2$$

- Style reconstruction loss

$$G_j^{\phi}(x)_{c,c'} = \frac{1}{C_j H_j W_j}\sum_{h=1}^{H_j}\sum_{w=1}^{W_j}\phi_j(x)_{h,w,c}\phi_j(x)_{h,w,c'}$$

$$\ell_{style}^{\phi,j}(\hat{y},y) = \|G_j^{\phi}(\hat{y}) - G_j^{\phi}(y)\|_F^2$$

```
# calculate gram matrices for style feature layer maps we care about
style_features = vgg(style)
style_gram = [utils.gram(fmap) for fmap in style_features]
```

how to get gram

```
# calculate style loss
y_hat_gram = [utils.gram(fmap) for fmap in y_hat_features]
style_loss = 0.0
for j in range(4):
    style_loss += loss_mse(y_hat_gram[j], style_gram[j])
```

I. Project I: Colorization

D. Perceptual Loss

- **Feature loss**

$$\ell_{feat}^{\phi,j}(\hat{y},y) = \frac{1}{C_j H_j W_j}\|\phi_j(\hat{y}) - \phi_j(y)\|_2^2$$

- **Style reconstruction loss**

$$G_j^{\phi}(x)_{c,c'} = \frac{1}{C_j H_j W_j}\sum_{h=1}^{H_j}\sum_{w=1}^{W_j}\phi_j(x)_{h,w,c}\phi_j(x)_{h,w,c'}$$

$$\ell_{style}^{\phi,j}(\hat{y},y) = \|G_j^{\phi}(\hat{y}) - G_j^{\phi}(y)\|_F^2$$

`torch.bmm(input, mat2, *, out=None)` → Tensor

Performs a batch matrix-matrix product of matrices stored in `input` and `mat2`.

`input` and `mat2` must be 3-D tensors each containing the same number of matrices.

If `input` is a $(b \times n \times m)$ tensor, `mat2` is a $(b \times m \times p)$ tensor, `out` will be a $(b \times n \times p)$ tensor

$$\text{out}_i = \text{input}_i \mathbin{@} \text{mat2}_i$$

```
# Calculate Gram matrix (G = FF^T)
def gram(x):
    (bs, ch, h, w) = x.size()
    f = x.view(bs, ch, w*h)
    f_T = f.transpose(1, 2)
    G = f.bmm(f_T) / (ch * h * w)
```

I. Project I: Colorization

D. Perceptual Loss

- Feature loss

$$\ell_{feat}^{\phi,j}(\hat{y},y) = \frac{1}{C_j H_j W_j}\|\phi_j(\hat{y}) - \phi_j(y)\|_2^2$$

Much more popular because

style reconstruction loss is much heavier

- Style reconstruction loss

$$G_j^{\phi}(x)_{c,c'} = \frac{1}{C_j H_j W_j}\sum_{h=1}^{H_j}\sum_{w=1}^{W_j}\phi_j(x)_{h,w,c}\phi_j(x)_{h,w,c'}$$

$$\ell_{style}^{\phi,j}(\hat{y},y) = \|G_j^{\phi}(\hat{y}) - G_j^{\phi}(y)\|_F^2$$

I. Project I: Colorization

D. Tricky Loss: tv Loss

- total variation loss/regularizer
  [image gradient loss/regularizer]

Theory:

$$\mathcal{R}_{V^\beta}(f) = \int_\Omega \left(\left(\frac{\partial f}{\partial u}(u,v) \right)^2 + \left(\frac{\partial f}{\partial v}(u,v) \right)^2 \right)^{\frac{\beta}{2}} du\, dv$$

$$\mathcal{R}_{V^\beta}(\mathbf{x}) = \sum_{i,j} \left((x_{i,j+1} - x_{ij})^2 + (x_{i+1,j} - x_{ij})^2 \right)^{\frac{\beta}{2}}$$

$\beta = 1$

Implementation: practical way

```
tv_loss = torch.mean(torch.abs(pred[:, :, :-1, :] -
                               pred[:, :, 1:, :])) + \
          torch.mean(torch.abs(pred[:, :, :, :-1] -
                               pred[:, :, :, 1:]))
```

II. Project II: Face Frontalization

II. Project II: Face Frontalization

- **Paper: Two-Pathway GAN [2017] (TP-GAN)**

  Authors: Rui Huang etc.

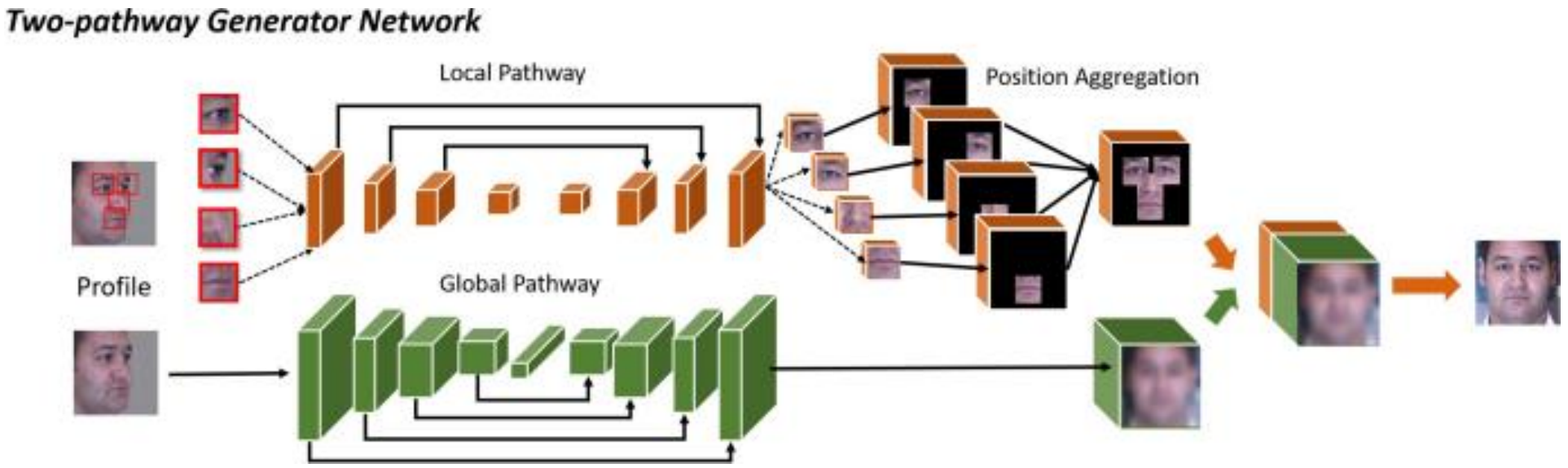
  Link: address

  Other resources: DR-GAN

I. Project II: Face Frontalization

E. Modified Backbone

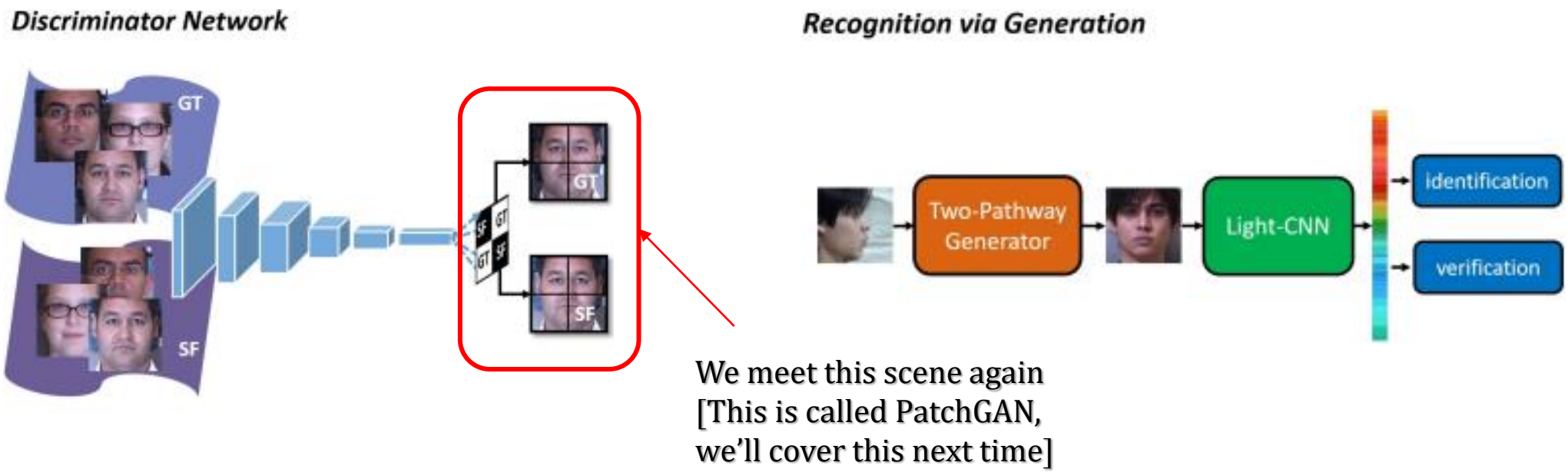
- **Several Parts**



I. Project II: Face Frontalization

E. Modified Backbone

- **Several Parts**



We meet this scene again
[This is called PatchGAN,
we'll cover this next time]

