main

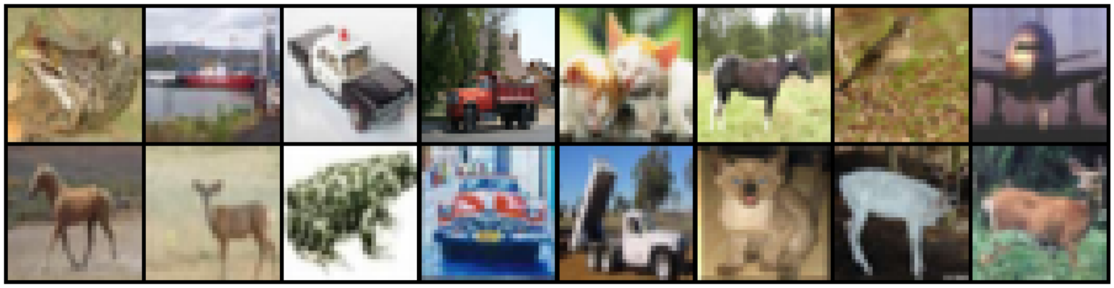
April 2, 2024

[2]:
```python
import torch
from torch import nn
import torchvision
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
import numpy as np
```

[ ]:
```python
batch_size = 16

transform = transforms.Compose([
    transforms.Resize((32, 32)),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
])

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
 ↪download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
 ↪shuffle=True)
```

```
Files already downloaded and verified
```

[ ]:
```python
for i, (imgs, labels) in enumerate(trainloader):
  images = imgs
  break

images = 0.5 * images + 0.5
grid_true = torchvision.utils.make_grid(images, nrow=8, padding=1,
 ↪normalize=False)

plt.figure(figsize=(15, 15))
plt.imshow(np.transpose(grid_true, (1, 2, 0)))
plt.axis('off')
plt.show()
```



[3]:
```python
class Generator(nn.Module):
    def __init__(self):
        super().__init__()
        self.label_emb = nn.Embedding(10, 10)
        self.fc = nn.Sequential(
            nn.Linear(128 + 10, 256),
            nn.LeakyReLU(0.2)
        )
        self.conv_layers = nn.Sequential(
            nn.ConvTranspose2d(256, 128, 4, 2, 1),
            nn.BatchNorm2d(128),
            nn.ReLU(True),

            nn.ConvTranspose2d(128, 64, 4, 2, 1),
            nn.BatchNorm2d(64),
            nn.ReLU(True),

            nn.ConvTranspose2d(64, 32, 4, 2, 1),
            nn.BatchNorm2d(32),
            nn.ReLU(True),

            nn.ConvTranspose2d(32, 16, 4, 2, 1),
            nn.BatchNorm2d(16),
            nn.ReLU(True),

            nn.ConvTranspose2d(16, 3, 4, 2, 1),
            nn.Tanh(),
        )

    def forward(self, z, labels):
        z = z.view(z.size(0), 128)
        c = self.label_emb(labels)
        # c = self.fc(c)
        x = torch.cat([z, c], 1)
        # print("G = ", x.shape)
        x = self.fc(x)
```

```python
        x = x.view(-1, 256, 1, 1)
        return self.conv_layers(x)


class Discriminator(nn.Module):
    def __init__(self):
        super().__init__()
        self.label_emb = nn.Embedding(10, 10)
        self.conv_layers = nn.Sequential(
            nn.Conv2d(3, 16, 4, 2, 1),
            nn.LeakyReLU(0.2, True),

            nn.Conv2d(16, 32, 4, 2, 1),
            nn.BatchNorm2d(32),
            nn.LeakyReLU(0.2, True),

            nn.Conv2d(32, 64, 4, 2, 1),
            nn.BatchNorm2d(64),
            nn.LeakyReLU(0.2, True),

            nn.Conv2d(64, 128, 4, 2, 1),
            nn.BatchNorm2d(128),
            nn.LeakyReLU(0.2, True),

            nn.Conv2d(128, 256, 4, 2, 1),
            nn.BatchNorm2d(256),
            nn.LeakyReLU(0.2, True)
            # nn.Sigmoid()


        )
        self.fc = nn.Sequential(
            nn.Linear(256 + 10, 128),
            nn.LeakyReLU(0.2),
            nn.Linear(128, 1),
            nn.Sigmoid()
        )

    def forward(self, x, labels):
        x = self.conv_layers(x)
        x = x.view(x.size(0), -1)
        c = self.label_emb(labels)
        x = torch.cat([x, c], 1)
        # x = torch.transpose(x, 0, 1)
        # print(x.shape)
        return self.fc(x)
```

[ ]:
```
G = Generator()
D = Discriminator()

criterion = nn.BCELoss()
optim_D = torch.optim.Adam(D.parameters(), lr=0.0002)
optim_G = torch.optim.Adam(G.parameters(), lr=0.0002)
```

