Video Anomaly Detection
(Coding practice)

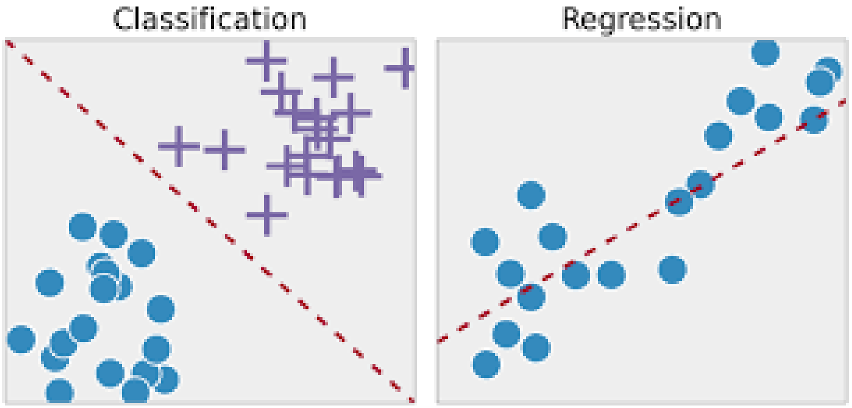
Application In Database Systems
(CSI8782.01-01)

Data Engineering Lab
Multi Modal Deep Learning Team

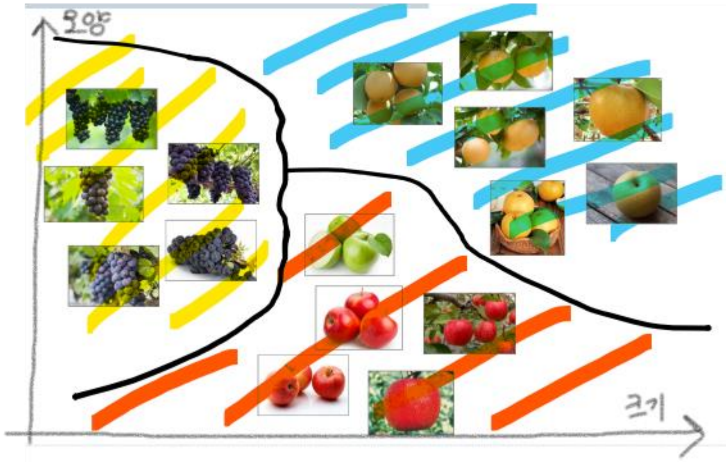
연세대학교

Supervised vs Unsupervised

- ## Supervised
  - ✓ Supervised learning is a machine learning approach that's defined by its use of **labelled datasets**.
  - ✓ That can be divided into classification and regression.
  - ✓ That **learns** from the training dataset by iteratively making predictions on the data and adjusting for the correct answer.



- ## Unsupervised
  - ✓ Unsupervised learning uses machine learning algorithms to analyze and cluster **unlabelled datasets**.
  - ✓ Unsupervised learning models are uses for three main tasks: clustering, association and dimensionality reduction.
  - ✓ That aims to **discover the inherent structure** of unlabeled data.

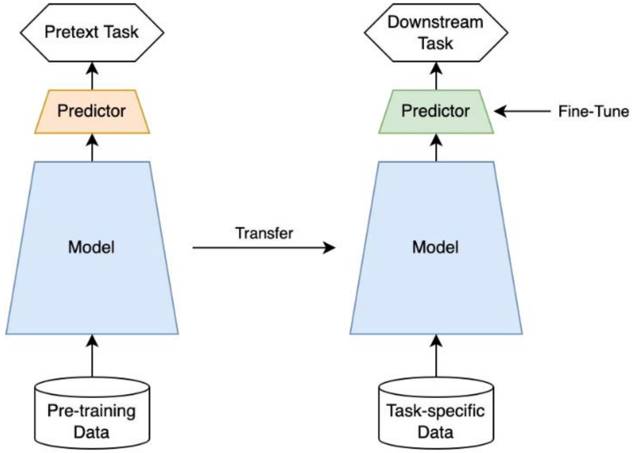


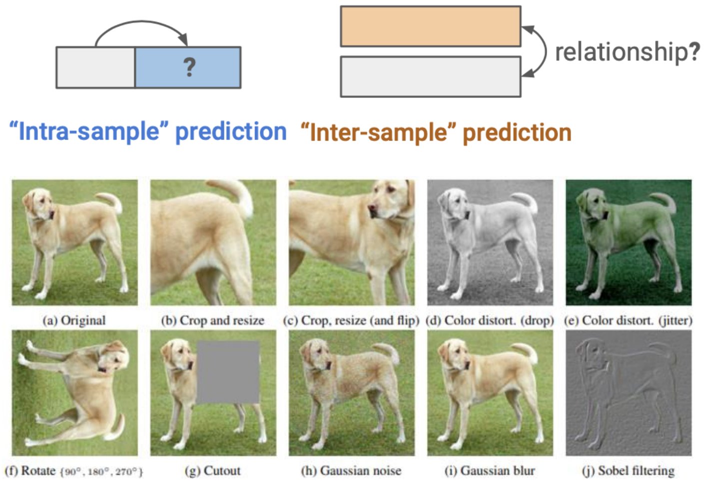
Supervised vs Self-Supervised vs Weakly Supervised

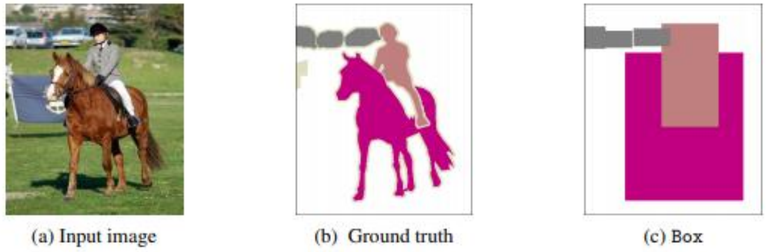
- **Supervised**
  - ✓ Supervised learning requires numerous high-quality labelled datasets, which can be costly to achieve accurate performance.

- **Self-Supervised**
  - ✓ Self-Supervised learning belongs to unsupervised learning.
  - ✓ Self-Supervised learning aims to obtain good representations from unlabelled datasets. Good representations are used to adapt downstream task.

- **Weakly Supervised**
  - ✓ Weakly supervised learning helps reduce the human involvement in training the models by using only **partially labelled datasets**, which typically requires less cost compared to supervised learning.

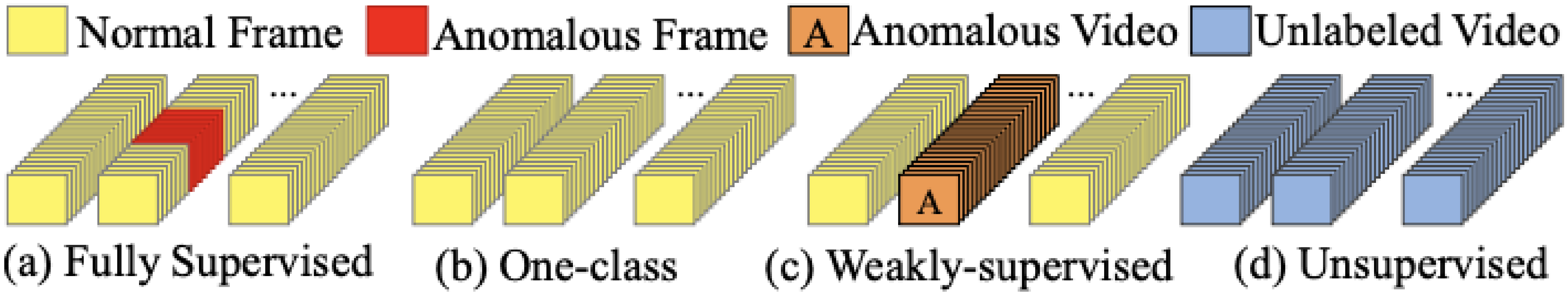






VAD Learning Method

- **Four Learning Method**
  - ✓ Fully Supervised: frame-level normal/abnormal annotations in the training data
  - ✓ One-class: only normal training data
  - ✓ Weakly-supervised: video-level normal/abnormal annotations
  - ✓ Unsupervised:no training data annotations



Zaheer, M. Zaigham, et al. "Generative cooperative learning for unsupervised video anomaly detection." Proceedings of the IEEE/CVF conference on c omputer vision and pattern recognition. 2022.

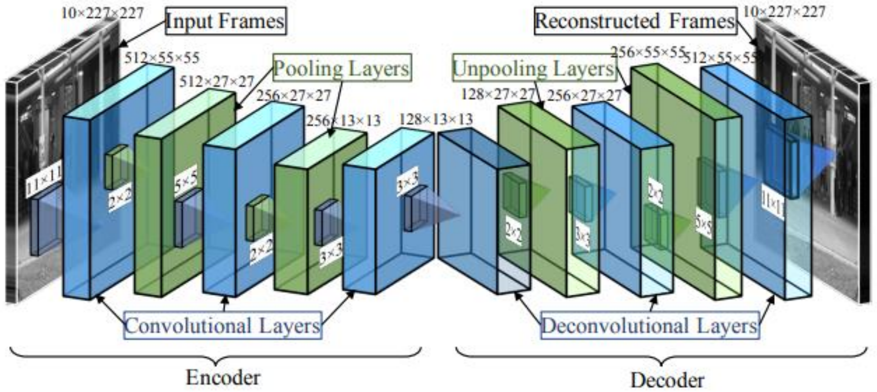
One-class classification

- ## Method

✓ A method of anomaly detection that trains only on normal data and categorizing anything dissimilar to the patterns in normal data as anomalies.

✓ In supervised learning, class imbalance may arise as the number of normal data is much higher than anomalies. Additionally, even with effective learning, overfitting can occur due to the unpredictable real-world anomaly data.

✓ Thus train with frame reconstruction or frame prediction methods using one-class classification and attempt to detect anomalies using similarities (e.g. PSNR) between reconstructed and real frames.