I. Project II: Face Frontalization

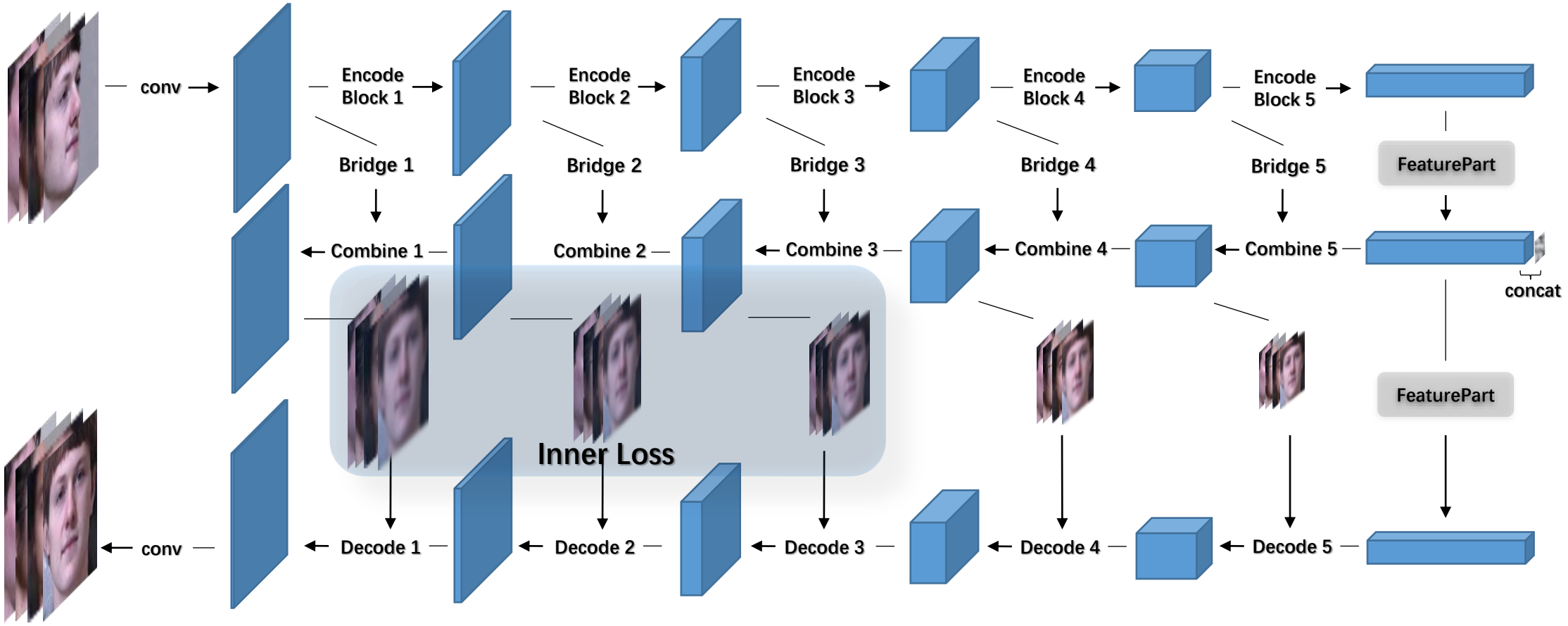
E. Modified Backbone

- **Key Parts: Backbone of Global Pathway: Updated from UNet**

I. Project II: Face Frontalization

E. Modified Backbone

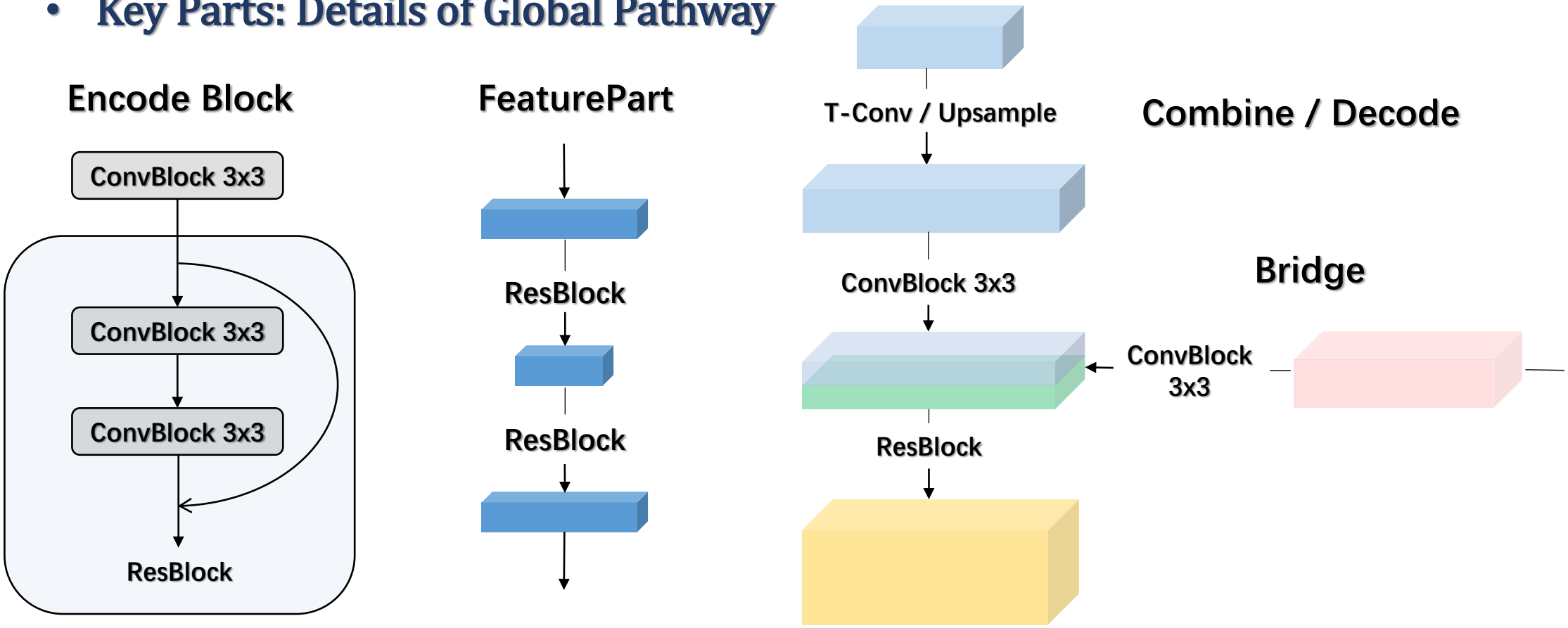
- **Key Parts: Backbone of Global Pathway: Updated from UNet**



I. Project II: Face Frontalization

E. Modified Backbone

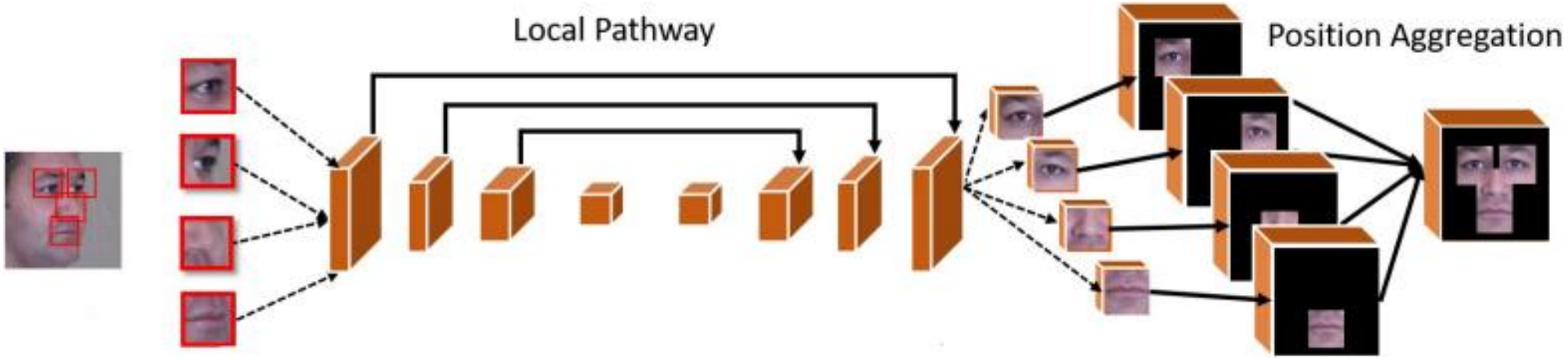
- **Key Parts: Details of Global Pathway**