[ ]:
```
epochs = 50
latent_dim = 128

for epoch in range(epochs):

  for i, (imgs, labels) in enumerate(trainloader):

      batch_size = imgs.shape[0]
      # print(real_imgs.shape)
      true = torch.ones((batch_size, 1))
      fake = torch.zeros((batch_size, 1))

      G.zero_grad()

      z = torch.randn(batch_size, latent_dim)
      gen_labels = torch.LongTensor(np.random.randint(0, 10, batch_size))

      gen_imgs = G(z, gen_labels)
      # print("Gen imgs = ", gen_imgs.shape)
      # Loss measures generator's ability to fool discriminator
      validity = D(gen_imgs, gen_labels)
      g_loss = criterion(validity, true)

      g_loss.backward()
      optim_G.step()

      D.zero_grad()

      validity_real = D(imgs, labels)
      d_real_loss = criterion(validity_real, true)

      validity_fake = D(gen_imgs.detach(), gen_labels)
      d_fake_loss = criterion(validity_fake, fake)

      d_loss = d_real_loss + d_fake_loss

      d_loss.backward()
      optim_D.step()
```

```python
    print("[Epoch %d/%d] [D loss: %f] [G loss: %f]" % (epoch, epochs, d_loss.
↪item(), g_loss.item()))
```

```
[Epoch 0/50] [D loss: 0.164573] [G loss: 3.329196]
[Epoch 1/50] [D loss: 0.257483] [G loss: 2.547272]
[Epoch 2/50] [D loss: 0.163862] [G loss: 3.975039]
[Epoch 3/50] [D loss: 0.041636] [G loss: 6.801376]
[Epoch 4/50] [D loss: 0.552134] [G loss: 1.744323]
[Epoch 5/50] [D loss: 0.204911] [G loss: 2.848992]
[Epoch 6/50] [D loss: 0.045504] [G loss: 5.052713]
[Epoch 7/50] [D loss: 0.343392] [G loss: 4.544432]
[Epoch 8/50] [D loss: 0.920826] [G loss: 6.756310]
[Epoch 9/50] [D loss: 0.111562] [G loss: 3.586954]
[Epoch 10/50] [D loss: 0.776823] [G loss: 0.874999]
[Epoch 11/50] [D loss: 0.072719] [G loss: 5.472248]
[Epoch 12/50] [D loss: 1.599412] [G loss: 0.528974]
[Epoch 13/50] [D loss: 0.128046] [G loss: 3.700593]
[Epoch 14/50] [D loss: 1.521552] [G loss: 5.323337]
[Epoch 15/50] [D loss: 0.054636] [G loss: 5.556524]
[Epoch 16/50] [D loss: 0.105962] [G loss: 4.669725]
[Epoch 17/50] [D loss: 0.344883] [G loss: 4.368725]
[Epoch 18/50] [D loss: 0.055671] [G loss: 3.954777]
[Epoch 19/50] [D loss: 0.069216] [G loss: 4.714697]
[Epoch 20/50] [D loss: 0.028943] [G loss: 4.665463]
[Epoch 21/50] [D loss: 0.030349] [G loss: 5.697176]
[Epoch 22/50] [D loss: 0.027657] [G loss: 4.558606]
[Epoch 23/50] [D loss: 0.142813] [G loss: 3.753667]
[Epoch 24/50] [D loss: 0.341330] [G loss: 3.010595]
[Epoch 25/50] [D loss: 0.184725] [G loss: 6.353989]
[Epoch 26/50] [D loss: 0.091592] [G loss: 3.875756]
[Epoch 27/50] [D loss: 0.042576] [G loss: 6.379745]
[Epoch 28/50] [D loss: 1.020889] [G loss: 5.324764]
[Epoch 29/50] [D loss: 0.058781] [G loss: 6.667848]
[Epoch 30/50] [D loss: 0.216747] [G loss: 4.292665]
[Epoch 31/50] [D loss: 0.037216] [G loss: 5.989934]
[Epoch 32/50] [D loss: 1.178798] [G loss: 1.130328]
[Epoch 33/50] [D loss: 0.092780] [G loss: 4.096793]
[Epoch 34/50] [D loss: 0.016140] [G loss: 6.844347]
[Epoch 35/50] [D loss: 0.015035] [G loss: 7.139268]
[Epoch 36/50] [D loss: 0.085746] [G loss: 3.911168]
[Epoch 37/50] [D loss: 0.195427] [G loss: 6.113157]
[Epoch 38/50] [D loss: 0.110029] [G loss: 3.522297]
[Epoch 39/50] [D loss: 0.059356] [G loss: 3.556822]
[Epoch 40/50] [D loss: 0.759340] [G loss: 6.028442]
[Epoch 41/50] [D loss: 0.287070] [G loss: 8.441288]
[Epoch 42/50] [D loss: 0.373361] [G loss: 3.273876]
[Epoch 43/50] [D loss: 0.151879] [G loss: 4.045653]
```

```
[Epoch 44/50] [D loss: 1.199750] [G loss: 3.404011]
[Epoch 45/50] [D loss: 0.011130] [G loss: 9.338118]
[Epoch 46/50] [D loss: 0.192980] [G loss: 3.975272]
[Epoch 47/50] [D loss: 0.426174] [G loss: 5.747356]
[Epoch 48/50] [D loss: 0.170626] [G loss: 4.414764]
[Epoch 49/50] [D loss: 1.431220] [G loss: 0.871859]
```

