AI VIETNAM
AI Insight Course

Image Domain Conversion

Quang-Vinh Dinh
Ph.D. in Computer Science

Year 20201

Motivation

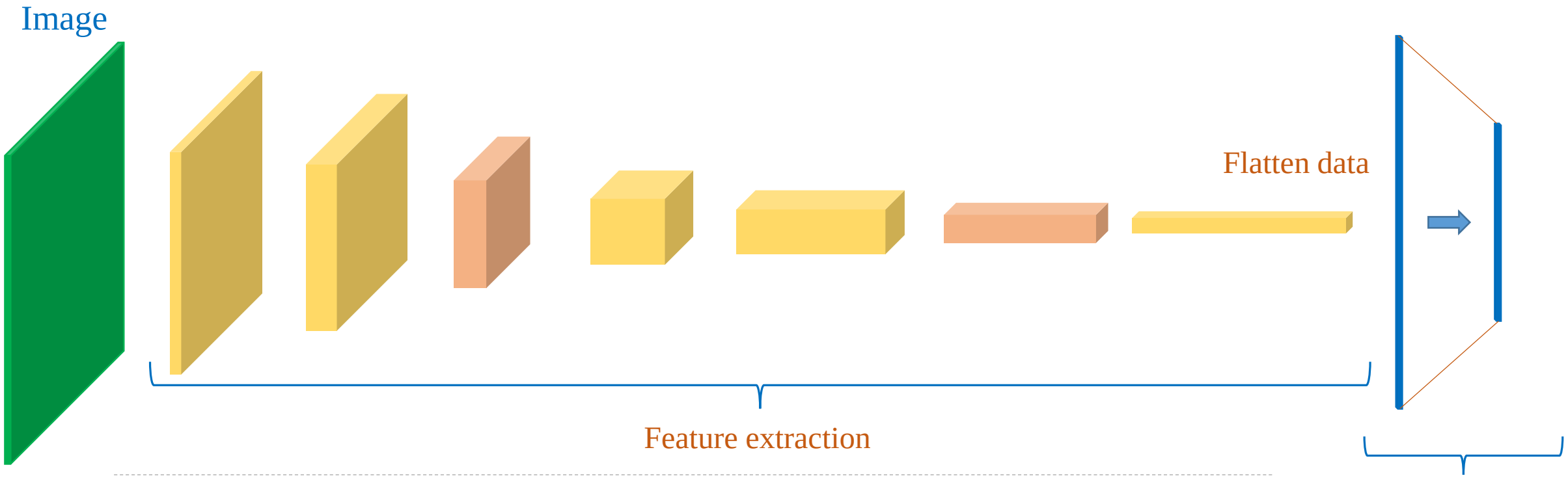


Image
Flatten data
Feature extraction
Classification

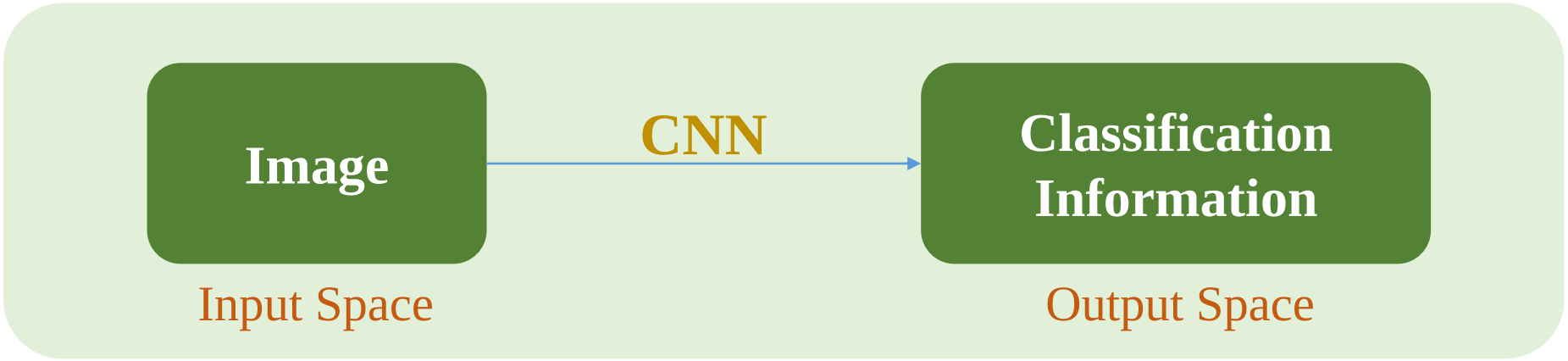


Image
CNN
Classification Information
Input Space
Output Space

Motivation

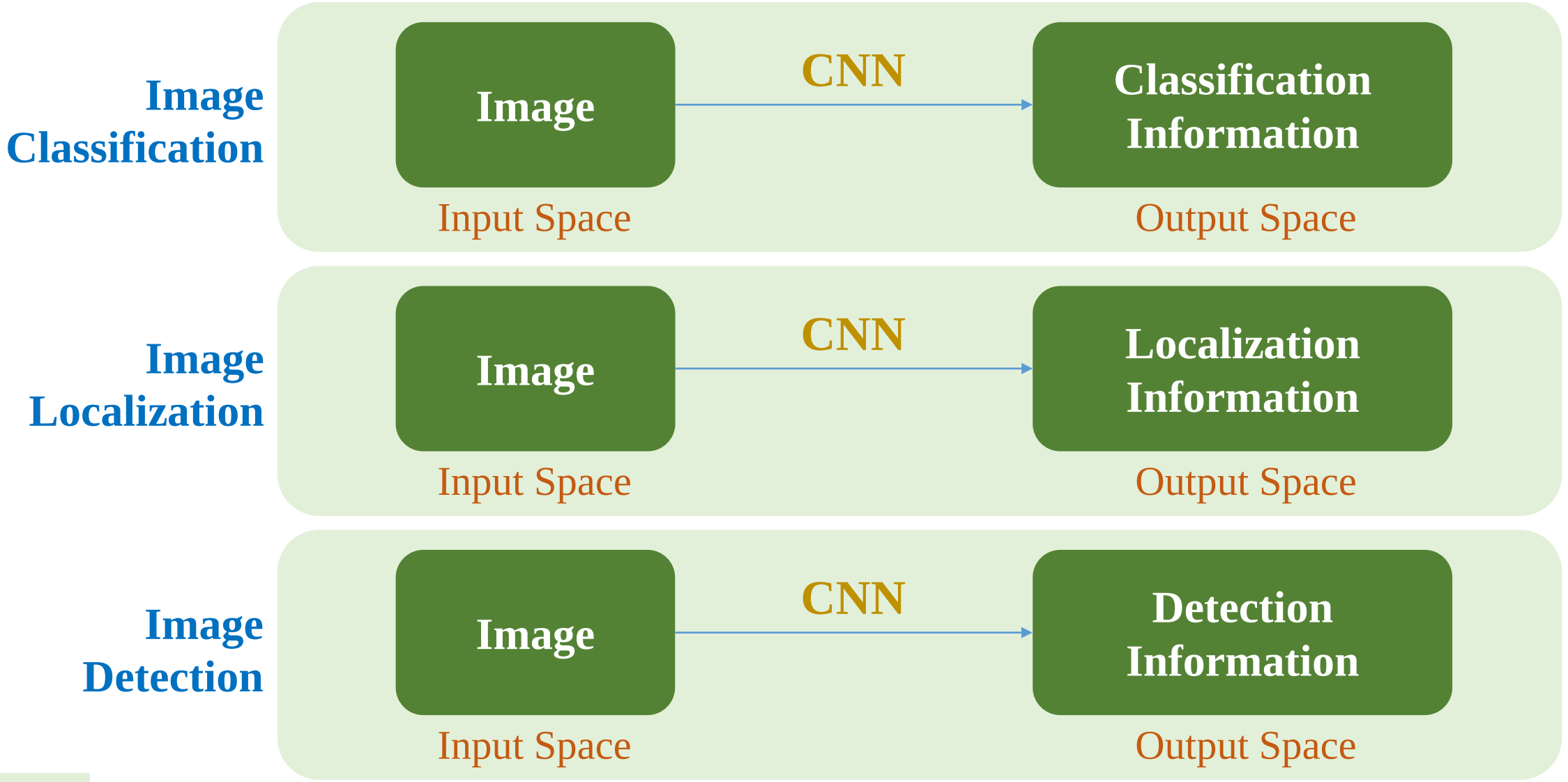
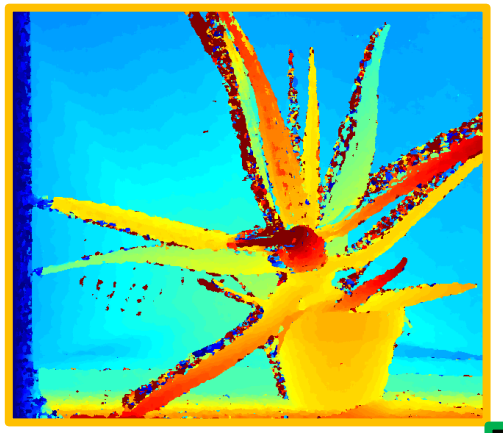


Image Classification
Image
CNN
Classification Information
Input Space
Output Space
Image Localization
Image
CNN
Localization Information
Input Space
Output Space
Image Detection
Image
CNN
Detection Information
Input Space
Output Space

Motivation



Image

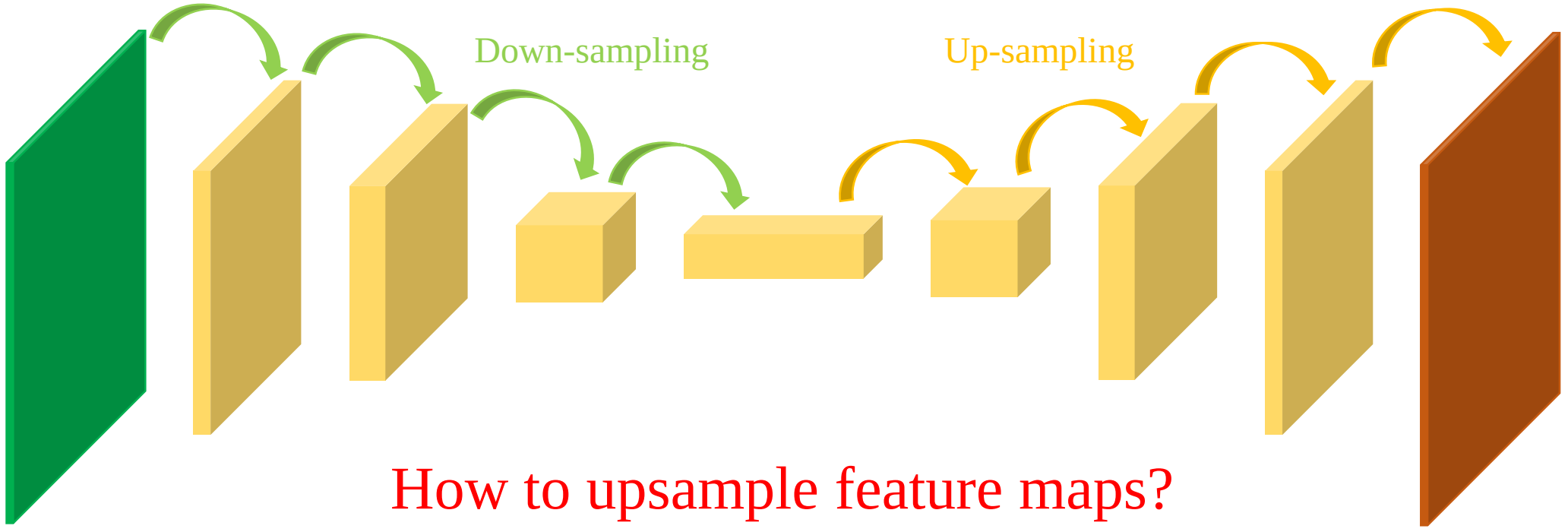
How?