I. Project II: Face Frontalization

E. Modified Backbone

- **Key Parts: Local Pathway**



keypoint detection

typical Small UNet

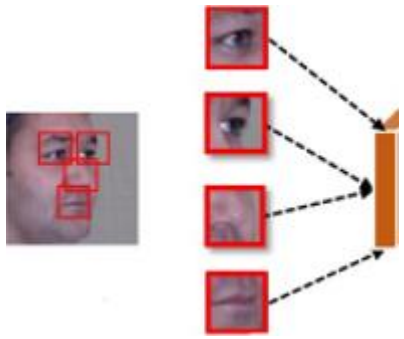
localization

combination

I. Project II: Face Frontalization

E. Modified Backbone

- **Key Parts: Local Pathway**



keypoint detection

[not our target in this class]

Method 1: Dlib (this is official website in C++)
[installation, install dlib before use]
[e.g., a good example to learn how to use in Python]

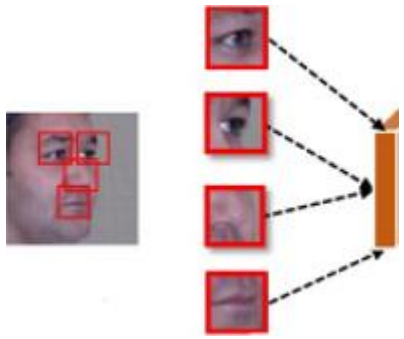
2: MTCNN (paper, code [old fashioned in TF])
[e.g., a good example we can directly use in PyTorch]

Details (original): [any ideas to upgrade?]
1. 5 landmarks: 2 eye-centers / 1 nose / 2 corners of the mouth
2. Crop a little patch around each detected landmark
   by default: img size: 128 / eye: 40 / nose: 40-32 / mouth: 48-32
3. Each path put into local pathway UNet

I. Project II: Face Frontalization

E. Modified Backbone

- **Key Parts: Local Pathway**



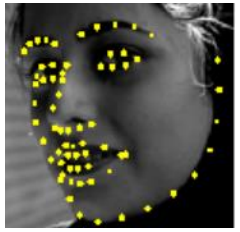
keypoint detection

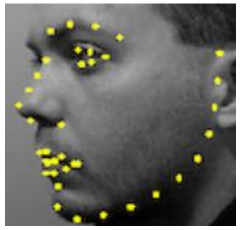
Details (original): [any ideas to upgrade?]

1. 5 landmarks: 2 eye-centers / 1 nose / 2 corners of the mouth
2. Crop a little patch around each detected landmark
   by default: img size: 128 / eye: 40 / nose: 40-32 / mouth: 48-32
3. Each path put into local pathway UNet

Details:

1. more landmarks
2. Crop an accurate patch around each detected landmark



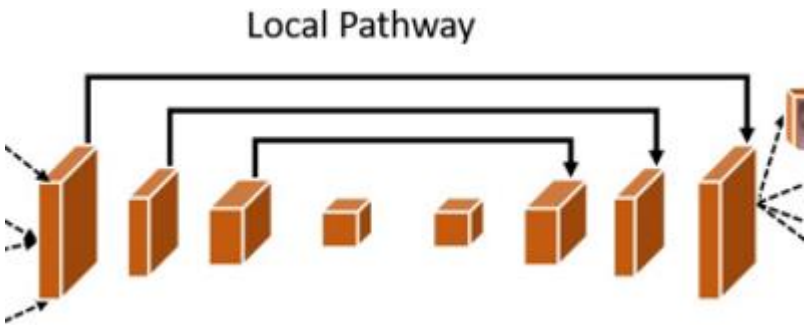


3. Each path put into local pathway UNet

I. Project II: Face Frontalization

E. Modified Backbone

- **Key Parts: Local Pathway**



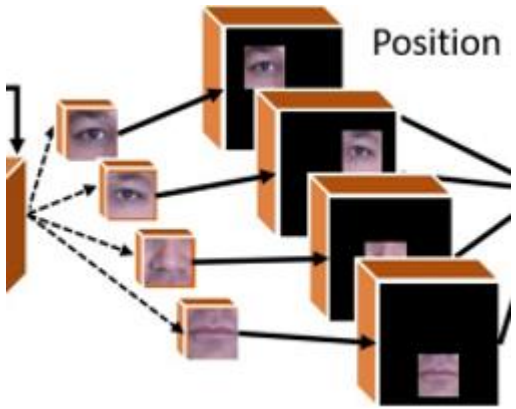
typical
Small UNet

Details:
For each patch, we instantiate a network
5 patches → 5 instantiated network

I. Project II: Face Frontalization

E. Modified Backbone

- **Key Parts: Local Pathway**



localization

Details

For each frontalized little patch, padding to greater tensors with same size

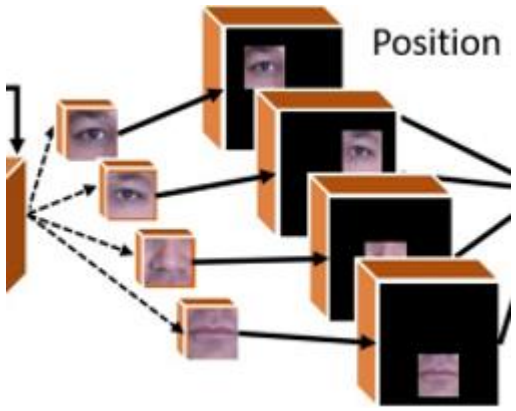
```
def forward(self, f_left_eye, f_right_eye, f_nose, f_mouth):
    f_left_eye = torch.nn.functional.pad(f_left_eye,                                    # input
                                         (leye_w - EYE_W // 2 - 1,                      # left
                                          input_img_size - (leye_w + EYE_W // 2 - 1),   # right
                                          leye_h - EYE_H // 2 - 1,                      # top
                                          input_img_size - (leye_h + EYE_H // 2 - 1)))  # bottom
    f_right_eye = torch.nn.functional.pad(f_right_eye,
                                          (reye_w - EYE_W // 2 - 1,
                                           input_img_size - (reye_w + EYE_W // 2 - 1),
                                           reye_h - EYE_H // 2 - 1,
```

How to localize from where to pad?

I. Project II: Face Frontalization

E. Modified Backbone

- **Key Parts: Local Pathway**



localization

Details

For each frontalized little patch, padding to greater tensors with same size

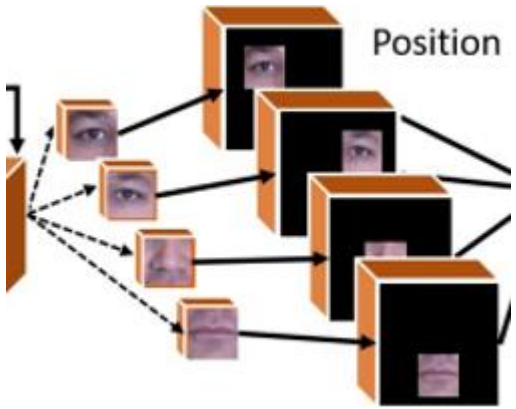
```
def forward(self, f_left_eye, f_right_eye, f_nose, f_mouth):
    f_left_eye = torch.nn.functional.pad(f_left_eye,                                    # input
                                         (leye_w - EYE_W // 2 - 1,                      # left
                                          input_img_size - (leye_w + EYE_W // 2 - 1),   # right
                                          leye_h - EYE_H // 2 - 1,                      # top
                                          input_img_size - (leye_h + EYE_H // 2 - 1)))  # bottom
    f_right_eye = torch.nn.functional.pad(f_right_eye,
                                          (reye_w - EYE_W // 2 - 1,
                                           input_img_size - (reye_w + EYE_W // 2 - 1),
                                           reye_h - EYE_H // 2 - 1,
```

How to localize from where to pad?