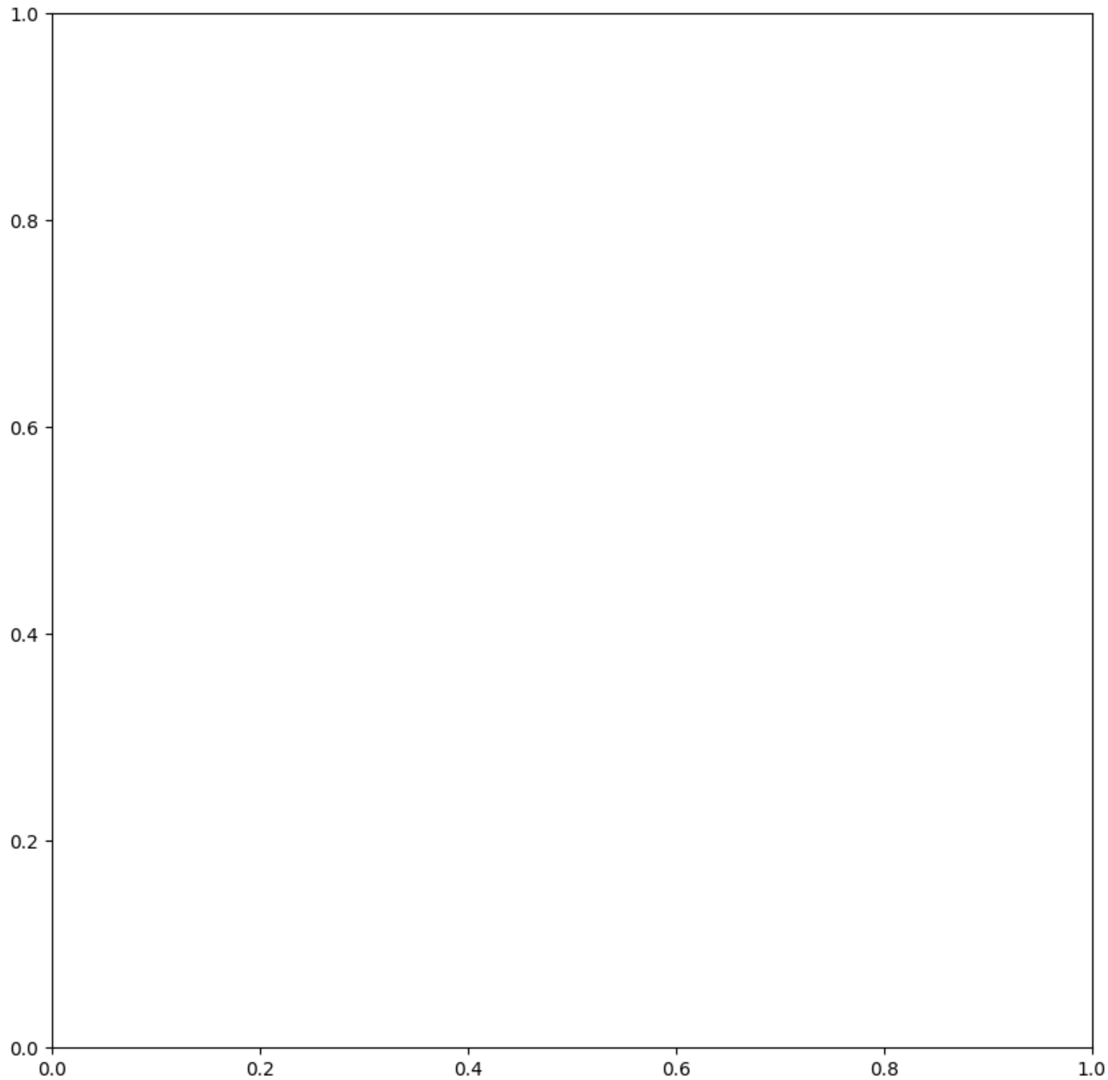
[ ]:
```
G.eval()

noise = torch.randn(32, latent_dim)
random_labels = torch.LongTensor([np.random.randint(0, 5) for _ in range(32)])

with torch.no_grad():
    generated_images = G(noise, random_labels).detach().cpu()

generated_images = 0.5 * generated_images + 0.5

grid = torchvision.utils.make_grid(generated_images, nrow=8, padding=1,␣
 ↪normalize=False)

plt.figure(figsize=(15, 15))
plt.imshow(np.transpose(grid, (1, 2, 0)))
plt.axis('off')
plt.show()
```

[ ]:
```
torch.save(G.state_dict(), 'Generator.pth')
```

[ ]:
```
torch.save(D.state_dict(), 'Discriminator.pth')
```

[ ]:
```python
class_labels = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog',␣
↪'horse', 'ship', 'truck']

for i in range(32):
    image_label = class_labels[random_labels[i]]

    print(str(i) + " = " + str(image_label))
```

```
0 = automobile
1 = airplane
2 = airplane
3 = airplane
4 = automobile
5 = automobile
6 = automobile
7 = cat
8 = bird
9 = bird
10 = bird
11 = bird
12 = deer
13 = cat
14 = airplane
15 = cat
16 = automobile
17 = cat
18 = deer
19 = bird
20 = automobile
21 = airplane
22 = airplane
23 = cat
24 = automobile
25 = automobile
26 = deer
27 = airplane
28 = deer
29 = bird
30 = cat
31 = deer
```



