Autoencoders

December 2023

http://link.koreatech.ac.kr

[Reference] https://deepinsight.tistory.com/126

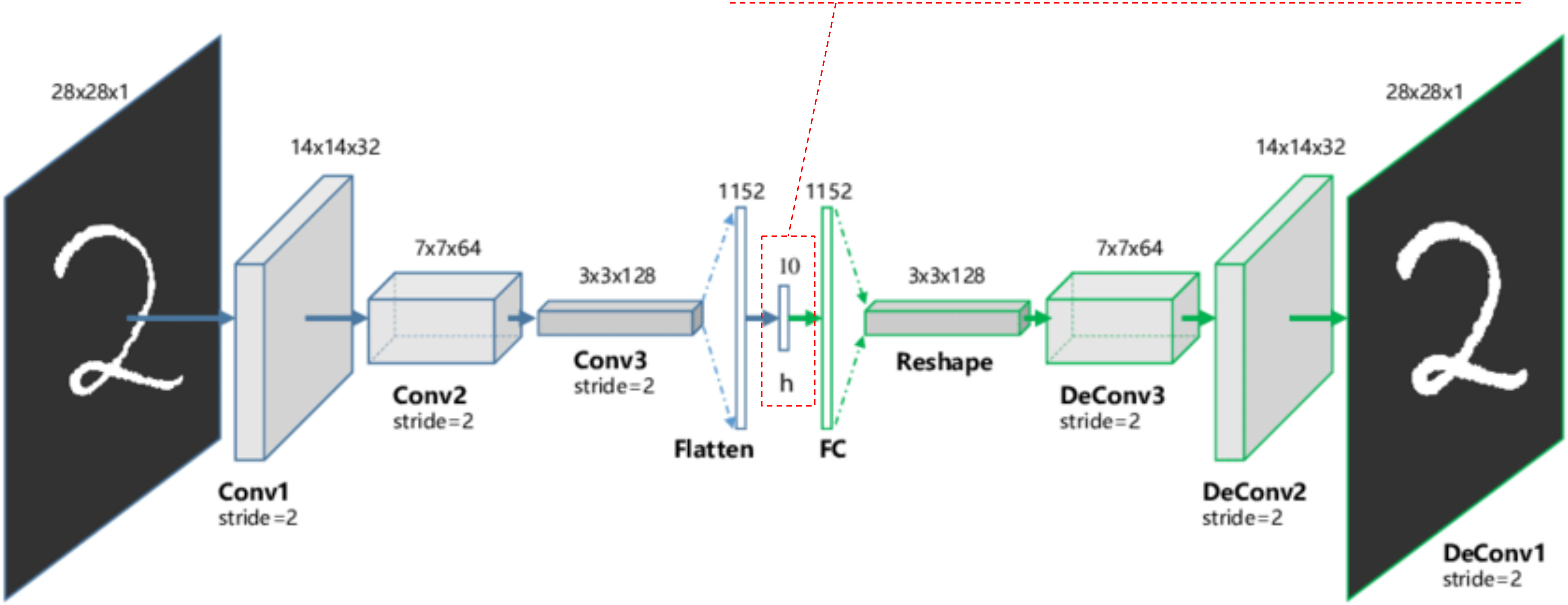
Autoencoders

Autoencoders

◈What is Autoencoders?

– It is designed to reproduce their input, especially for images.

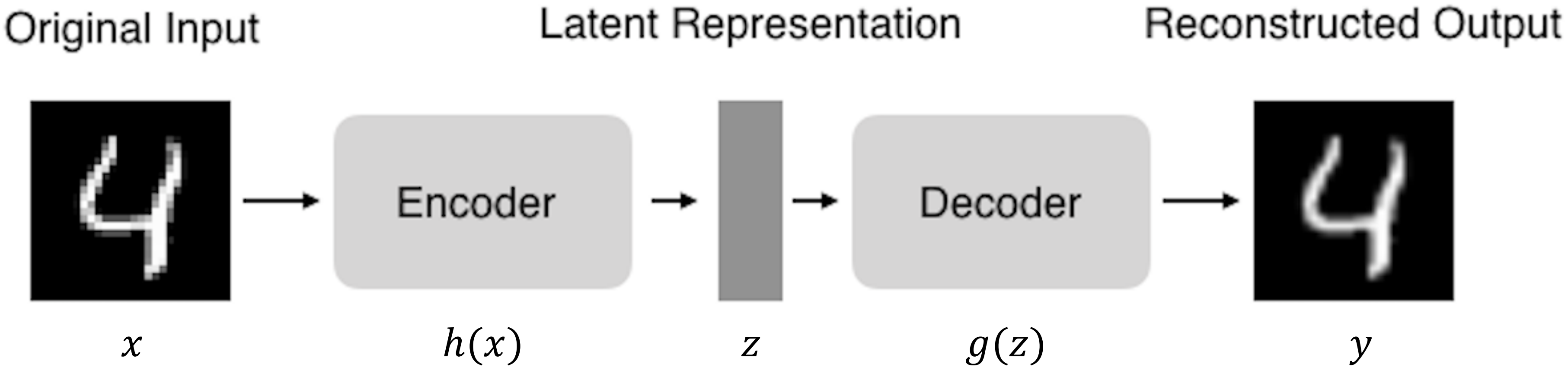
- Key point is to reproduce the input from a **learned** encoding (or embedding, latent representation)



Autoencoders

◈What is Autoencoders?

- Encoder: compress input into a latent-space of usually smaller dimension
  - $z = h(x) \in \mathbb{R}^{d_z}$
- Decoder: reconstruct input from the latent space.
  - $y = g(z) = g(h(x))$
- Reconstruction Loss Function
  - $L_{AE} = \sum_{x \in D} L(x, y) = \sum_{x \in D} \|x - y\|^2 = \sum_{x \in D} \|x - g(h(x))\|^2$

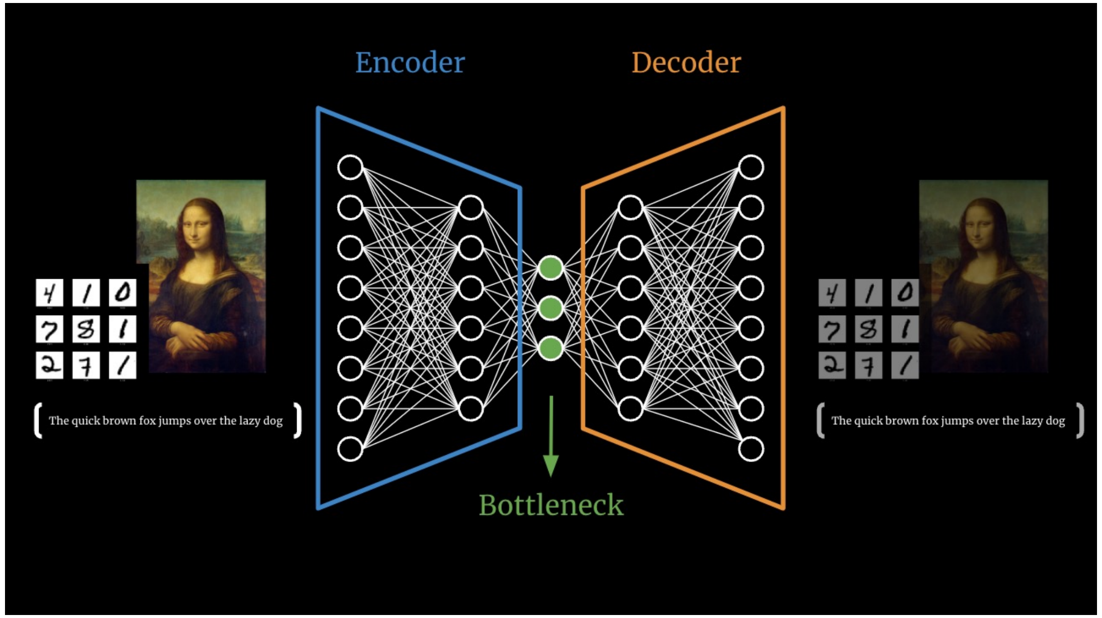


[Source] https://towardsdatascience.com/deep-inside-autoencoders-7e41f319999f

Autoencoders

◈What is the Point of Autoencoders?

We are squeezing the information into fewer dimensions (hence the bottleneck) while trying to ensure that we can still get back to the original values. Therefore, we are creating a custom function that compresses the data, which is a way to reduce the dimensionality and extract meaningful information. After training the Undercomplete Autoencoder, we typically discard the Decoder and only use the Encoder part.

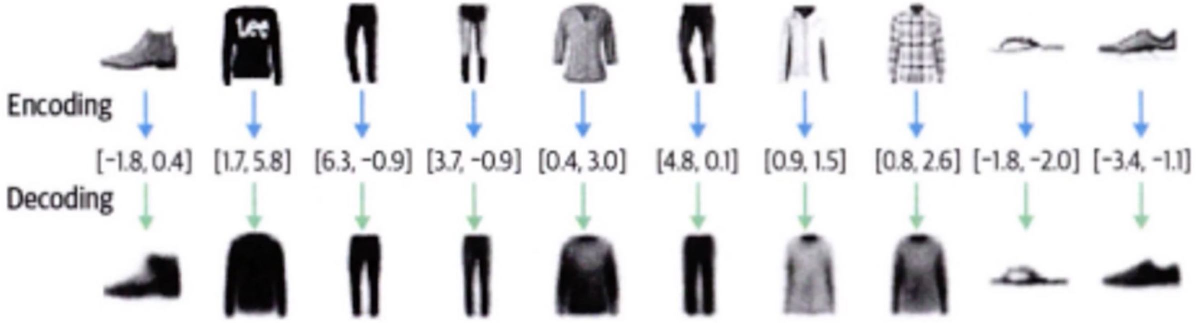

Encoder
Decoder
Bottleneck
4 1 0
7 8 1
2 7 1
The quick brown fox jumps over the lazy dog
4 1 0
7 8 1
2 7 1
The quick brown fox jumps over the lazy dog

[Source] https://www.saberhq.com/blog/autoencoders

Autoencoders

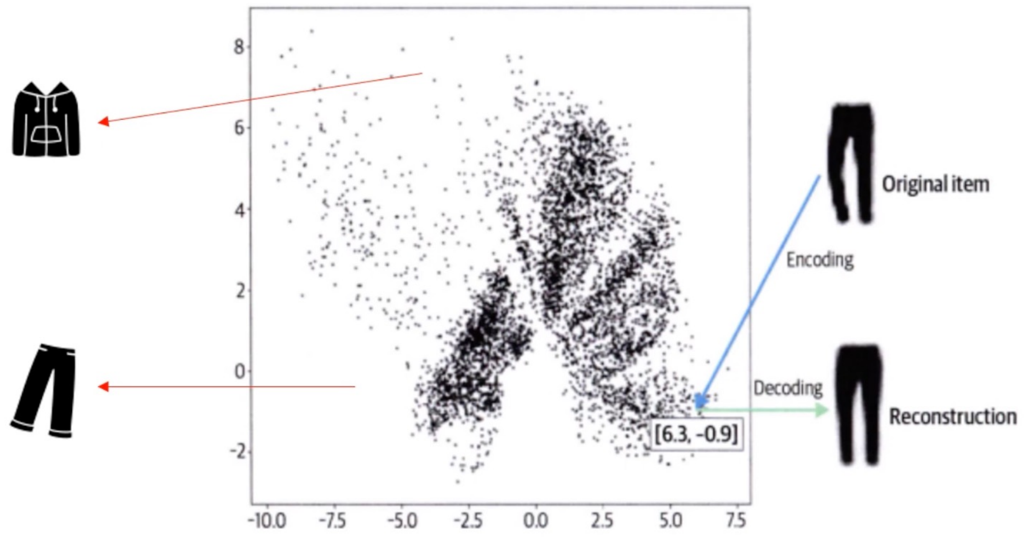
◈ Autoencoders Keywords

- Unsupervised (or Self-Supervised) Learning
- Representation Learning
- Dimensionality Reduction
- Generative Model Learning



◈ Applications of Autoencoders (1/2)

- Visualization of Data feature
  - Getting "Features with Reduced Dimension" of input data and visualizing the input data by using the reduced features