Image

Input Space

Output Space

Motivation



How to upsample feature maps?

Input Image

Output Image

Encoding phase

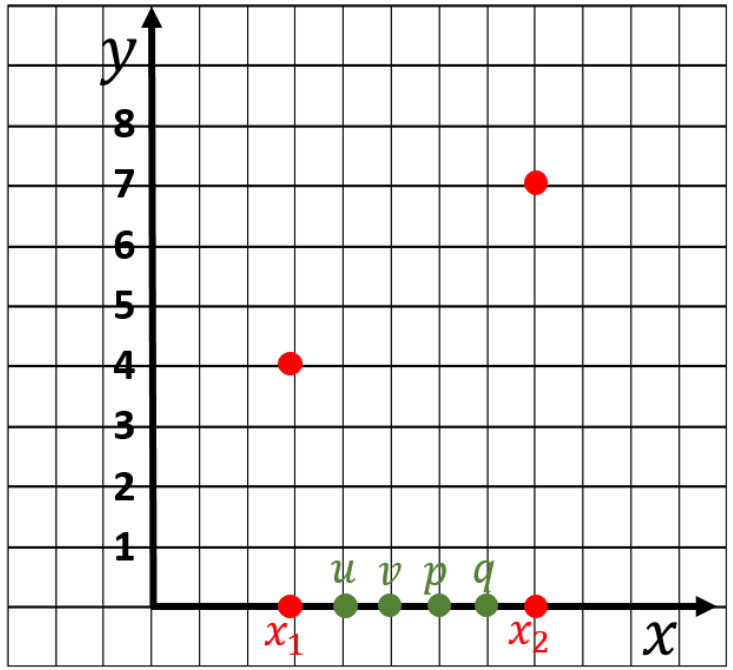
Decoding phase

#channels increases	#channels decreases
resolution decreases	resolution increases

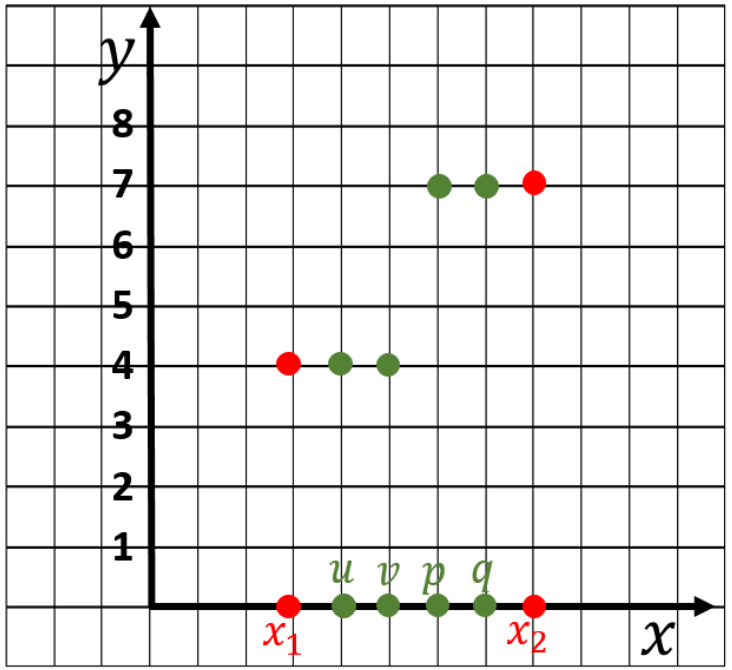
How to Upsample Feature Maps

❖ Solution 1: Image upsampling

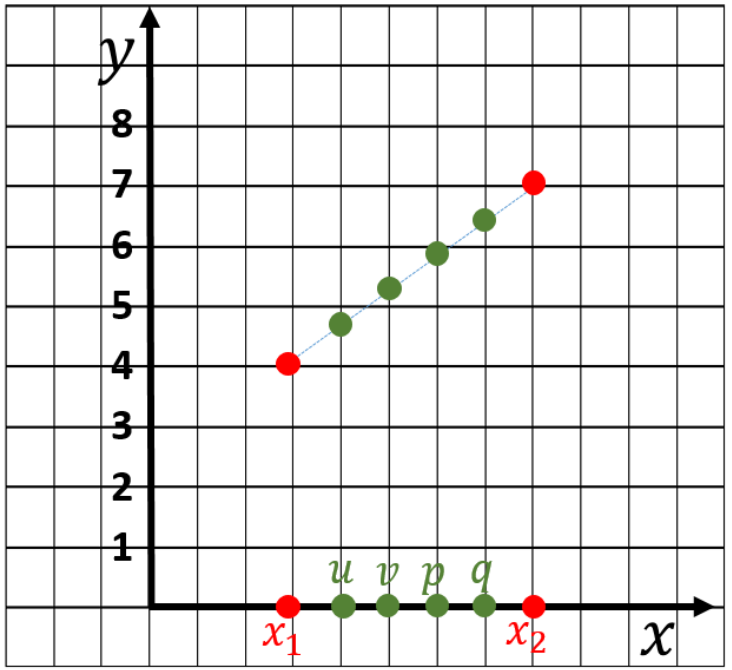
❖ Data interpolation



Tìm giá trị cho các vị trí u, v, p và q



Nearest neighbor: Tính khoảng cách đến x_1 và x_2, và lấy giá trị của x gần hơn



Nội suy theo hàm tuyến tính

How to Upsample Feature Maps

- **Solution 1: Feature upsampling**
  - **Data interpolation**

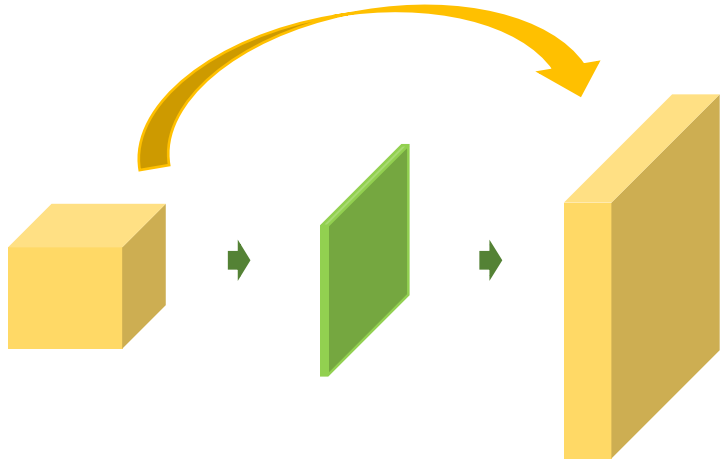
Ảnh gốc

Ảnh phóng to dùng nearest neighbor

Ảnh phóng to dùng hàm tuyến tính

How to Upsample Feature Maps

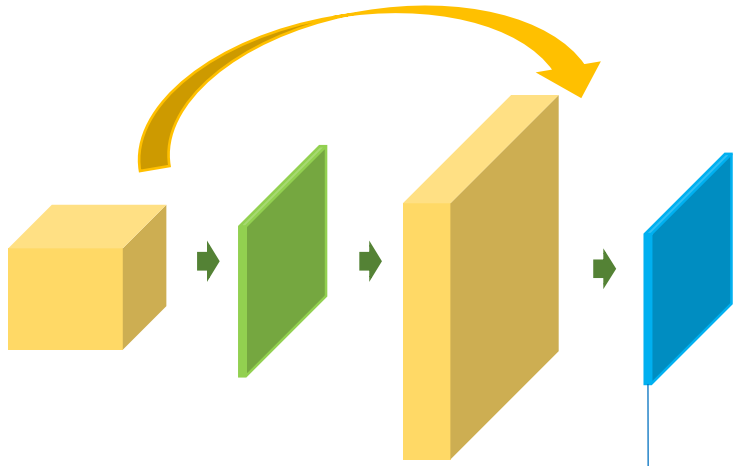
Naïve approach: Only use ‘image upsampling’



Output feature maps are lack of details

```
model = tf.keras.Sequential()
model.add(tf.keras.layers.UpSampling2D(interpolation='bilinear'))
model.add(tf.keras.layers.Conv2D(num_filters,
                                 kernel_size,
                                 padding='same',
                                 kernel_initializer=initializer))
```