[ ]:
```
from google.colab import widgets
grid = widgets.Grid(4,8)
j=0
for i in range(32):
  with grid.output_to(i//8, i%8):
    image_label = class_labels[random_labels[i]]
    print(image_label)
```

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>

```
<IPython.core.display.Javascript object>

automobile

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>

airplane

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>

airplane

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>

airplane

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>

automobile

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>

automobile

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>

automobile
```

```
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
cat
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
bird
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
bird
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
bird
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
bird
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
deer
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
cat
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
airplane
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
cat
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
automobile
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
cat
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
deer
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
```

```
bird
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
automobile
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
airplane
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
airplane
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
cat
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
automobile
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
automobile
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
deer
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
airplane
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
deer
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
bird
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
cat
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
deer
<IPython.core.display.Javascript object>
```

[ ]:

[2]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[3]:
```
res_folder = "/content/drive/MyDrive/Colab Projects/DL/cfg_res"
dataset_folder = "/content/drive/MyDrive/Colab Projects/DL/train"
check_point_dir = "/content/drive/MyDrive/Colab Projects/DL/cfg_res"
```

[4]:
```
import os
import numpy as np
import torch
import torch.nn as nn
import matplotlib.pyplot as plt
```

[5]:
```
device = "cuda" if torch.cuda.is_available() else "cpu"
```

[6]:
```
def inverse_transform(tensors):
    return (((tensors.clamp(-1, 1) + 1.0) / 2.0) * 255.0).type(torch.uint8)
```

[7]:
```
class DiffusionHelper:
    def __init__(self,
                 inverse_transform,
                 noise_steps = 1000,
                 beta_start = 0.0001,
                 beta_end = 0.02,
                 img_size = 64,
                 device = "cuda",
                 ):
        self.noise_steps = noise_steps
        self.beta_start = beta_start
        self.beta_end = beta_end
        self.img_size = img_size
        self.device = device
        self.inverse_transform = inverse_transform

        self.beta = torch.linspace(self.beta_start, self.beta_end, self.
↪noise_steps).to(device)
        self.alpha = 1 - self.beta

        self.alpha_hat = torch.cumprod(self.alpha, dim=0)


    def add_noise(self, x, t):
        sqrt_alpha_hat = torch.sqrt(self.alpha_hat[t])[:, None, None, None]
        sqrt_one_minus_alpha_hat = torch.sqrt(1-self.alpha_hat[t])[:, None,␣
↪None, None]
```

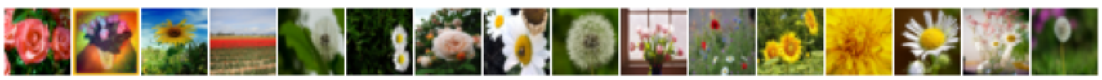
```
        eps = torch.randn_like(x)
        return sqrt_alpha_hat*x + sqrt_one_minus_alpha_hat*eps, eps


    def reverse_diffuse(self, model, batch_size, labels, guidance_strength):
        model.eval()
        with torch.no_grad():
            x = torch.randn((batch_size, 3, self.img_size, self.img_size)).
↪to(self.device)
            for timestep in reversed(range(1, self.noise_steps)):
                t = (torch.ones(batch_size) * timestep).long().to(self.device)

                predicted_noise = model(x, t, labels)
                unconditioned_pred_noise = model(x, t, None)

                predicted_noise = torch.lerp(predicted_noise,␣
↪unconditioned_pred_noise, guidance_strength)

                alpha = self.alpha[t][:, None, None, None]
                alpha_hat = self.alpha_hat[t][:, None, None, None]
                beta = self.beta[t][:, None, None, None]
                noise = torch.randn_like(x) if timestep > 1 else torch.
↪zeros_like(x)
                x = 1 / torch.sqrt(alpha) * (x - ((1 - alpha) / (torch.sqrt(1 -␣
↪alpha_hat))) * predicted_noise) + torch.sqrt(beta) * noise
        model.train()

        return self.inverse_transform(x)
```

[8]:

```
import torchvision.datasets as datasets
from torch.utils.data import Dataset, DataLoader
import torchvision.transforms
from torchvision.datasets import ImageFolder

transforms = torchvision.transforms.Compose(
        [
            torchvision.transforms.Resize((32, 32)),
            torchvision.transforms.ToTensor(),
            torchvision.transforms.Lambda(lambda t: (t * 2) - 1)
        ]
    )


# dataset = datasets.CIFAR10(root="data", train=True, download=True,␣
↪transform=transforms)
dataset = ImageFolder(dataset_folder, transform=transforms)
```

[9]:
```
dataloader = DataLoader(dataset, batch_size=16,
                            shuffle=True)
vis_loader = DataLoader(dataset, batch_size=6,
                            shuffle=True)
```

[10]:
```
from torchvision.utils import make_grid

plt.figure(figsize=(10, 4), facecolor='white')

for b_image, _ in dataloader:
    b_image = inverse_transform(b_image)
    grid_img = make_grid(b_image / 255.0, nrow=16, padding=True, pad_value=1)
    plt.imshow(grid_img.permute(1, 2, 0))
    plt.axis("off")
    break
```