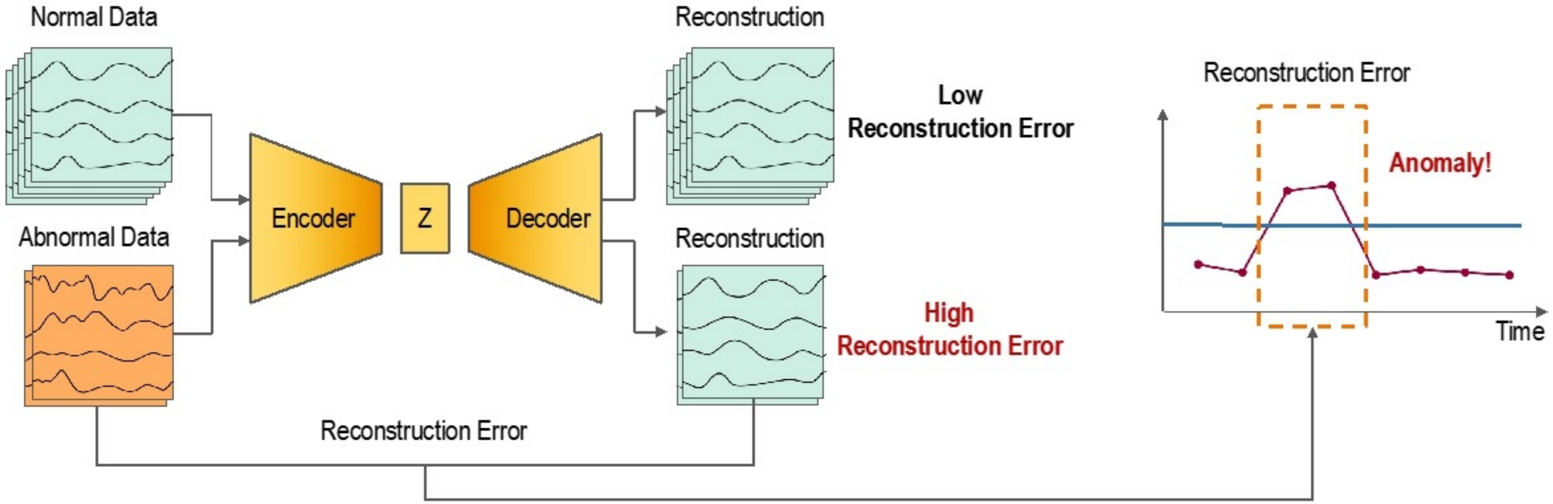


Autoencoders

◈Applications of Autoencoders (2/2)

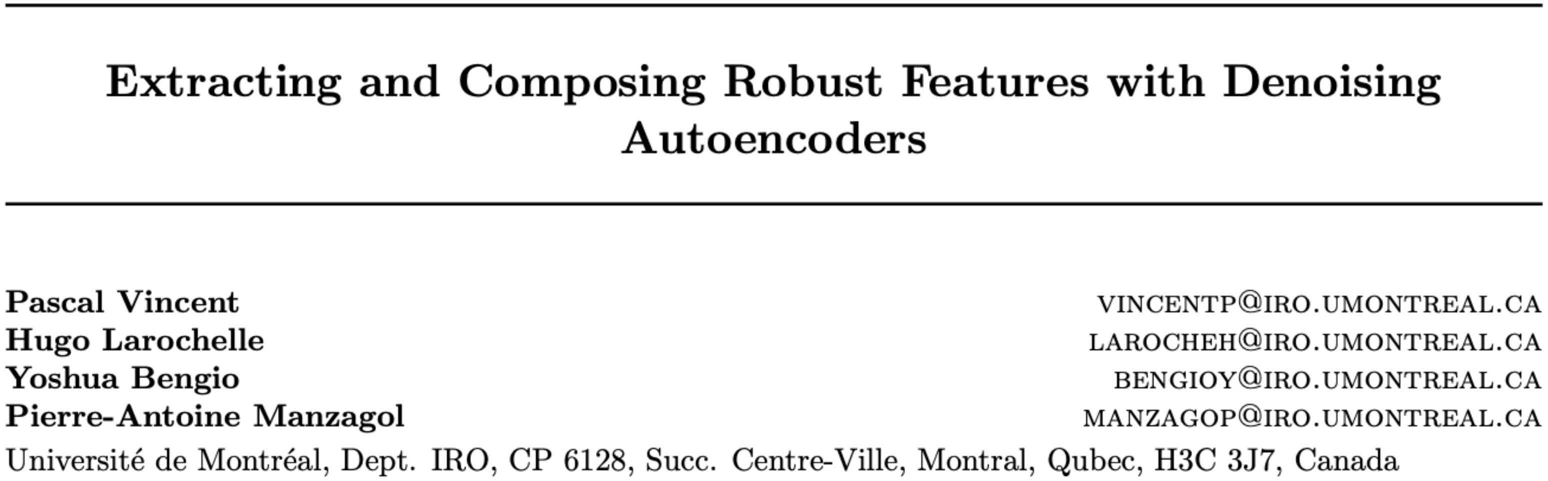
– Anomaly Detection

- We train the model on normal instances so that if we feed it any outliers, they will be detected easily.
- we can use the reconstruction loss (or Error) as a metric to detect outliers.



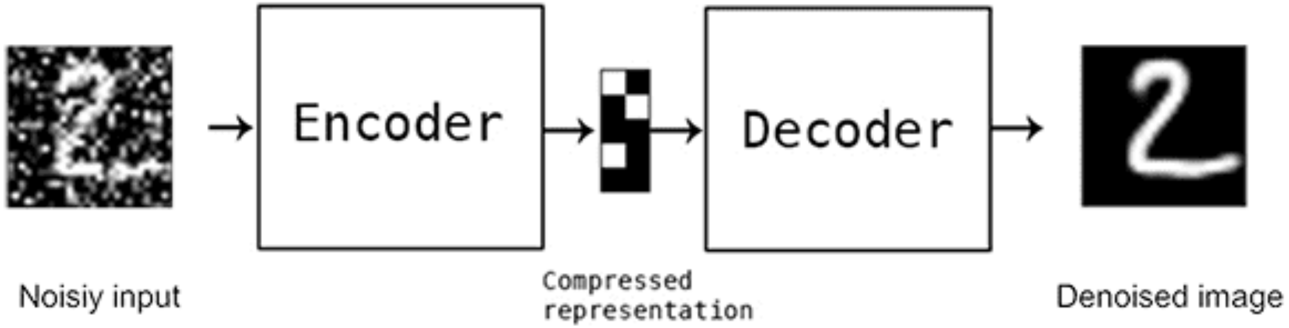
Denoising Autoencoders

- ◈What is Denoising Autoencoders (DAE)?
  - – It trains an autoencoder to **identify and remove noise** from a set of data
  - – It **extract and compose robust features** from a set of data



Extracting and Composing Robust Features with Denoising Autoencoders

Pascal Vincent VINCENTP@IRO.UMONTREAL.CA
Hugo Larochelle LAROCHEH@IRO.UMONTREAL.CA
Yoshua Bengio BENGIOY@IRO.UMONTREAL.CA
Pierre-Antoine Manzagol MANZAGOP@IRO.UMONTREAL.CA
Université de Montréal, Dept. IRO, CP 6128, Succ. Centre-Ville, Montral, Qubec, H3C 3J7, Canada

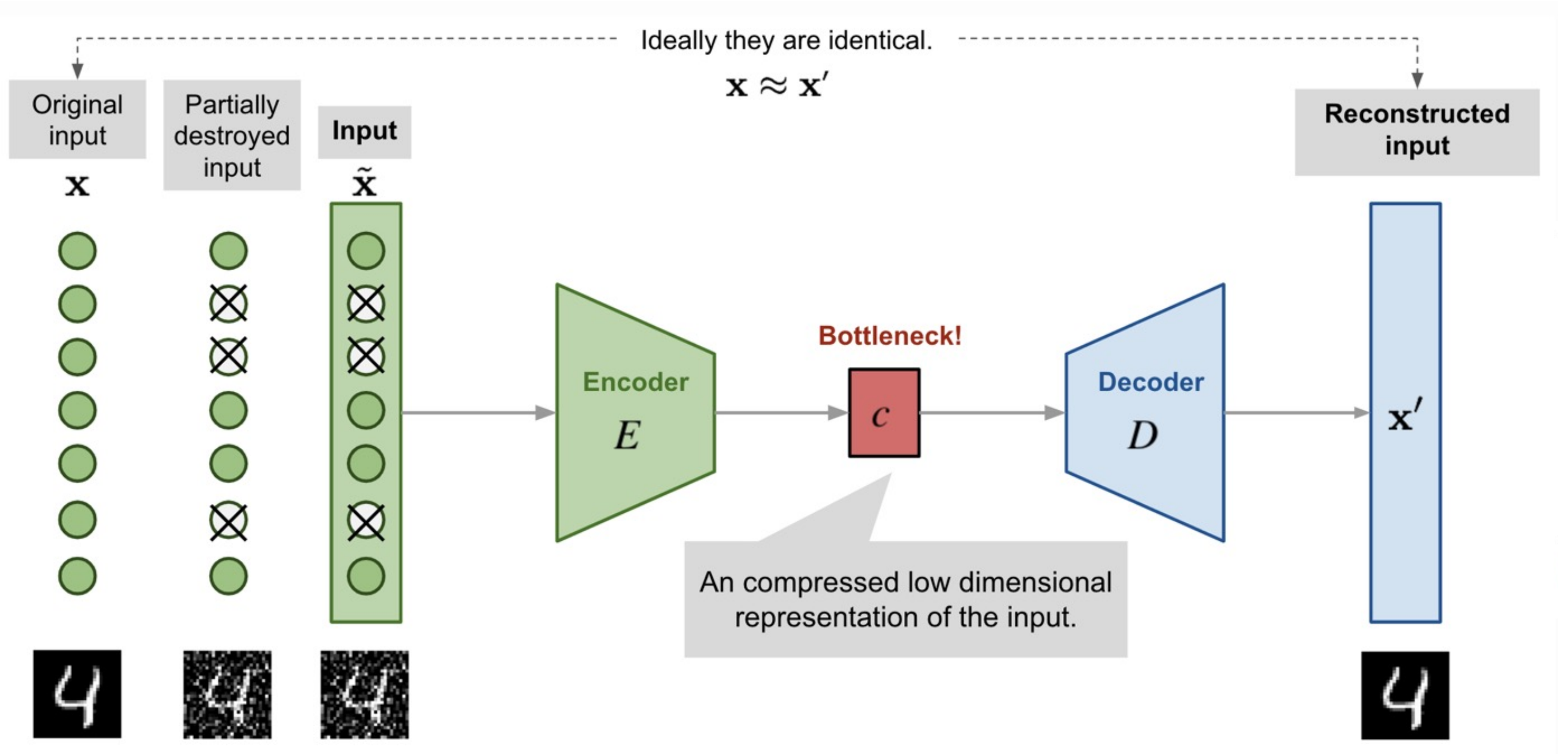


Denoising Autoencoders

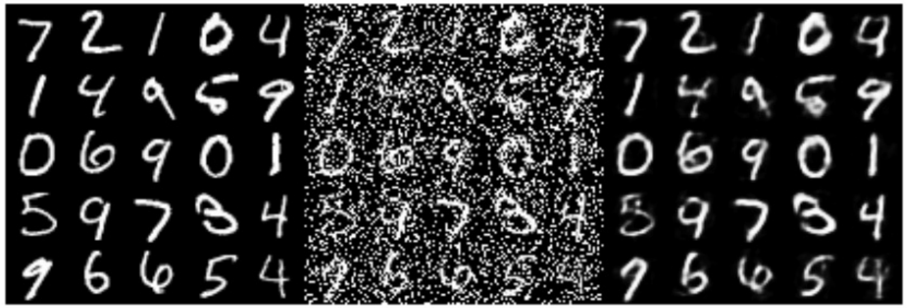
◈What is Denoising Autoencoders (DAE)?

– DAE is a "customized" denoising algorithm tuned to a set of data

- On a specific set of data, DAE is optimized to remove noise from similar data
- For example, if we train it to remove noise from a collection of images, it will work well on similar images but will not be suitable for cleaning text data.



An compressed low dimensional representation of the input.

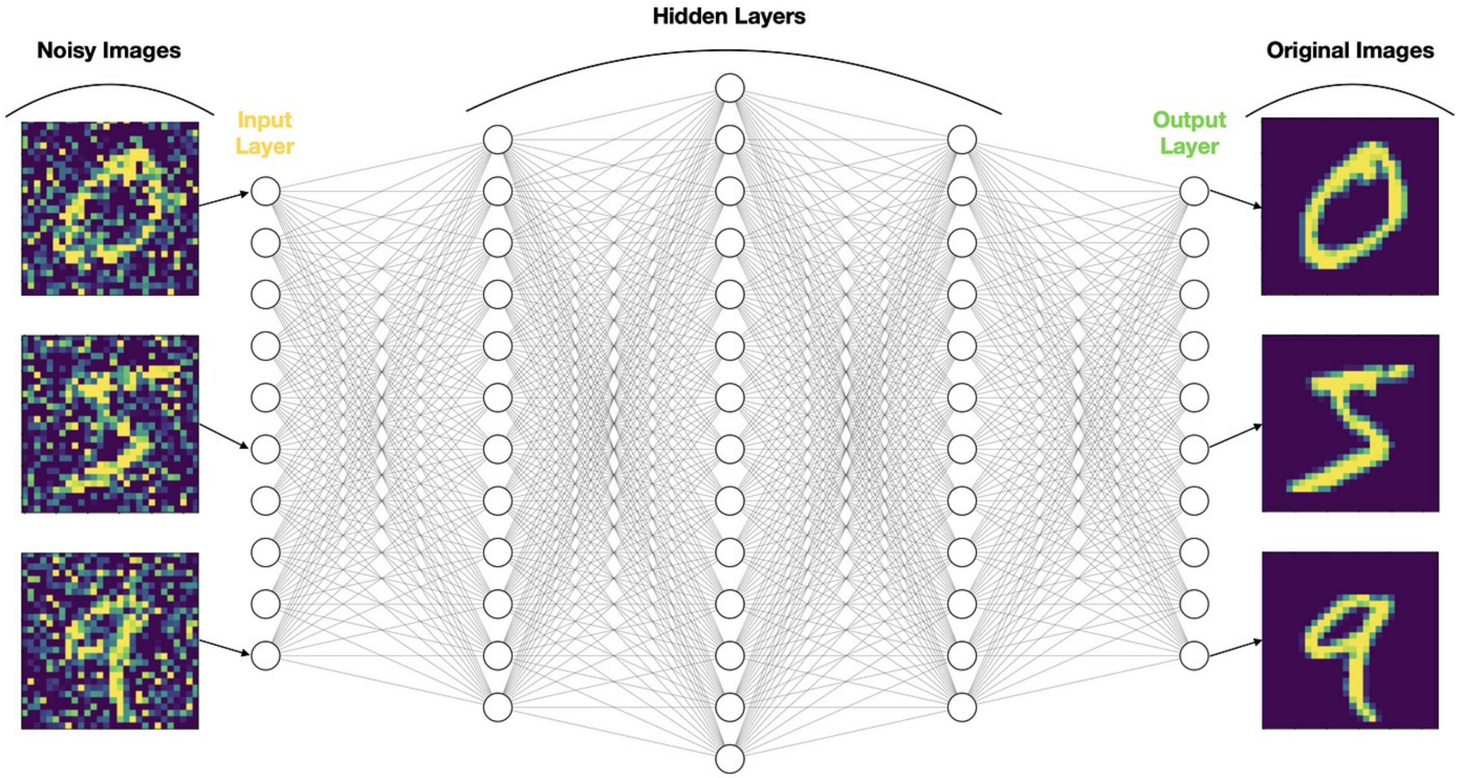


Original input, corrupted data, reconstructed data

Denoising Autoencoders

◈What is Denoising Autoencoders (DAE)?