Use ‘image upsampling’+Conv



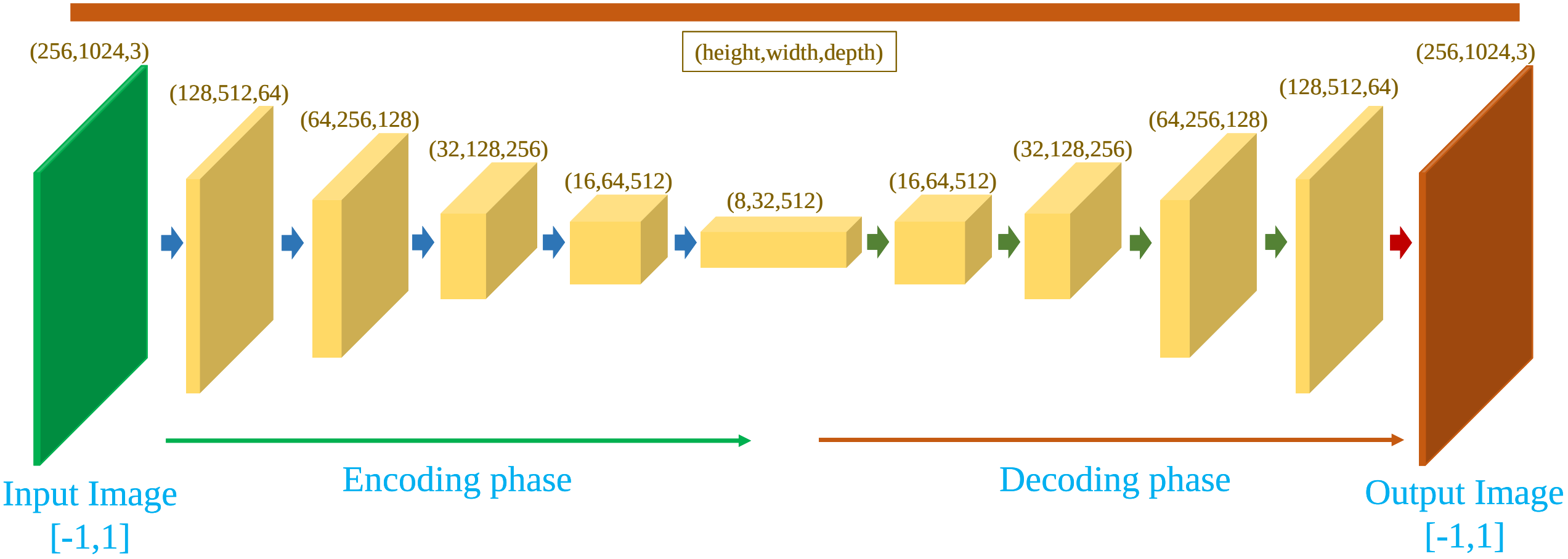
Reduce the weakness from upsampling

upsampling

conv
padding=‘same’
stride=1

AI VIETNAM
AI Insight Course

How to Upsample Feature Maps



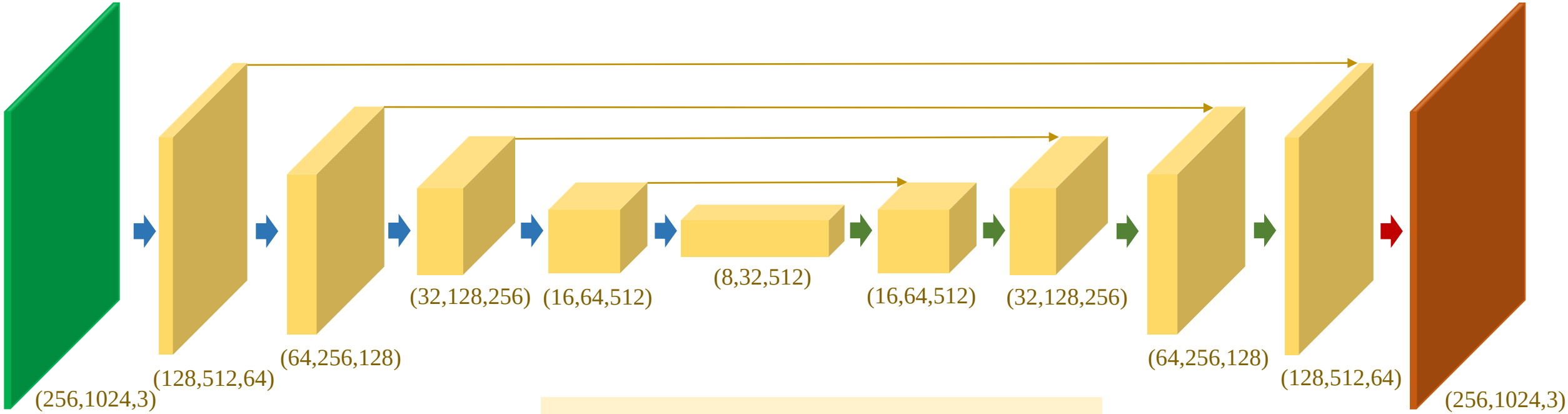
(3x3) Convolution padding = 'same' stride = 1 + ReLU + Batch Norm + (2x2) max pooling

Feature Map Upsamping + (3x3) Convolution padding = 'same' stride = 1 + ReLU + Batch Norm

Feature Map Upsamping + (3x3) Convolution padding = 'same' stride = 1 + Tanh

AI VIETNAM
AI Insight Course

How to Upsample Feature Maps



Input Image
[-1,1]

Skip connection (UNet)

Output Image
[-1,1]

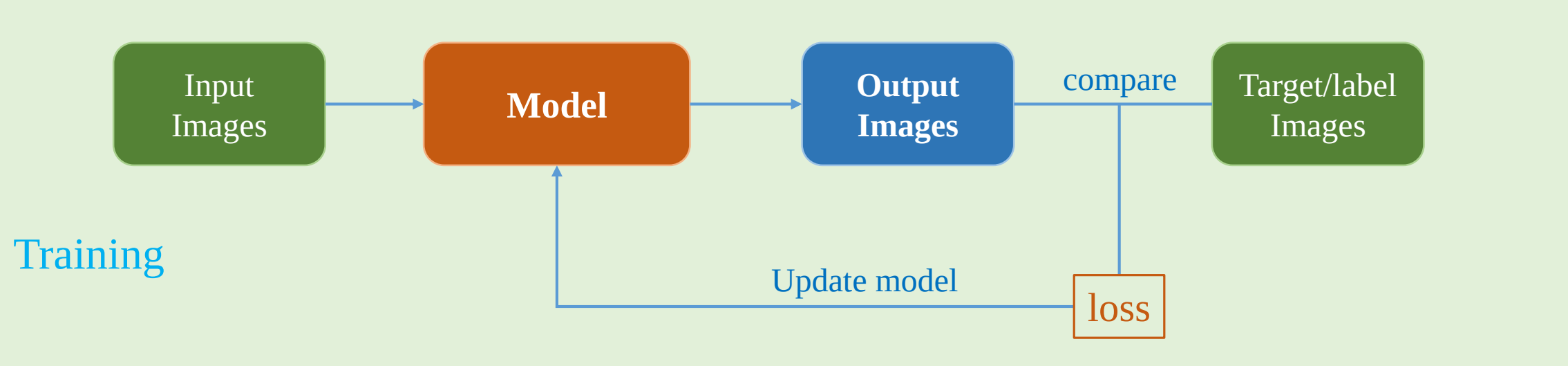
(3x3) Convolution padding = 'same' stride = 1 + ReLU + Batch Norm + (2x2) max pooling

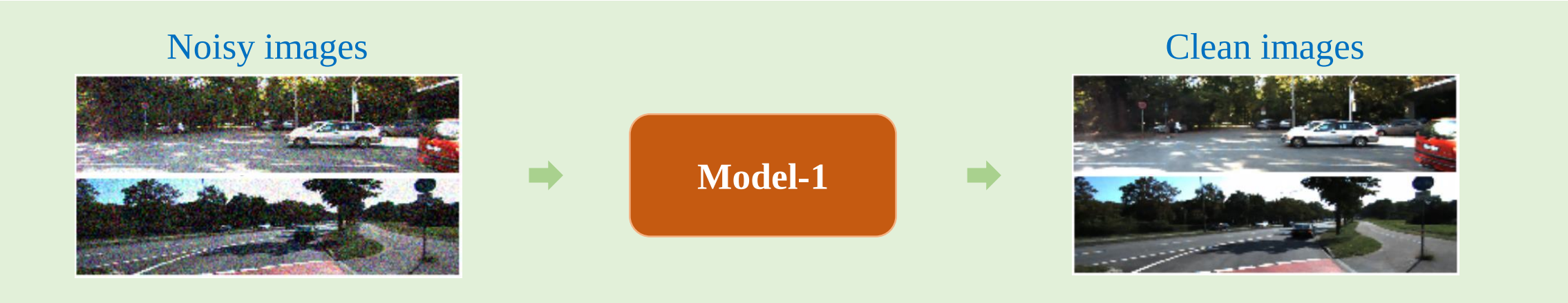
Feature Map Upsamping + (3x3) Convolution padding = 'same' stride = 1 + ReLU + Batch Norm

Feature Map Upsamping + (3x3) Convolution padding = 'same' stride = 1 + Tanh

How to Upsample Feature Maps

❖ Solution 1: Feature upsampling





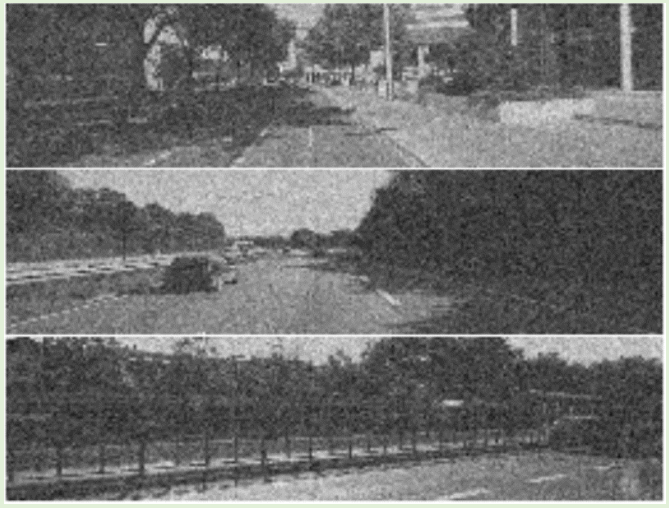
How to Upsample Feature Maps

Grayscale images

Model-1

Color images

Noisy and grayscale images



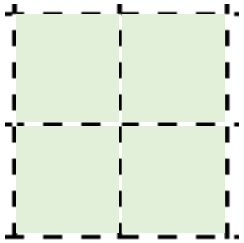
Model-1

Clean and color images

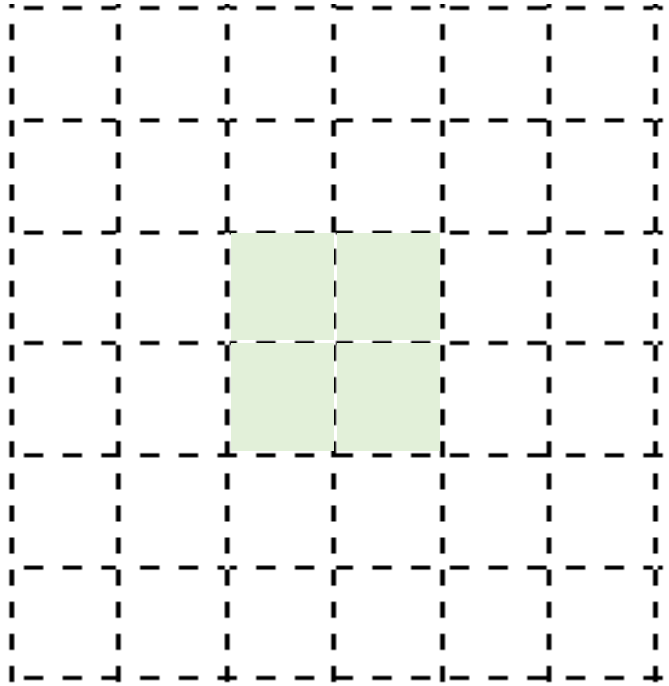
How to Upsample Feature Maps

❖ Solution 2: Convolution Transpose (Use convolution to upsample feature maps)