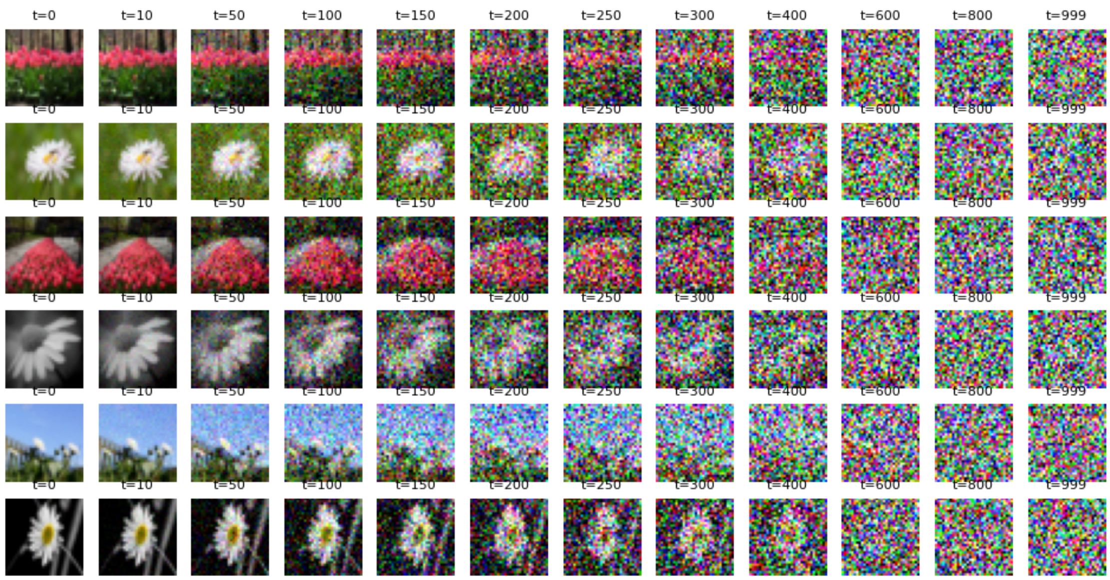


[11]:
```
diffusion_helper = DiffusionHelper(inverse_transform=inverse_transform, img_size=32, device=device)
```

[12]:
```
noisy_images = []
specific_timesteps = [0, 10, 50, 100, 150, 200, 250, 300, 400, 600, 800, 999]

images, labels = next(iter(vis_loader))

for timestep in specific_timesteps:
    timestep = torch.as_tensor([[[timestep]]], dtype=torch.long)

    xts, _ = diffusion_helper.add_noise(images.to(device), timestep)
    xts = inverse_transform(xts[:, None, None, None])

    noisy_images.append(xts)
```

[13]:
```
_, ax = plt.subplots(6, len(noisy_images), figsize=(12, 6))

for j, (timestep, noisy_sample) in enumerate(zip(specific_timesteps, noisy_images)):
    noisy_sample = torch.squeeze(noisy_sample, dim=(0, 1, 2, 3, 4))
    for i in range(len(noisy_sample)):
        ax[i][j].imshow((noisy_sample[i].permute(1, 2, 0)/255.0).cpu())
        ax[i][j].set_title(f"t={timestep}", fontsize=8)
        ax[i][j].axis("off")
```

```
        ax[i][j].grid(False)

plt.axis("off")
plt.show()
```



[14]:
```
NUM_CLASSES = len(np.unique(dataloader.dataset.targets))
```

[15]:
```
class SelfAttention(nn.Module):
    def __init__(self, channels, size):
        super(SelfAttention, self).__init__()
        self.channels = channels
        self.size = size
        self.mha = nn.MultiheadAttention(channels, 4, batch_first=True)
        self.ln = nn.LayerNorm([channels])
        self.ff_self = nn.Sequential(
            nn.LayerNorm([channels]),
            nn.Linear(channels, channels),
            nn.GELU(),
            nn.Linear(channels, channels),
        )

    def forward(self, x):
        x = x.view(-1, self.channels, self.size * self.size).swapaxes(1, 2)
        x_ln = self.ln(x)
        attention_value, _ = self.mha(x_ln, x_ln, x_ln)
        attention_value = attention_value + x
        attention_value = self.ff_self(attention_value) + attention_value
```

```
        return attention_value.swapaxes(2, 1).view(-1, self.channels, self.
↪size, self.size)
```

[16]:
```
class ResidualDoubleConv(nn.Module):
    def __init__(self, in_channels, out_channels, mid_channels):
        super(ResidualDoubleConv, self).__init__()

        self.double_conv = nn.Sequential(
            nn.Conv2d(in_channels, mid_channels, kernel_size=3, padding=1,␣
↪bias=False),
            nn.GroupNorm(1, mid_channels),
            nn.GELU(),
            nn.Conv2d(mid_channels, out_channels, kernel_size=3, padding=1,␣
↪bias=False),
            nn.GroupNorm(1, out_channels),
        )

    def forward(self, x):
        return nn.GELU()(x + self.double_conv(x))
```