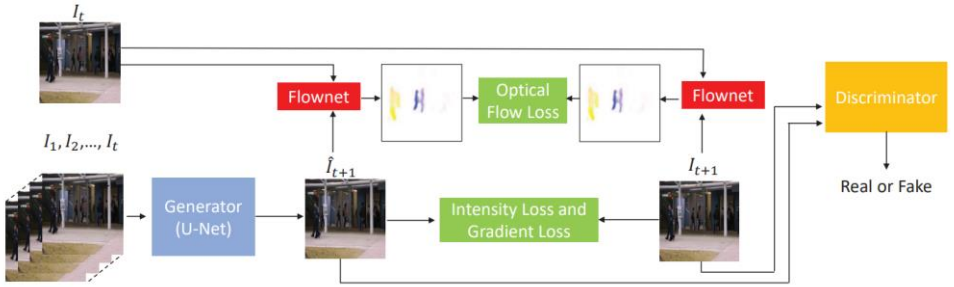


Figure 2. The pipeline of our video frame prediction network. Here we adopt U-Net as generator to predict next frame. To generate high quality image, we adopt the constraints in terms of appearance (intensity loss and gradient loss) and motion (optical flow loss). Here Flownet is a pretrained network used to calculate optical flow. We also leverage the adversarial training to discriminate whether the prediction is real or fake.

Future Frame Prediction for Anomaly Detection - A New Baseline(CVPR, 2018)

- **Concept**
  - ✓ Deep learning model derived from the assumption that <u>abnormal future frames cannot be predicted</u> if trained only on predicting future frames of normal data.

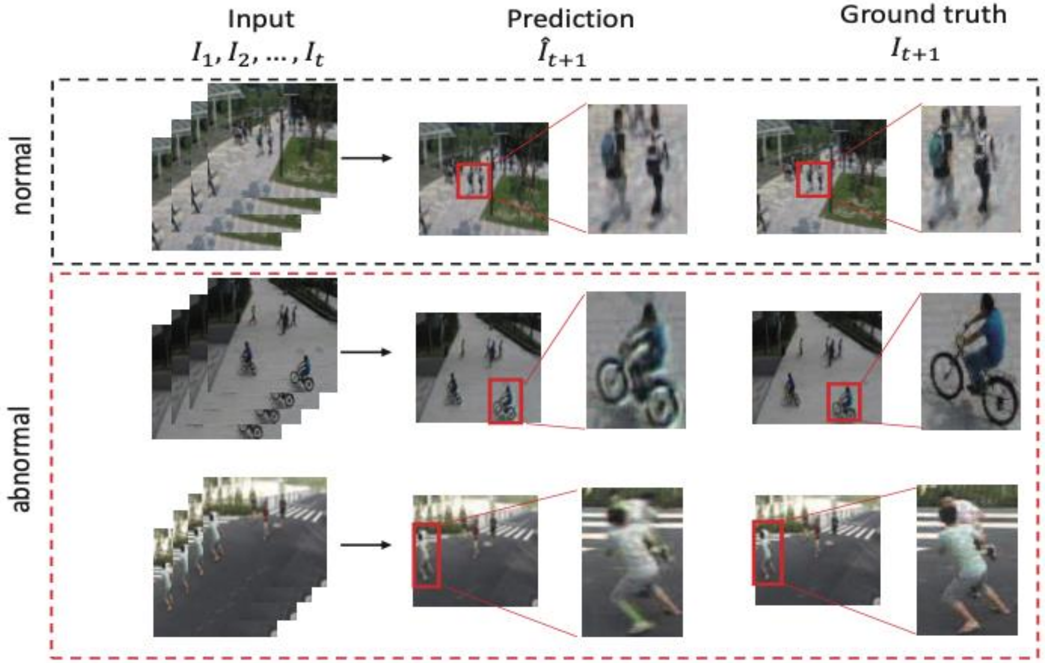


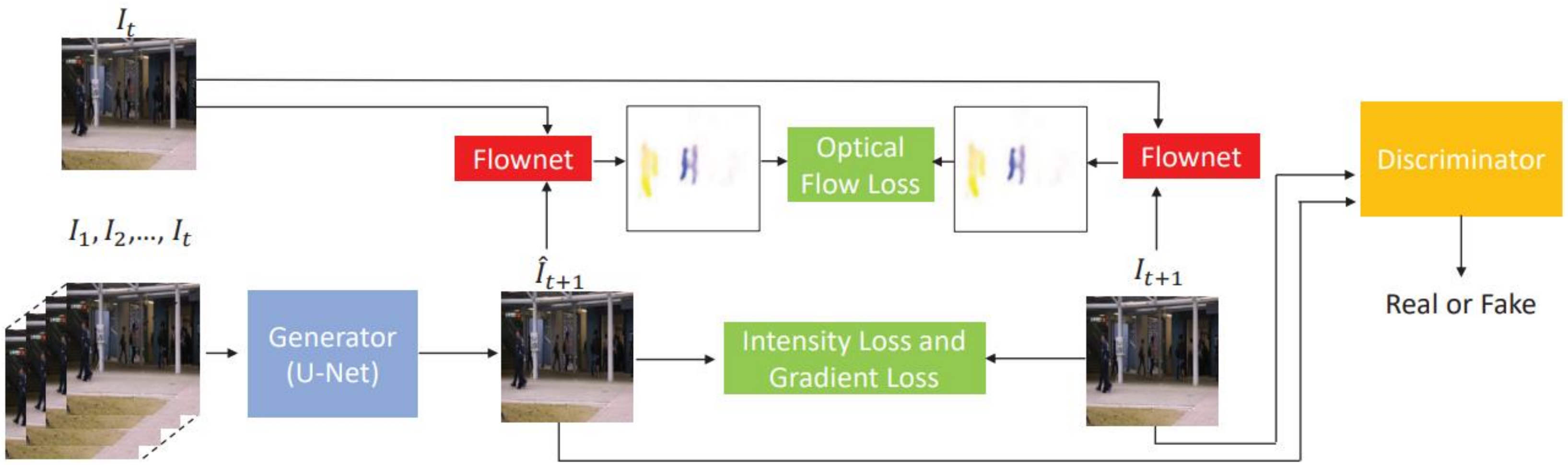
Figure 1. Some predicted frames and their ground truth in normal and abnormal events. Here the region is walking zone. When pedestrians are walking in the area, the frames can be well predicted. While for some abnormal events (a bicycle intrudes/ two men are fighting), the predictions are blurred and with color distortion. Best viewed in color.

Future Frame Prediction for Anomaly Detection

- **Training and Testing**

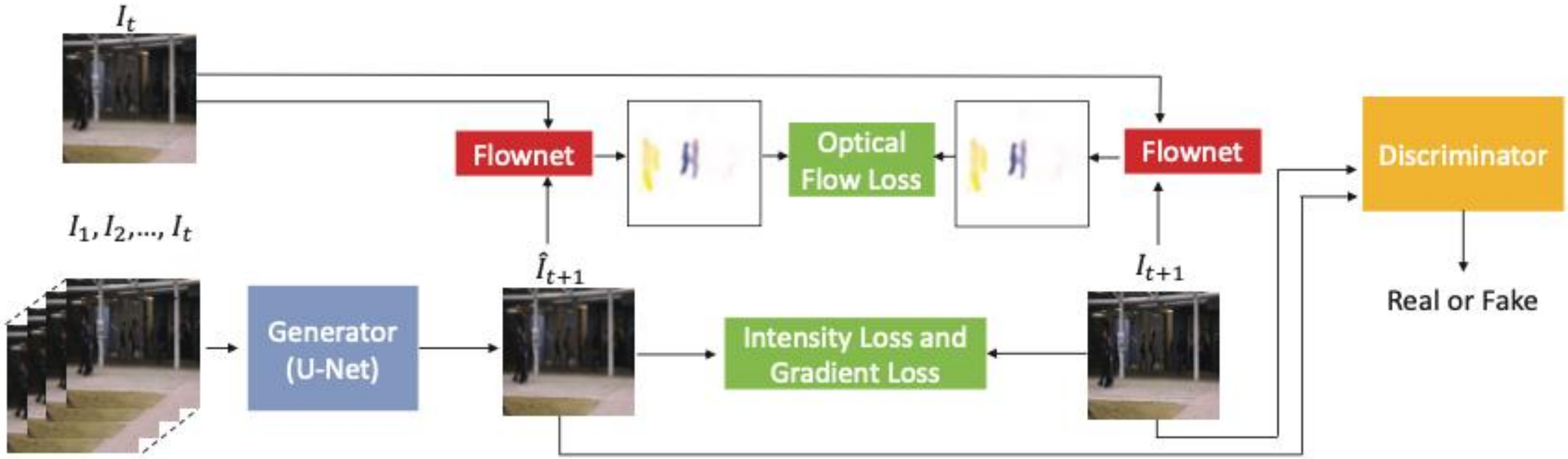
1. Using the frames from 1 to t as input to the generator, the model learns to generate the (t+1)-th frame through Intensity Loss and Gradient Loss.
2. Generated (t+1)-th frame is forwarded into the Discriminator to learn to deceive the Discriminator. *[to make image sharply]*
3. Calculate the optical flow between consecutive frames and train to minimize the difference between the real optical flow and the fake optical flow. *[to make two consecutive frames appear smoothly]*

✓ Compute the Anomaly Score through PSNR between frames generated by the generator and real frames.



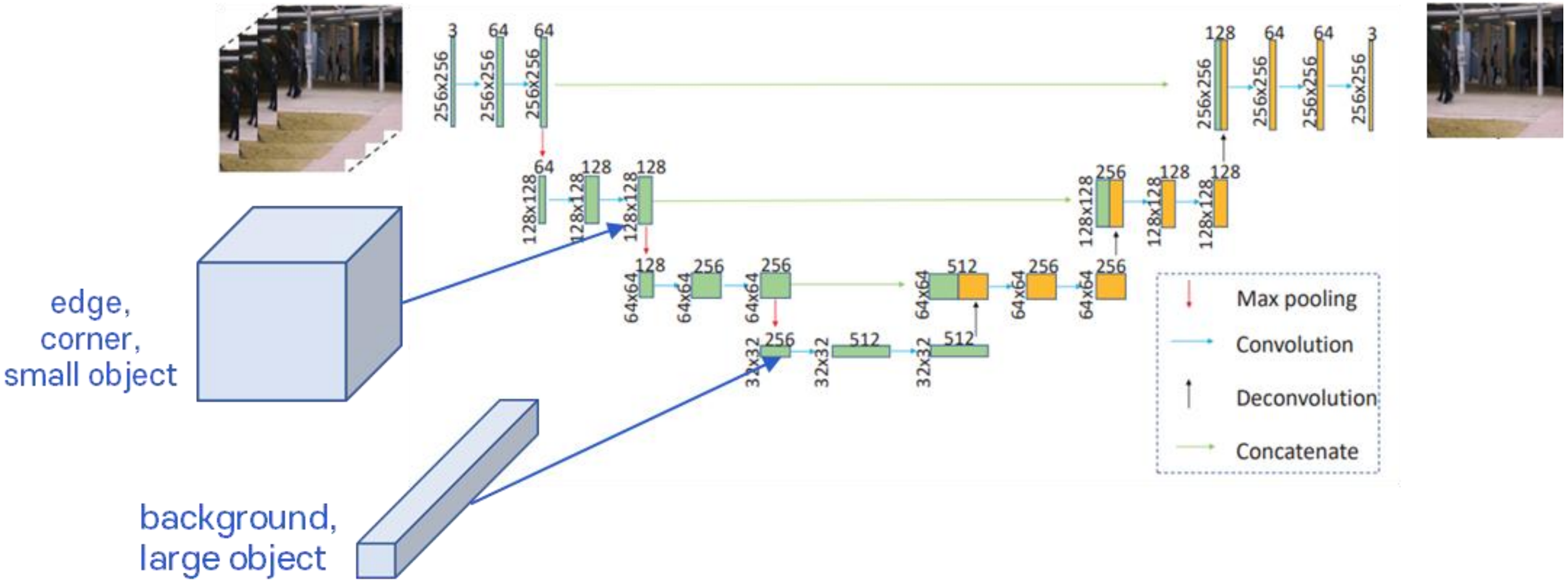
Future Frame Prediction for Anomaly Detection