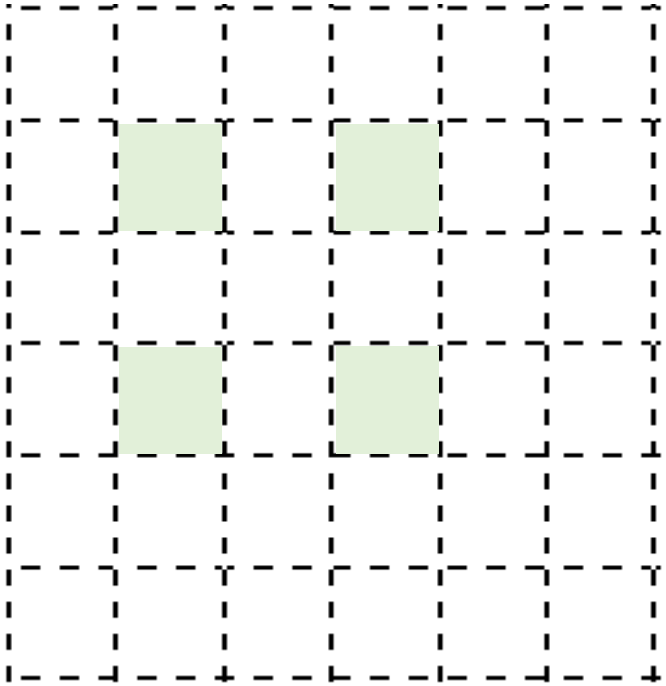
Increase Data from 2x2 to 4x4



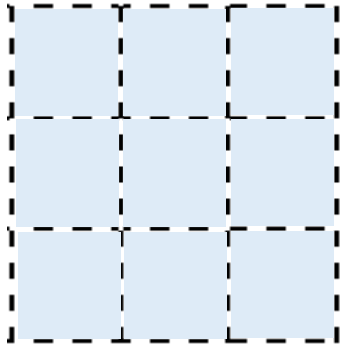
Data



Data + padding=2 (D_1)



Data + stride=2 + padding=(1,1,2,2) (D_2)



Kernel K

In Keras

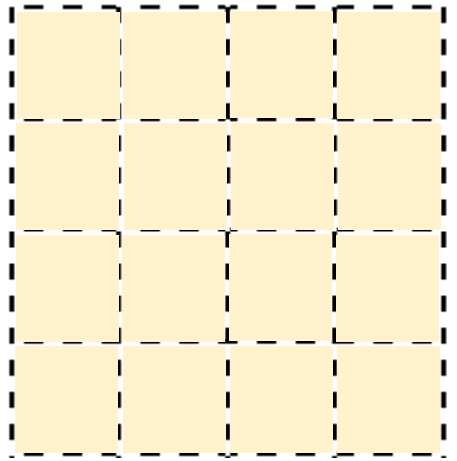
```
kernel_size = 4
Conv2DTranspose(num_filters, kernel_size,
                strides=2, padding='same')
```

Convolution(D_1 with K)

Convolution(D_2 with K)

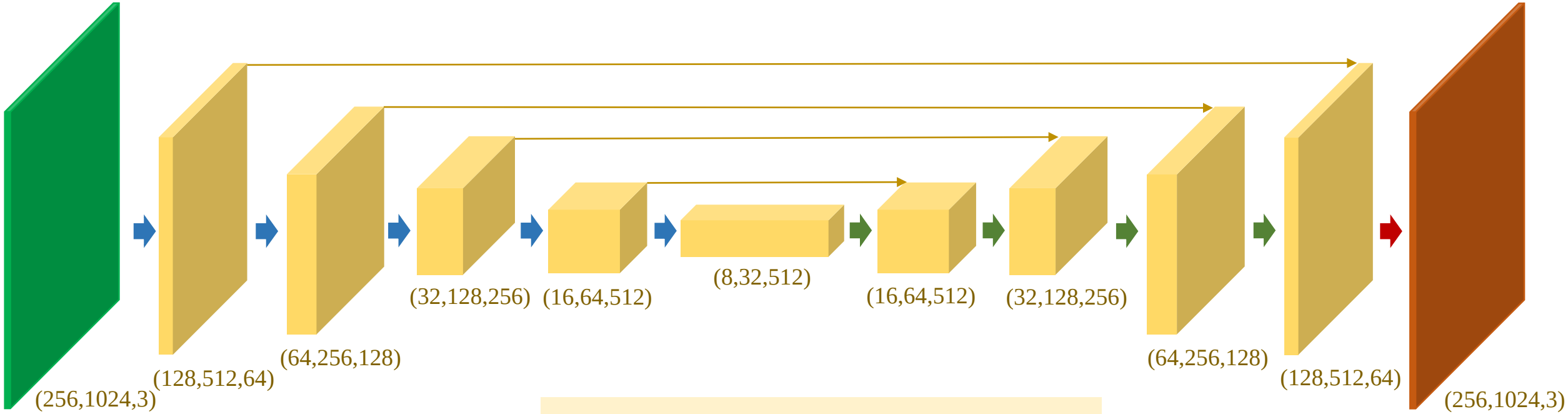
→

Output



AI VIETNAM
AI Insight Course

How to Upsample Feature Maps



Input Image
[-1,1]

Skip connection (UNet)

Output Image
[-1,1]

(3x3) Convolution padding = 'same' stride = 1 + ReLU + Batch Norm + (2x2) max pooling