– Unlike Undercomplete AE, sometimes, we may use the same or higher number of neurons within the hidden layer, making the DAE overcomplete



Denoising Autoencoders

◈What is Denoising Autoencoders (DAE)?

– Encoder: compress input into a latent-space of usually smaller dimension

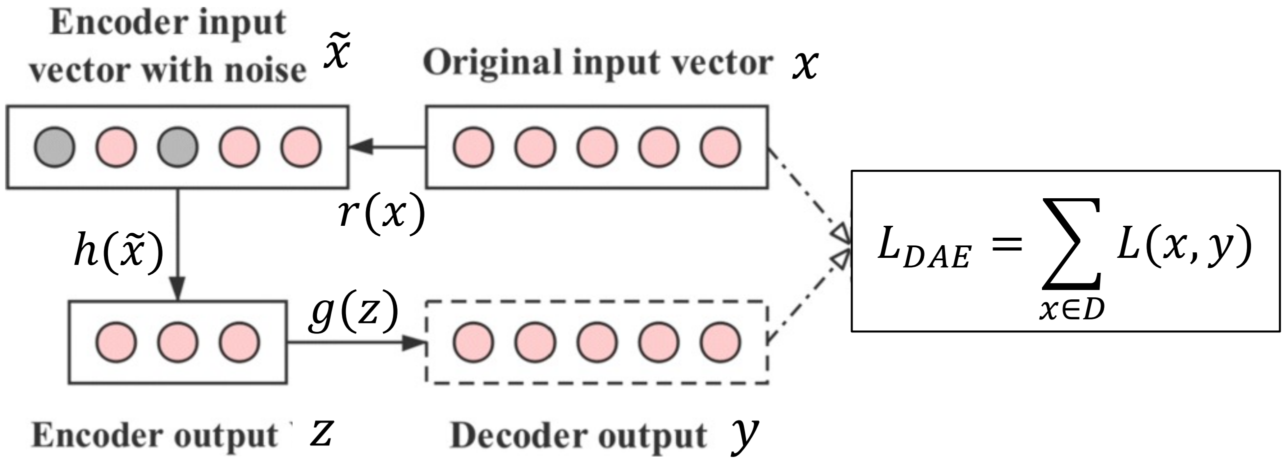
- $\tilde{x} = r(x)$ for a random noise function $r(\cdot)$
- $z = h(\tilde{x}) \in \mathbb{R}^{d_z}$

– Decoder: reconstruct input from the latent space.

- $y = g(z) = g(h(\tilde{x})) = g(h(r(x)))$

– Reconstruction Loss Function

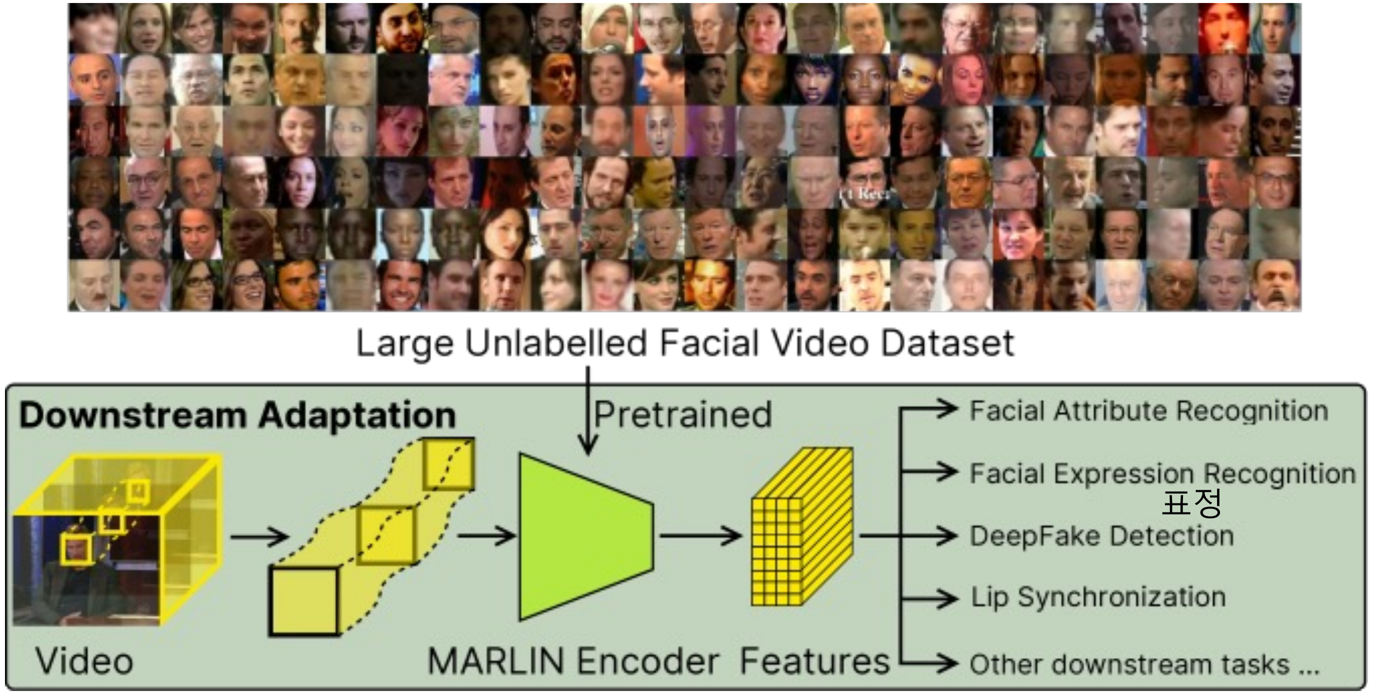
- $L_{DAE} = \sum_{x \in D} L(x, y) = \sum_{x \in D} \|x - y\|^2 = \sum_{x \in D} \|x - g(h(\tilde{x}))\|^2 = \sum_{x \in D} \|x - g(h(r(x)))\|^2$


Encoder input vector with noise $\tilde{x}$
Original input vector x
$r(x)$
$h(\tilde{x})$
$g(z)$
$L_{DAE} = \sum_{x \in D} L(x, y)$
Encoder output z
Decoder output y

Denoising Autoencoders

◈How to use DAE?

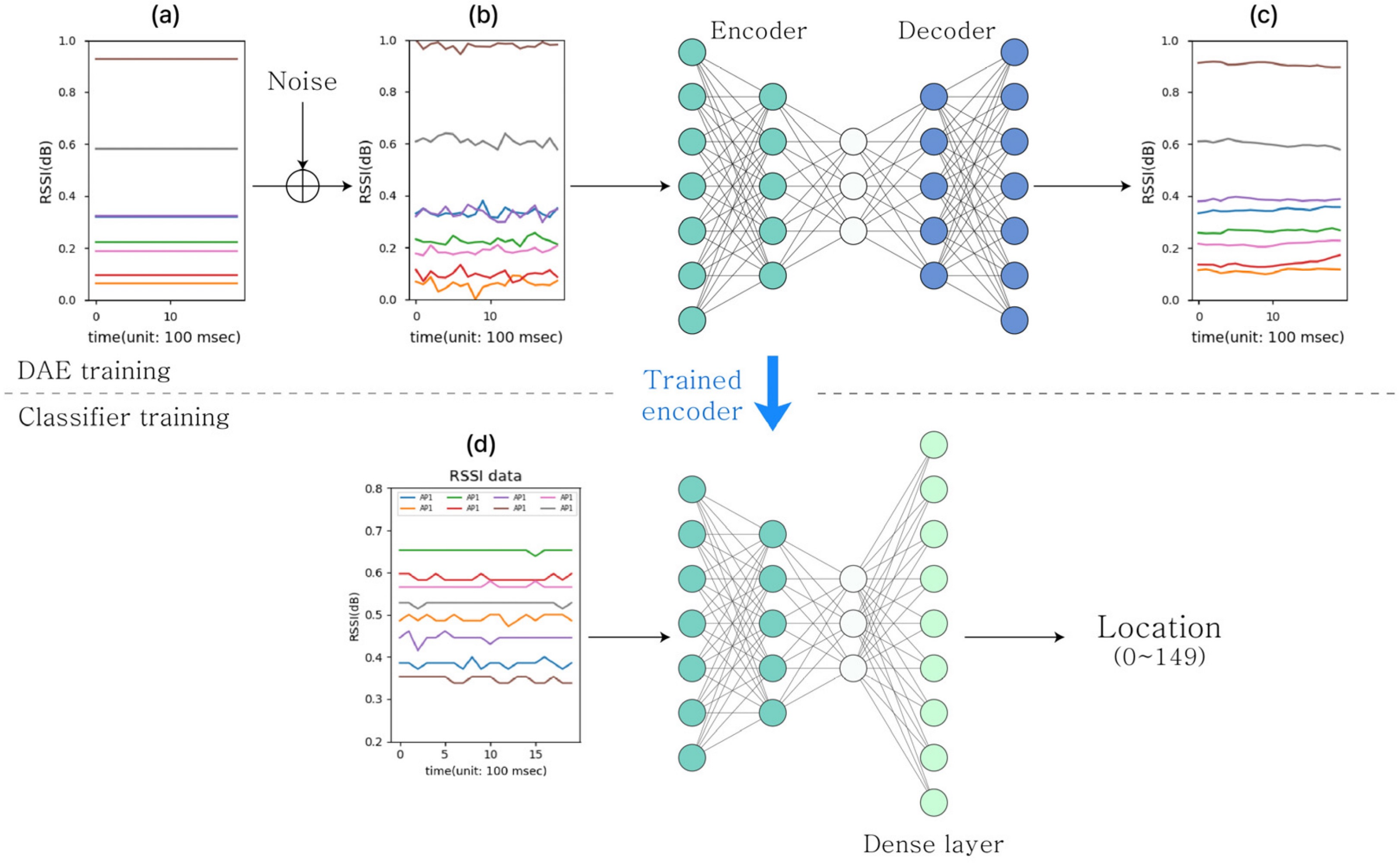
- Noise Reduction
  - DAE is trained to reconstruct a clean or "denoised" version of the input from a corrupted version.
  - This is particularly useful where removing noise from signals is required
- Feature Learning for Downstream Tasks
  - By forcing the network to recover the original signal from a corrupted version, DAE can learn more robust features of the data.
  - These features are often more useful for downstream tasks like classification or anomaly detection
- Improving Robustness of Models


Large Unlabelled Facial Video Dataset
Downstream Adaptation
Pretrained
Video
MARLIN Encoder
Features
Facial Attribute Recognition
Facial Expression Recognition
표정
DeepFake Detection
Lip Synchronization
Other downstream tasks ...

Denoising Autoencoders

◈How to use DAE?