I. Project II: Face Frontalization

E. Modified Backbone

- **Key Parts: Local Pathway**



Details

For each frontalized little patch, padding to greater tensors with same size

```
def forward(self, f_left_eye, f_right_eye, f_nose, f_mouth):
    f_left_eye = torch.nn.functional.pad(f_left_eye,                                    # input
                                         (leye_w - EYE_W // 2 - 1,                      # left
                                          input_img_size - (leye_w + EYE_W // 2 - 1),   # right
                                          leye_h - EYE_H // 2 - 1,                      # top
                                          input_img_size - (leye_h + EYE_H // 2 - 1)))  # bottom
    f_right_eye = torch.nn.functional.pad(f_right_eye,
                                          (reye_w - EYE_W // 2 - 1,
                                           input_img_size - (reye_w + EYE_W // 2 - 1),
                                           reye_h - EYE_H // 2 - 1,
```

localization

```
_parser.add_argument("--input_img_size", default=128, type=int)
_parser.add_argument("--EYE_W", default=40, type=int)
_parser.add_argument("--EYE_H", default=40, type=int)
_parser.add_argument("--NOSE_W", default=40, type=int)
_parser.add_argument("--NOSE_H", default=32, type=int)
_parser.add_argument("--reye_w", default=44, type=int)
_parser.add_argument("--reye_h", default=41, type=int)
_parser.add_argument("--leye_w", default=86, type=int)
_parser.add_argument("--leye_h", default=41, type=int)
```

I. Project II: Face Frontalization

E. Modified Backbone

- **Key Parts: Local Pathway**

combination

Details

Combine/Merge padded frontalized patch/tensor as a whole

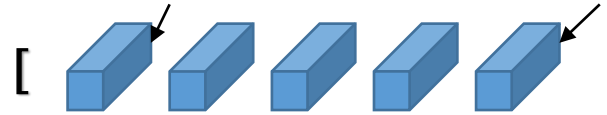
```
return torch.max(
            torch.stack([f_left_eye, f_right_eye, f_nose, f_mouth], dim=0),
            dim=0
        )[0]
```

1. torch.stack([a, b, c, d], dim=0), 在第dim维入栈

So if `A` and `B` are of shape (3, 4):

- `torch.cat([A, B], dim=0)` will be of shape (6, 4)
- `torch.stack([A, B], dim=0)` will be of shape (2, 3, 4)

Left eye: [1, 64, 128, 128] Left mouth: [1, 64, 128, 128]



Stacked tensor: [5, 1, 64, 128, 128]

I. Project II: Face Frontalization

E. Modified Backbone

- **Key Parts: Local Pathway**

combination

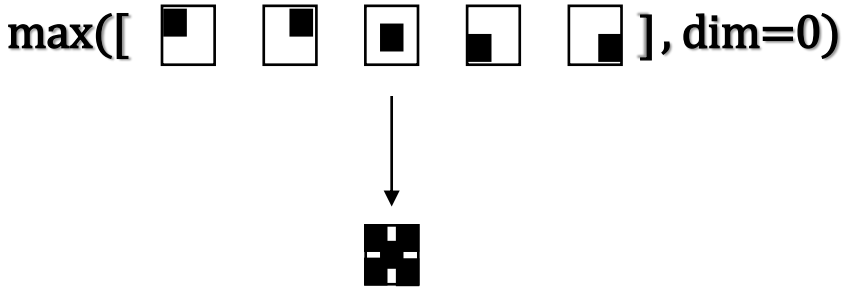
Details

Combine/Merge padded frontalized patch/tensor as a whole

```
return torch.max(
            torch.stack([f_left_eye, f_right_eye, f_nose, f_mouth], dim=0),
            dim=0
        )[0]
```

2. torch. max(tensor, dim=0),沿第dim找最大值并存出来

```
>>> a = torch.randn(2, 3, 4)
>>> a
tensor([[[-1.3002,  0.0548,  1.0300,  0.6392],
         [-0.1983,  0.1366, -0.0305, -0.0889],
         [ 2.6973,  1.4979, -0.2619, -0.3422]],

        [[-0.9631,  0.2432, -0.0634,  0.2243],
         [-0.6535,  0.5095,  1.6183,  0.3808],
         [-1.0251,  2.0204,  2.4084,  1.6377]]])
>>> torch.max(a, 0)
torch.return_types.max(
values=tensor([[-0.9631,  0.2432,  1.0300,  0.6392],
        [-0.1983,  0.5095,  1.6183,  0.3808],
        [ 2.6973,  2.0204,  2.4084,  1.6377]]),
```



I. Project II: Face Frontalization

E. Modified Backbone

- **Key Parts: Local Pathway**

combination

Details

Combine/Merge padded frontalized patch/tensor as a whole

```
return torch.max(
            torch.stack([f_left_eye, f_right_eye, f_nose, f_mouth], dim=0),
            dim=0
        )[0]
```

3. torch. max(tensor, dim=0)[0], torch.max返回2个值，最大值[0] 和 对应位置[1]

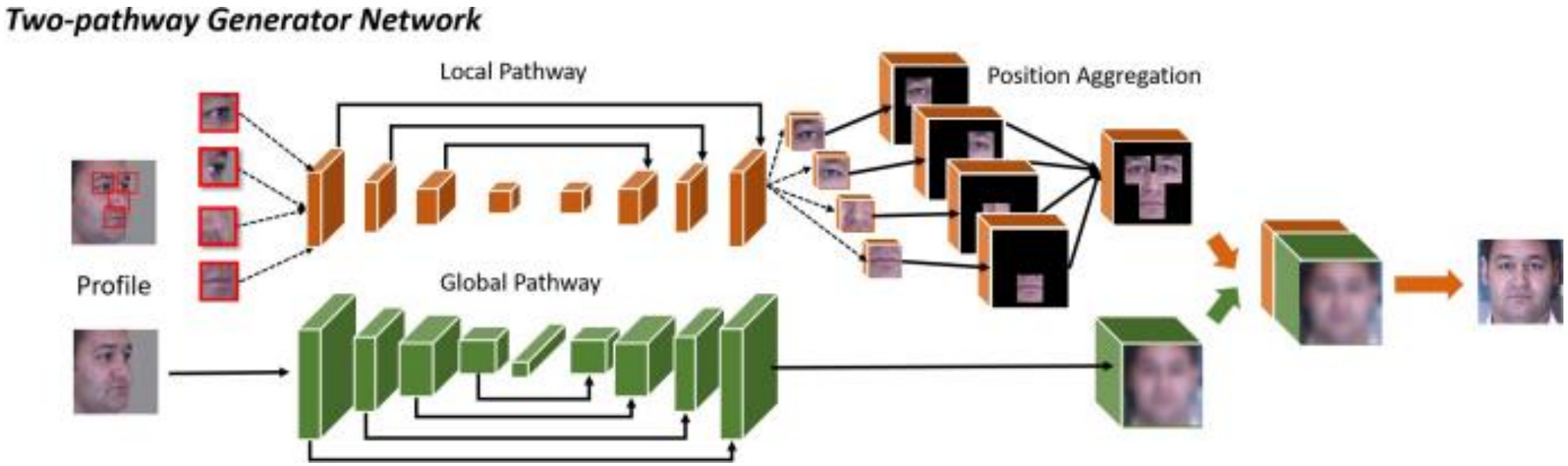
```
>>> a = torch.randn(2, 3, 4)
>>> a
tensor([[[-1.3002,  0.0548,  1.0300,  0.6392],
         [-0.1983,  0.1366, -0.0305, -0.0889],
         [ 2.6973,  1.4979, -0.2619, -0.3422]],

        [[-0.9631,  0.2432, -0.0634,  0.2243],
         [-0.6535,  0.5095,  1.6183,  0.3808],
         [-1.0251,  2.0204,  2.4084,  1.6377]]])
>>> torch.max(a, 0)
torch.return_types.max(
values=tensor([[-0.9631,  0.2432,  1.0300,  0.6392],
        [-0.1983,  0.5095,  1.6183,  0.3808],
        [ 2.6973,  2.0204,  2.4084,  1.6377]]),
```

```
>>> torch.max(a, 0)
torch.return_types.max(
values=tensor([[-0.9631,  0.2432,  1.0300,  0.6392],
        [-0.1983,  0.5095,  1.6183,  0.3808],
        [ 2.6973,  2.0204,  2.4084,  1.6377]]),
indices=tensor([[1, 1, 0, 0],
        [0, 1, 1, 1],
        [0, 1, 1, 1]]))
```