4x4 ConvTranspose padding = 'same' stride = 2 + ReLU + Batch Norm

Feature Map Upsamping + 4x4 ConvTranspose padding = 'same' stride = 2 + Tanh

How to Upsample Feature Maps

Noisy images

Model-2

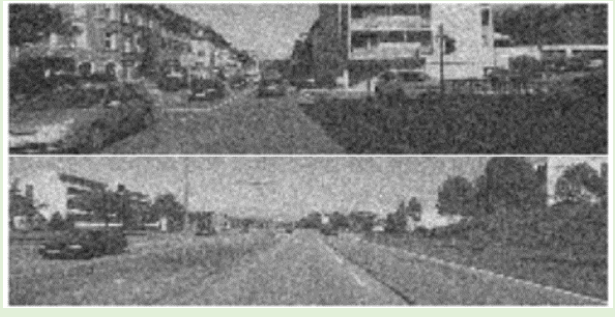
Clean images

Grayscale images

Model-2

Color images

Noisy and grayscale images



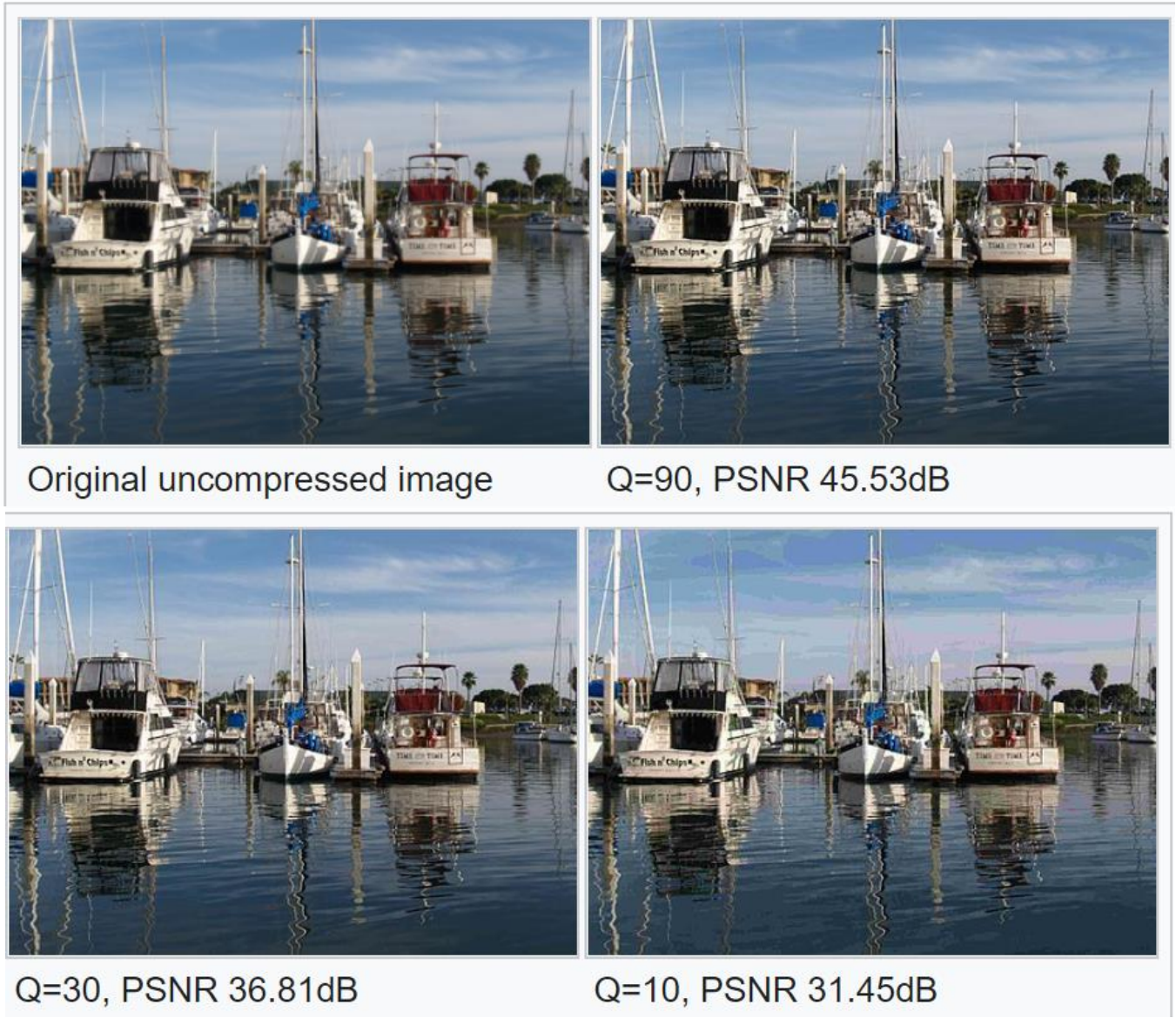
Model-2

Clean and color images

Similarity Metric

❖ **Peak signal-to-noise ratio (PSNR)**

https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio



Applications

❖ Denoise

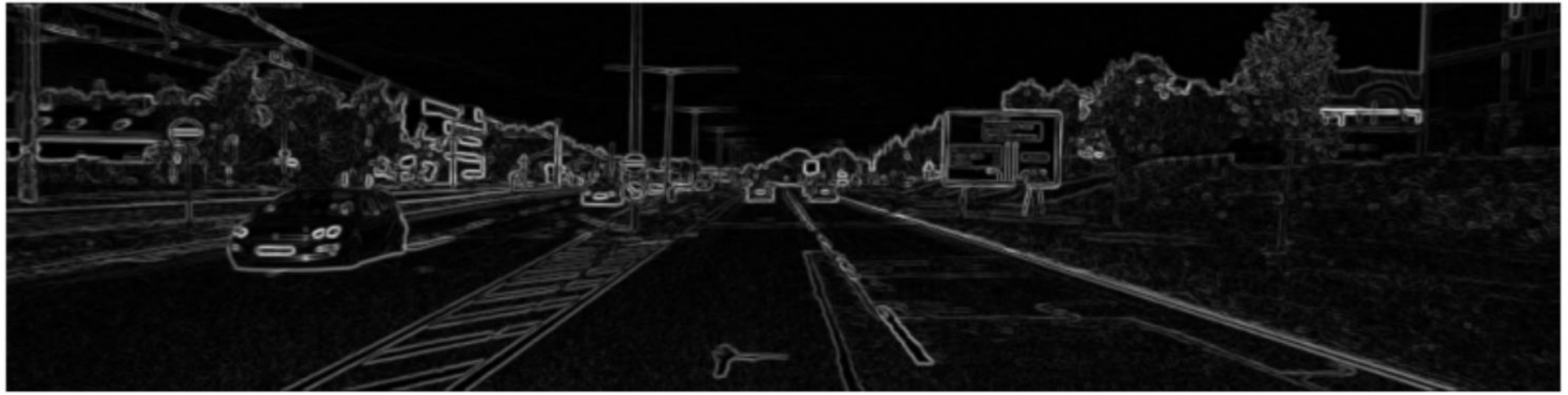
Input Left

Predicted Left

Applications

❖ Edge2Scene

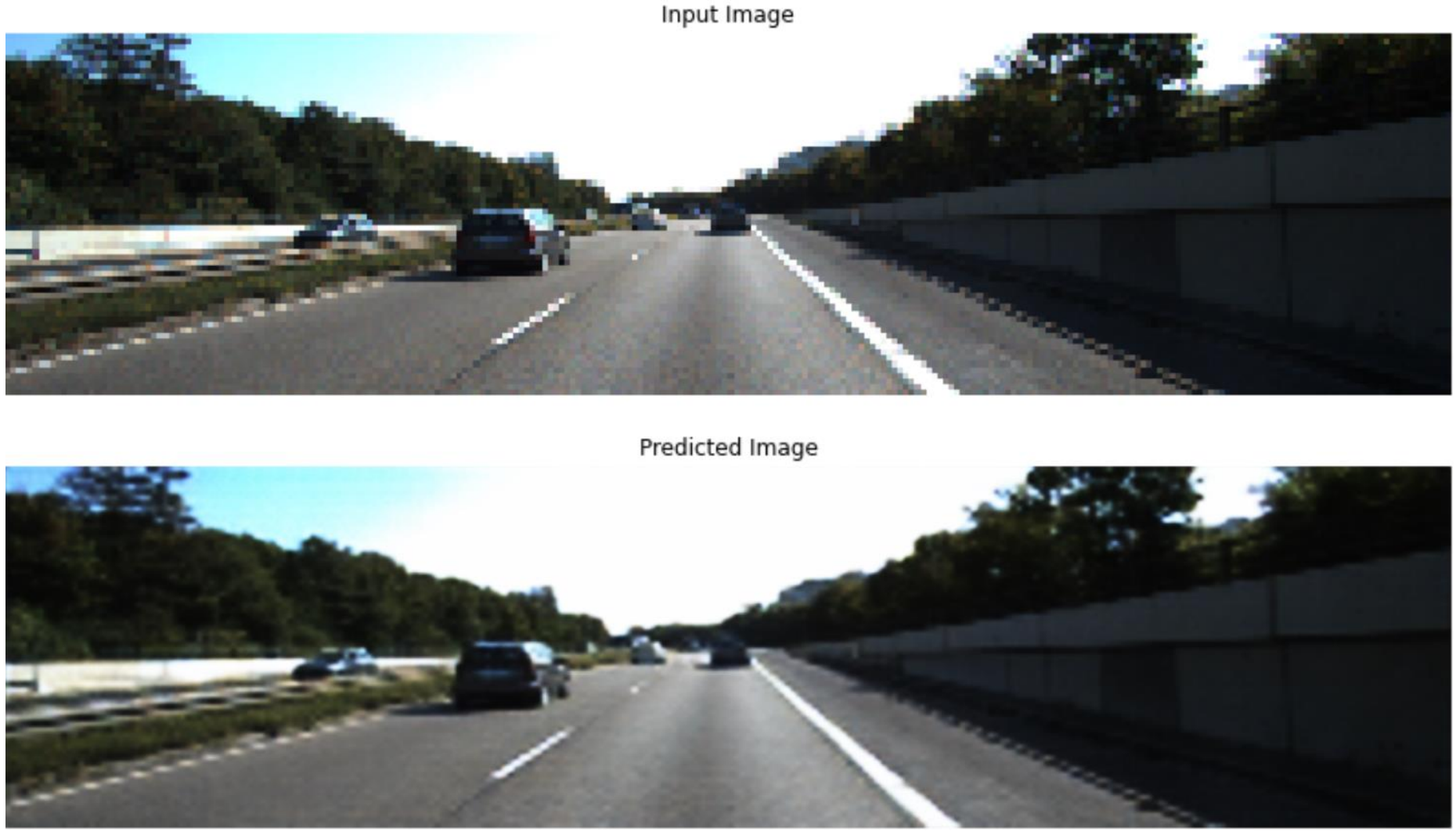
Input Image



Predicted Image

Applications

❖ Super-resolution



Applications

❖ Denoise+Colorization

Input Image

Predicted Image

Applications

❖ Data preparation

```
PATH = '/content/gdrive/My Drive/data/'
IMG_WIDTH    = 1024
IMG_HEIGHT   = 256

def load(image_file):
    image = tf.io.read_file(image_file)
    image = tf.image.decode_jpeg(image)

    image = tf.image.resize(image, (IMG_HEIGHT, IMG_WIDTH))

    return image
```

```
image = load(PATH+'unet/kitti_train/000008_10.png')

# Show the image
plt.figure()
plt.axis('off')
plt.imshow(image/255.0)
```

```
<matplotlib.image.AxesImage at 0x7fae94bf9450>
```

Applications

❖ **Data preparation**

```python
# normalizing the images to [-1, 1]
def normalize(image):
    image  = (image / 127.5) - 1

    return image

def random_jitter(image):
    if tf.random.uniform(()) > 0.5:
        # random mirroring
        image  = tf.image.flip_left_right(image)

    return image

def load_image_train(image_file):
    image = load(image_file)
    image = random_jitter(image)
    image = normalize(image)

    return image
```

Applications

❖ Data preparation

```
# normalizing the images to [-1, 1]
def normalize(image):
    image  = (image / 127.5) - 1

    return image

def random_jitter(image):
    if tf.random.uniform(()) > 0.5:
        # random mirroring
        image  = tf.image.flip_left_right(image)

    return image

def load_image_test(image_file):
    image = load(image_file)
    image = normalize(image)

    return image
```

Applications

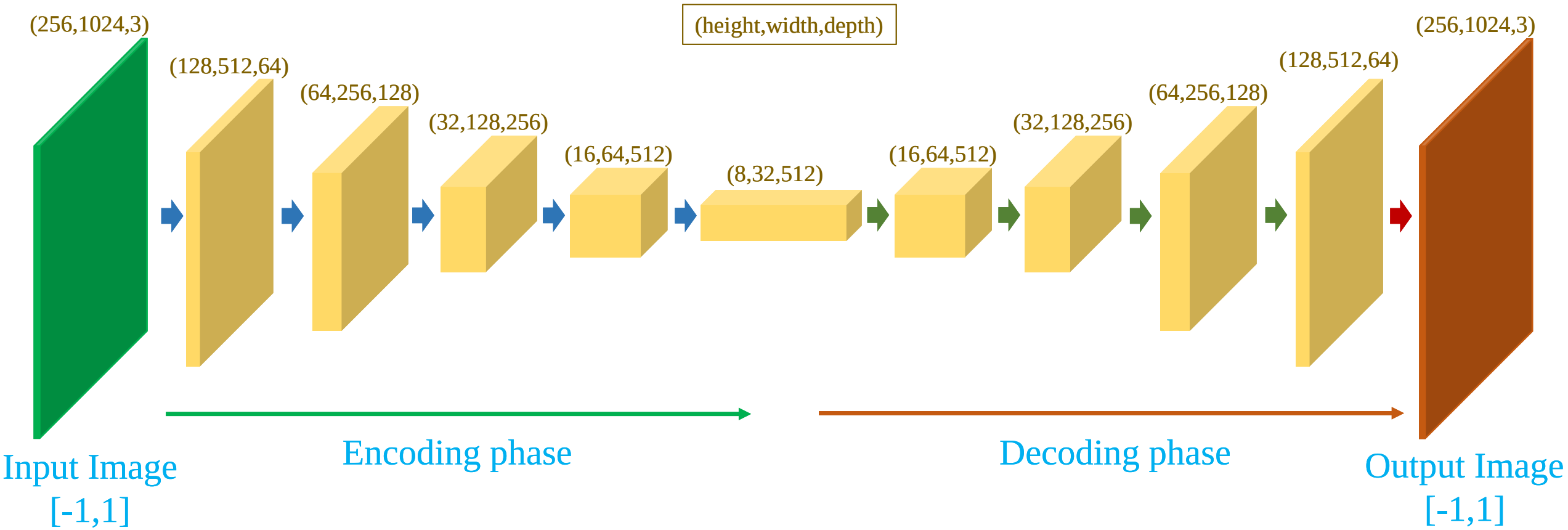
❖ Data preparation

```
BUFFER_SIZE = 50
BATCH_SIZE  = 1

# train_dataset
train_dataset = tf.data.Dataset.list_files(PATH+'unet/kitti_train/*.png')
train_dataset = train_dataset.map(load_image_train,
                                  num_parallel_calls=tf.data.experimental.AUTOTUNE)
train_dataset = train_dataset.shuffle(BUFFER_SIZE)
train_dataset = train_dataset.batch(1)

# test_dataset
test_dataset = tf.data.Dataset.list_files(PATH+'unet/kitti_test/*.png')
test_dataset = test_dataset.map(load_image_test)
test_dataset = test_dataset.batch(1)
```