- **Architecture**
  - ✓ FFP is consisted of Generator, Discriminator, FlowNet
  - ✓ Generator: A predictor that can well predict the future frame for normal training data
  - ✓ Discriminator: discriminate real or fake frame to generate more successful future frame
  - ✓ Flownet: pretrained network to estimate optical flow that means pixel by pixel motion



Future Frame Prediction for Anomaly Detection

- **Generator (U-Net)**
  - ✓ A convolutional neural network in the form of a U-shape, composed of an encoder and a decoder.
    - Encoder: Takes input frames from 1 to t to capture contextual information at various scales and transmits feature maps to the decoder through skip connections.
    - Decoder: Utilizes the received feature maps to generate the detailed (t+1)-th frame.



Future Frame Prediction for Anomaly Detection

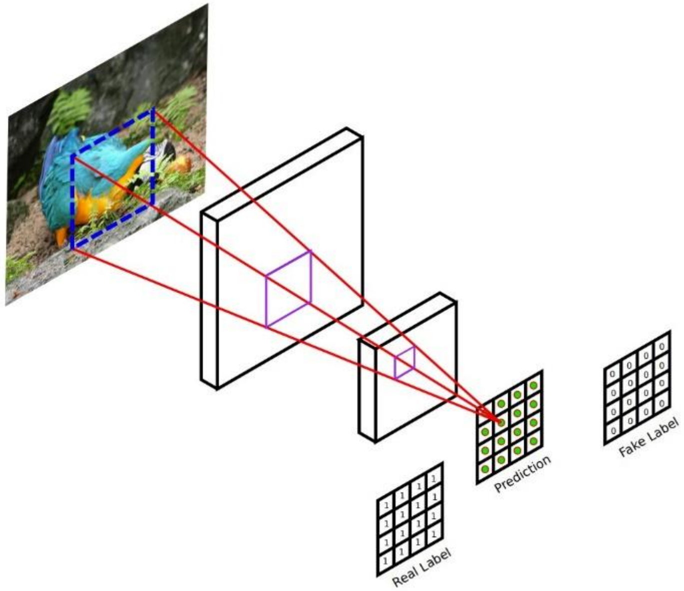
- ## Generator (U-Net)

```python
class UNet(nn.Module):
    def __init__(self, input_channels, output_channel=3):
        super(UNet, self).__init__()
        self.inc = inconv(input_channels, 64)
        self.down1 = down(64, 128)
        self.down2 = down(128, 256)
        self.down3 = down(256, 512)
        self.up1 = up(512, 256)
        self.up2 = up(256, 128)
        self.up3 = up(128, 64)
        self.outc = nn.Conv2d(64, output_channel, kernel_size=3, padding=1)

    def forward(self, x):
        x1 = self.inc(x)
        x2 = self.down1(x1)
        x3 = self.down2(x2)
        x4 = self.down3(x3)
        x = self.up1(x4, x3)
        x = self.up2(x, x2)
        x = self.up3(x, x1)
        x = self.outc(x)

        return torch.tanh(x)
```

```python
class inconv(nn.Module):
    def __init__(self, in_ch, out_ch):
        super().__init__()
        self.conv = double_conv(in_ch, out_ch)

    def forward(self, x):
        x = self.conv(x)
        return x

class down(nn.Module):
    def __init__(self, in_ch, out_ch):
        super().__init__()
        self.mpconv = nn.Sequential(nn.MaxPool2d(2),
                                    double_conv(in_ch, out_ch))

    def forward(self, x):
        x = self.mpconv(x)
        return x

class up(nn.Module):
    def __init__(self, in_ch, out_ch):
        super().__init__()
        self.up = nn.ConvTranspose2d(in_ch, in_ch // 2, 2, stride=2)
        self.conv = double_conv(in_ch, out_ch)

    def forward(self, x1, x2):
        x1 = self.up(x1)
        x = torch.cat([x2, x1], dim=1)
        x = self.conv(x)
        return x
```

Future Frame Prediction for Anomaly Detection

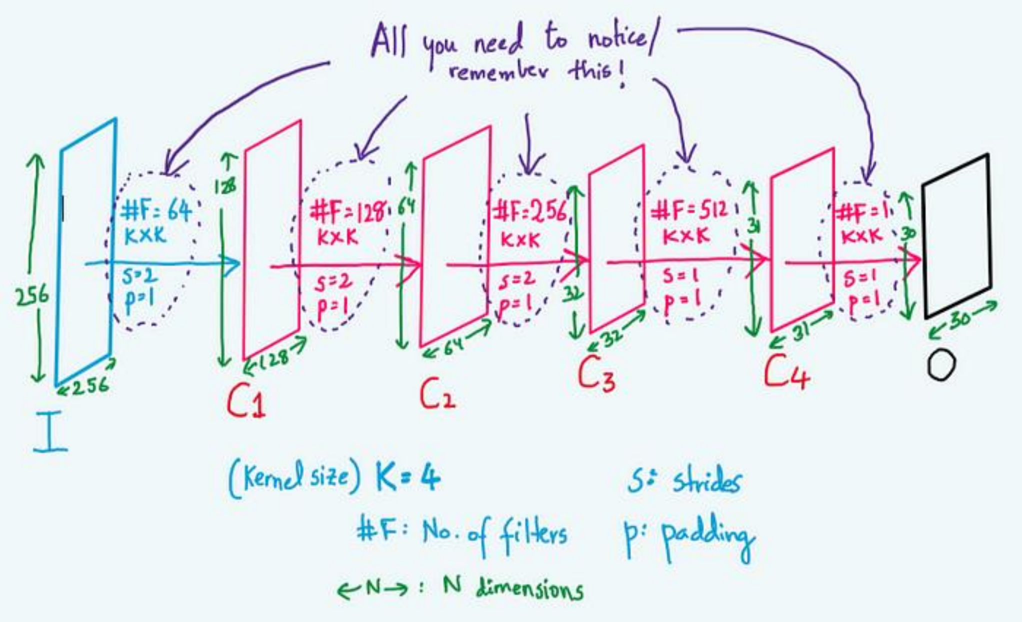
- **Discriminator (PatchGAN)**
  - ✓ The size of patch (receptive field) significantly impact the performance of the generative model. Therefore, it is crucial to train a GAN with an appropriate patch configuration.
  - ✓ If the receptive field is 1x1, each pixel is generated independently, leading to a diminished expressive power for representing continuous objects. If the receptive field is 286x286, only the global information of the image is emphasized, leading to a lack of detail in the image.



Discriminator receptive field	Per-pixel acc.	Per-class acc.	Class IOU
1×1	0.39	0.15	0.10
16×16	0.65	0.21	**0.17**
70×70	**0.66**	**0.23**	**0.17**
286×286	0.42	0.16	0.11

Future Frame Prediction for Anomaly Detection

- **Discriminator (PatchGAN)**




```python
class PixelDiscriminator(nn.Module):
    """Defines a PatchGAN discriminator (pixelGAN)"""

    def __init__(self, input_nc, num_filters=(128, 256, 512, 512)):
        """Construct a PatchGAN discriminator

        Parameters:
            input_nc (int)  -- the number of channels in input images
            num_filters (int)       -- the number of filters in the conv layer
        """
        super().__init__()

        self.conv1 = nn.Sequential(nn.Conv2d(input_nc, num_filters[0], kernel_size=4, padding=1, stride=2, bias=True),
                                   nn.LeakyReLU(0.2, True))

        self.conv2 = nn.Sequential(nn.Conv2d(num_filters[0], num_filters[1], kernel_size=4, padding=1, stride=2, bias=True),
                                   nn.LeakyReLU(0.2, True))

        self.conv3 = nn.Sequential(nn.Conv2d(num_filters[1], num_filters[2], kernel_size=4, padding=1, stride=2, bias=True),
                                   nn.LeakyReLU(0.2, True))

        self.conv4 = nn.Sequential(nn.Conv2d(num_filters[2], num_filters[3], kernel_size=4, padding=1, stride=1, bias=True),
                                   nn.LeakyReLU(0.2, True))

        self.out_conv = nn.Sequential(nn.Conv2d(num_filters[3], 1, kernel_size=4, padding=1, stride=1, bias=True))
```

```python
def forward(self, x):
    x = self.conv1(x)
    x = self.conv2(x)
    x = self.conv3(x)
    x = self.conv4(x)
    x = self.out_conv(x)
    out = torch.sigmoid(x)
    return out
```

Future Frame Prediction for Anomaly Detection

- Discriminator (PatchGAN)

Formula

$$K_{l-1} = K + S\,(K_l - 1)$$

Receptive field of $(l-1)^{th}$ layer; Kernel size; Stride value; Output receptive field of l^{th} layer

1) $K_l = 1 \times 1 \leftarrow$ 0 layer

$K = 4 \times 4$

$S = 1 \times 1$

$$K_{l-1} = (\underset{r}{4} \times \underset{c}{4}) + (\underset{r}{1} \times \underset{c}{1})(\underset{r}{1} \times \underset{c}{1} - 1)$$

$= 4 \times 4 \leftarrow$ C4 layer

2) $K_l = 4 \times 4 \leftarrow$ C4 layer

$K = 4 \times 4$

$S = 1 \times 1$

$$K_{l-1} = (\underset{r}{4} \times \underset{c}{4}) + (\underset{r}{1} \times \underset{c}{1})(\underset{r}{4} \times \underset{c}{4} - 1)$$