I. Project II: Face Frontalization

E. Modified Backbone

- **Key Parts: Global Pathway: UNet Upgrade + Combination with Local**



I. Project II: Face Frontalization

F. Tricky Losses: Symmetrical Loss + Extra Loss

- **Symmetrical Loss**

Definition:
Common faces are symmetrical at most time.
Why not use this feature as a training target

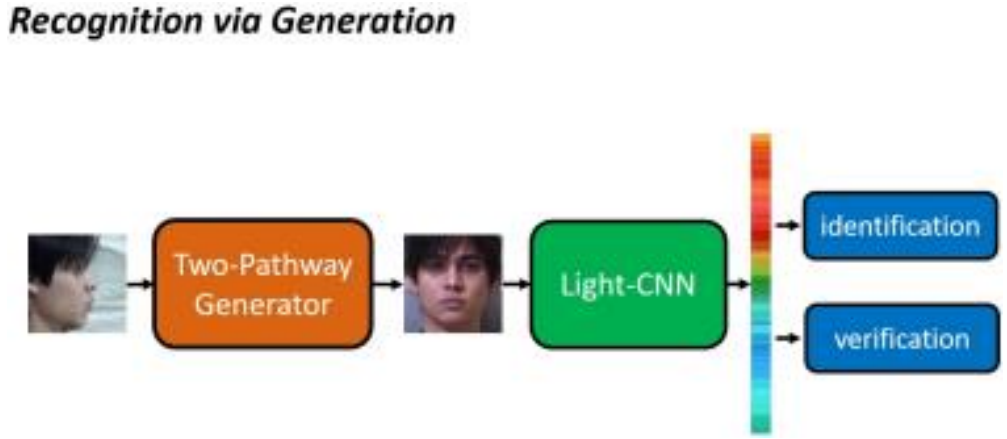
How to use:

```
class SymLoss(nn.modules.loss._Loss):

    def __init__(self):
        super(SymLoss, self).__init__()
        self.l1 = nn.L1Loss(reduction="sum")

    def forward(self, x):
        half_w = x.size(-1) // 2
        return self.l1(x[..., :half_w], x[..., half_w:].flip(-1))
```

I. Project II: Face Frontalization

F. Tricky Losses: Symmetrical Loss + Extra Loss

- **Extra Loss: Recognition Loss**



Reason:
We hope the generated frontalized faces are faces of the same people

How to use:
1. Push generated frontalized faces into face recognition network to get corresponding features
2. Then compare the features as Perceptual Loss

I. Project II: Face Frontalization

F. Tricky Losses

- **Summary**

  You can use as many losses as possible.

  The final loss is the combination of different partial losses which are your learning targets

  You need to find out weights corresponding to each loss in order their values are balanced

III. Project I: Revisit

III. Project I: Revisit

G. Temporal Consistency

III. Project I: Revisit

G. Temporal Consistency

III. Project I: Revisit

G. Temporal Consistency

- **How to constrain the temporal consistency**

Optical Flow: Definition

the pattern of apparent motion of image objects between two consecutive frames caused by the movement of object or camera.

2D vector field where each vector is a displacement vector showing the movement of points from first frame to second

III. Project I: Revisit

G. Temporal Consistency

- **How to constrain the temporal consistency**

Optical Flow: Assumption

1. The pixel intensities of an object do not change between consecutive frames.

III. Project I: Revisit

G. Temporal Consistency

- **How to constrain the temporal consistency**

Optical Flow: Calculation

$$I(x, y, t) = I(x + dx, y + dy, t + dt)$$

$$= I(x, y, t) + \frac{\partial I}{\partial x}dx + \frac{\partial I}{\partial y}dy + \frac{\partial I}{\partial t}dt + o^n$$

$$\Rightarrow$$

$$\frac{\partial I}{\partial x}\frac{dx}{dt} + \frac{\partial I}{\partial y}\frac{dy}{dt} + \frac{\partial I}{\partial t}\frac{dt}{dt} = \frac{\partial I}{\partial x}\frac{dx}{dt} + \frac{\partial I}{\partial y}\frac{dy}{dt} + \frac{\partial I}{\partial t} = 0$$

$$\Rightarrow$$

$I_x u + I_y v + I_t = 0$ where (u, v) is the optical vector and I_x, I_y and I_t can get from two consecutive frames

Difficulty:
1 equation with 2 parameters (u, v)

So, it means we still need at least another equation

III. Project I: Revisit

G. Temporal Consistency

- **How to constrain the temporal consistency**

Optical Flow: Assumption

1. The pixel intensities of an object do not change between consecutive frames.

2. Neighboring pixels have similar motion

(There are many methods to get other constraints to find out (u, v), here we only provide the most common method.)

You need to understand it before your interview

III. Project I: Revisit

G. Temporal Consistency

- **How to constrain the temporal consistency**

Optical Flow: Lucas-Kanade (LK) Algorithm [1981]

Critical idea:
Pixels inside a $N \times N$ window (neighboring pixels) have the same movement

$$\boldsymbol{A^T v = b}$$

$$\begin{aligned} I_x(q_1)u + I_y(q_1)v &= -I_t(q_1) \\ I_x(q_2)u + I_y(q_2)v &= -I_t(q_2) \\ &\vdots \\ I_x(q_n)u + I_y(q_n)v &= -I_t(q_n) \end{aligned} \Rightarrow \quad \boldsymbol{A} = \begin{bmatrix} I_x(q_1) & I_x(q_2) & \dots & I_x(q_n) \\ I_y(q_1) & I_y(q_2) & & I_y(q_n) \end{bmatrix}$$

$$\boldsymbol{v} = \begin{bmatrix} u \\ v \end{bmatrix}$$

$$\boldsymbol{b} = [-I_t(q_1) \quad -I_t(q_1) \quad \cdots \quad -I_t(q_1)]^{\boldsymbol{T}}$$

III. Project I: Revisit

G. Temporal Consistency

- **How to constrain the temporal consistency**

Optical Flow: Lucas-Kanade (LK) Algorithm [1981] (Sparse Optical Flow)

$$\boldsymbol{v} = \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \sum_i I_x(q_i)^2 & \sum_i I_x(q_i) I_y(q_i) \\ \sum_i I_x(q_i) I_y(q_i) & \sum_i I_y(q_i)^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i I_x(q_i) I_t(q_i) \\ -\sum_i I_y(q_i) I_t(q_i) \end{bmatrix}$$

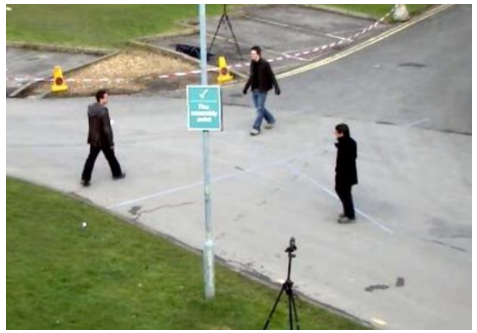
III. Project I: Revisit

G. Temporal Consistency

- **How to constrain the temporal consistency**

  Optical Flow: Some other materials

  Dense Optical Flow: Farneback Optical Flow (2003): (we can use this in light project)



  Optical Flow in Deep Learning: (we can use this in real project)
  Flownet 2.0: Evolution of Optical Flow Estimation with Deep Networks (Eddy Ilg, 2016)