- Noise Reduction
- Feature Learning for Downstream Tasks
- Improving Robustness of Models



Denoising Autoencoder by PyTorch

link_dl/_01_code/_13_autoencoders/c_autoencoder_trainer.py

Denoising Autoencoder with PyTorch

◈Autoencoder Trainer (1/8)

```
class AutoencoderTrainer:
  def __init__(
    self, project_name, model, optimizer, train_data_loader, validation_data_loader, transforms,
    run_time_str, wandb, device, checkpoint_file_path, test_dataset, test_transforms, denoising=True
  ):
    self.project_name = project_name
    self.model = model
    self.optimizer = optimizer
    self.train_data_loader = train_data_loader
    self.validation_data_loader = validation_data_loader
    self.transforms = transforms
    self.run_time_str = run_time_str
    self.wandb = wandb
    self.device = device
    self.checkpoint_file_path = checkpoint_file_path
    self.test_dataset = test_dataset
    self.test_transforms = test_transforms
    self.denoising = denoising

    # Use a built-in loss function
    self.loss_fn = nn.MSELoss()
```

link_dl/_01_code/_13_autoencoders/c_autoencoder_trainer.py

Denoising Autoencoder with PyTorch

◈Autoencoder Trainer (2/8)

```
class AutoencoderTrainer:
  ...

  def add_noise(self, inputs, noise_factor=0.1):
    noisy = inputs + torch.randn(inputs.size()) * noise_factor
    noisy = torch.clip(noisy, 0., 1.)
    return noisy

  def do_train(self):
    self.model.train()

    loss_train = 0.0
    num_trains = 0


    # Iterate the dataloader (we do not need the label values, this is unsupervised learning)
    for train_batch in self.train_data_loader:
      # with "_" we just ignore the target labels
      input_train, _ = train_batch
      if self.denoising is True:
        input_train = self.add_noise(input_train)
      input_train = input_train.to(device=self.device)
```

link_dl/_01_code/_13_autoencoders/c_autoencoder_trainer.py

Denoising Autoencoder with PyTorch

◈Autoencoder Trainer (3/8)

```
class AutoencoderTrainer:
  ...

  def do_train(self):
    ...
    for train_batch in self.train_data_loader:
      ...

      if self.transforms:
        input_train = self.transforms(input_train)
      decoded_input_train = self.model(input_train)
      loss = self.loss_fn(decoded_input_train, input_train)
      loss_train += loss.item()
      num_trains += 1

      self.optimizer.zero_grad()
      loss.backward()
      self.optimizer.step()

    train_loss = loss_train / num_trains
    return train_loss
```

link_dl/_01_code/_13_autoencoders/c_autoencoder_trainer.py

Denoising Autoencoder with PyTorch

◈Autoencoder Trainer (4/8)

```
class AutoencoderTrainer:
  ...

  def do_validation(self):
    self.model.eval()

    loss_validation = 0.0
    num_validations = 0

    with torch.no_grad():
      for validation_batch in self.validation_data_loader:
        input_validation, _ = validation_batch

        if self.denoising is True:
          input_validation = self.add_noise(input_validation)

        input_validation = input_validation.to(device=self.device)

        if self.transforms:
          input_validation = self.transforms(input_validation)
        decoded_input_validation = self.model(input_validation)
```

link_dl/_01_code/_13_autoencoders/c_autoencoder_trainer.py

Denoising Autoencoder with PyTorch

◈Autoencoder Trainer (5/8)

```python
class AutoencoderTrainer:
  ...

  def do_validation(self):
    ...
    with torch.no_grad():
      for validation_batch in self.validation_data_loader:
        ...
        loss_validation += self.loss_fn(decoded_input_validation, input_validation).item()
        num_validations += 1

    validation_loss = loss_validation / num_validations

    return validation_loss

  def plot_denoising_autoencoders_outputs(self, n=10, noise_factor=0.1):
    self.model.eval()

    plt.figure(figsize=(16, 4.5))
    ...
    # refer to the codes in the pycharm
```

link_dl/_01_code/_13_autoencoders/c_autoencoder_trainer.py

Denoising Autoencoder with PyTorch

◈Autoencoder Trainer (6/8)

```
class AutoencoderTrainer:
  ...
  def train_loop(self):
    early_stopping = EarlyStopping(
      patience=self.wandb.config.early_stop_patience,
      delta=self.wandb.config.early_stop_delta,
      project_name=self.project_name,
      checkpoint_file_path=self.checkpoint_file_path,
      run_time_str=self.run_time_str
    )
    n_epochs = self.wandb.config.epochs
    training_start_time = datetime.now()

    for epoch in range(1, n_epochs + 1):
      train_loss = self.do_train()

      if epoch == 1 or epoch % self.wandb.config.validation_intervals == 0:
        validation_loss = self.do_validation()

        elapsed_time = datetime.now() - training_start_time
        epoch_per_second = 1000 * epoch / elapsed_time.microseconds
```

link_dl/_01_code/_13_autoencoders/c_autoencoder_trainer.py

Denoising Autoencoder with PyTorch

◈Autoencoder Trainer (7/8)

```
class AutoencoderTrainer:
  ...
  def train_loop(self):
    ...
    for epoch in range(1, n_epochs + 1):
      ...
      if epoch == 1 or epoch % self.wandb.config.validation_intervals == 0:
        ...
        message, early_stop = early_stopping.check_and_save(validation_loss, self.model)

        print(
          f"[Epoch {epoch:>3}] "
          f"T_loss: {train_loss:7.5f}, "
          f"V_loss: {validation_loss:7.5f}, "
          f"{message} | "
          f"T_time: {strfdelta(elapsed_time, '%H:%M:%S')}, "
          f"T_speed: {epoch_per_second:4.3f}"
        )
```

link_dl/_01_code/_13_autoencoders/c_autoencoder_trainer.py

Denoising Autoencoder with PyTorch

◈Autoencoder Trainer (8/8)

```
class AutoencoderTrainer:
  ...
  def train_loop(self):
    ...
    for epoch in range(1, n_epochs + 1):
      ...
      if epoch == 1 or epoch % self.wandb.config.validation_intervals == 0:
        ...
        self.wandb.log({
          "Epoch": epoch,
          "Training loss": train_loss,
          "Validation loss": validation_loss,
          "Training speed (epochs/sec.)": epoch_per_second,
        })

        self.plot_denoising_autoencoders_outputs(n=10, noise_factor=0.3)
        if early_stop:
          break

    elapsed_time = datetime.now() - training_start_time
    print(f"Final training time: {strfdelta(elapsed_time, '%H:%M:%S')}")
```

link_dl/_01_code/_13_autoencoders/d_denoising_autoencoder_train_and_test.py

Denoising Autoencoder with PyTorch

◇Autoencoder Model (1/5)

```
def get_model(encoded_space_dim=2):

  class Encoder(nn.Module):
    def __init__(self):
      super(Encoder, self).__init__()

      self.encoder = nn.Sequential(
        # B x 1 x 28 x 28 --> B x 32 x (28 - 3 + 2 + 1) x (28 - 3 + 2 + 1) = B x 32 x 28 x 28
        nn.Conv2d(in_channels=1, out_channels=32, kernel_size=3, padding=1, stride=1),
        nn.BatchNorm2d(32),
        nn.LeakyReLU(),

        # B x 32 x 28 x 28 --> B x 64 x (|(28 - 3 + 2) / 2| + 1) x (|(28 - 3 + 2) / 2| + 1) = B x 64 x 14 x 14
        nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, padding=1, stride=2),
        nn.BatchNorm2d(64),
        nn.LeakyReLU(),

        # B x 64 x 14 x 14 --> B x 64 x (|(14 - 3 + 2) / 2| + 1) x (|(28 - 3 + 2) / 2| + 1) = B x 64 x 7 x 7
        nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, padding=1, stride=2),
        nn.BatchNorm2d(64),
        nn.LeakyReLU(),
```

link_dl/_01_code/_13_autoencoders/d_denoising_autoencoder_train_and_test.py

Denoising Autoencoder with PyTorch

◈Autoencoder Model (2/5)

```
def get_model(encoded_space_dim=2):

 class Encoder(nn.Module):
  def __init__(self):
   ...

   self.encoder = nn.Sequential(
    ...
    nn.Flatten(),

    nn.Linear(64 * 7 * 7, encoded_space_dim),
    nn.LeakyReLU(),
   )

  def forward(self, x):
   x = self.encoder(x)
   return x
```

link_dl/_01_code/_13_autoencoders/d_denoising_autoencoder_train_and_test.py

Denoising Autoencoder with PyTorch

◈Autoencoder Model (3/5)

```
def get_model(encoded_space_dim=2):
  ...

  class Decoder(nn.Module):
    def __init__(self):
      super(Decoder, self).__init__()

      self.decoder = nn.Sequential(
        nn.Linear(encoded_space_dim, 64 * 7 * 7),
        nn.Unflatten(1, (64, 7, 7)),
        nn.LeakyReLU(),

        # B x 64 x 7 x 7 --> B x 64 x ((7 - 1) x 2 - 2 x 1 + 3 + 1) x ((7 - 1) x 2 - 2 x 1 + 3 + 1) =
        # B x 64 x (12 - 2 + 3 + 1) x (12 - 2 + 3 + 1) = B x 64 x 14 x 14
        nn.ConvTranspose2d(
          in_channels=64, out_channels=64, kernel_size=3, padding=1, stride=2, output_padding=1
        ),
        nn.BatchNorm2d(64),
        nn.LeakyReLU(),
```

link_dl/_01_code/_13_autoencoders/d_denoising_autoencoder_train_and_test.py

Denoising Autoencoder with PyTorch

◈Autoencoder Model (4/5)

```
def get_model(encoded_space_dim=2):
  ...

  class Decoder(nn.Module):
    def __init__(self):
      ...
      self.decoder = nn.Sequential(
        ...
        # B x 64 x 14 x 14 --> B x 32 x ((14 - 1) x 2 - 2 x 1 + 3 + 1) x ((15 - 1) x 2 - 2 x 1 + 3 + 1) =
        # B x 32 x (26 - 2 + 3 + 1) x (26 - 2 + 3 + 1) = B x 32 x 28 x 28
        nn.ConvTranspose2d(
          in_channels=64, out_channels=32, kernel_size=3, padding=1, stride=2, output_padding=1
        ),
        nn.BatchNorm2d(32),
        nn.LeakyReLU(),

        # B x 32 x 28 x 28 --> B x 1 x ((28 - 1) x 1 - 2 x 1 + 3 + 0) x ((28 - 1) x 1 - 2 x 1 + 3 + 0) =
        # B x 1 x (27 - 2 + 3) x (27 - 2 + 3) = B x 1 x 28 x 28
        nn.ConvTranspose2d(in_channels=32, out_channels=1, kernel_size=3, padding=1),
        nn.Sigmoid()
      )
```

link_dl/_01_code/_13_autoencoders/d_denoising_autoencoder_train_and_test.py

Denoising Autoencoder with PyTorch

◈Autoencoder Model (5/5)

```
def get_model(encoded_space_dim=2):
  ...

  class Decoder(nn.Module):
    ...
    def forward(self, x):
      x = self.decoder(x)
      return x

  class Autoencoder(torch.nn.Module):
    def __init__(self):
      super(Autoencoder, self).__init__()
      self.encoder = Encoder()
      self.decoder = Decoder()

    def forward(self, x):
      x = self.encoder(x)  # x.shape: [B, 256, 7, 7]
      x = self.decoder(x)
      return x

  autoencoder = Autoencoder()
  return autoencoder
```

[Appendix]
Image Upsampling &
Transposed Convolution

[Source] https://arxiv.org/pdf/1603.07285.pdf

Image Upsampling

- Image Upsampling (= upsizing)
  - The image resolution increasing
    - Number of pixels in the given image is increased
      - 24 megapixel image to 48 megapixels, 96 megapixels, or 240 megapixels
  - Basic Upsampling
    - Nearest Neighbors

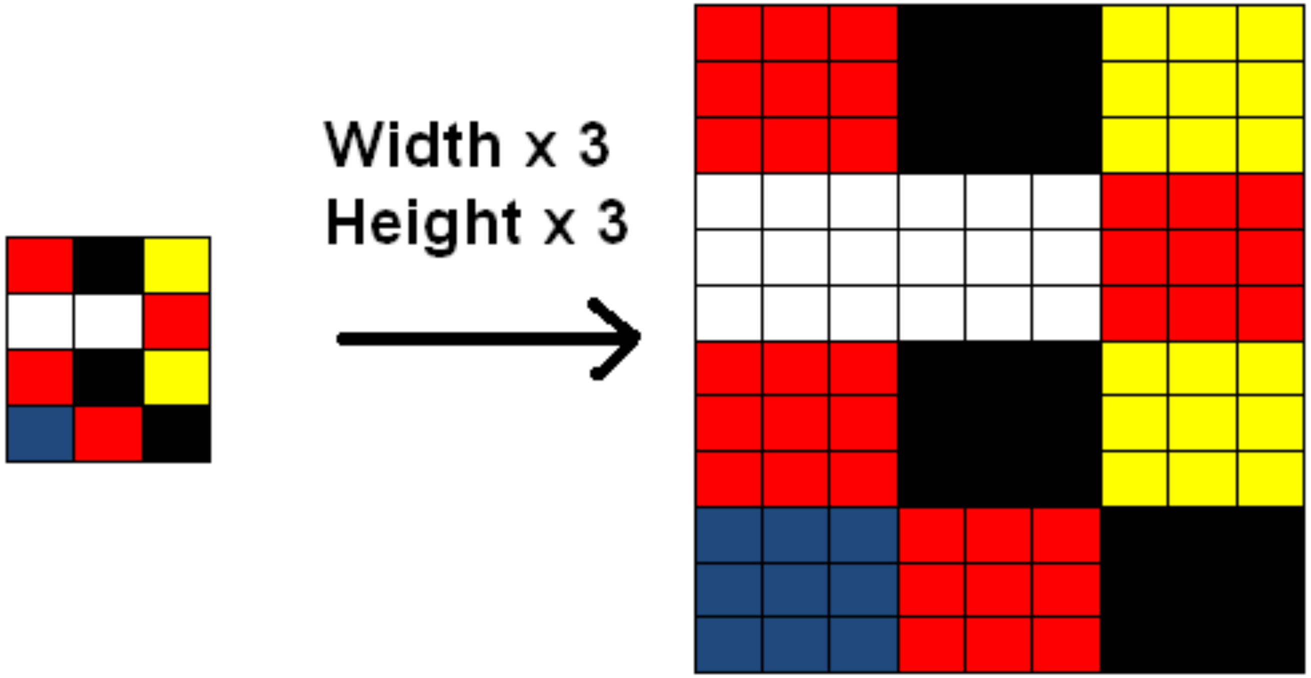
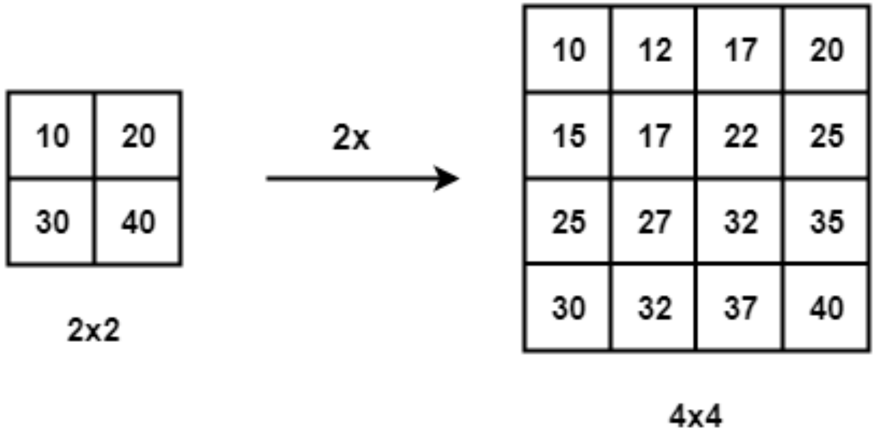