$= 7 \times 7 \leftarrow$ C3 layer

3) $K_l = 7 \times 7 \leftarrow$ C3 layer

$K = 4 \times 4$

$S = 2 \times 2$

$$K_{l-1} = (\underset{r}{4} \times \underset{c}{4}) + (\underset{r}{2} \times \underset{c}{2})(\underset{r}{7} \times \underset{c}{7} - 1)$$

$= 16 \times 16 \leftarrow$ C2 layer

4) $K_l = 16 \times 16 \leftarrow$ C2 layer

$K = 4 \times 4$

$S = 2 \times 2$

$$K_{l-1} = (\underset{r}{4} \times \underset{c}{4}) + (\underset{r}{2} \times \underset{c}{2})(\underset{r}{16} \times \underset{c}{16} - 1)$$

$= 34 \times 34 \leftarrow$ C1 layer

5) $K_l = 34 \times 34 \leftarrow$ C1 layer

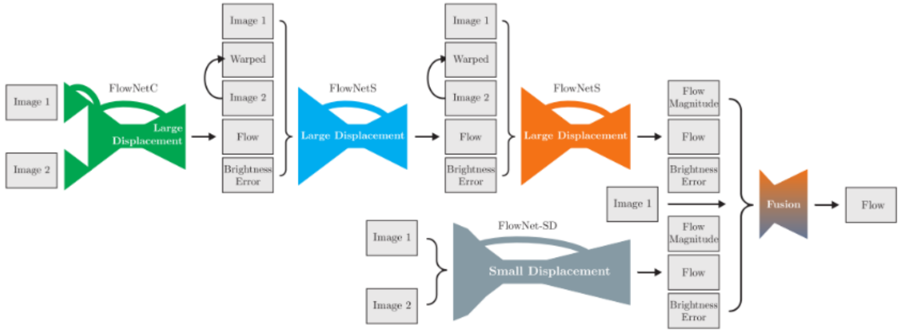
$K = 4 \times 4$

$S = 2 \times 2$

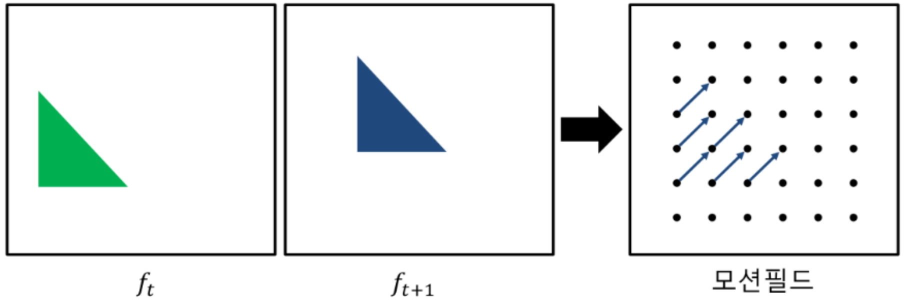
$$K_{l-1} = (\underset{r}{4} \times \underset{c}{4}) + (\underset{r}{2} \times \underset{c}{2})(\underset{r}{34} \times \underset{c}{34} - 1)$$

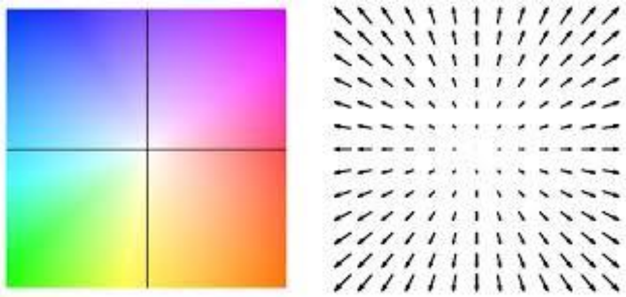
$= 70 \times 70 \leftarrow$ I layer.

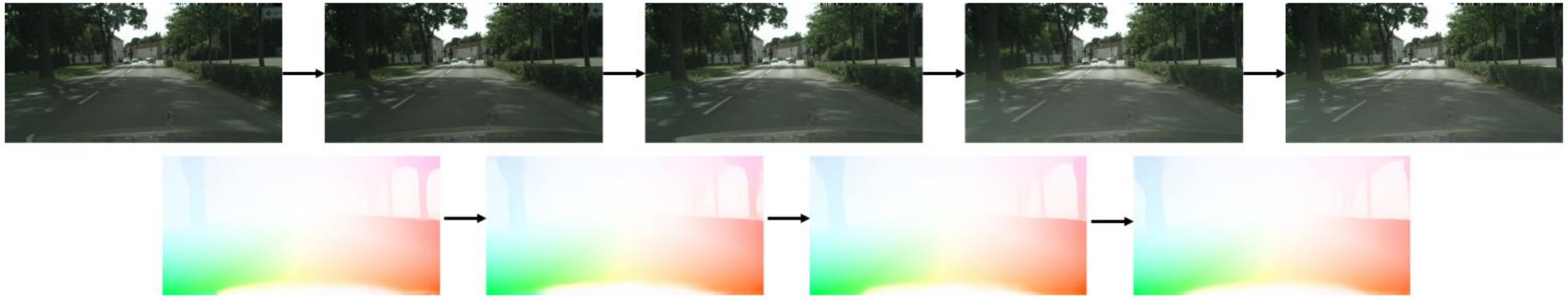
Future Frame Prediction for Anomaly Detection



- **FlowNet**
  - ✓ A deep learning model that generates FlowMap, an image that represents the change in the x-direction and the change in the y-direction between the two frames per pixel.







Future Frame Prediction for Anomaly Detection

- FlowNet

```
def def_models():
    # generator
    generator = UNet(12,3).cuda()
    # discriminator
    discriminator = PixelDiscriminator(input_nc=3).cuda()
    # flownet
    flownet = FlowNet2SD().cuda()

    return generator, discriminator, flownet
```

```
def load_models(cfg, generator, discriminator, flownet, optimizer_G, optimizer_D):
    if cfg.resume:
        generator.load_state_dict(torch.load(cfg.resume)['net_g'])
        discriminator.load_state_dict(torch.load(cfg.resume)['net_d'])
        optimizer_G.load_state_dict(torch.load(cfg.resume)['optimizer_g'])
        optimizer_D.load_state_dict(torch.load(cfg.resume)['optimizer_d'])
    else:
        generator.apply(weights_init_normal)
        discriminator.apply(weights_init_normal)

    # flownet
    flownet.load_state_dict(torch.load('flownet/pretrained/FlowNet2-SD.pth')['state_dict'])
    flownet.eval()
```

Future Frame Prediction for Anomaly Detection

- **Loss Function**
  - ✓ Loss Functions are consisted of Intensity loss, gradient loss, optical flow loss, adversarial loss
  - ✓ Intensity loss: increase similarity between pixels
  - ✓ gradient loss: make the differences between consecutive pairs of pixels similar, thereby smoothing out the generated images
  - ✓ optical flow loss : minimize the difference between the real the fake optical flow, thereby making two consecutive frames appear smoothly.
  - ✓ Adversarial loss: compete generator and discriminator to create better images

$$L_{int}(\hat{I}, I) = \|\hat{I} - I\|_2^2 \qquad L_{gd}(\hat{I}, I) = \sum_{i,j} \left\| |\hat{I}_{i,j} - \hat{I}_{i-1,j}| - |I_{i,j} - I_{i-1,j}| \right\|_1 + \left\| |\hat{I}_{i,j} - \hat{I}_{i,j-1}| - |I_{i,j} - I_{i,j-1}| \right\|_1$$

$$L_{adv}^{G}(\hat{I}) = \sum_{i,j} \frac{1}{2} L_{MSE}\left(D(\hat{I})_{i,j}, 1\right) \qquad L_{adv}^{D}(\hat{I}, I) = \sum_{i,j} \frac{1}{2} L_{MSE}(D(I)_{i,j}, 1) + \sum_{i,j} \frac{1}{2} L_{MSE}(D(\hat{I})_{i,j}, 0)$$

$$L_{op}(\hat{I}_{t+1}, I_{t+1}, I_t) = \|f(\hat{I}_{t+1}, I_t) - f(I_{t+1}, I_t)\|_1$$

Future Frame Prediction for Anomaly Detection

- **Intensity loss**
  - ✓ increase similarity between pixels.

$$L_{int}(\hat{I}, I) = \|\hat{I} - I\|_2^2$$

```python
class Intensity_Loss(nn.Module):
    def __init__(self):
        super().__init__()

    def forward(self, gen_frames, gt_frames):
        return torch.mean(torch.abs((gen_frames - gt_frames) ** 2))
```

generated frame

ground truth

Future Frame Prediction for Anomaly Detection

- **Gradient loss**

$$L_{gd}(\hat{I},I)=\sum_{i,j}\left\||\hat{I}_{i,j}-\hat{I}_{i-1,j}|-|I_{i,j}-I_{i-1,j}|\right\|_1+\left\||\hat{I}_{i,j}-\hat{I}_{i,j-1}|-|I_{i,j}-I_{i,j-1}|\right\|_1$$