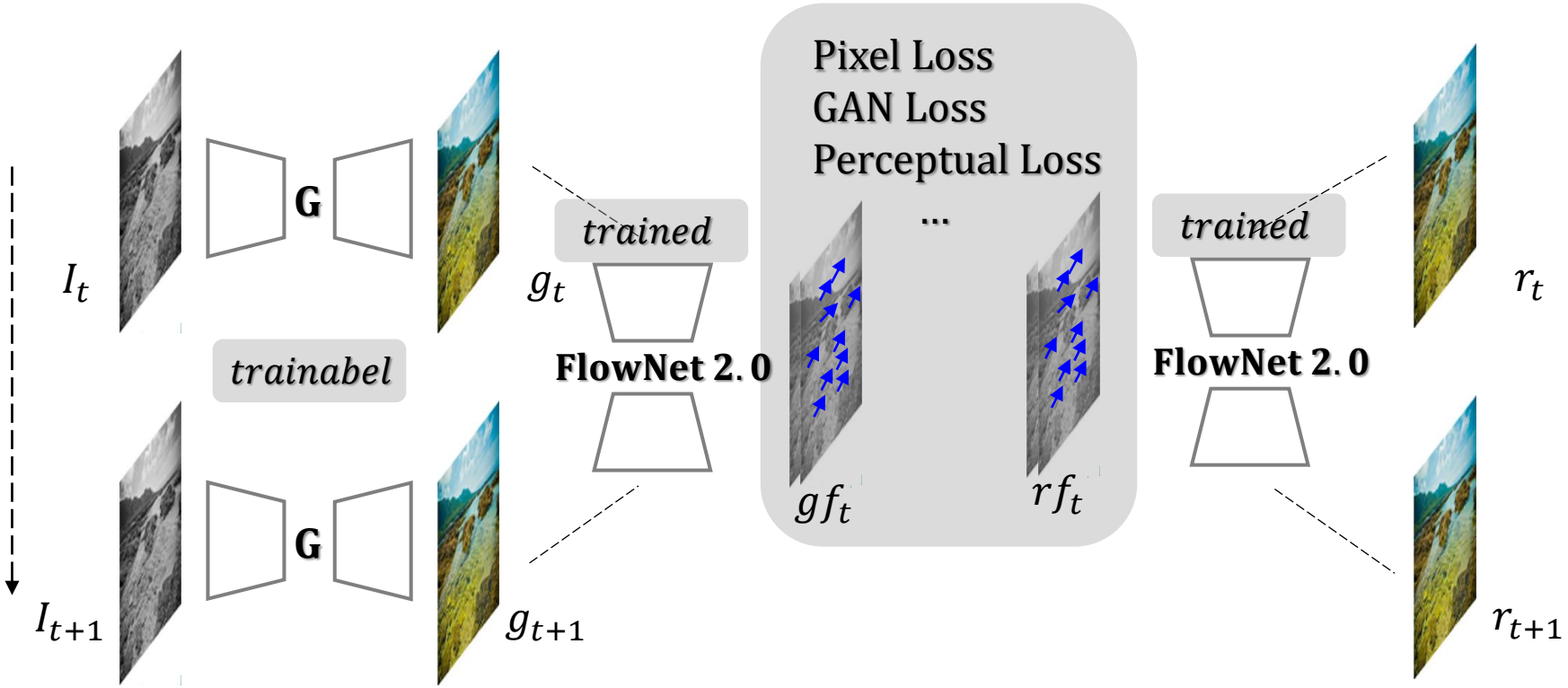
  Related papers: link

III. Project I: Revisit

G. Temporal Consistency

- **How to integrate optical flow into colorizing video**

method 1: directly add optical flow constraint



III. Project I: Revisit

G. Temporal Consistency

- **How to integrate optical flow into colorizing video**

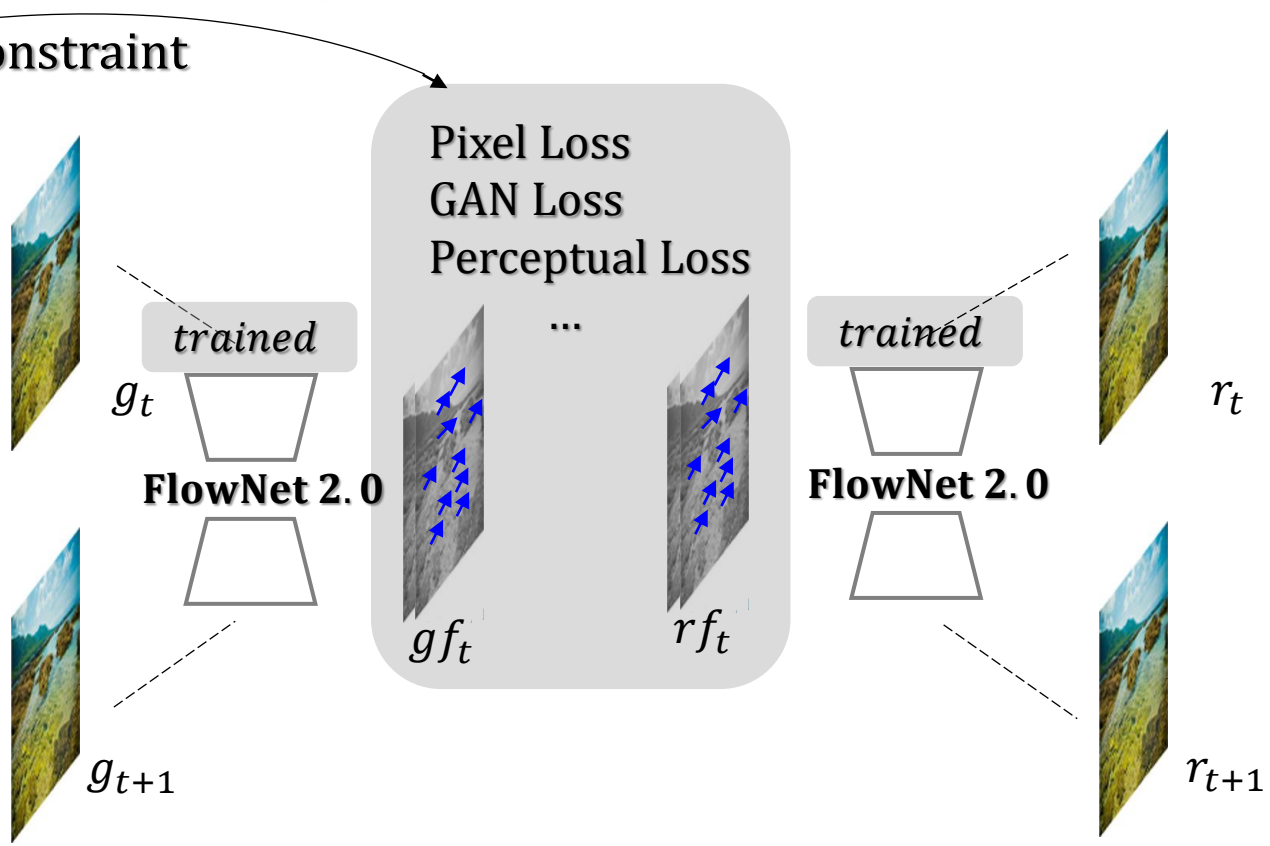
method 1: directly add optical flow constraint

Generalized tricks:

1. Optical flow result has 2 channels: [displacement_x, displacement_y]
2. For flow_loss, we can make it more robust by warping the output like:

$$\mathcal{L}_{\text{gf}} = \text{losses\_we\_have\_known} + \| \mathbf{gf}_t(\mathbf{r}_t) - \mathbf{r}_{t+1} \|$$

An example to show how to implement (from line 108-115)



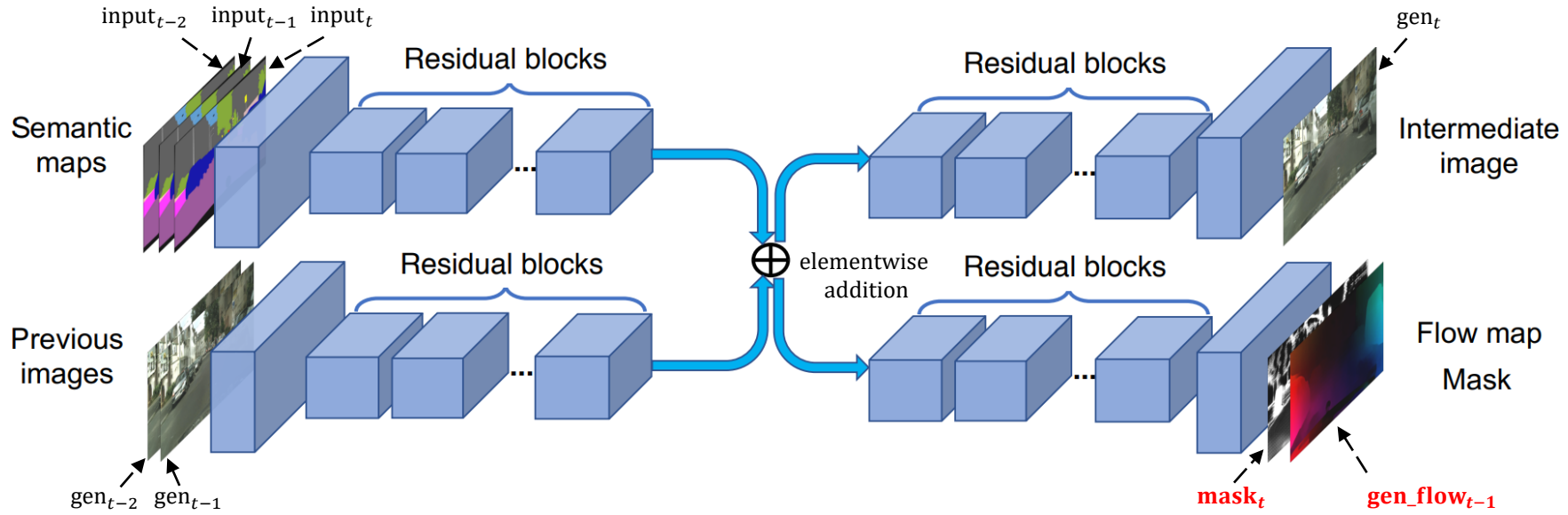
III. Project I: Revisit

G. Temporal Consistency

- **How to integrate optical flow into colorizing video**

method 2: use output as input

[vid2vid: Video-to-Video Synthesis (2018, Wang)]