[17]:
```
class DoubleConv(nn.Module):
    def __init__(self, in_channels, out_channels, mid_channels):
        super(DoubleConv, self).__init__()

        self.double_conv = nn.Sequential(
            nn.Conv2d(in_channels, mid_channels, kernel_size=3, padding=1,␣
↪bias=False),
            nn.GroupNorm(1, mid_channels),
            nn.GELU(),
            nn.Conv2d(mid_channels, out_channels, kernel_size=3, padding=1,␣
↪bias=False),
            nn.GroupNorm(1, out_channels),
        )

    def forward(self, x):
       return self.double_conv(x)
```

[18]:
```
class Up(nn.Module):
    def __init__(self, in_channels, out_channels, emb_dim=256):
        super().__init__()

        self.up = nn.Upsample(scale_factor=2, mode="bilinear",␣
↪align_corners=True)
        self.conv = nn.Sequential(
            ResidualDoubleConv(in_channels, in_channels, in_channels),
            DoubleConv(in_channels, out_channels, in_channels // 2),
```

```
        )

        self.emb_layer = nn.Sequential(
            nn.SiLU(),
            nn.Linear(
                emb_dim,
                out_channels
            ),
        )

    def forward(self, x, skip_x, t):
        x = self.up(x)
        x = torch.cat([skip_x, x], dim=1)
        x = self.conv(x)
        emb = self.emb_layer(t)[:, :, None, None].repeat(1, 1, x.shape[-2], x.
 ↪shape[-1])
        return x + emb
```

[19]:
```
class Down(nn.Module):
    def __init__(self, in_channels, out_channels, emb_dim=256):
        super().__init__()
        self.maxpool_conv = nn.Sequential(
            nn.MaxPool2d(2),
            ResidualDoubleConv(in_channels, in_channels, in_channels),
            DoubleConv(in_channels, out_channels, out_channels),
        )

        self.emb_layer = nn.Sequential(
            nn.SiLU(),
            nn.Linear(
                emb_dim,
                out_channels
            ),
        )

    def forward(self, x, t):
        x = self.maxpool_conv(x)
        emb = self.emb_layer(t)[:, :, None, None].repeat(1, 1, x.shape[-2], x.
 ↪shape[-1])
        return x + emb
```

[20]:
```
class UNetEncoder(nn.Module):
    def __init__(self):
        super(UNetEncoder, self).__init__()

        self.down1 = Down(64, 128)
        self.sa1 = SelfAttention(128, 16)
```

```
        self.down2 = Down(128, 128)
        self.sa2 = SelfAttention(128, 8)


    def forward(self, x, t):

        x2 = self.down1(x, t)
        x2 = self.sa1(x2)
        x3 = self.down2(x2, t)
        x3 = self.sa2(x3)

        return x2, x3
```

[21]:
```
class UNetBottleneck(nn.Module):
    def __init__(self):
        super(UNetBottleneck, self).__init__()
        self.bot1 = DoubleConv(128, 256, 256)
        self.bot2 = DoubleConv(256, 256, 256)
        self.bot3 = DoubleConv(256, 128, 128)

    def forward(self, x):
        x = self.bot1(x)
        x = self.bot2(x)
        x = self.bot3(x)
        return x
```

[22]:
```
class UNetDecoder(nn.Module):
    def __init__(self):
        super(UNetDecoder, self).__init__()
        self.up1 = Up(256, 64)
        self.sa3 = SelfAttention(64, 16)
        self.up2 = Up(128, 32)
        self.sa4 = SelfAttention(32, 32)

    def forward(self, x1, x2, x3, t):
        x = self.up1(x3, x2, t)
        x = self.sa3(x)
        x = self.up2(x, x1, t)
        x = self.sa4(x)
        return x
```

[23]:
```
class UNet(nn.Module):
    def __init__(self, c_in=3, c_out=3, time_dim=256, num_classes = None,␣
 ↪device="cuda"):
        super().__init__()
        self.device = device
        self.time_dim = time_dim
```

```
        self.inc = DoubleConv(c_in, 64, 64)

        self.encoder = UNetEncoder()
        self.bottleneck = UNetBottleneck()
        self.decoder = UNetDecoder()

        self.outc = nn.Conv2d(32, c_out, kernel_size=1)

        if num_classes:
            self.label_emb = nn.Embedding(num_classes, time_dim)

    def pos_encoding(self, t, channels):
        inv_freq = 1.0 / (
            10000
            ** (torch.arange(0, channels, 2, device=self.device).float() /
 ↪channels)
        )
        pos_enc_a = torch.sin(t.repeat(1, channels // 2) * inv_freq)
        pos_enc_b = torch.cos(t.repeat(1, channels // 2) * inv_freq)
        pos_enc = torch.cat([pos_enc_a, pos_enc_b], dim=-1)
        return pos_enc

    def forward(self, x, t, y):
        t = t.unsqueeze(-1).type(torch.float)
        t = self.pos_encoding(t, self.time_dim)

        if y is not None:
            t += self.label_emb(y)

        x1 = self.inc(x)

        x2, x3 = self.encoder(x1, t)
        x3 = self.bottleneck(x3)
        x = self.decoder(x1, x2, x3, t)

        output = self.outc(x)
        return output
```

[24]:
```
model = UNet(3, 3, num_classes=NUM_CLASSES).to(device)
```

[29]:
```
from PIL import Image

def save_images(images, path, **kwargs):
  grid = torchvision.utils.make_grid(images, **kwargs)
  ndarr = grid.permute(1, 2, 0).to('cpu').numpy()
  im = Image.fromarray(ndarr)
```

```
    im.save(path)
```