✓ make the differences between consecutive pairs of pixels similar, thereby smoothing out the generated images.

```
class Gradient_Loss(nn.Module):
    def __init__(self, channels):
        super().__init__()

        pos = torch.from_numpy(np.identity(channels, dtype=np.float32))
        neg = -1 * pos
        # Note: when doing conv2d, the channel order is different from tensorflow, so do permutation.
        self.filter_x = torch.stack((neg, pos)).unsqueeze(0).permute(3, 2, 0, 1).cuda()
        self.filter_y = torch.stack((pos.unsqueeze(0), neg.unsqueeze(0))).permute(3, 2, 0, 1).cuda()

    def forward(self, gen_frames, gt_frames):
        # Do padding to match the  result of the original tensorflow implementation
        gen_frames_x = nn.functional.pad(gen_frames, [0, 1, 0, 0])
        gen_frames_y = nn.functional.pad(gen_frames, [0, 0, 0, 1])
        gt_frames_x = nn.functional.pad(gt_frames, [0, 1, 0, 0])
        gt_frames_y = nn.functional.pad(gt_frames, [0, 0, 0, 1])

        gen_dx = torch.abs(nn.functional.conv2d(gen_frames_x, self.filter_x))
        gen_dy = torch.abs(nn.functional.conv2d(gen_frames_y, self.filter_y))
        gt_dx = torch.abs(nn.functional.conv2d(gt_frames_x, self.filter_x))
        gt_dy = torch.abs(nn.functional.conv2d(gt_frames_y, self.filter_y))

        grad_diff_x = torch.abs(gt_dx - gen_dx)
        grad_diff_y = torch.abs(gt_dy - gen_dy)

        return torch.mean(grad_diff_x + grad_diff_y)
```

generated frame

```
tensor([[[[1., 2., 3.],
          [4., 5., 6.],
          [7., 8., 9.]]]])
```

ground truth

```
tensor([[[[2., 4., 6.],
          [6., 8., 9.],
          [3., 2., 1.]]]])
```

```
tensor([[[[-1.,  1.]]]], device='cuda:0')
tensor([[[[ 1.],
          [-1.]]]], device='cuda:0')
```

filter_x,
filter_y

```
tensor([[[[1., 2., 3., 0.],
          [4., 5., 6., 0.],
          [7., 8., 9., 0.]]]], device='cuda:0')
tensor([[[[1., 2., 3.],
          [4., 5., 6.],
          [7., 8., 9.],
          [0., 0., 0.]]]], device='cuda:0')
```

gen_frame_x,
gen_frame_y

```
tensor([[[[1., 1., 3.],
          [1., 1., 6.],
          [1., 1., 9.]]]], device='cuda:0')
tensor([[[[3., 3., 3.],
          [3., 3., 3.],
          [7., 8., 9.]]]], device='cuda:0')
```

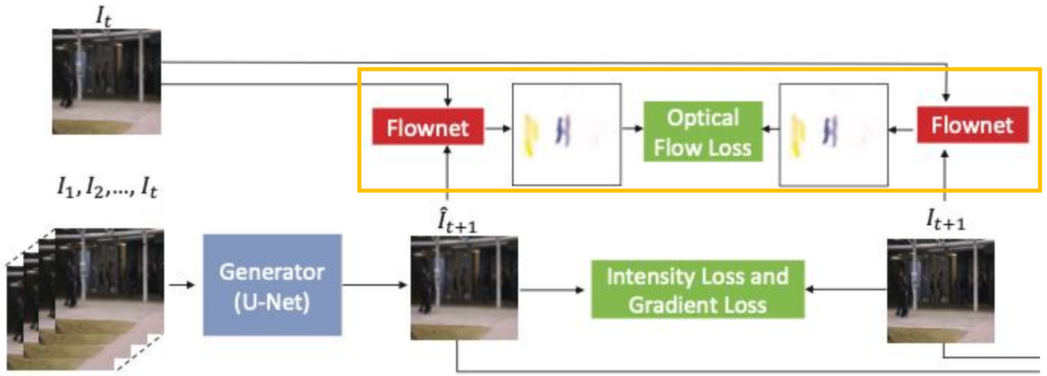
gen_dx,
gen_dy

Future Frame Prediction for Anomaly Detection

- **Optical Flow loss**
  - ✓ minimize the difference between the real the fake optical flow, thereby making two consecutive frames appear smoothly.

$$L_{op}(\hat{I}_{t+1}, I_{t+1}, I_t) = \| f(\hat{I}_{t+1}, I_t) - f(I_{t+1}, I_t) \|_1$$

```python
class Flow_Loss(nn.Module):
    def __init__(self):
        super().__init__()

    def forward(self, gen_flows, gt_flows):
        return torch.mean(torch.abs(gen_flows - gt_flows))
```



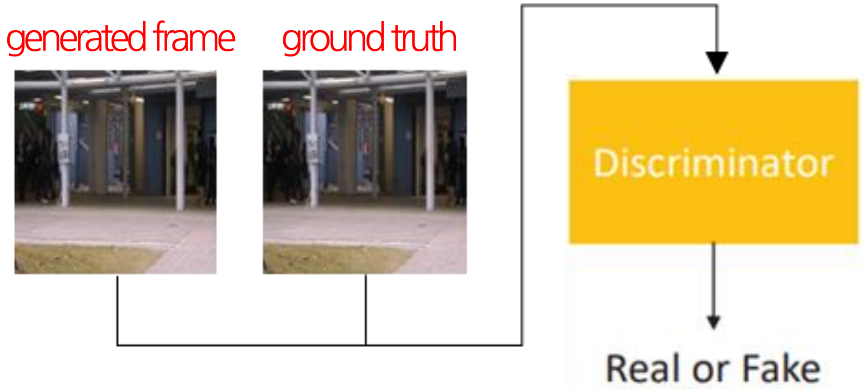
Future Frame Prediction for Anomaly Detection

- **Discriminator adversarial loss**

✓ distinguish real and fake future frame

$$L_{adv}^{D}(\hat{I}, I) = \sum_{i,j} \frac{1}{2} L_{MSE}\left(D(I)_{i,j}, 1\right) + \sum_{i,j} \frac{1}{2} L_{MSE}\left(D(\hat{I})_{i,j}, 0\right)$$

```python
class Discriminate_Loss(nn.Module):
    def __init__(self):
        super().__init__()

    def forward(self, real_outputs, fake_outputs):
        return torch.mean((real_outputs - 1) ** 2 / 2) + torch.mean(fake_outputs ** 2 / 2)
```



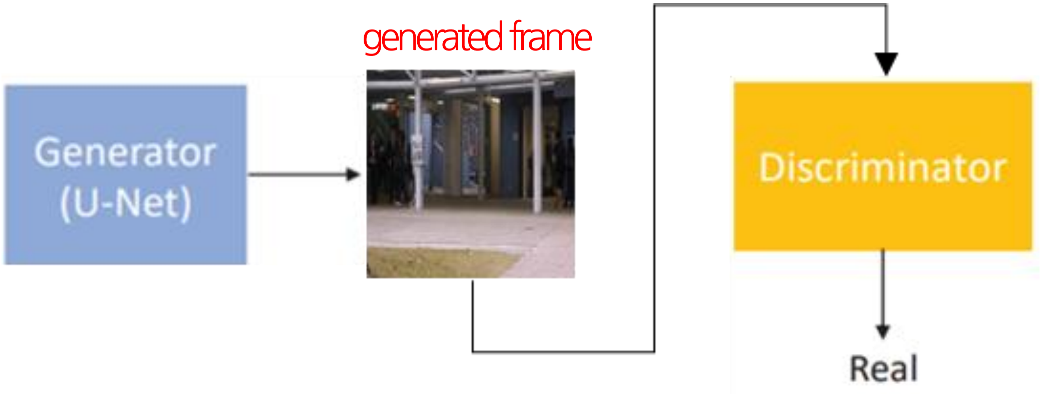
Future Frame Prediction for Anomaly Detection

- **Generator adversarial loss**

  ✓ predict future frames well enough to deceive Discriminator

$$L^G_{adv}(\hat{I}) = \sum_{i,j} \frac{1}{2} L_{MSE}\left(D(\hat{I})_{i,j}, 1\right)$$

```python
class Adversarial_Loss(nn.Module):
    def __init__(self):
        super().__init__()

    def forward(self, fake_outputs):
        # TODO: compare with torch.nn.MSELoss ?
        return torch.mean((fake_outputs - 1) ** 2 / 2)
```



Future Frame Prediction for Anomaly Detection

- Total loss

3.5. Objective Function

We combine all these constraints regarding appearance, motion, and adversarial training, into our objective function, and arrive at the following objective function:

$$L_{\mathcal{G}} = \lambda_{int} L_{int}(\hat{I}_{t+1}, I_{t+1}) + \lambda_{gd} L_{gd}(\hat{I}_{t+1}, I_{t+1}) + \lambda_{op} L_{op}(\hat{I}_{t+1}, I_{t+1}, I_t) + \lambda_{adv} L^{\mathcal{G}}_{adv}(\hat{I}_{t+1}) \tag{7}$$

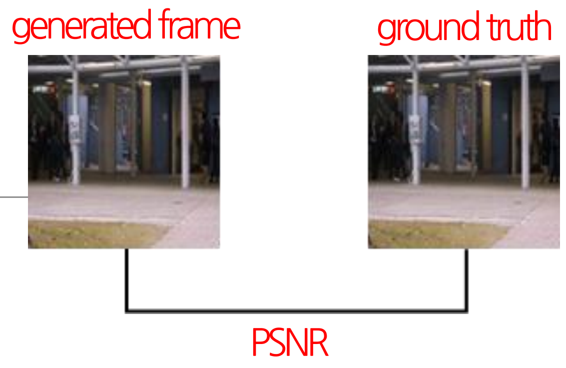
When we train $\mathcal{D}$, we use the following loss function:

$$L_{\mathcal{D}} = L^{\mathcal{D}}_{adv}(\hat{I}_{t+1}, I_{t+1}) \tag{8}$$

To train the network, the intensity of pixels in all frames are normalized to [-1, 1] and the size of each frame is resized to 256×256. Similar to [27], we set $t = 4$ and randomly clip 5 sequential frames. Adam [19] based Stochastic Gradient Descent method is used for parameter optimization. The mini-batch size is 4. For gray scale datasets, the learning rate of generator and discriminator are set to 0.0001 and 0.00001, separately. While for color scale datasets, they start from 0.0002 and 0.00002, respectively. λ_{int}, λ_{gd}, λ_{op} and λ_{adv} slightly vary from datasets and an easy way is to set them as 1.0, 2.0 and 0.05, respectively.

```
discriminate_loss = losses['discriminate_loss']
intensity_loss = losses['intensity_loss']
gradient_loss = losses['gradient_loss']
adversarial_loss = losses['adversarial_loss']
flow_loss = losses['flow_loss']

coefs = [1, 1, 0.05, 2] # inte_l, grad_l, adv_l, flow_l

# future frame prediction and get loss
pred  = generator(input)
inte_l = intensity_loss(pred, target)
grad_l = gradient_loss(pred, target)
adv_l = adversarial_loss(discriminator(pred))

# flowmap prediction and get loss
gt_flow_input = torch.cat([input_last.unsqueeze(2), target.unsqueeze(2)], 2)
pred_flow_input = torch.cat([input_last.unsqueeze(2), pred.unsqueeze(2)], 2)

flow_gt = (flownet(gt_flow_input * 255.) / 255.).detach()  # Input for flownet2sd is in (0, 255).
flow_pred = (flownet(pred_flow_input * 255.) / 255.).detach()
flow_l = flow_loss(flow_pred, flow_gt)

loss_gen = coefs[0] * inte_l + \
           coefs[1] * grad_l + \
           coefs[2] * adv_l + \
           coefs[3] * flow_l

# discriminator
loss_dis = discriminate_loss(discriminator(target),
                             discriminator(pred.detach()))
```

