AIL 862

Lecture 10

Application – super resolution (SRGAN)

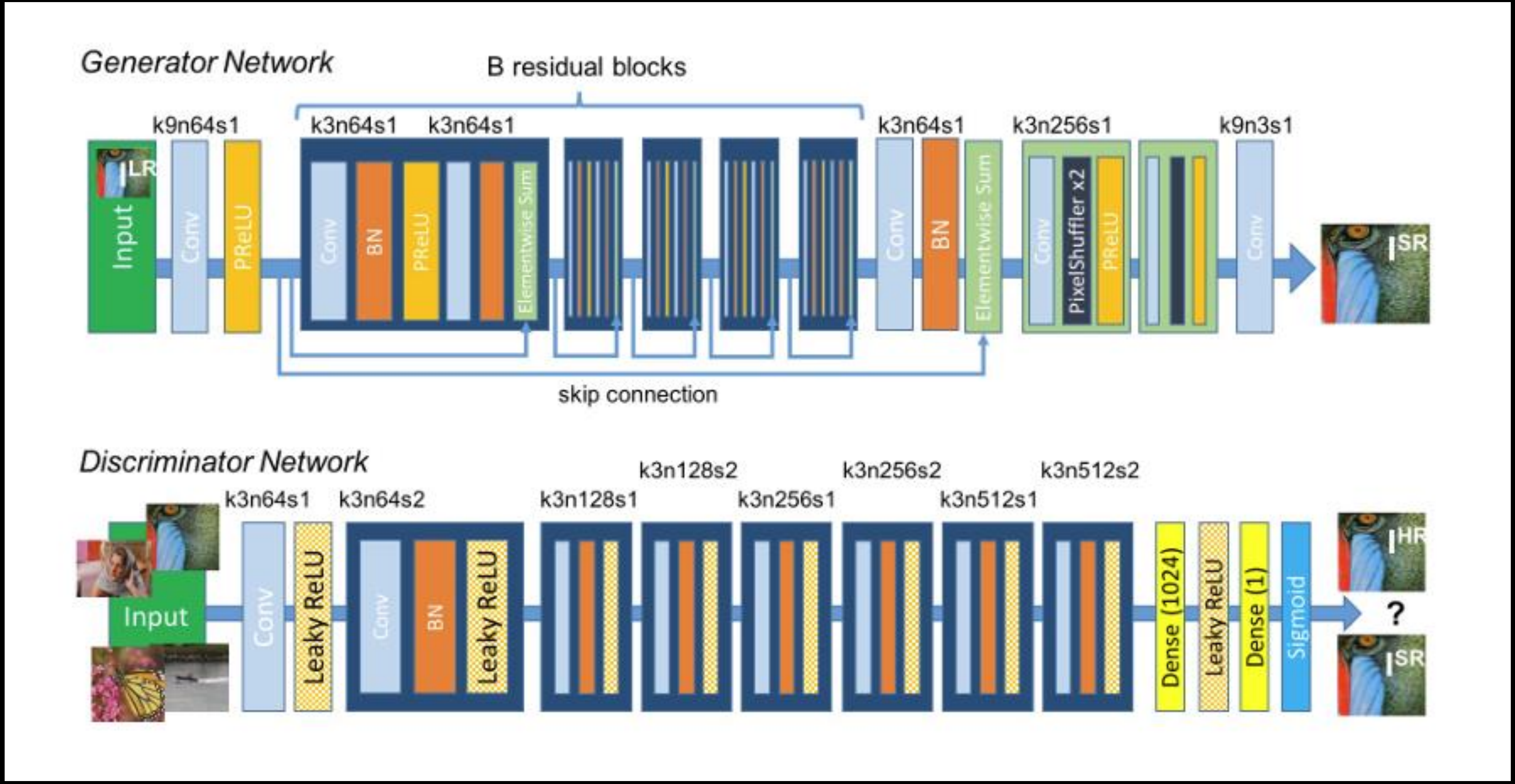
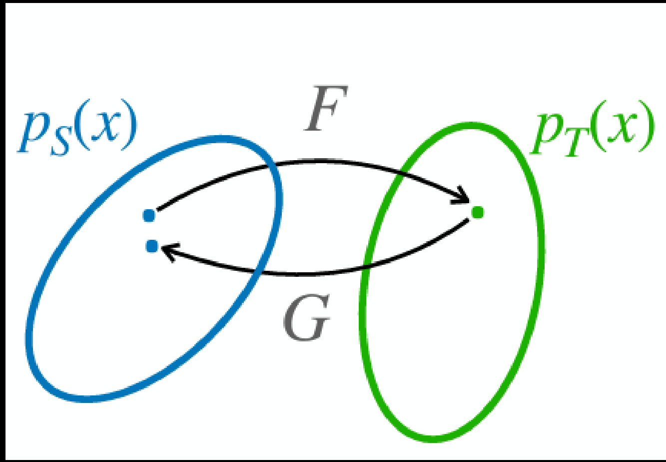


Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network, 2017

Domain Translation with CycleGAN



CycleGAN

Domain Translation

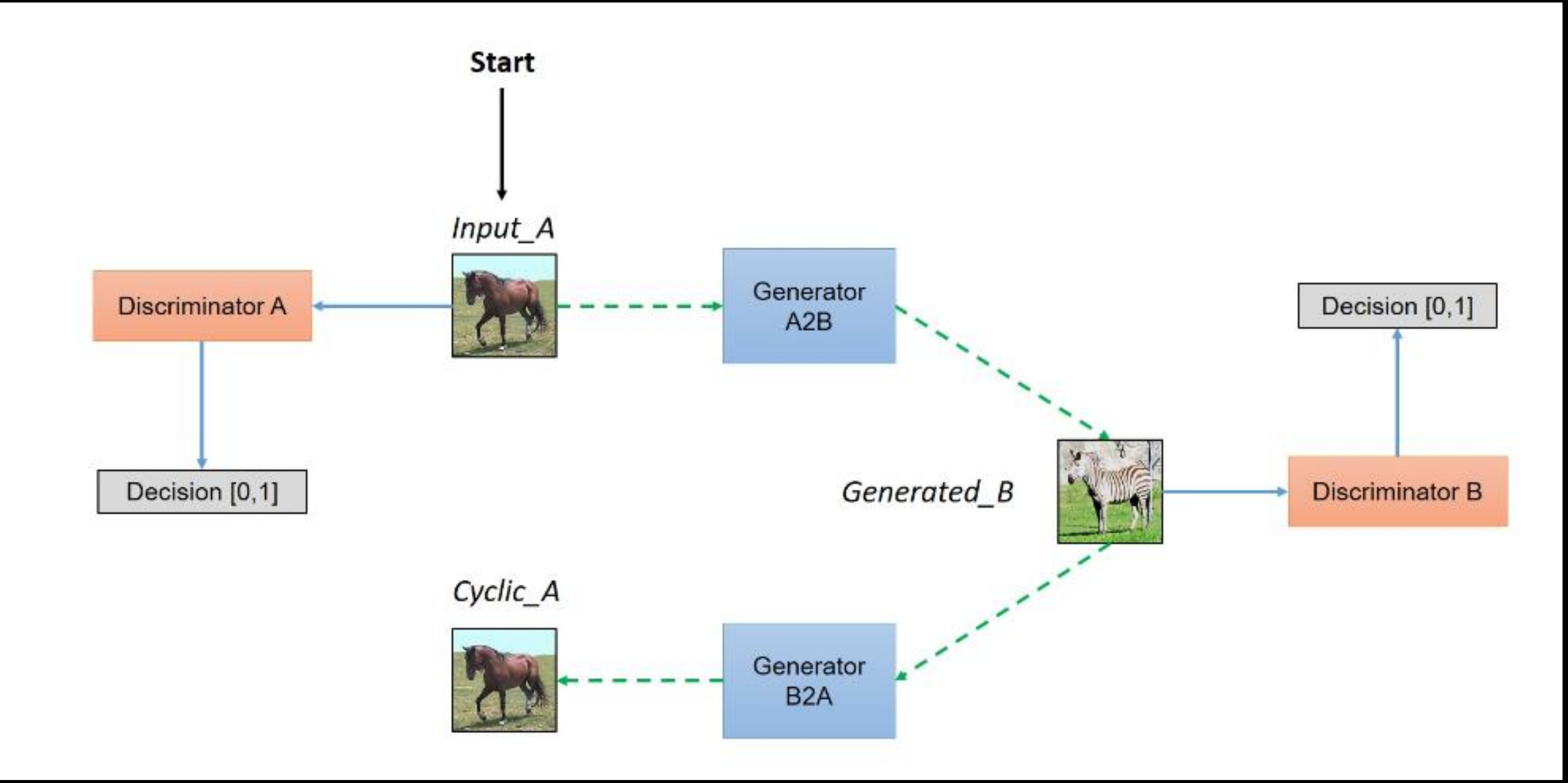


Figure from https://hardikbansal.github.io/CycleGANBlog/

Domain Translation

No noise input

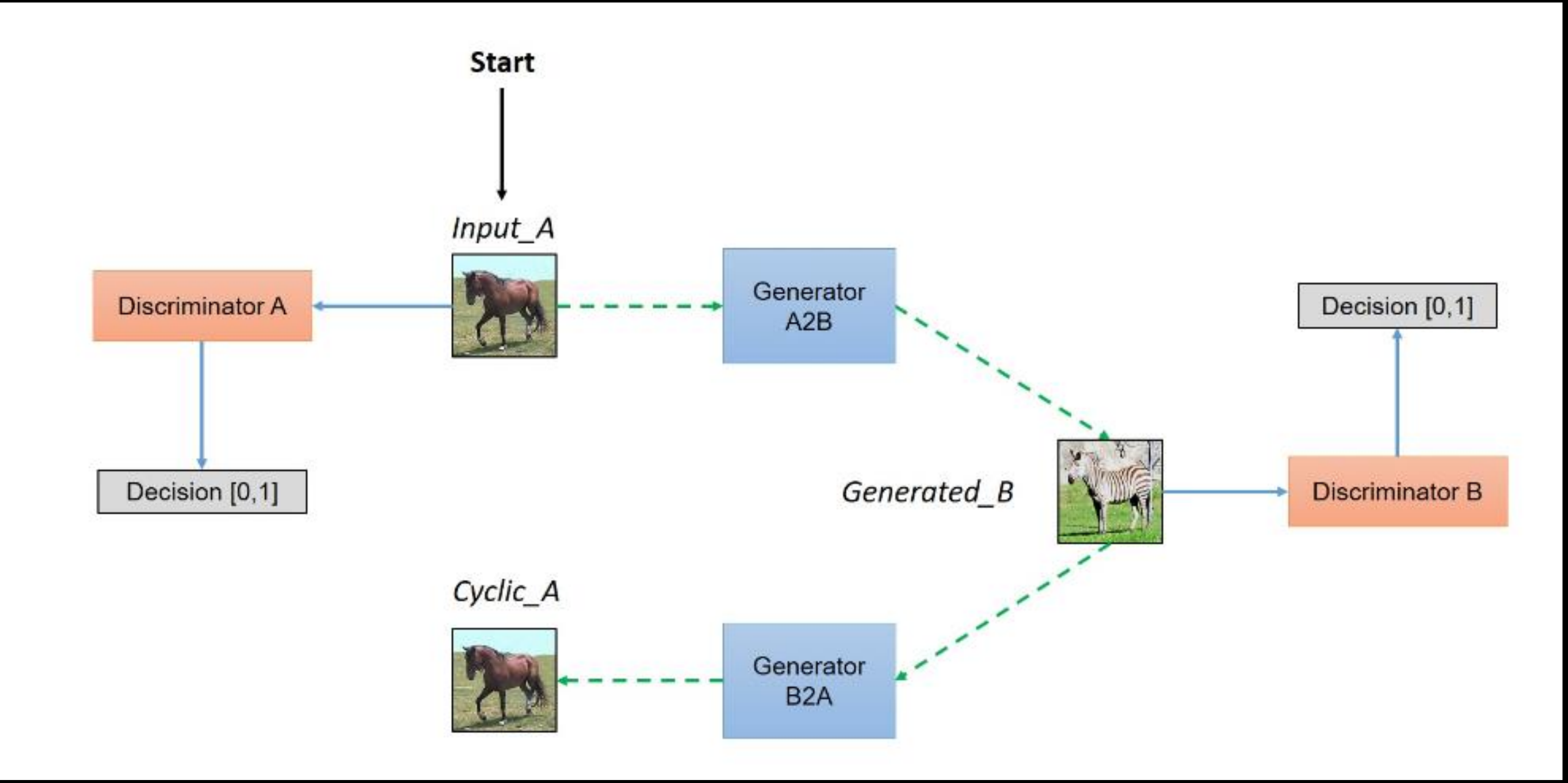


Figure from https://hardikbansal.github.io/CycleGANBlog/

CycleGAN

```python
def set_input(self, input):
    """Unpack input data from the dataloader and perform necessary pre-processing steps.

    Parameters:
        input (dict): include the data itself and its metadata information.

    The option 'direction' can be used to swap domain A and domain B.
    """
    AtoB = self.opt.direction == 'AtoB'
    self.real_A = input['A' if AtoB else 'B'].to(self.device)
    self.real_B = input['B' if AtoB else 'A'].to(self.device)
    self.image_paths = input['A_paths' if AtoB else 'B_paths']

def forward(self):
    """Run forward pass; called by both functions <optimize_parameters> and <test>."""
    self.fake_B = self.netG_A(self.real_A)  # G_A(A)
    self.rec_A = self.netG_B(self.fake_B)   # G_B(G_A(A))
    self.fake_A = self.netG_B(self.real_B)  # G_B(B)
    self.rec_B = self.netG_A(self.fake_A)   # G_A(G_B(B))
```