[25]:
```
EPOCHS = 500
criterion = nn.MSELoss()
optim = torch.optim.Adam(model.parameters(), lr=2e-4)

loss_list = []

for epoch in range(EPOCHS):
    batch = 1
    for x0s, labels in dataloader:
        ts = torch.randint(low=1, high=1000, size=(x0s.shape[0],),␣
  ↪device=device)

        xts, added_noise = diffusion_helper.add_noise(x0s.to(device), ts)

        labels = labels.to(device)

        if torch.rand(1) < 0.1:
           labels = None

        pred_noise = model(xts, ts, labels)

        loss = criterion(added_noise, pred_noise)

        optim.zero_grad()
        loss.backward()
        optim.step()

        # if batch % 100 == 0:
        #   print(f"Epoch {epoch+1}, Batch {batch}/{len(dataloader)}")

        batch += 1

    print(f"Epoch {epoch+1}, Loss: {loss.item():.4f}")
    loss_list.append(round(loss.item(), 5))

    if (epoch+1) % 50 == 0:
      samp_images = diffusion_helper.reverse_diffuse(model, x0s.shape[0],␣
  ↪labels, 3)
      save_images(samp_images, os.path.join(res_folder, f"{epoch+1}.png"))
      torch.save(model.state_dict(), os.path.
  ↪join(check_point_dir,f"{epoch+1}_ckpt.pt"))


print("Training done")
```

```
Epoch 1, Loss: 0.1499
Epoch 2, Loss: 0.0868
Epoch 3, Loss: 0.0423
Epoch 4, Loss: 0.0615
Epoch 5, Loss: 0.0643
Epoch 6, Loss: 0.1839
Epoch 7, Loss: 0.0265
Epoch 8, Loss: 0.1686
Epoch 9, Loss: 0.0617
Epoch 10, Loss: 0.0228
Epoch 11, Loss: 0.0219
Epoch 12, Loss: 0.0893
Epoch 13, Loss: 0.0267
Epoch 14, Loss: 0.0199
Epoch 15, Loss: 0.0197
Epoch 16, Loss: 0.0185
Epoch 17, Loss: 0.0572
Epoch 18, Loss: 0.0660
Epoch 19, Loss: 0.0382
Epoch 20, Loss: 0.0292
Epoch 21, Loss: 0.0392
Epoch 22, Loss: 0.0391
Epoch 23, Loss: 0.1008
Epoch 24, Loss: 0.0960
Epoch 25, Loss: 0.0397
Epoch 26, Loss: 0.0407
Epoch 27, Loss: 0.0688
Epoch 28, Loss: 0.0202
Epoch 29, Loss: 0.1521
Epoch 30, Loss: 0.0182
Epoch 31, Loss: 0.0936
Epoch 32, Loss: 0.0445
Epoch 33, Loss: 0.0141
Epoch 34, Loss: 0.0183
Epoch 35, Loss: 0.0208
Epoch 36, Loss: 0.0637
Epoch 37, Loss: 0.0480
Epoch 38, Loss: 0.0429
Epoch 39, Loss: 0.0105
Epoch 40, Loss: 0.0794
Epoch 41, Loss: 0.0553
Epoch 42, Loss: 0.0376
Epoch 43, Loss: 0.0370
Epoch 44, Loss: 0.0165
Epoch 45, Loss: 0.0829
Epoch 46, Loss: 0.0583
Epoch 47, Loss: 0.0722
Epoch 48, Loss: 0.0333
```

```
Epoch 49, Loss: 0.0231
Epoch 50, Loss: 0.0198
Epoch 51, Loss: 0.0222
Epoch 52, Loss: 0.0578
Epoch 53, Loss: 0.0812
Epoch 54, Loss: 0.0192
Epoch 55, Loss: 0.0354
Epoch 56, Loss: 0.0428
Epoch 57, Loss: 0.0725
Epoch 58, Loss: 0.0295
Epoch 59, Loss: 0.0534
Epoch 60, Loss: 0.0456
Epoch 61, Loss: 0.0211
Epoch 62, Loss: 0.0293
Epoch 63, Loss: 0.0068
Epoch 64, Loss: 0.0215
Epoch 65, Loss: 0.0178
Epoch 66, Loss: 0.0363
Epoch 67, Loss: 0.0731
Epoch 68, Loss: 0.0514
Epoch 69, Loss: 0.0209
Epoch 70, Loss: 0.0476
Epoch 71, Loss: 0.0402
Epoch 72, Loss: 0.0306
Epoch 73, Loss: 0.0183
Epoch 74, Loss: 0.0233
Epoch 75, Loss: 0.0126
Epoch 76, Loss: 0.0500
Epoch 77, Loss: 0.0390
Epoch 78, Loss: 0.0718
Epoch 79, Loss: 0.0387
Epoch 80, Loss: 0.0983
Epoch 81, Loss: 0.0052
Epoch 82, Loss: 0.0762