Future Frame Prediction for Anomaly Detection



- **Testing**
  - ✓ Peak Signal-to-noise ratio (PSNR)
    - represents the quality of an image by normalizing Mean Squared Error (MSE) to pixel information scale; higher values indicate better quality.

$$PSNR(I, \hat{I}) = 10 \log_{10} \frac{[\max_{\hat{I}}]^2}{\frac{1}{N}\sum_{i=0}^{N}(I_i - \hat{I}_i)^2}$$

JPEG

Original image | PSNR 34.8227dB | PSNR 30.9394dB | PSNR 25.8699dB

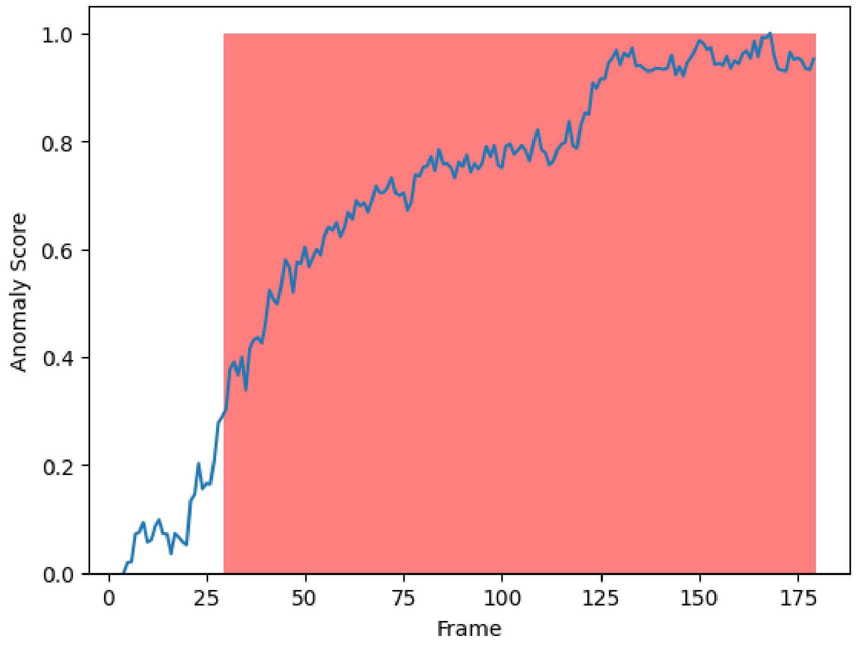
```
def psnr_error(gen_frames, gt_frames):
    """
    Computes the Peak Signal to Noise Ratio error between the generated images and the ground
    truth images.
    @param gen_frames: A tensor of shape [batch_size, 3, height, width]. The frames generated by the
                       generator model.
    @param gt_frames: A tensor of shape [batch_size, 3, height, width]. The ground-truth frames for
                      each frame in gen_frames.
    @return: A scalar tensor. The mean Peak Signal to Noise Ratio error over each frame in the
             batch.
    """
    shape = list(gen_frames.shape)
    num_pixels = (shape[1] * shape[2] * shape[3])
    gt_frames = (gt_frames + 1.0) / 2.0  # if the generate ouuput is sigmoid output, modify here.
    gen_frames = (gen_frames + 1.0) / 2.0
    square_diff = (gt_frames - gen_frames) ** 2
    batch_errors = 10 * log10(1. / ((1. / num_pixels) * torch.sum(square_diff, [1, 2, 3])))
    return torch.mean(batch_errors)
```

Future Frame Prediction for Anomaly Detection

- **Testing**
  - ✓ Anomaly Score
    - PSNR is min-max normalized to calculate the Anomaly Score, also known as the Regular Score.

$$S(t) = 1 - \frac{PSNR(I_t, \hat{I}_t) - \min PSNR\,(I_t, \hat{I}_t)}{\max PSNR(I_t, \hat{I}_t) - \min PSNR\,(I_t, \hat{I}_t)}$$

```python
def min_max_normalize(arr, eps=1e-8):
    min_val = np.min(arr)
    max_val = np.max(arr)
    denominator = max_val - min_val + eps  # Avoid division by zero

    normalized_arr = (arr - min_val) / denominator
    return normalized_arr
```



연세대학교

Future Frame Prediction for Anomaly Detection

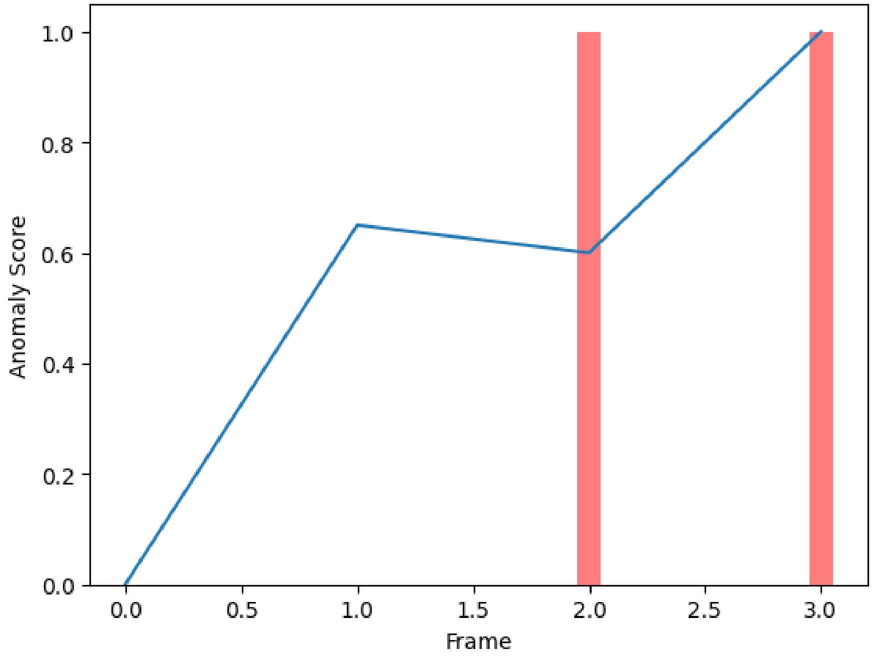
- **Testing**
  - ✓ Area Under the ROC Curve (AUROC) [1]
    - AUROC is an evaluation metric for anomaly detection, and it is calculated based on the Anomaly Score.

```python
import numpy as np
from sklearn import metrics
import matplotlib.pyplot as plt
```

```python
y = np.array([0, 0, 1, 1])
scores = np.array([0, 0.65, 0.6, 1.0])

length = len(scores)
anomalies_idx = [i for i,l in enumerate(y) if l==1]

print("length:", length)
print("anomalies_idx", anomalies_idx)
```

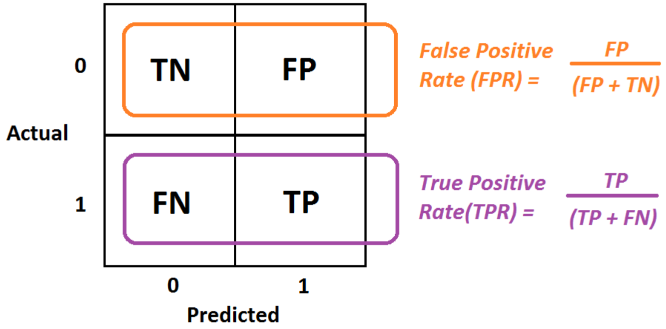
```
length: 4
anomalies_idx [2, 3]
```

```python
plt.clf()
plt.plot([num for num in range(length)], [score for score in scores]) # plotting
plt.bar(anomalies_idx, max(scores), width=0.1, color='r', alpha=0.5) # check answer
plt.xlabel("Frame")
plt.ylabel("Anomaly Score")
```



This is an example where abnormal is considered positive(1) and normal is considered negative(0).

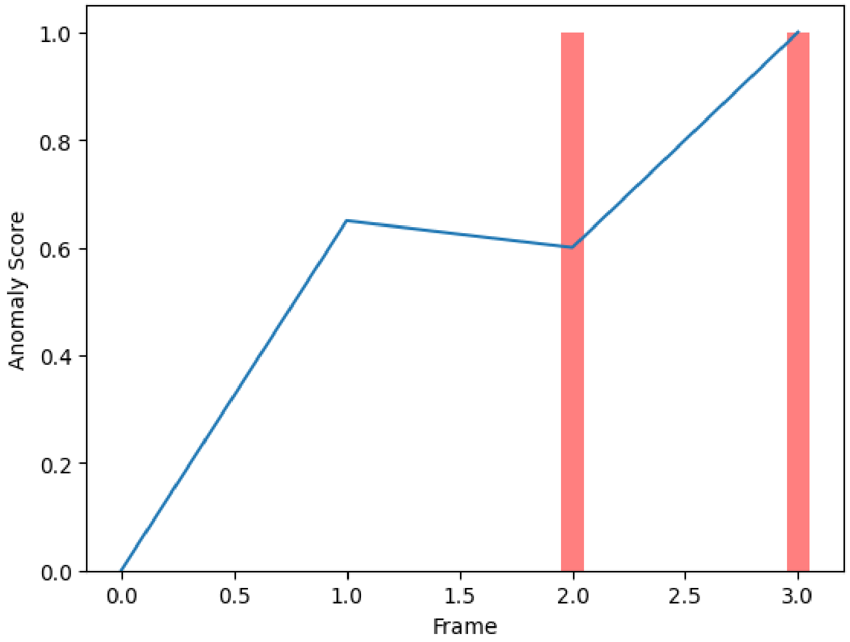
Future Frame Prediction for Anomaly Detection



- **Testing**
  - ✓ Area Under the ROC Curve (AUROC) [2]
    - The ROC curve is a graph that records True Positive Rate (TPR) and False Positive Rate (FPR) while changing the threshold.
    - TPR (True Positive Rate): The proportion of data correctly predicted as positive in actual positive data.
    - FPR (False Positive Rate): The proportion of data incorrectly predicted as positive in actual negative data.

```
fpr, tpr, thresholds = metrics.roc_curve(y, scores, pos_label=1)
auc = metrics.auc(fpr, tpr)

print("thresholds", thresholds)
print("tpr:", tpr)
print("fpr", fpr)
print("auc", auc)
```

```
thresholds [2.   1.   0.65 0.6  0.  ]
tpr: [0.  0.5 0.5 1.  1. ]
fpr [0.  0.  0.5 0.5 1. ]
auc 0.75
```