22.1.UNet-Data.ipynb

Applications

❖ Network construction



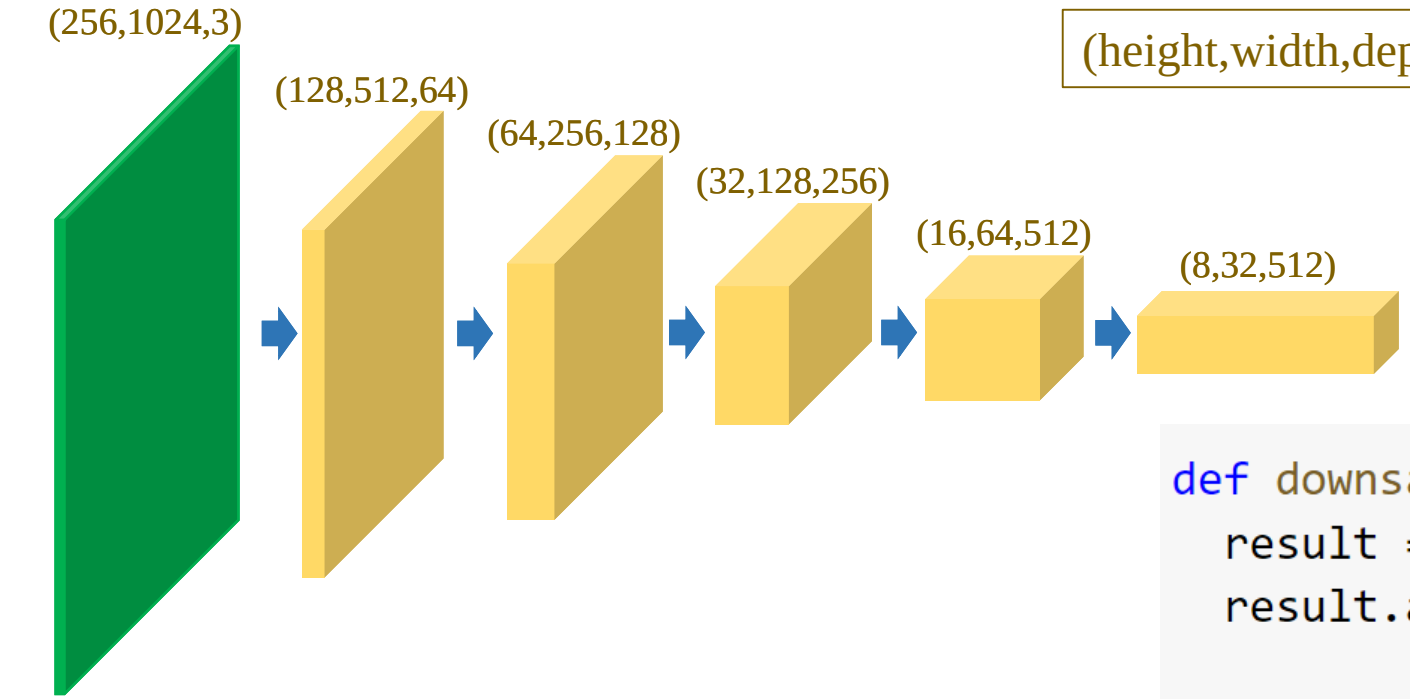
(3x3) Convolution padding = 'same' stride = 1 + ReLU + Batch Norm + (2x2) max pooling

Feature Map Upsamping + (3x3) Convolution padding = 'same' stride = 1 + ReLU + Batch Norm

Feature Map Upsamping + (3x3) Convolution padding = 'same' stride = 1 + Tanh

Applications

❖ Network construction

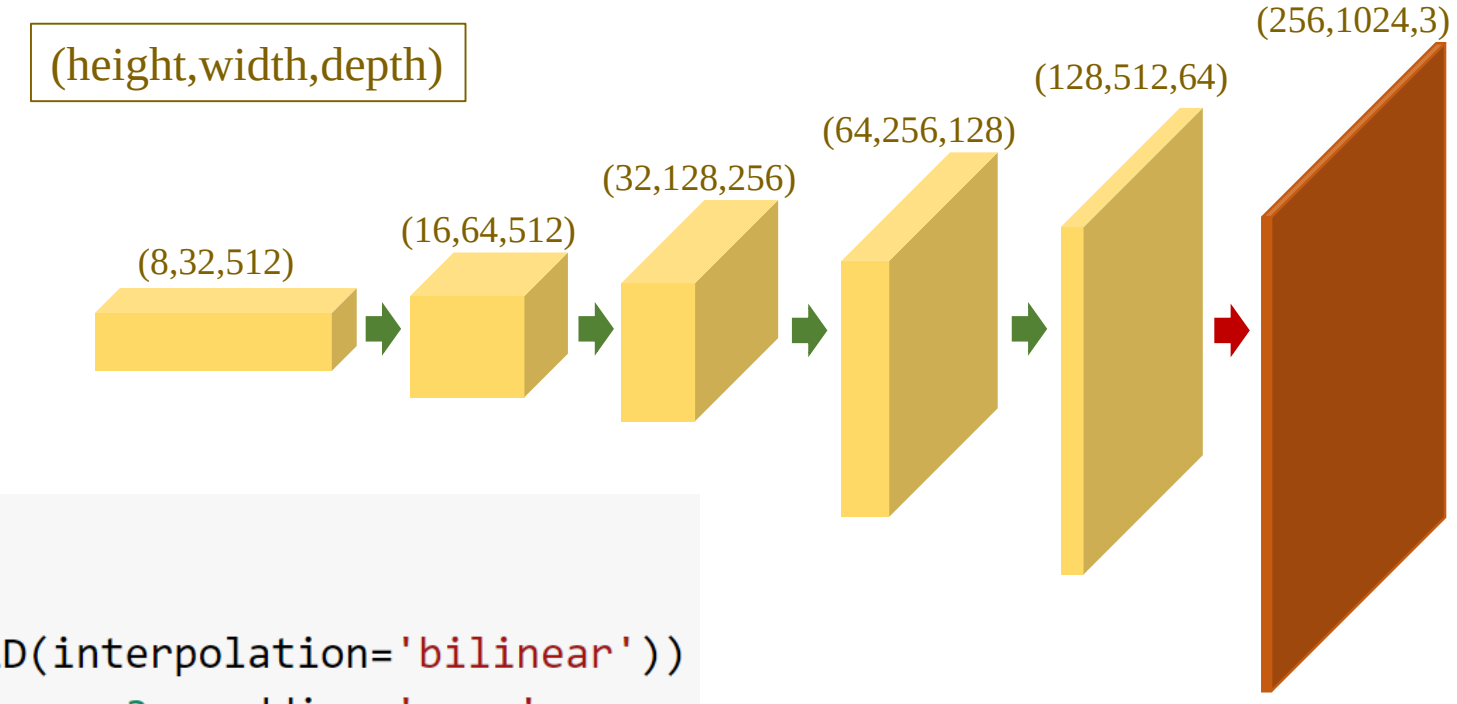


(3x3) Convolution padding = 'same' stride = 1 + ReLU + Batch Norm + (2x2) max pooling

```
def downsample(filters, size):
  result = tf.keras.Sequential()
  result.add(tf.keras.layers.Conv2D(filters, size,
                                    strides=2,
                                    padding='same',
                                    use_bias=False))
  result.add(tf.keras.layers.ReLU())
  result.add(tf.keras.layers.BatchNormalization())

  return result
```

Applications

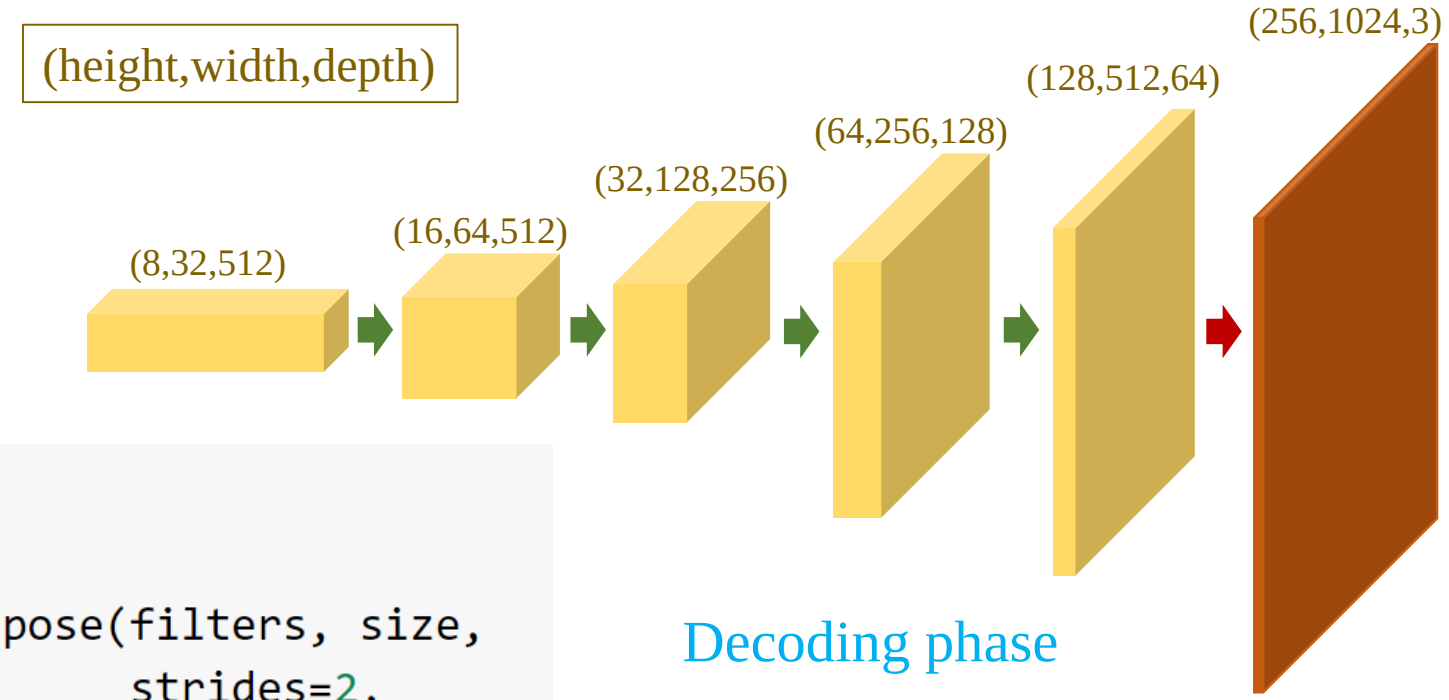
❖ Network construction




```
def upsample(filters, size):
    result = tf.keras.Sequential()
    result.add(tf.keras.layers.UpSampling2D(interpolation='bilinear'))
    result.add(tf.keras.layers.Conv2D(filters, 3, padding='same',
                                      kernel_initializer=initializer,
                                      use_bias=False))
    result.add(tf.keras.layers.ReLU())
    result.add(tf.keras.layers.BatchNormalization())

    return result
```