CycleGAN

```
def backward_D_basic(self, netD, real, fake):
    """Calculate GAN loss for the discriminator

    Parameters:
        netD (network)      -- the discriminator D
        real (tensor array) -- real images
        fake (tensor array) -- images generated by a generator

    Return the discriminator loss.
    We also call loss_D.backward() to calculate the gradients.
    """
    # Real
    pred_real = netD(real)
    loss_D_real = self.criterionGAN(pred_real, True)
    # Fake
    pred_fake = netD(fake.detach())
    loss_D_fake = self.criterionGAN(pred_fake, False)
    # Combined loss and calculate gradients
    loss_D = (loss_D_real + loss_D_fake) * 0.5
    loss_D.backward()
    return loss_D
```

CycleGAN

```
def backward_D_A(self):
    """Calculate GAN loss for discriminator D_A"""
    fake_B = self.fake_B_pool.query(self.fake_B)
    self.loss_D_A = self.backward_D_basic(self.netD_A, self.real_B, fake_B)

def backward_D_B(self):
    """Calculate GAN loss for discriminator D_B"""
    fake_A = self.fake_A_pool.query(self.fake_A)
    self.loss_D_B = self.backward_D_basic(self.netD_B, self.real_A, fake_A)
```

CycleGAN

```python
def backward_G(self):
    """Calculate the loss for generators G_A and G_B"""
    lambda_idt = self.opt.lambda_identity
    lambda_A = self.opt.lambda_A
    lambda_B = self.opt.lambda_B
    # Identity loss
    if lambda_idt > 0:
        # G_A should be identity if real_B is fed: ||G_A(B) - B||
        self.idt_A = self.netG_A(self.real_B)
        self.loss_idt_A = self.criterionIdt(self.idt_A, self.real_B) * lambda_B * lambda_idt
        # G_B should be identity if real_A is fed: ||G_B(A) - A||
        self.idt_B = self.netG_B(self.real_A)
        self.loss_idt_B = self.criterionIdt(self.idt_B, self.real_A) * lambda_A * lambda_idt
    else:
        self.loss_idt_A = 0
        self.loss_idt_B = 0

    # GAN loss D_A(G_A(A))
    self.loss_G_A = self.criterionGAN(self.netD_A(self.fake_B), True)
    # GAN loss D_B(G_B(B))
    self.loss_G_B = self.criterionGAN(self.netD_B(self.fake_A), True)
    # Forward cycle loss || G_B(G_A(A)) - A||
    self.loss_cycle_A = self.criterionCycle(self.rec_A, self.real_A) * lambda_A
    # Backward cycle loss || G_A(G_B(B)) - B||
    self.loss_cycle_B = self.criterionCycle(self.rec_B, self.real_B) * lambda_B
    # combined loss and calculate gradients
    self.loss_G = self.loss_G_A + self.loss_G_B + self.loss_cycle_A + self.loss_cycle_B + self.loss_idt_A + self.loss_idt_B
    self.loss_G.backward()
```

```python
if gan_mode == 'lsgan':
    self.loss = nn.MSELoss()
elif gan_mode == 'vanilla':
    self.loss = nn.BCEWithLogitsLoss()
```

LSGAN

$$\min_G \max_D V_{\text{GAN}}(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))].$$

$$\min_D V_{\text{LSGAN}}(D) = \frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}\big[(D(\boldsymbol{x}) - b)^2\big] + \frac{1}{2}\mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}\big[(D(G(\boldsymbol{z})) - a)^2\big]$$

$$\min_G V_{\text{LSGAN}}(G) = \frac{1}{2}\mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}\big[(D(G(\boldsymbol{z})) - c)^2\big],$$