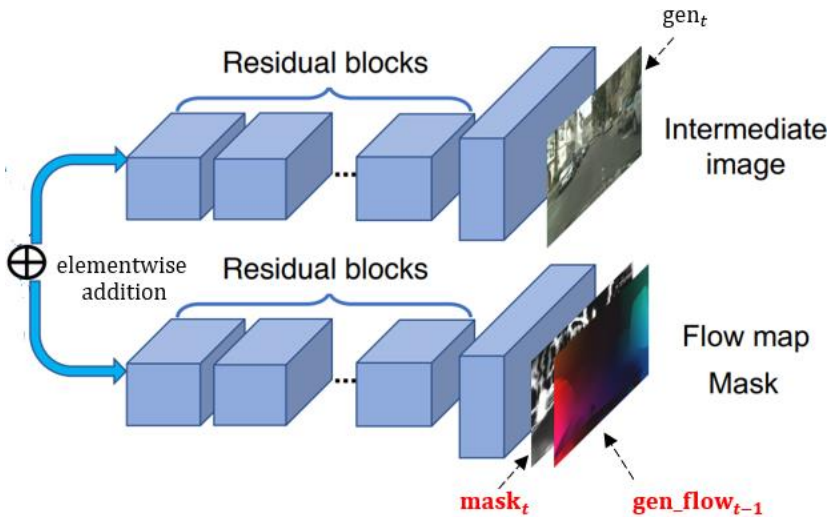
III. Project I: Revisit

G. Temporal Consistency

- **How to integrate optical flow into colorizing video**

method 2: use output as input

[vid2vid: Video-to-Video Synthesis (2018, Wang)]



Final Generate Image

$$= (1 - mask_t) \odot gen_{flow_{t-1}}(gen_{t-1}) + mask_t \odot gen_{t-1}$$

III. Project I: Revisit

G. Temporal Consistency

- **How to integrate optical flow into colorizing video**

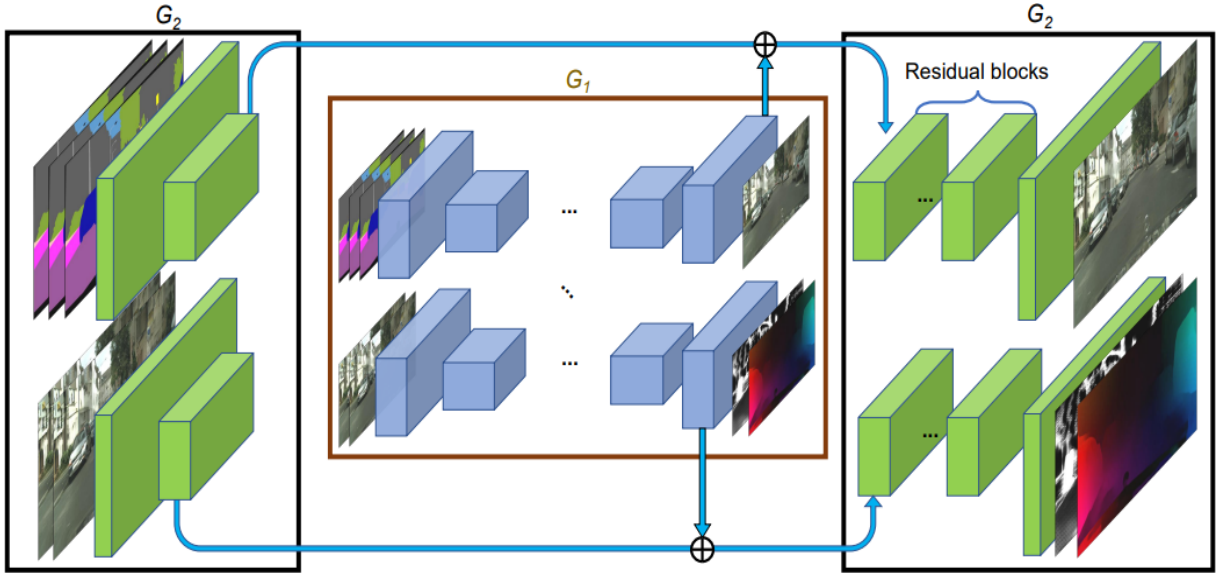
method 2: use output as input

[vid2vid: Video-to-Video Synthesis (2018, Wang)]

Generalized tricks:

1. Multiscale Generator
2. Multiscale Discriminator
3. Inner Loss of Discriminator

[Just like SPADE]



III. Project I: Revisit

G. Temporal Consistency

- **How to integrate optical flow into colorizing video**

  method 3: other related methods: link

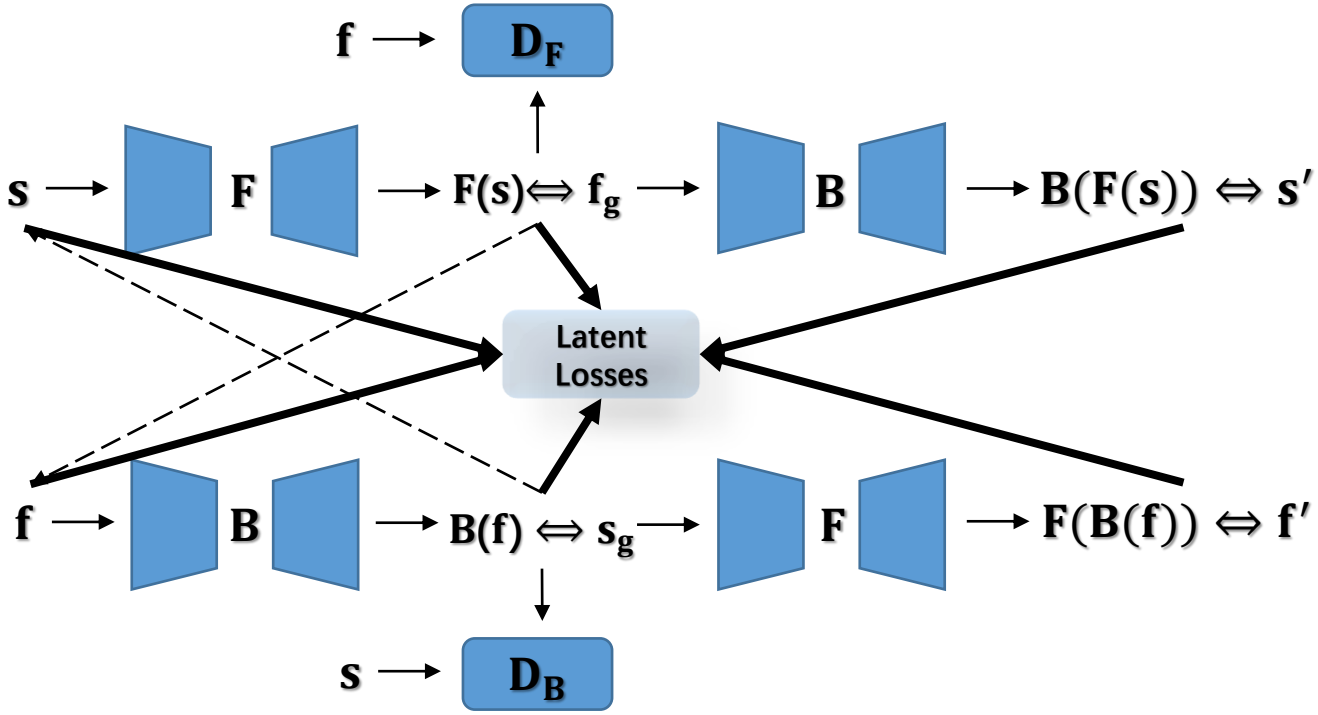
IV. Project II: Revisit

H. Pair with Unpair

- **Structure: Outline**

Input

$$\begin{pmatrix} s^1_{unpair} & f^1_{unpair} \\ s^2_{unpair} & f^2_{unpair} \\ s^1_{pair} & f^1_{pair} \\ s^2_{pair} & f^2_{pair} \end{pmatrix}$$