Applications

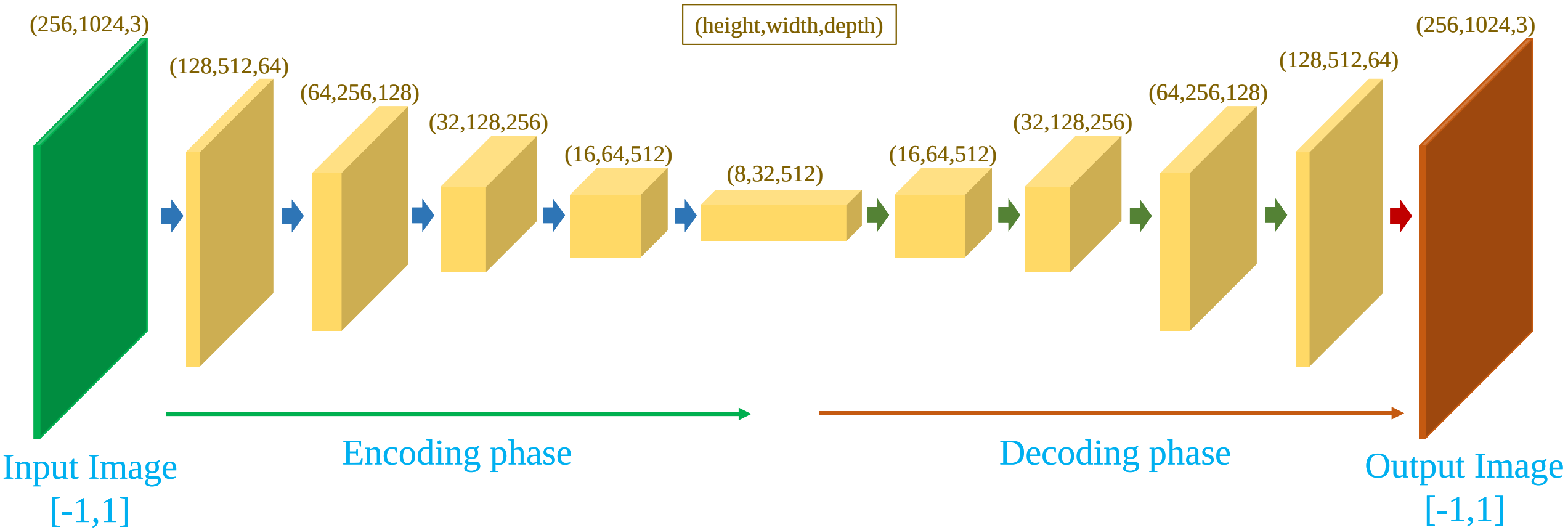
❖ Network construction




```
def upsample(filters, size):
  result = tf.keras.Sequential()
  result.add(tf.keras.layers.Conv2DTranspose(filters, size,
                                             strides=2,
                                             padding='same',
                                             use_bias=False))
  result.add(tf.keras.layers.ReLU())
  result.add(tf.keras.layers.BatchNormalization())

  return result
```

Applications

❖ Network construction



(3x3) Convolution padding = 'same' stride = 1 + ReLU + Batch Norm + (2x2) max pooling

Feature Map Upsamping + (3x3) Convolution padding = 'same' stride = 1 + ReLU + Batch Norm

Feature Map Upsamping + (3x3) Convolution padding = 'same' stride = 1 + Tanh

Applications

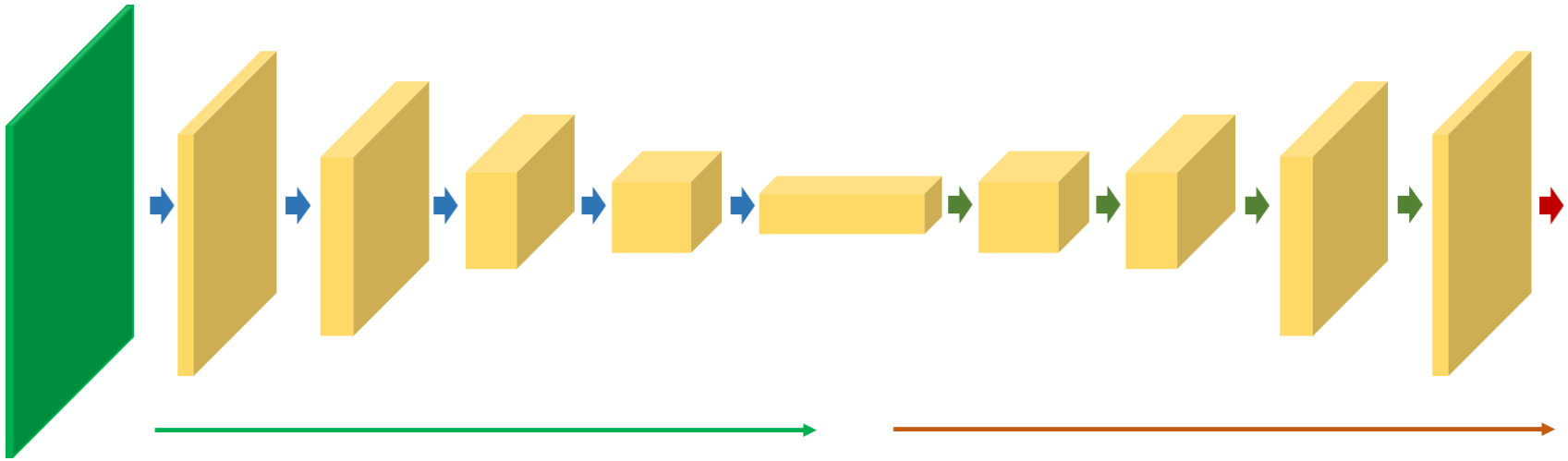
❖ Network construction

```
def UNet_process(x): # (1, 256, 1024, 3)
  # encoding
  down_stack = [
    downsample(64, 4),  # (bs, 128, 512, 64)
    downsample(256, 4), # (bs, 64, 256, 256)
    downsample(512, 4), # (bs, 32, 128, 512)
    downsample(512, 4), # (bs, 16, 64, 512)
    downsample(512, 4), # (bs, 8, 32, 512)
  ]

  for down in down_stack:
    x = down(x)
```

```
# decoding
up_stack = [
  upsample(512, 4), # (bs, 16, 64, 512)
  upsample(512, 4), # (bs, 32, 128, 512)
  upsample(256, 4), # (bs, 64, 256, 256)
  upsample(64, 4),  # (bs, 128, 512, 64)
]

for up in up_stack:
  x = up(x)

# last layer
OUTPUT_CHANNELS = 3
last = tf.keras.layers.Conv2DTranspose(OUTPUT_CHANNELS,
                                       4, strides=2,
                                       padding='same',
                                       activation='tanh')

x = last(x)

return x
```

Applications

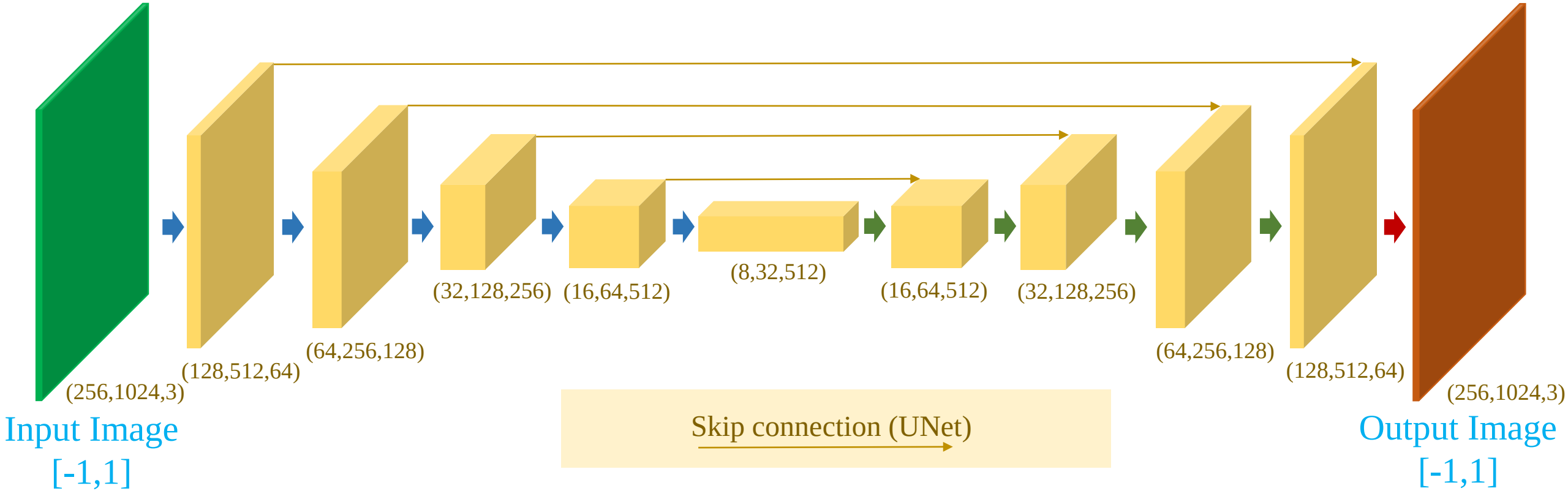
❖ Network construction



```
def Generator():
  # inputs
  inputs = tf.keras.layers.Input(shape=[256, 1024, 3])
  x = inputs

  # UNet_process
  output = UNet_process(x)

  return tf.keras.Model(inputs=[inputs], outputs=[output])
```