Image Upsampling

◈Conventional Image Upsampling

- – Nearest Neighbor
- – Bilinear Interpolation



10	20
30	40

2x2

10	12	17	20
15	17	22	25
25	27	32	35
30	32	37	40

4x4

- – Bicubic Spline Interpolation
- – Generalized Bicubic Interpolation

Image Upsampling

◈2D Transposed Convolution

- Also known as deconvolution or fractionally-strided convolution
- It increases the spatial dimensions of input image through learnable parameters for the specified purpose
  - Like standard convolution, it has parameters like kernel size, stride, and padding, but they are used in a way to expand rather than reduce the dimensions of the input
- Usage
  - Upsample the feature maps, such as in the decoder part of an autoencoder or in the generator of a GAN

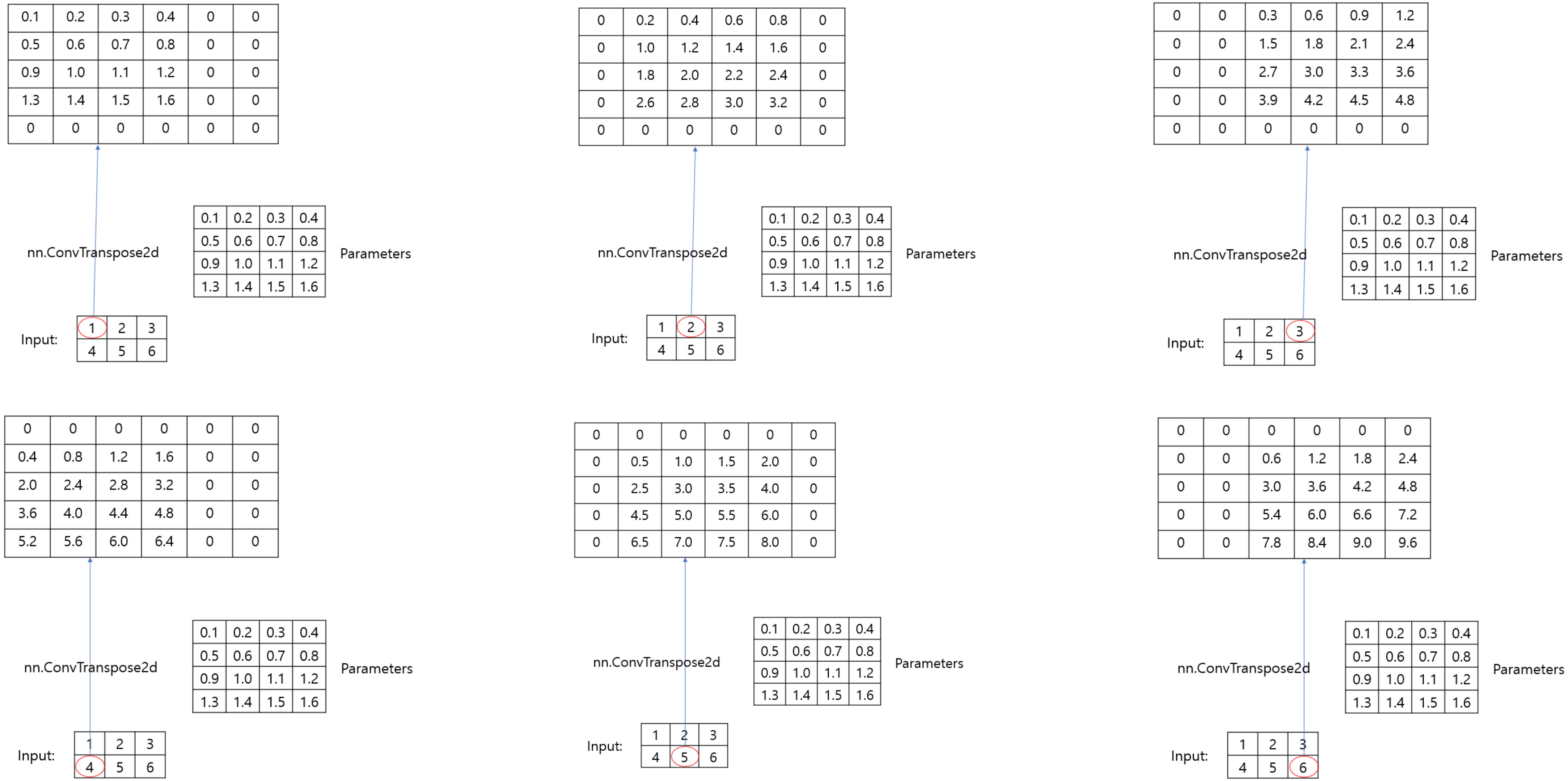
◈How does it work?

- It maps each input pixel to a higher dimensional space using a kernel (filter)
- This process involves inserting zeros (known as up-sampling or up-striding) between the pixels of the input feature map, and then applying a standard convolution

[Source] https://cumulu-s.tistory.com/29

Transposed Convolution

◈nn.ConvTranspose2d (stride=1, padding=0): Operation Details for a Channel



0.1	0.2	0.3	0.4	0	0
0.5	0.6	0.7	0.8	0	0
0.9	1.0	1.1	1.2	0	0
1.3	1.4	1.5	1.6	0	0
0	0	0	0	0	0

nn.ConvTranspose2d

0.1	0.2	0.3	0.4
0.5	0.6	0.7	0.8
0.9	1.0	1.1	1.2
1.3	1.4	1.5	1.6

Parameters

Input:

1	2	3
4	5	6

0	0.2	0.4	0.6	0.8	0
0	1.0	1.2	1.4	1.6	0
0	1.8	2.0	2.2	2.4	0
0	2.6	2.8	3.0	3.2	0
0	0	0	0	0	0

nn.ConvTranspose2d

0.1	0.2	0.3	0.4
0.5	0.6	0.7	0.8
0.9	1.0	1.1	1.2
1.3	1.4	1.5	1.6

Parameters

Input:

1	2	3
4	5	6

0	0	0.3	0.6	0.9	1.2
0	0	1.5	1.8	2.1	2.4
0	0	2.7	3.0	3.3	3.6
0	0	3.9	4.2	4.5	4.8
0	0	0	0	0	0

nn.ConvTranspose2d

0.1	0.2	0.3	0.4
0.5	0.6	0.7	0.8
0.9	1.0	1.1	1.2
1.3	1.4	1.5	1.6

Parameters

Input:

1	2	3
4	5	6

0	0	0	0	0	0
0.4	0.8	1.2	1.6	0	0
2.0	2.4	2.8	3.2	0	0
3.6	4.0	4.4	4.8	0	0
5.2	5.6	6.0	6.4	0	0

nn.ConvTranspose2d

0.1	0.2	0.3	0.4
0.5	0.6	0.7	0.8
0.9	1.0	1.1	1.2
1.3	1.4	1.5	1.6

Parameters

Input:

1	2	3
4	5	6

0	0	0	0	0	0
0	0.5	1.0	1.5	2.0	0
0	2.5	3.0	3.5	4.0	0
0	4.5	5.0	5.5	6.0	0
0	6.5	7.0	7.5	8.0	0

nn.ConvTranspose2d

0.1	0.2	0.3	0.4
0.5	0.6	0.7	0.8
0.9	1.0	1.1	1.2
1.3	1.4	1.5	1.6

Parameters

Input:

1	2	3
4	5	6

0	0	0	0	0	0
0	0	0.6	1.2	1.8	2.4
0	0	3.0	3.6	4.2	4.8
0	0	5.4	6.0	6.6	7.2
0	0	7.8	8.4	9.0	9.6

nn.ConvTranspose2d

0.1	0.2	0.3	0.4
0.5	0.6	0.7	0.8
0.9	1.0	1.1	1.2
1.3	1.4	1.5	1.6

Parameters

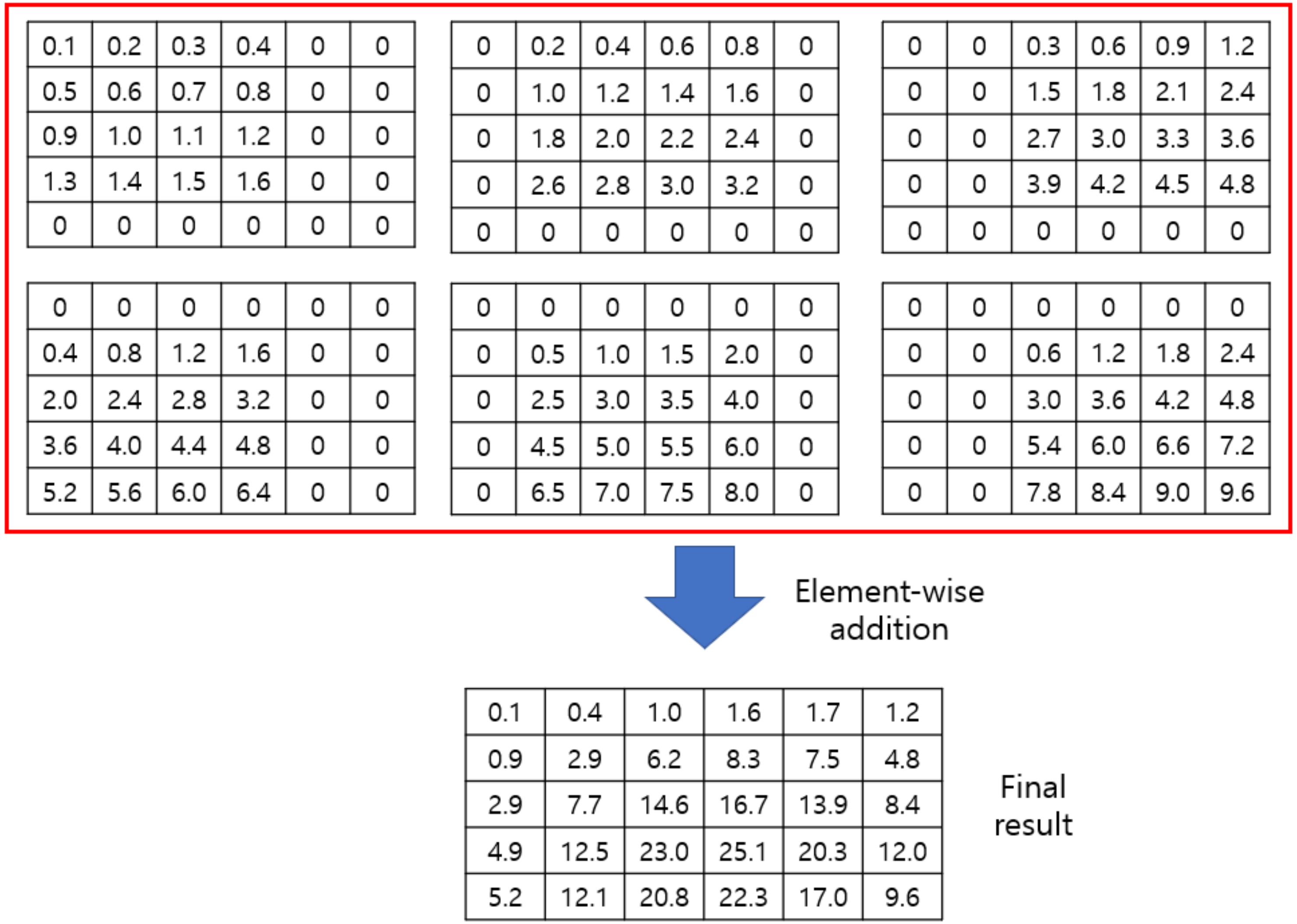
Input:

1	2	3
4	5	6

[Source] https://cumulu-s.tistory.com/29

Transposed Convolution

◈nn.ConvTranspose2d (stride=1, padding=0): Operation Details for a Channel



0.1	0.2	0.3	0.4	0	0
0.5	0.6	0.7	0.8	0	0
0.9	1.0	1.1	1.2	0	0
1.3	1.4	1.5	1.6	0	0
0	0	0	0	0	0

0	0.2	0.4	0.6	0.8	0
0	1.0	1.2	1.4	1.6	0
0	1.8	2.0	2.2	2.4	0
0	2.6	2.8	3.0	3.2	0
0	0	0	0	0	0

0	0	0.3	0.6	0.9	1.2
0	0	1.5	1.8	2.1	2.4
0	0	2.7	3.0	3.3	3.6
0	0	3.9	4.2	4.5	4.8
0	0	0	0	0	0

0	0	0	0	0	0
0.4	0.8	1.2	1.6	0	0
2.0	2.4	2.8	3.2	0	0
3.6	4.0	4.4	4.8	0	0
5.2	5.6	6.0	6.4	0	0

0	0	0	0	0	0
0	0.5	1.0	1.5	2.0	0
0	2.5	3.0	3.5	4.0	0
0	4.5	5.0	5.5	6.0	0
0	6.5	7.0	7.5	8.0	0

0	0	0	0	0	0
0	0	0.6	1.2	1.8	2.4
0	0	3.0	3.6	4.2	4.8
0	0	5.4	6.0	6.6	7.2
0	0	7.8	8.4	9.0	9.6

Results

Element-wise addition

0.1	0.4	1.0	1.6	1.7	1.2
0.9	2.9	6.2	8.3	7.5	4.8
2.9	7.7	14.6	16.7	13.9	8.4
4.9	12.5	23.0	25.1	20.3	12.0
5.2	12.1	20.8	22.3	17.0	9.6

Final result

Transposed Convolution

2D Transposed Convolution

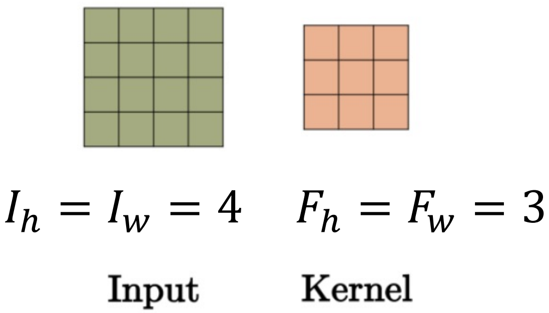
- Given
  - I_h, I_w: Input Size (C Channels)
  - F_h, F_w: Filter Size (K Filters)
  - P_h, P_w: Padding Size
  - S_h, S_w: Stride Size
  - V_h, V_w: Out Padding Size
    - Out Padding
      - » an additional padding added to one side of the output
- Output
  - O_h, O_w: Output Size (K Channels)
- In many cases,
  - $F_h = F_w = F, \ I_h = I_w = I$
  - $P_h = P_w = P, \ S_h = S_w = S, V_h = V_w = V$
  - $O_h = O_w = O$

Transposed Convolution

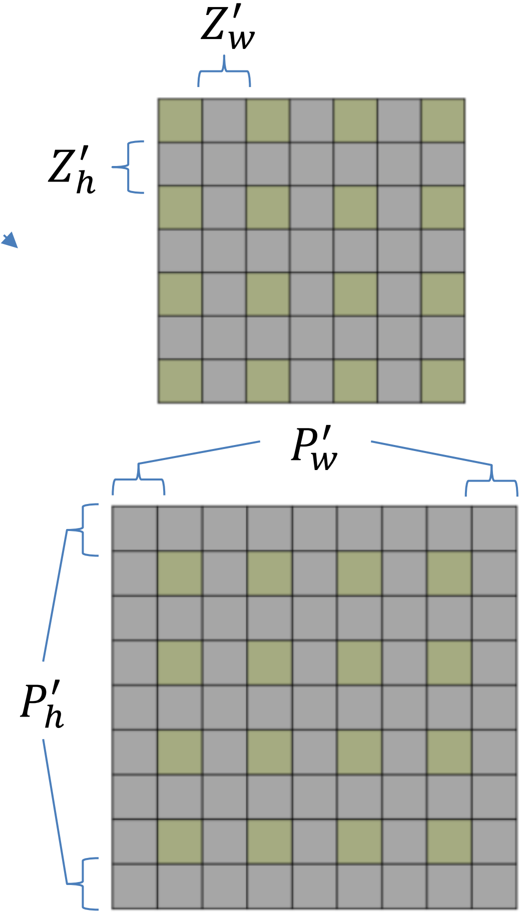
$P_h = P_w = 1$
$S_h = S_w = 2$
$V_h = V_w = 0$

$$\boxed{\begin{array}{l} Z'_h = Z'_w = 2 - 1 = 1 \\ P'_h = P'_w = 3 - 1 - 1 = 1 \end{array}}$$

◈2D Transposed Convolution



- Step 1.
  - $Z'_h = S_h - 1, Z'_w = S_w - 1$
  - $P'_h = F_h - P_h - 1, P'_w = F_w - P_w - 1$
- Step 2.
  - <u>inserting (Z'_h, Z'_w) zeros between the pixels of the input feature map</u>
- Step 3.
  - <u>pad $(P'_h,\ P'_h)$ zeros to the side of the changed input feature map</u>



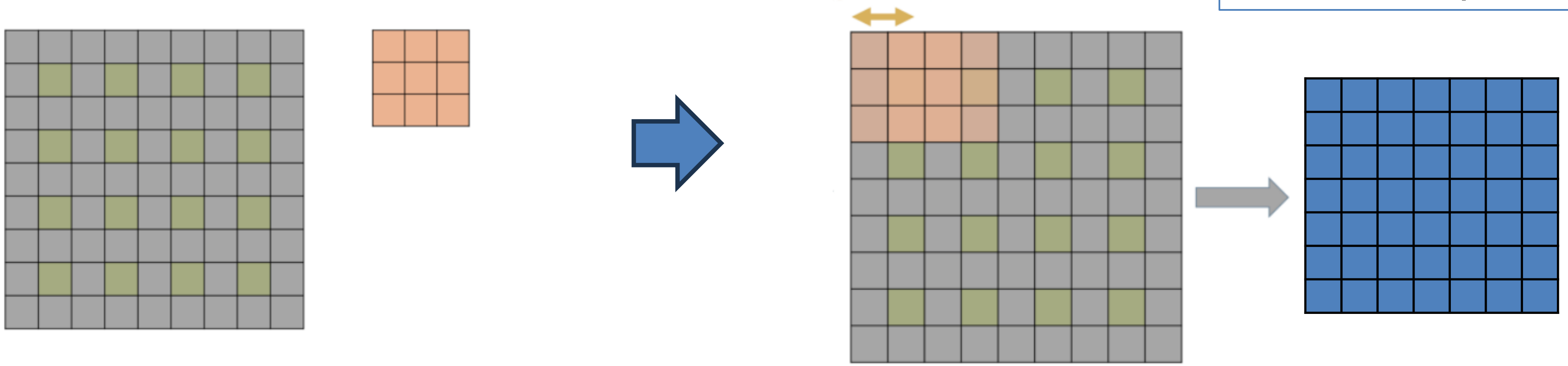
Transposed Convolution

$$O_h = (I_h - 1)\times S_h - 2\times P_h + F_h + V_h$$
$$O_w = (I_w - 1)\times S_w - 2\times P_w + F_w + V_w$$

◈2D Transposed Convolution

– Step 4.

- Convolve the filters into the changed input feature map
- ALWAYS stride = 1

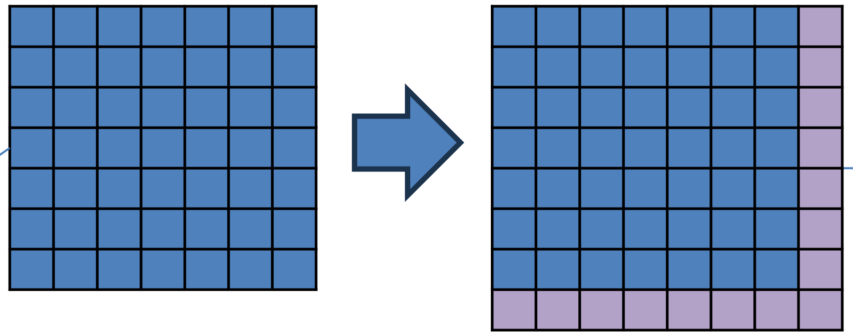


Final Output Size

– Step 5.

- If the "Out Padding Size" V_h and V_w are above zeros, pad (V_h, V_w) zeros into the output feature map

$$O_h = (4-1)\times 2 - 2\times 1 + 3 + 0 = 7$$
$$O_w = (4-1)\times 2 - 2\times 1 + 3 + 0 = 7$$



Transposed Convolution

◈2D Transposed Convolution

– Example 1 $P_h = P_w = 0$
$S_h = S_w = 1$
$V_h = V_w = 1$

$$O_h = (I_h - 1) \times S_h - 2 \times P_h + F_h + V_h$$
$$O_w = (I_w - 1) \times S_w - 2 \times P_w + F_w + V_w$$

0.1	0.2	0.3	0.4	0	0
0.5	0.6	0.7	0.8	0	0
0.9	1.0	1.1	1.2	0	0
1.3	1.4	1.5	1.6	0	0
0	0	0	0	0	0

0.1	0.4	1.0	1.6	1.7	1.2
0.9	2.9	6.2	8.3	7.5	4.8
2.9	7.7	14.6	16.7	13.9	8.4
4.9	12.5	23.0	25.1	20.3	12.0
5.2	12.1	20.8	22.3	17.0	9.6

Final result

nn.ConvTranspose2d

0.1	0.2	0.3	0.4
0.5	0.6	0.7	0.8
0.9	1.0	1.1	1.2
1.3	1.4	1.5	1.6

Parameters

Input:

1	2	3
4	5	6

$$O_h = (2 - 1) \times 1 - 2 \times 0 + 4 + 0 = 5$$
$$O_w = (3 - 1) \times 1 - 2 \times 0 + 4 + 0 = 6$$

$I_h = 2, I_w = 3$ $F_h = F_w = 4$

$$Z'_h = S_h - 1 = 0, Z'_w = S_w - 1 = 0$$
$$P'_h = F_h - P_h - 1 = 4 - 0 - 1 = 3,$$
$$P'_w = F_w - P_w - 1 = 4 - 0 - 1 = 3$$

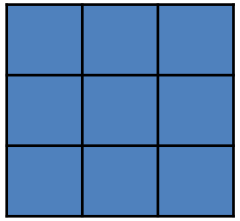
Transposed Convolution

◈2D Transposed Convolution

– Example 2

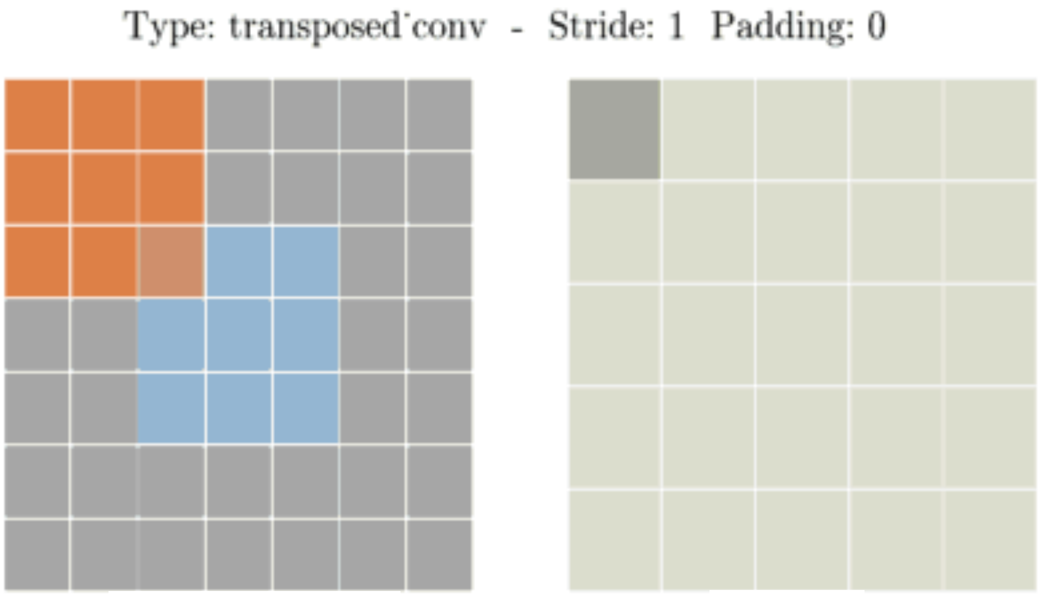
$I_h = I_w = 3$ $F_h = F_w = 3$

$P_h = P_w = 0$
$S_h = S_w = 1$
$V_h = V_w = 0$



Input

$Z'_h = S_h - 1 = 0, Z'_w = S_w - 1 = 0$
$P'_h = F_h - P_h - 1 = 3 - 0 - 1 = 2,$
$P'_w = F_w - P_w - 1 = 3 - 0 - 1 = 2$