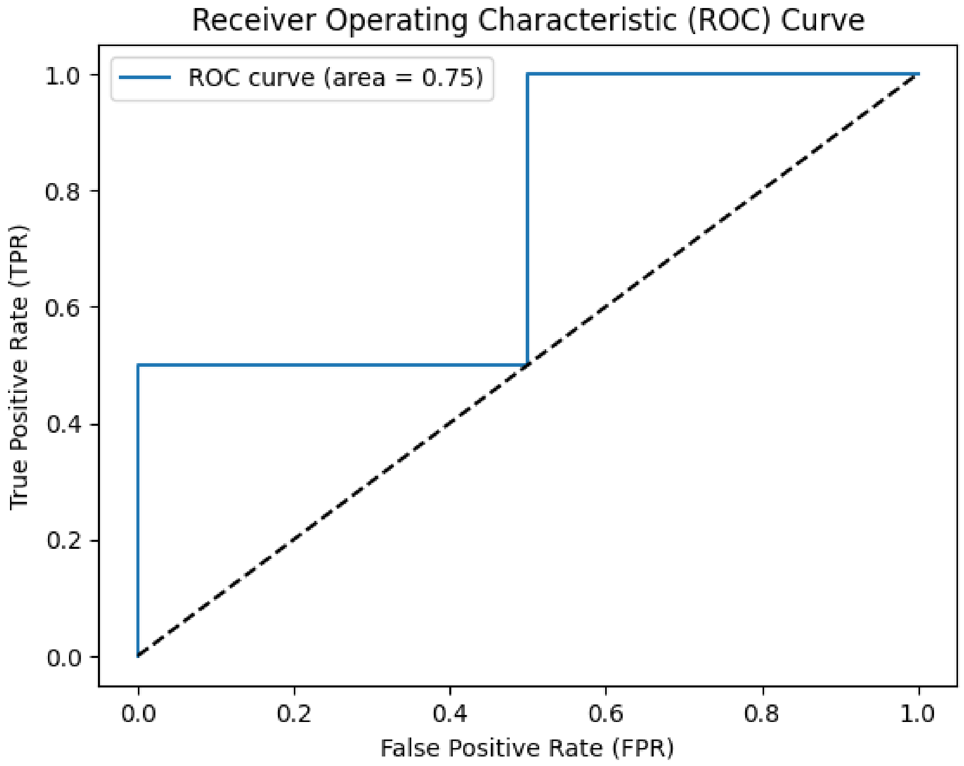
This is an example where abnormal is considered positive(1) and normal is considered negative(0).

연세대학교

Future Frame Prediction for Anomaly Detection

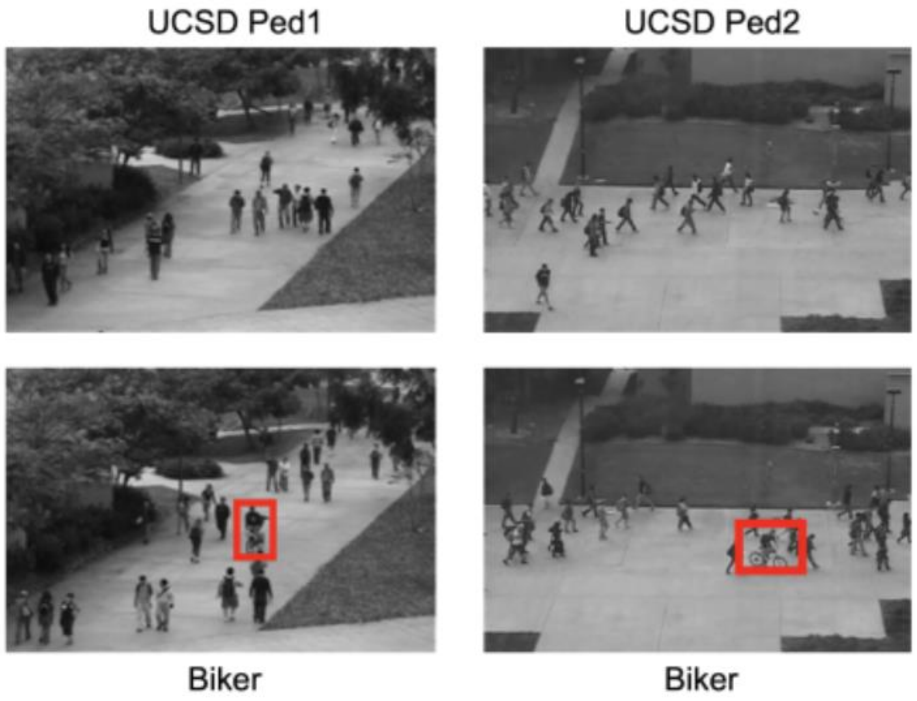
- **Testing**
  - ✓ Area Under the ROC Curve (AUROC) [3]
    - The AUC (Area Under the Curve) of the ROC curve represents the overall performance of the model, with a higher AUC value indicating better performance, approaching 1.

```
plt.clf()
plt.plot(fpr, tpr, label=f'ROC curve (area = {auc:.2f})')
plt.plot([0,1], [0,1], 'k--')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
```



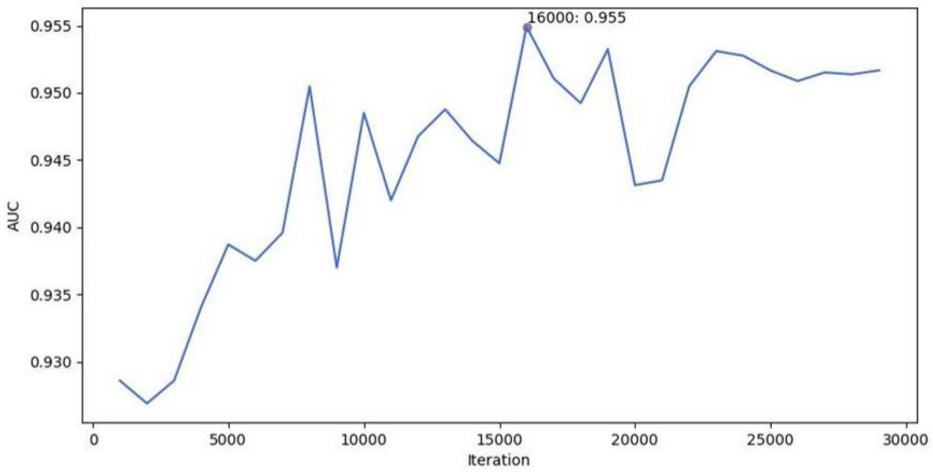
Future Frame Prediction for Anomaly Detection

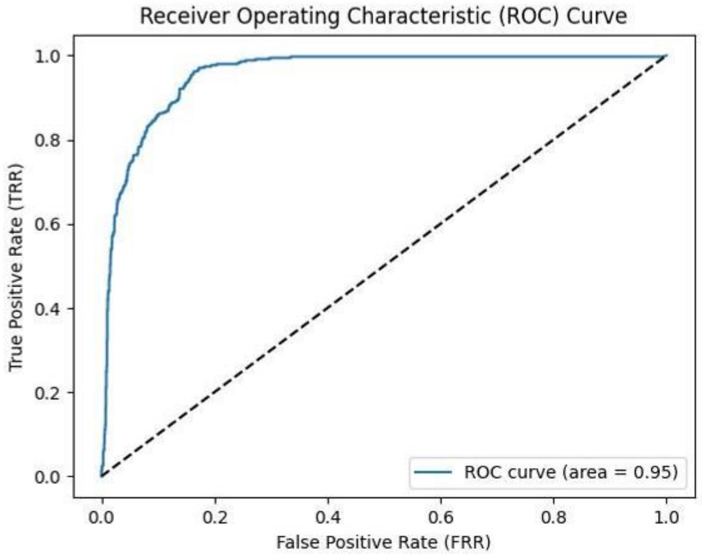
- **Experiments**
  - ✓ Pedestrian Road CCTV Dataset
    - UCSD Ped2 dataset consists of 16 training videos and 12 testing videos. Abnormal situations in this dataset involve videos of pedestrians or vehicles moving on roads.