LSGAN

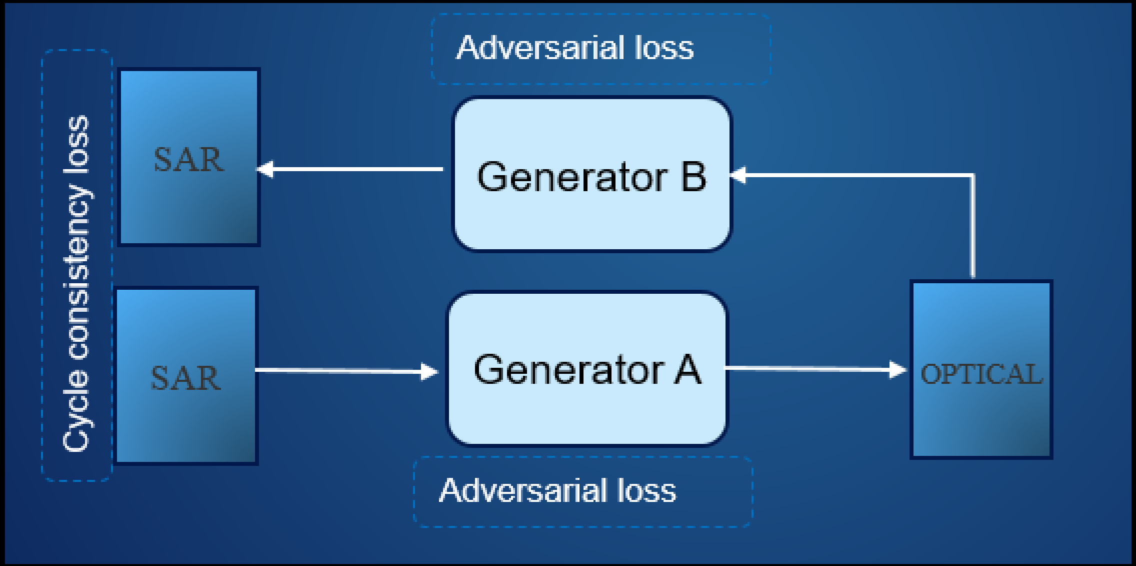
$$\min_G \max_D V_{\text{GAN}}(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))].$$

$$\min_D V_{\text{LSGAN}}(D) = \frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}\left[(D(\boldsymbol{x}) - b)^2\right] + \frac{1}{2}\mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}\left[(D(G(\boldsymbol{z})) - a)^2\right]$$

$$\min_G V_{\text{LSGAN}}(G) = \frac{1}{2}\mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}\left[(D(G(\boldsymbol{z})) - c)^2\right],$$

A possible choice of parameters: a=0, b=1, c=1

SAR and Optical



https://ieeexplore.ieee.org/abstract/document/9120230

Auxiliary Classifier GAN

GAN Evaluation

- Diversity
- Discriminability
- (Related to above two) How good are the intermediate features?

GAN Evaluation

- Amazon Mechanical Turk perception study (see CycleGAN paper)

- FCN score
- Inception Score

Assignment 3

Marks: 15

Problem Statement

- Choose a few distinct image classes of your choice.
- Use a text-to-image generation model to generate synthetic images for each class.
- Divide your synthetic dataset into two splits - training and validation.
- Train a deep learning classifier using only the synthetic dataset.
- Test your classifier on a real dataset consisting of the given classes. Measure performance gap of your model between real dataset and synthetic dataset validation set.
- See if this performance gap can be reduced by merely increasing image numbers in the synthetic dataset or some other simple trick during text to image generation.
- Consider that you have an unlabeled dataset that mostly has images from the classes of your interest but 10% of this dataset are images from other classes.
- Use the above-mentioned unlabeled dataset, potentially with some domain adaptation technique, to further improve the model trained on synthetic data.

Report Format

Refer to IEEE conference (two column) format, please submit 1-2 page report.

Submission Instruction

In .zip folder (code and report), similar to previous Assignment.

Submission Deadline

March 2, 6 pm

```python
import os
import torch
from diffusers import StableDiffusionPipeline

# Define model and device
model_id = "runwayml/stable-diffusion-v1-5"
device = "cuda"

# Initialize the pipeline
pipe = StableDiffusionPipeline.from_pretrained(model_id, torch_dtype=torch.float16)
pipe = pipe.to(device)

# Directories for saving images
tallBuildingsDir = "./diffusionGeneratedBuildings/tallBuildings"



# Define number of images to generate
numImages = 200

# Function to generate and save images
def generate_and_save_images(prompt, folder, num_images):
    for i in range(num_images):
        image = pipe(prompt).images[0]
        image_path = os.path.join(folder, f"{prompt.replace(' ', '_')}_{i + 1}.png")
        image.save(image_path)
        print(f"Saved {image_path}")

# Generate and save images for tall buildings
tallBuildingsPrompt = "top view satellite image of urban scene with tall buildings"
generate_and_save_images(tallBuildingsPrompt, tallBuildingsDir, numImages)
```