Changed Input

Output

$$O_h = (I_h - 1)\times S_h - 2\times P_h + F_h + V_h$$
$$O_w = (I_w - 1)\times S_w - 2\times P_w + F_w + V_w$$

$$O_h = (3 - 1)\times 1 - 2\times 0 + 3 + 0 = 5$$
$$O_w = (3 - 1)\times 1 - 2\times 0 + 3 + 0 = 5$$

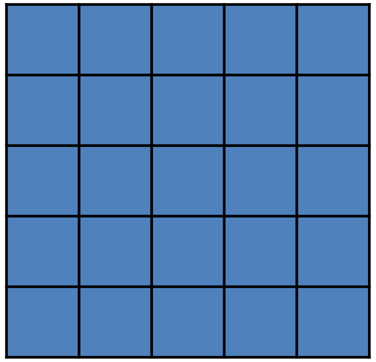
Transposed Convolution

◈2D Transposed Convolution

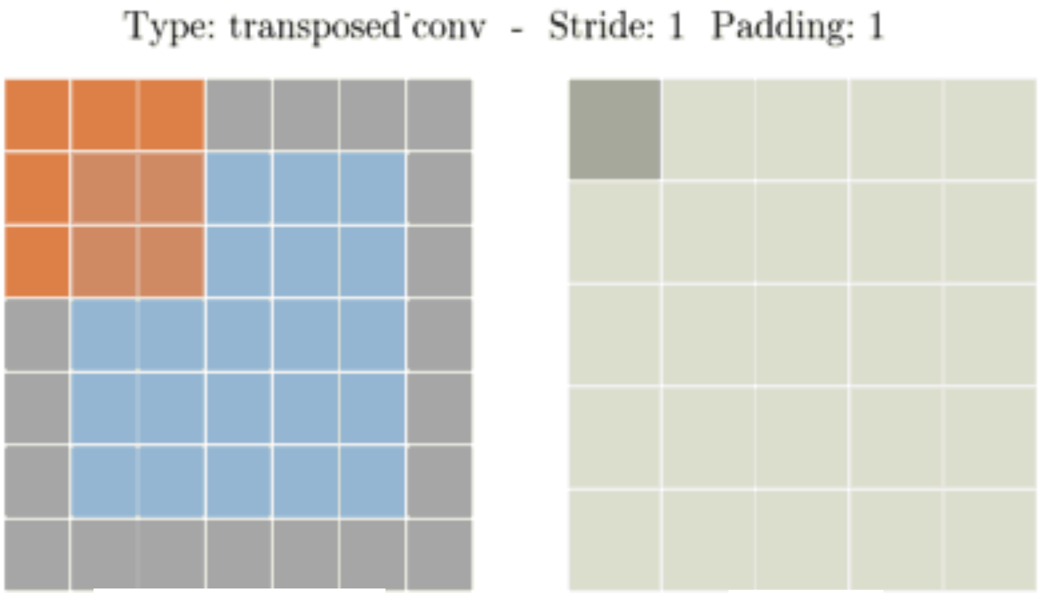
– Example 3

$I_h = I_w = 5$ $F_h = F_w = 3$

$P_h = P_w = 1$
$S_h = S_w = 1$
$V_h = V_w = 0$



Input



Changed Input

Output

$$Z'_h = S_h - 1 = 0, Z'_w = S_w - 1 = 0$$
$$P'_h = F_h - P_h - 1 = 3 - 1 - 1 = 1,$$
$$P'_w = F_w - P_w - 1 = 3 - 1 - 1 = 1$$

$$O_h = (I_h - 1)\times S_h - 2\times P_h + F_h + V_h$$
$$O_w = (I_w - 1)\times S_w - 2\times P_w + F_w + V_w$$

$$O_h = (5 - 1)\times 1 - 2\times 1 + 3 + 0 = 5$$
$$O_w = (5 - 1)\times 1 - 2\times 1 + 3 + 0 = 5$$

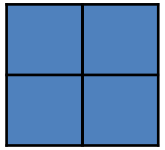
Transposed Convolution

◈2D Transposed Convolution

– Example 4

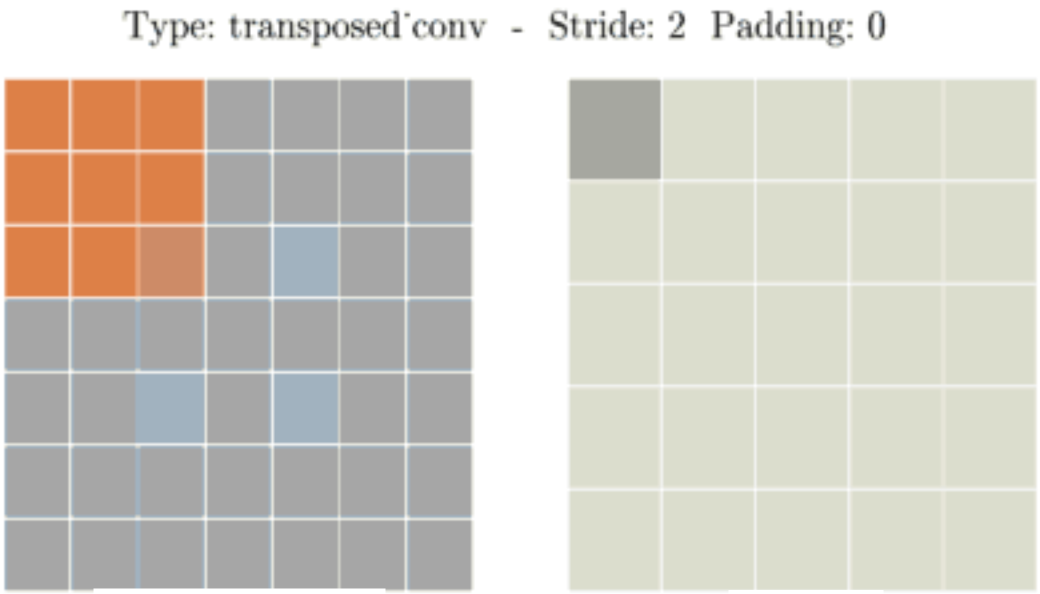
$I_h = I_w = 2$ $\qquad F_h = F_w = 3$

$P_h = P_w = 0$
$S_h = S_w = 2$
$V_h = V_w = 0$



Input

$$Z'_h = S_h - 1 = 1, Z'_w = S_w - 1 = 1$$
$$P'_h = F_h - P_h - 1 = 3 - 0 - 1 = 2,$$
$$P'_w = F_w - P_w - 1 = 3 - 0 - 1 = 2$$



Changed Input

Output

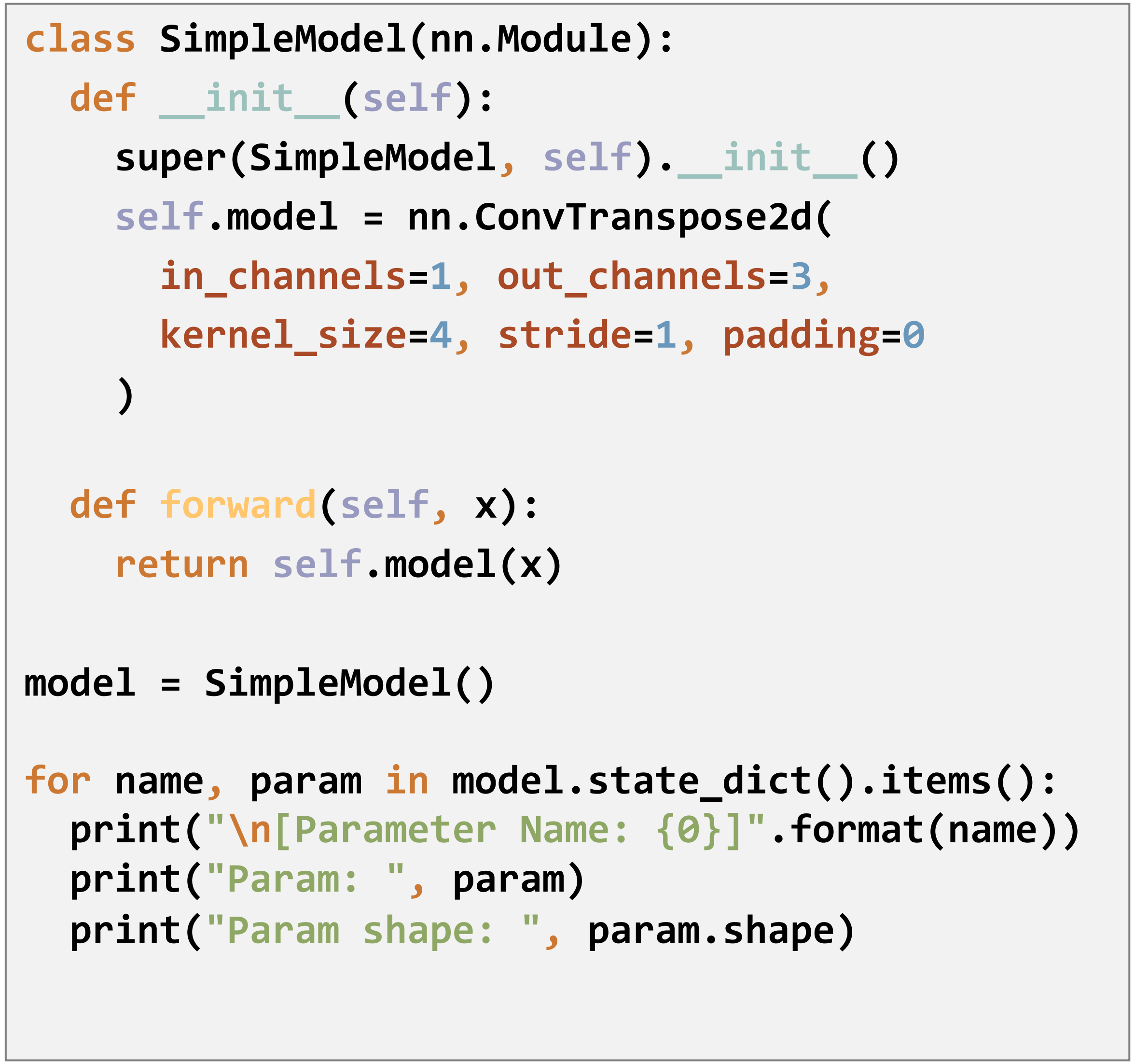
$$O_h = (I_h - 1) \times S_h - 2 \times P_h + F_h + V_h$$
$$O_w = (I_w - 1) \times S_w - 2 \times P_w + F_w + V_w$$

$$O_h = (2 - 1) \times 2 - 2 \times 0 + 3 + 0 = 5$$
$$O_w = (2 - 1) \times 2 - 2 \times 0 + 3 + 0 = 5$$

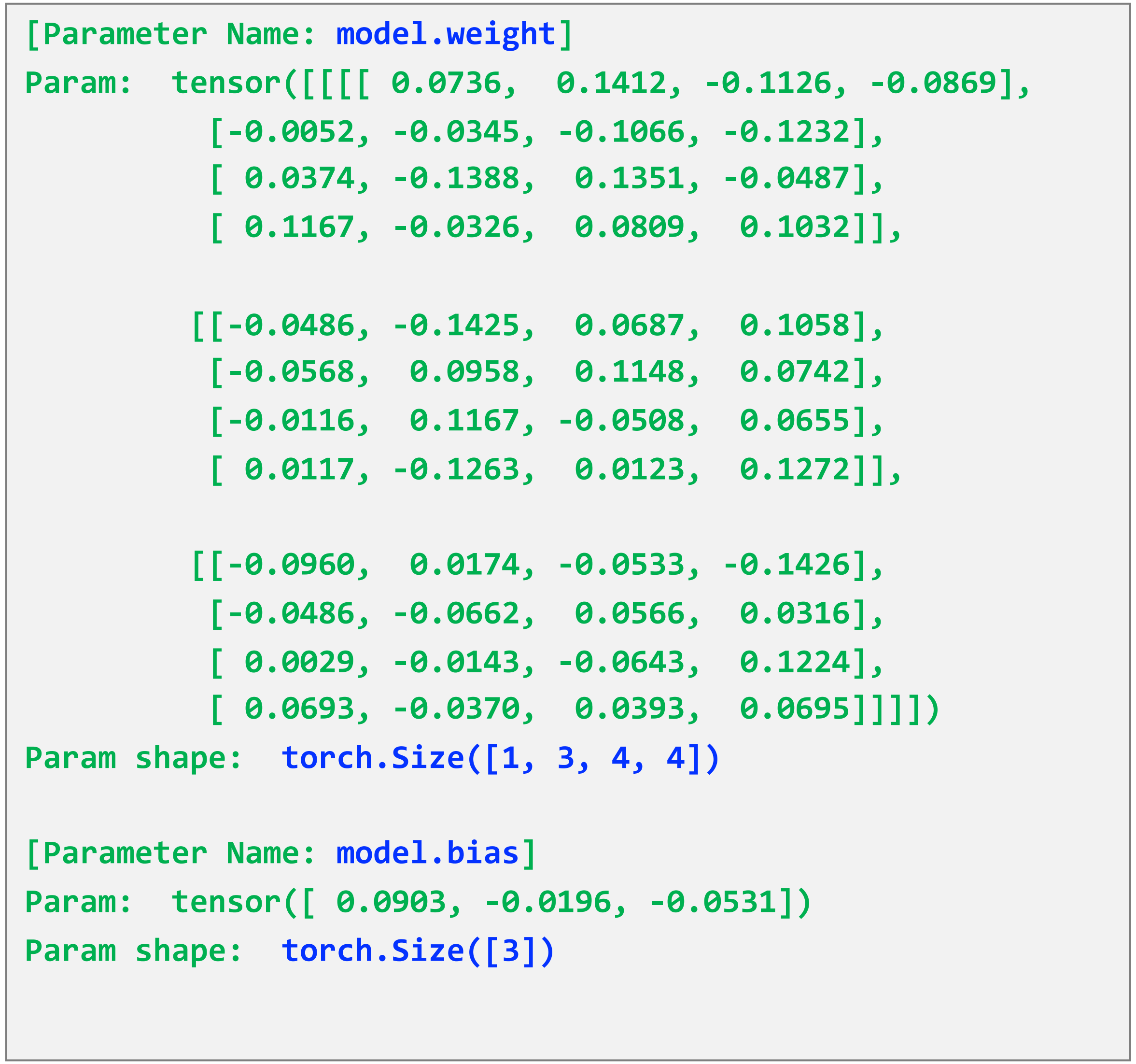
[Source] https://cumulu-s.tistory.com/29