Future Frame Prediction for Anomaly Detection

- **Experiments**
  - ✓ AUC & Heatmap

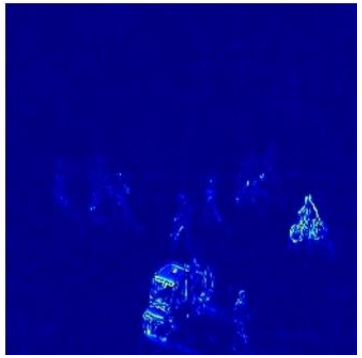




Real frame

Fake frame

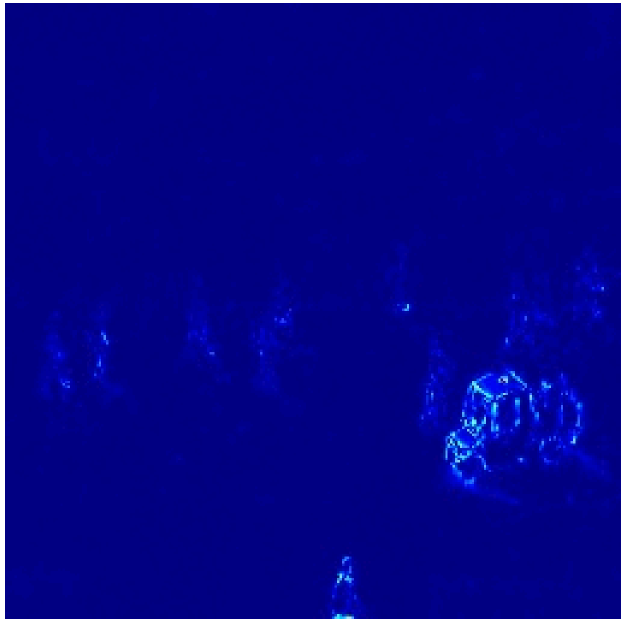
Heatmap



Future Frame Prediction for Anomaly Detection

- **Experiments**
  - ✓ Heatmap Video

Frame

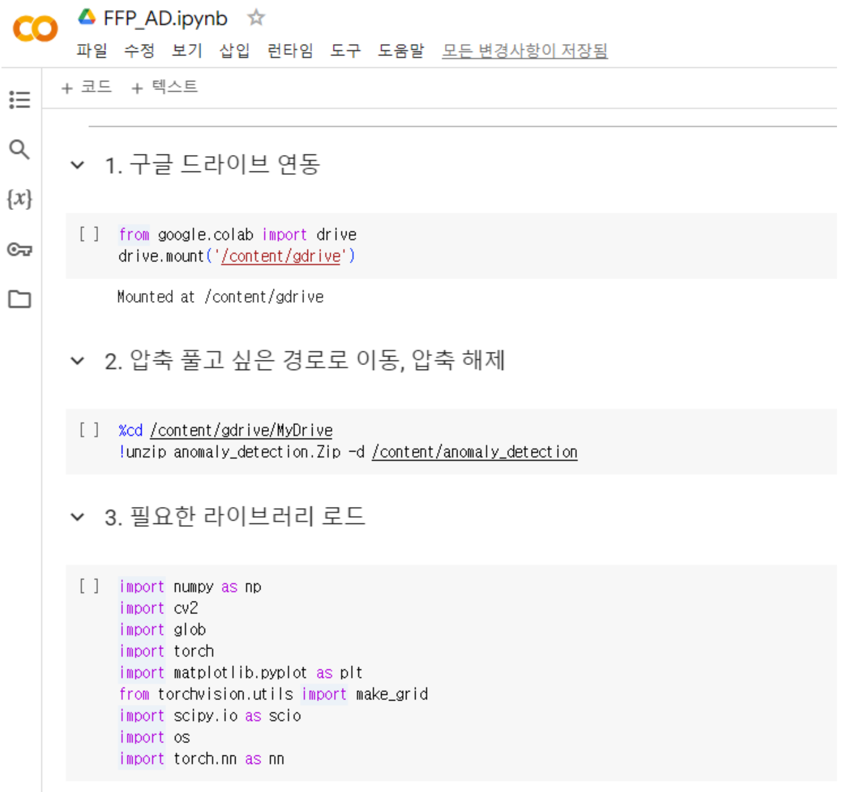


Heatmap

Frame x Heatmap

Future Frame Prediction for Anomaly Detection

- Colab page

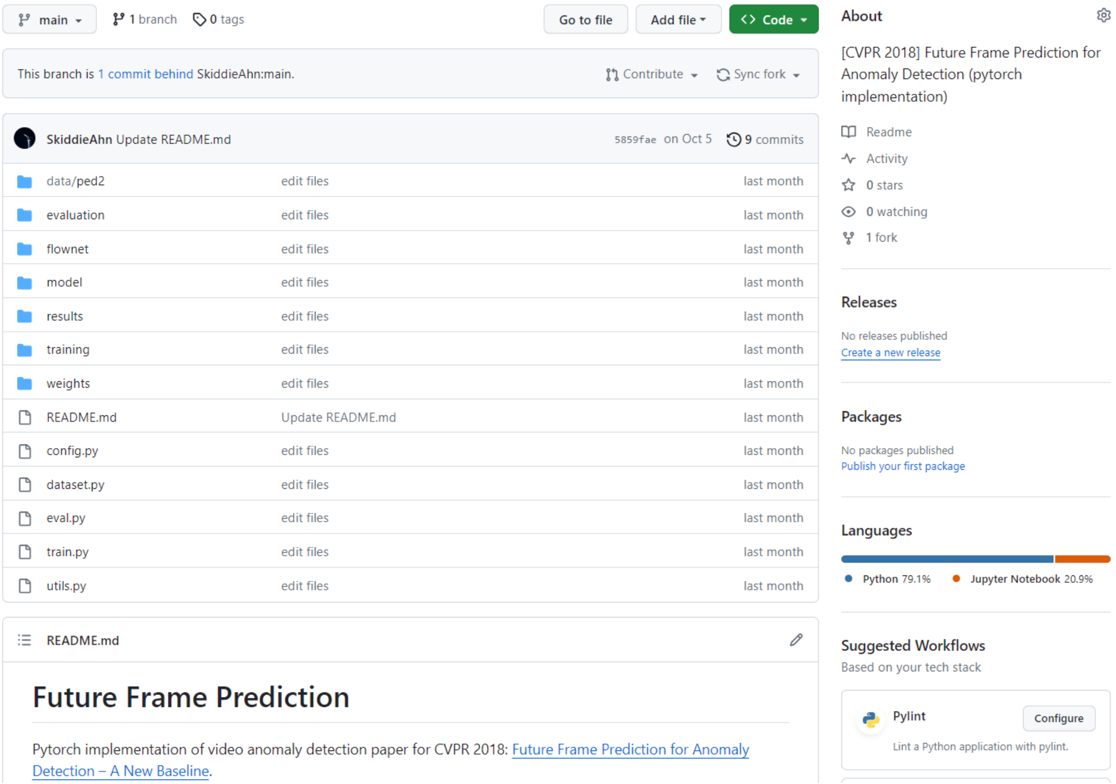


https://colab.research.google.com/drive/1ckH-16QVRLmTjnocvmFTga5h0zSyT4_u?usp=sharing

https://drive.google.com/file/d/1vaMQAU_QZ-tlXGq0cwbA9DIVLnFiy1Hq/view?usp=drive_link

Future Frame Prediction for Anomaly Detection

- ## GitHub page



https://github.com/ad-yonsei/Code-Future-Frame-Prediction

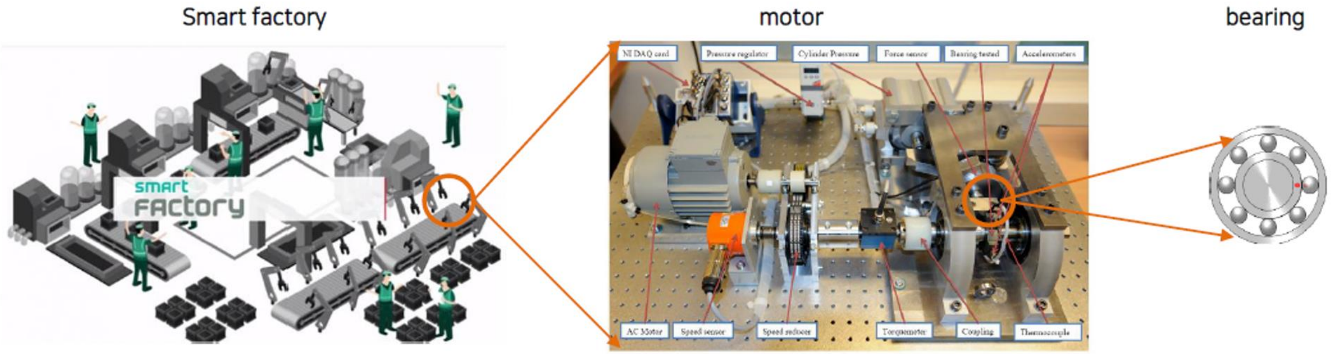
Anomaly detection for application

- **Smart factory**

✓ Smart factory's main task is Predictive Maintenance service.

✓ Predictive Maintenance service aims to conduct maintenance before equipment failure by anomaly detection.

✓ In industrial processes, about 60% of motor failures are due to bearing faults, making it crucial for manufacturing companies to detect these bearing failures effectively when adopting smart factory solutions.



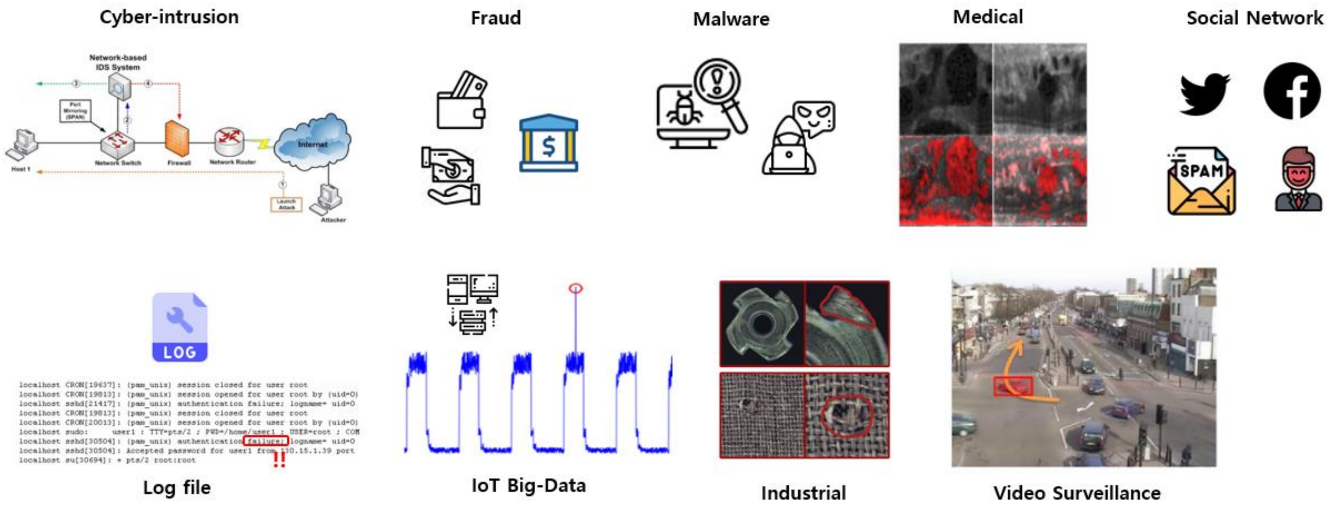
Anomaly detection for application

- **Surveillance camera**
  - ✓ Security cameras are available in a wide range of styles and features, and they are a common component in a security system.
  - ✓ Video surveillance involves the act of observing a scene or scenes and looking for specific behaviors that are improper or that may indicate the emergence or existence of improper behavior.
  - ✓ Video surveillance is commonly used for:
    - ✓ Remote video monitoring
    - ✓ Vandalism deterrence
    - ✓ Employee safety
    - ✓ Outdoor perimeter security

https://alchera.ai/technology/visual-ai/behavior-analysis

Anomaly detection for application

- **The others**
  - ✓ Fraud Detection: cases of detecting illegal activities in insurance, credit, and financial-related data.
  - ✓ Malware Detection: cases of detecting malware.
  - ✓ Cyber-Intrusion Detection: cases of detecting intrusions on computer systems.
  - ✓ Medical Anomaly Detection: cases of detecting in medical data such as medical images and EEG records.
  - ✓ Social Networks Anomaly Detection: cases of detecting anomalies on social networks such as spam.
  - ✓ Log Anomaly Detection: cases of detecting failure causes by examining logs recorded by a system
  - ✓ IoT Big-Data Anomaly Detection: cases of detecting anomalies in data generated by sensors.



Thank you

연세대학교