Applications

❖ Network construction



Applications

❖ Network construction

```
def UNet_process(x): # (1, 256, 1024, 3)
  # encoding
  down_stack = [
    downsample(64, 4),  # (bs, 128, 512, 64)
    downsample(256, 4), # (bs, 64, 256, 256)
    downsample(512, 4), # (bs, 32, 128, 512)
    downsample(512, 4), # (bs, 16, 64, 512)
    downsample(512, 4), # (bs, 8, 32, 512)
  ]

  skips = []
  for down in down_stack:
    x = down(x)
    skips.append(x)

  skips = reversed(skips[:-1])
```

```
# decoding
up_stack = [
  upsample(512, 4), # (bs, 16, 64, 512)
  upsample(512, 4), # (bs, 32, 128, 512)
  upsample(256, 4), # (bs, 64, 256, 256)
  upsample(64, 4),  # (bs, 128, 512, 64)
]

concat = tf.keras.layers.Concatenate()
for up, skip in zip(up_stack, skips):
  x = up(x)
  x = concat([x, skip])


# last layer
OUTPUT_CHANNELS = 3
last = tf.keras.layers.Conv2DTranspose(OUTPUT_CHANNELS,
                                       4, strides=2,
                                       padding='same',
                                       activation='tanh')

x = last(x)

return x
```

Applications

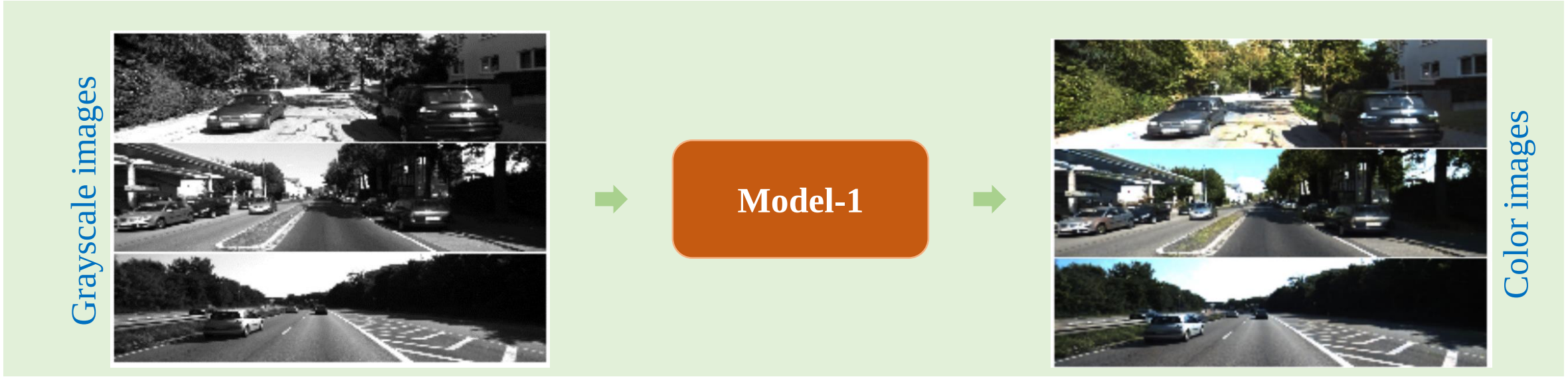
❖ Network construction

```
def Generator():
  # inputs
  inputs = tf.keras.layers.Input(shape=[256, 1024, 3])
  x = inputs

  # extract_first_features
  fextract = extract_first_features(64, 3)  # (1, 256, 1024, 64)
  x = fextract(x) # (1,256,1024,64)

  # UNet_process
  output = UNet_process(x)

  return tf.keras.Model(inputs=[inputs], outputs=[output])
```

Applications

❖ Colorization

1	Data preparation
2	Network construction
3	Loss and optimizer
4	Training



Applications

1 Data preparation

❖ Colorization

```
PATH = '/content/gdrive/My Drive/data/'
IMG_WIDTH  = 1024
IMG_HEIGHT  = 256

def load(image_file):
    image = tf.io.read_file(image_file)
    image = tf.image.decode_jpeg(image)

    # resize
    image = tf.image.resize(image, (IMG_HEIGHT, IMG_WIDTH))

    # copy
    gray = tf.identity(image)
    color= tf.identity(image)

    # convert to gray
    gray = tf.image.rgb_to_grayscale(gray)

    gray  = tf.cast(gray, tf.float32)
    color = tf.cast(color, tf.float32)

    return gray, color
```

```
gray, color = load(PATH+'unet/kitti_train/000008_10.png')

# Show the image
plt.figure()
plt.axis('off')
plt.imshow(gray[:,:,0]/255.0, cmap='gray')

plt.figure()
plt.axis('off')
plt.imshow(color/255.0)
```

```
<matplotlib.image.AxesImage at 0x7f7f0324e550>
```

Applications

❖ Colorization

```
BUFFER_SIZE = 50
BATCH_SIZE  = 1

# train_dataset
train_dataset = tf.data.Dataset.list_files(PATH+'unet/kitti_train/0000*_10.png')
train_dataset = train_dataset.map(load_image_train,
                                  num_parallel_calls=tf.data.experimental.AUTOTUNE)
train_dataset = train_dataset.shuffle(BUFFER_SIZE)
train_dataset = train_dataset.batch(BATCH_SIZE)

# test_dataset
test_dataset = tf.data.Dataset.list_files(PATH+'unet/kitti_test/0000*_10.png')
test_dataset = test_dataset.map(load_image_test)
test_dataset = test_dataset.batch(1)
```

```
# normalizing the images to [-1, 1]
def normalize(gray, color):
    gray  = (gray / 127.5) - 1
    color = (color / 127.5) - 1

    return gray, color

def random_jitter(gray, color):
    if tf.random.uniform(()) > 0.5:
        # random mirroring
        gray  = tf.image.flip_left_right(gray)
        color = tf.image.flip_left_right(color)

    return gray, color

def load_image_train(image_file):
    gray, color = load(image_file)
    gray, color = random_jitter(gray, color)
    gray, color = normalize(gray, color)

    return gray, color

def load_image_test(image_file):
    gray, color = load(image_file)
    gray, color = normalize(gray, color)

    return gray, color
```

Applications

❖ Network construction

```
def UNet_process(x): # (1, 256, 1024, 3)
  # encoding
  down_stack = [
    downsample(64, 4),  # (bs, 128, 512, 64)
    downsample(256, 4), # (bs, 64, 256, 256)
    downsample(512, 4), # (bs, 32, 128, 512)
    downsample(512, 4), # (bs, 16, 64, 512)
    downsample(512, 4), # (bs, 8, 32, 512)
  ]

  skips = []
  for down in down_stack:
    x = down(x)
    skips.append(x)

  skips = reversed(skips[:-1])
```

```
# decoding
up_stack = [
  upsample(512, 4), # (bs, 16, 64, 512)
  upsample(512, 4), # (bs, 32, 128, 512)
  upsample(256, 4), # (bs, 64, 256, 256)
  upsample(64, 4),  # (bs, 128, 512, 64)
]

concat = tf.keras.layers.Concatenate()
for up, skip in zip(up_stack, skips):
  x = up(x)
  x = concat([x, skip])


# last layer
OUTPUT_CHANNELS = 3
last = tf.keras.layers.Conv2DTranspose(OUTPUT_CHANNELS,
                                       4, strides=2,
                                       padding='same',
                                       activation='tanh')

x = last(x)

return x
```

Applications

❖ Colorization

3 Loss and optimizer

```
# loss function
def compute_loss(img1, img2):
  return tf.reduce_mean(tf.abs(img1-img2))

#optimizer
optimizer = tf.keras.optimizers.Adam(1e-4, beta_1=0.5)
```

Applications

❖ Colorization

4 Training

```
@tf.function
def train_step(gray, color):
    with tf.GradientTape() as gen_tape:
        # output
        fake_color = generator([gray], training=True)
        loss = compute_loss(fake_color, color)

    gradients = gen_tape.gradient(loss, generator.trainable_variables)
    optimizer.apply_gradients(zip(gradients, generator.trainable_variables))

    return loss

def fit(train_ds, epochs, test_ds):
    for epoch in range(epochs):
        # Train
        for gray, color in train_ds:
            loss = train_step(gray, color)
```

Applications

❖ Colorization

5 For debugging

```python
def generate_images(model, gray, real):
    fake = model([gray], training=True)
    plt.figure(figsize=(15,20))

    display_list = [gray[0,:,:,0], real[0], fake[0]]
    title = ['Input Left', 'Real Left', 'Predicted Left']

    for i in range(3):
        plt.subplot(1, 3, i+1)
        plt.title(title[i])
        plt.imshow(display_list[i] * 0.5 + 0.5)
        plt.axis('off')
    plt.show()
```

```python
def evaluate(model, epoch):
    psnr_mean = 0.0
    count = 0
    for gray, real in test_dataset:
        fake = model([gray], training=True)

        psnr = tf.image.psnr(fake*0.5 + 0.5,
                             real*0.5 + 0.5,
                             max_val=1.0)
        psnr = tf.math.reduce_mean(psnr)

        psnr_mean += psnr
        count =count + 1

    psnr_mean = psnr_mean/count
    return psnr_mean
```

Applications

❖ **Colorization**

4 Training

22.3.UNet-Colorization_v1_showGrayImage.ipynb

```python
@tf.function
def train_step(gray, color):
    with tf.GradientTape() as gen_tape:
        # output
        fake_color = generator([gray], training=True)
        loss = compute_loss(fake_color, color)

    gradients = gen_tape.gradient(loss, generator.trainable_variables)
    optimizer.apply_gradients(zip(gradients, generator.trainable_variables))

    return loss

def fit(train_ds, epochs, test_ds):
    best_pnsr = 0.0
    for epoch in range(epochs):
        # Train
        for gray, color in train_ds:
            loss = train_step(gray, color)

        # for debug
        pnsr = evaluate(generator, epoch)
        if best_pnsr < pnsr:
            best_pnsr = pnsr
            print(best_pnsr)

            for gray, color in test_ds.take(1):
                generate_images(generator, gray, color)
```

Applications

❖ Colorization

5 For debugging

22.3.UNet-Colorization_v3.ipynb

```
def generate_images(model, gray, real):
    fake = model([gray], training=True)
    plt.figure(figsize=(15,20))

    display_list = [gray[0,:,:,0], real[0], fake[0]]
    title = ['Input Left', 'Real Left', 'Predicted Left']

    i = 0
    plt.subplot(1, 3, i+1)
    plt.title(title[i])
    plt.imshow(display_list[i]*0.5 + 0.5, cmap='gray')
    plt.axis('off')

    for i in range(1,3):
        plt.subplot(1, 3, i+1)
        plt.title(title[i])
        plt.imshow(display_list[i] * 0.5 + 0.5)
        plt.axis('off')
    plt.show()
```


Thank you!