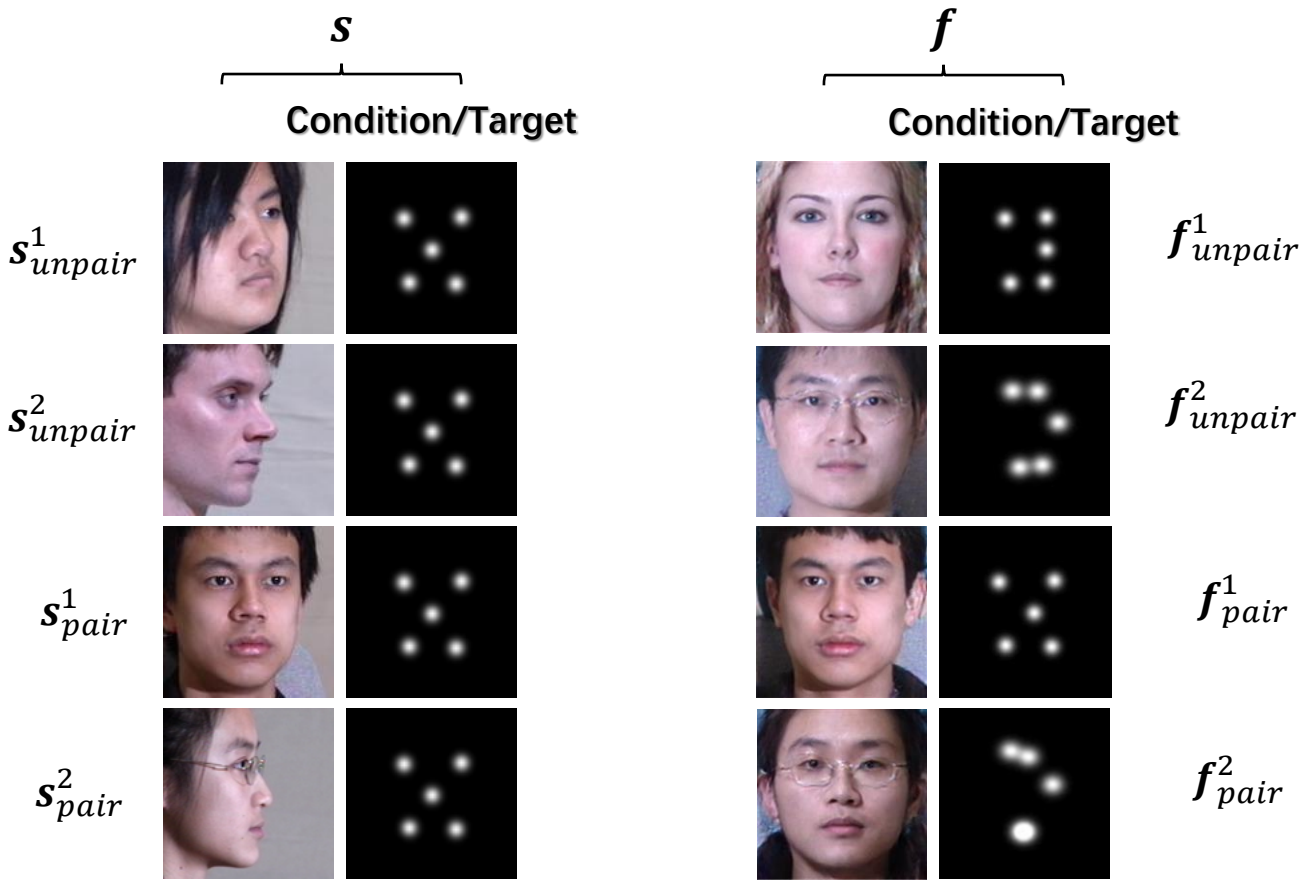
IV. Project II: Revisit

H. Pair with Unpair

- **Details: Input**

Input

$$\begin{pmatrix} \boldsymbol{s}^1_{unpair} & \boldsymbol{f}^1_{unpair} \\ \boldsymbol{s}^2_{unpair} & \boldsymbol{f}^2_{unpair} \\ \boldsymbol{s}^1_{pair} & \boldsymbol{f}^1_{pair} \\ \boldsymbol{s}^2_{pair} & \boldsymbol{f}^2_{pair} \end{pmatrix}$$



IV. Project II: Revisit

H. Pair with Unpair

- **Losses**

1. GAN Losses

For **F**: $s \rightarrow f$

$$\min_F \max_{D_F} \mathcal{L}_{GAN}(F, D_F, s, f)$$

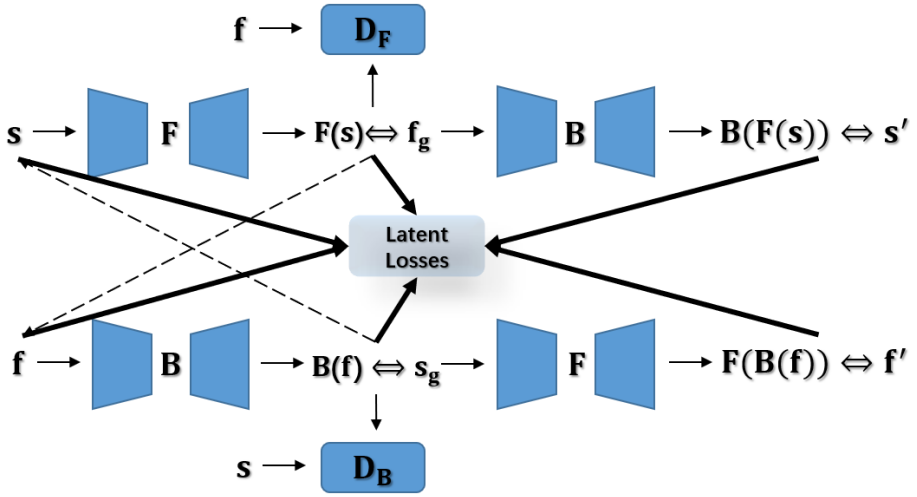
For **B**: $f \rightarrow s$

$$\min_B \max_{D_B} \mathcal{L}_{GAN}(B, D_B, f, s)$$

$$\mathcal{L}_{GAN}(F, D_F, s, f) = \mathbb{E}_{f \sim p_{data}(f)}[\log D_F(f)] + \mathbb{E}_{s \sim p_{data}(s)}[\log(1 - D_F(F(s)))]$$

$$\mathcal{L}_{GAN}(B, D_B, f, s) = \mathbb{E}_{s \sim p_{data}(s)}[\log D_B(s)] + \mathbb{E}_{f \sim p_{data}(f)}[\log(1 - D_B(B(f)))]$$

Implementation was completed by WGAN-GP



2. Cycle Loss

$$\mathcal{L}_{cyc}(F, B) = \mathbb{E}_{s \sim p_{data}(s)}[\| s' - s \|_1] + \mathbb{E}_{f \sim p_{data}(f)}[\| f' - f \|_1]$$

3. Identity Loss

4. tv (total variation) Loss

$$\mathcal{L}_{tv}(F, B) = {f_g}^2_{grad} + {f'}^2_{grad} + {s_g}^2_{grad} + {s'}^2_{grad}$$

IV. Project II: Revisit

H. Pair with Unpair

- **Losses**

5. Latent Losses

For **pair data**:

adv loss pixel loss perceptual loss

$$\text{inner loss} \sum_{l} \Big(\begin{matrix} \mathbf{f} \,/\, \mathbf{f_g} \longrightarrow \\ \mathbf{s} \longrightarrow \end{matrix} \boxed{\mathbf{D_{fs}}} + \begin{matrix} \mathbf{s} \,/\, \mathbf{s_g} \longrightarrow \\ \mathbf{f} \longrightarrow \end{matrix} \boxed{\mathbf{D_{sf}}} + \| f_g - f \|_1 + \| s_g - s \|_1 \Big) + \begin{matrix} \mathbf{f} \longrightarrow \\ \mathbf{f_g} \longrightarrow \end{matrix} \boxed{\textbf{Light CNN}}$$

$+$

For **unpair data**:

$$\text{inner loss} \sum_{l} \Big(\begin{matrix} \mathbf{s} \,/\, \mathbf{s'} \longrightarrow \\ \mathbf{f_g} \longrightarrow \end{matrix} \boxed{\mathbf{D_{sf}}} + \begin{matrix} \mathbf{f} \,/\, \mathbf{f'} \longrightarrow \\ \mathbf{s_g} \longrightarrow \end{matrix} \boxed{\mathbf{D_{fs}}} \Big)$$