Transposed Convolution

◈2D Transposed Convolution - nn.ConvTranspose2d



```
class SimpleModel(nn.Module):
  def __init__(self):
    super(SimpleModel, self).__init__()
    self.model = nn.ConvTranspose2d(
      in_channels=1, out_channels=3,
      kernel_size=4, stride=1, padding=0
    )

  def forward(self, x):
    return self.model(x)

model = SimpleModel()

for name, param in model.state_dict().items():
  print("\n[Parameter Name: {0}]".format(name))
  print("Param: ", param)
  print("Param shape: ", param.shape)
```

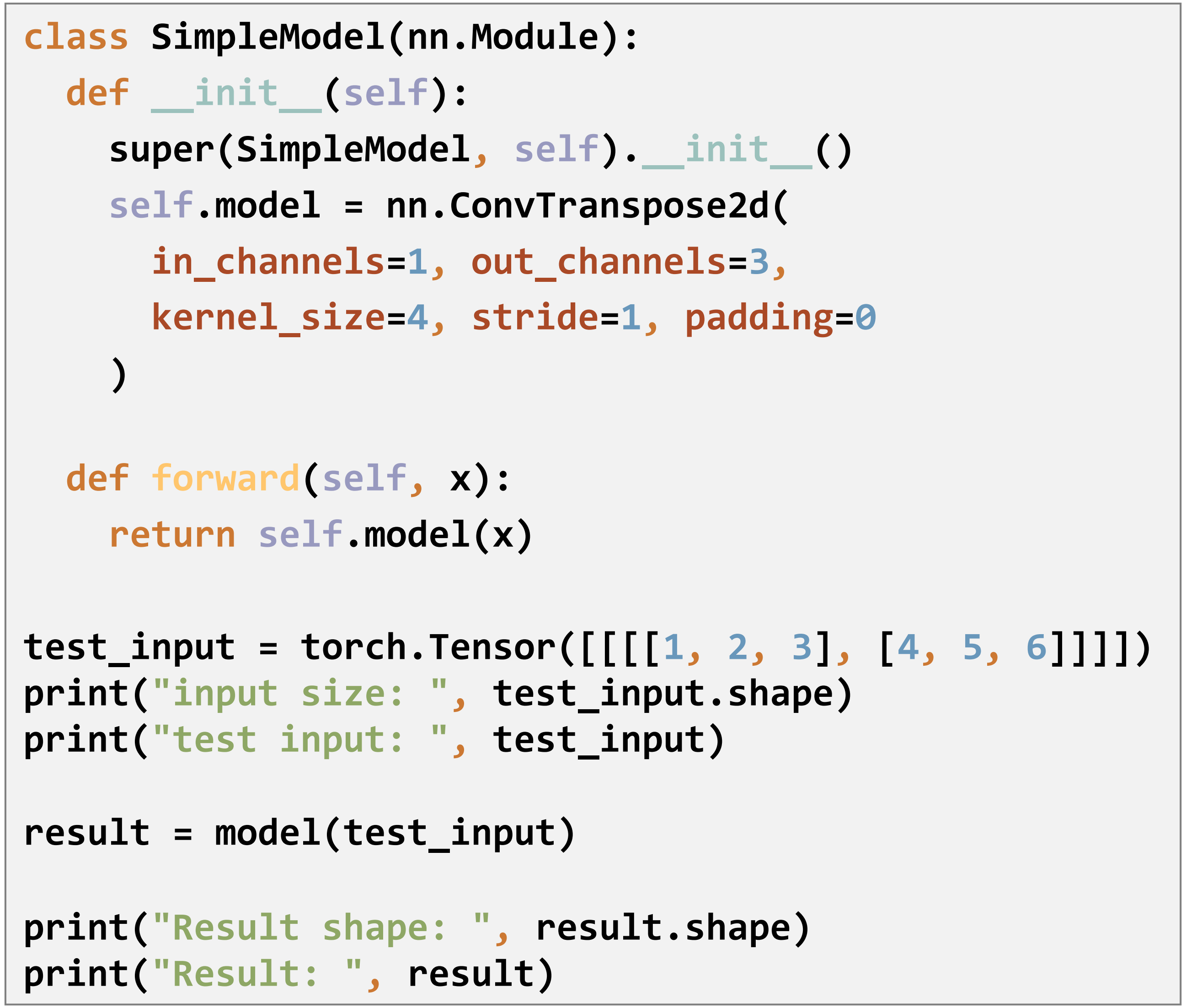


```
[Parameter Name: model.weight]
Param:  tensor([[[[ 0.0736,  0.1412, -0.1126, -0.0869],
          [-0.0052, -0.0345, -0.1066, -0.1232],
          [ 0.0374, -0.1388,  0.1351, -0.0487],
          [ 0.1167, -0.0326,  0.0809,  0.1032]],

         [[-0.0486, -0.1425,  0.0687,  0.1058],
          [-0.0568,  0.0958,  0.1148,  0.0742],
          [-0.0116,  0.1167, -0.0508,  0.0655],
          [ 0.0117, -0.1263,  0.0123,  0.1272]],

         [[-0.0960,  0.0174, -0.0533, -0.1426],
          [-0.0486, -0.0662,  0.0566,  0.0316],
          [ 0.0029, -0.0143, -0.0643,  0.1224],
          [ 0.0693, -0.0370,  0.0393,  0.0695]]]])
Param shape:  torch.Size([1, 3, 4, 4])

[Parameter Name: model.bias]
Param:  tensor([ 0.0903, -0.0196, -0.0531])
Param shape:  torch.Size([3])
```

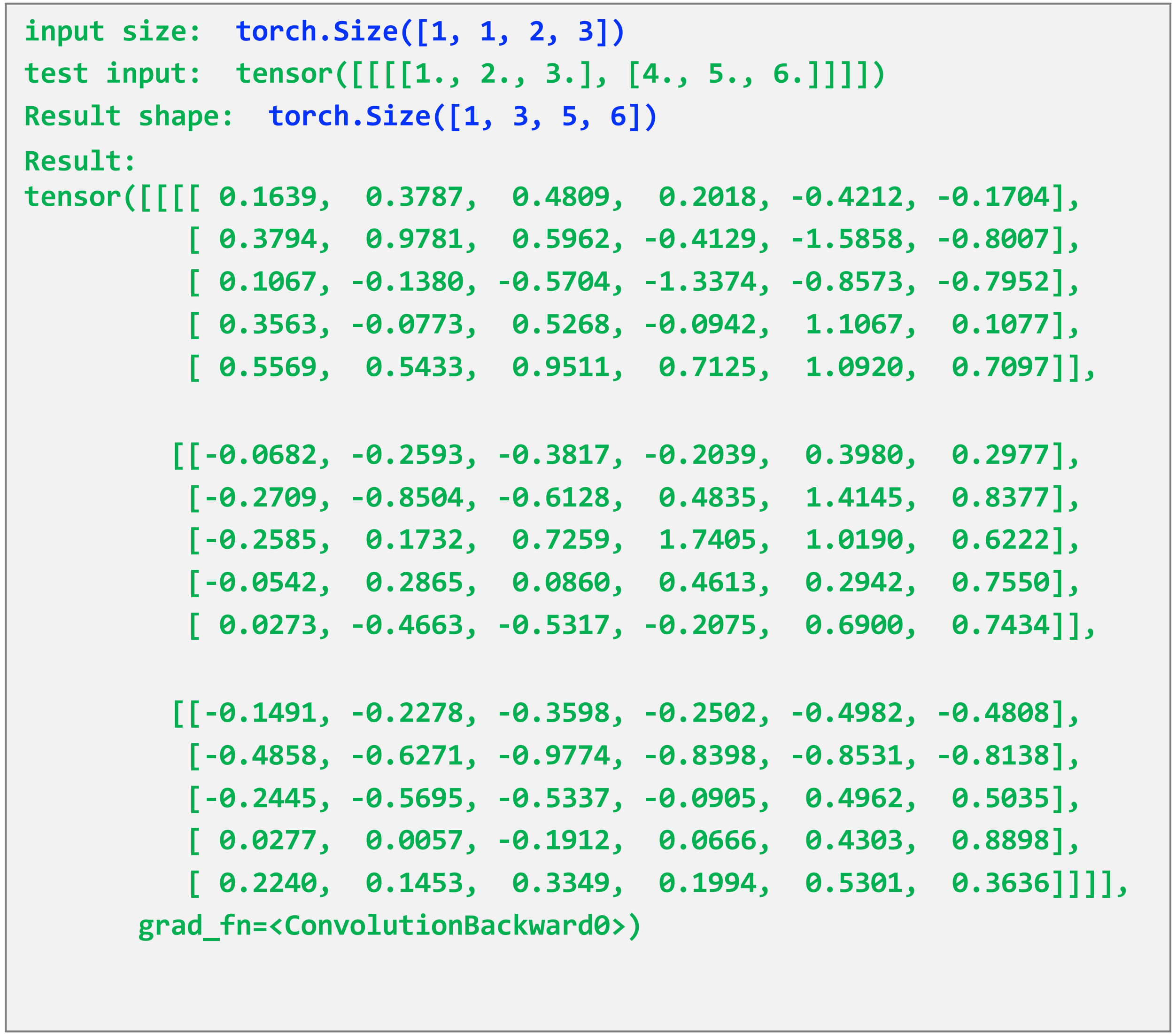
[Source] https://cumulu-s.tistory.com/29

Transposed Convolution

◈2D Transposed Convolution - nn.ConvTranspose2d



```
class SimpleModel(nn.Module):
  def __init__(self):
    super(SimpleModel, self).__init__()
    self.model = nn.ConvTranspose2d(
      in_channels=1, out_channels=3,
      kernel_size=4, stride=1, padding=0
    )

  def forward(self, x):
    return self.model(x)

test_input = torch.Tensor([[[[1, 2, 3], [4, 5, 6]]]])
print("input size: ", test_input.shape)
print("test input: ", test_input)

result = model(test_input)

print("Result shape: ", result.shape)
print("Result: ", result)
```



```
input size:  torch.Size([1, 1, 2, 3])
test input:  tensor([[[[1., 2., 3.], [4., 5., 6.]]]])
Result shape:  torch.Size([1, 3, 5, 6])
Result:
tensor([[[[ 0.1639,  0.3787,  0.4809,  0.2018, -0.4212, -0.1704],
          [ 0.3794,  0.9781,  0.5962, -0.4129, -1.5858, -0.8007],
          [ 0.1067, -0.1380, -0.5704, -1.3374, -0.8573, -0.7952],
          [ 0.3563, -0.0773,  0.5268, -0.0942,  1.1067,  0.1077],
          [ 0.5569,  0.5433,  0.9511,  0.7125,  1.0920,  0.7097]],

         [[-0.0682, -0.2593, -0.3817, -0.2039,  0.3980,  0.2977],
          [-0.2709, -0.8504, -0.6128,  0.4835,  1.4145,  0.8377],
          [-0.2585,  0.1732,  0.7259,  1.7405,  1.0190,  0.6222],
          [-0.0542,  0.2865,  0.0860,  0.4613,  0.2942,  0.7550],
          [ 0.0273, -0.4663, -0.5317, -0.2075,  0.6900,  0.7434]],

         [[-0.1491, -0.2278, -0.3598, -0.2502, -0.4982, -0.4808],
          [-0.4858, -0.6271, -0.9774, -0.8398, -0.8531, -0.8138],
          [-0.2445, -0.5695, -0.5337, -0.0905,  0.4962,  0.5035],
          [ 0.0277,  0.0057, -0.1912,  0.0666,  0.4303,  0.8898],
          [ 0.2240,  0.1453,  0.3349,  0.1994,  0.5301,  0.3636]]]],
       grad_fn=<ConvolutionBackward0>)
```