Epoch 83, Loss: 0.0267
Epoch 84, Loss: 0.0137
Epoch 85, Loss: 0.0156
Epoch 86, Loss: 0.1061
Epoch 87, Loss: 0.0797
Epoch 88, Loss: 0.0131
Epoch 89, Loss: 0.0377
Epoch 90, Loss: 0.0171
Epoch 91, Loss: 0.0374
Epoch 92, Loss: 0.0357
Epoch 93, Loss: 0.0125
Epoch 94, Loss: 0.0584
Epoch 95, Loss: 0.0241
Epoch 96, Loss: 0.0464
```

```
Epoch 97, Loss: 0.0283
Epoch 98, Loss: 0.0853
Epoch 99, Loss: 0.0079
Epoch 100, Loss: 0.0290
Epoch 101, Loss: 0.0193
Epoch 102, Loss: 0.0241
Epoch 103, Loss: 0.0367
Epoch 104, Loss: 0.0165
Epoch 105, Loss: 0.0575
Epoch 106, Loss: 0.0140
Epoch 107, Loss: 0.0591
Epoch 108, Loss: 0.0251
Epoch 109, Loss: 0.0177
Epoch 110, Loss: 0.0232
Epoch 111, Loss: 0.0291
Epoch 112, Loss: 0.0024
Epoch 113, Loss: 0.0429
Epoch 114, Loss: 0.0577
Epoch 115, Loss: 0.0427
Epoch 116, Loss: 0.0120
Epoch 117, Loss: 0.0855
Epoch 118, Loss: 0.0180
Epoch 119, Loss: 0.0328
Epoch 120, Loss: 0.0390
Epoch 121, Loss: 0.0114
Epoch 122, Loss: 0.0307
Epoch 123, Loss: 0.0596
Epoch 124, Loss: 0.0336
Epoch 125, Loss: 0.0370
Epoch 126, Loss: 0.0700
Epoch 127, Loss: 0.0277
Epoch 128, Loss: 0.0270
Epoch 129, Loss: 0.0143
Epoch 130, Loss: 0.0258
Epoch 131, Loss: 0.0354
Epoch 132, Loss: 0.0438
Epoch 133, Loss: 0.0305
Epoch 134, Loss: 0.0544
Epoch 135, Loss: 0.0485
Epoch 136, Loss: 0.0052
Epoch 137, Loss: 0.0713
Epoch 138, Loss: 0.0451
Epoch 139, Loss: 0.0050
Epoch 140, Loss: 0.0410
Epoch 141, Loss: 0.0297
Epoch 142, Loss: 0.0667
Epoch 143, Loss: 0.0608
Epoch 144, Loss: 0.0703
```

```
Epoch 145, Loss: 0.0616
Epoch 146, Loss: 0.0140
Epoch 147, Loss: 0.0231
Epoch 148, Loss: 0.0382
Epoch 149, Loss: 0.0539
Epoch 150, Loss: 0.0146
Epoch 151, Loss: 0.0029
Epoch 152, Loss: 0.0060
Epoch 153, Loss: 0.1160
Epoch 154, Loss: 0.0124
Epoch 155, Loss: 0.0253
Epoch 156, Loss: 0.0444
Epoch 157, Loss: 0.0365
Epoch 158, Loss: 0.0508
Epoch 159, Loss: 0.0043
Epoch 160, Loss: 0.0884
Epoch 161, Loss: 0.0204
Epoch 162, Loss: 0.0260
Epoch 163, Loss: 0.0425
Epoch 164, Loss: 0.1267
Epoch 165, Loss: 0.0320
Epoch 166, Loss: 0.0212
Epoch 167, Loss: 0.0328
Epoch 168, Loss: 0.0639
Epoch 169, Loss: 0.0197
Epoch 170, Loss: 0.0466
Epoch 171, Loss: 0.0306
Epoch 172, Loss: 0.0708
Epoch 173, Loss: 0.0471
Epoch 174, Loss: 0.0446
Epoch 175, Loss: 0.0329
Epoch 176, Loss: 0.0214
Epoch 177, Loss: 0.0829
Epoch 178, Loss: 0.0245
Epoch 179, Loss: 0.0441
Epoch 180, Loss: 0.0490
Epoch 181, Loss: 0.0205
Epoch 182, Loss: 0.0205
Epoch 183, Loss: 0.0572
Epoch 184, Loss: 0.0220
Epoch 185, Loss: 0.0164
Epoch 186, Loss: 0.0258
Epoch 187, Loss: 0.0152
Epoch 188, Loss: 0.0244
Epoch 189, Loss: 0.0161
Epoch 190, Loss: 0.0265
Epoch 191, Loss: 0.0336
Epoch 192, Loss: 0.0079
```

```
Epoch 193, Loss: 0.0469
Epoch 194, Loss: 0.0232
Epoch 195, Loss: 0.0781
Epoch 196, Loss: 0.0170
Epoch 197, Loss: 0.0610
Epoch 198, Loss: 0.0204
Epoch 199, Loss: 0.0335
Epoch 200, Loss: 0.1180
Epoch 201, Loss: 0.0315
Epoch 202, Loss: 0.0392
Epoch 203, Loss: 0.0363
Epoch 204, Loss: 0.0086
Epoch 205, Loss: 0.0210
Epoch 206, Loss: 0.0220
Epoch 207, Loss: 0.0515
Epoch 208, Loss: 0.0120
Epoch 209, Loss: 0.0162
Epoch 210, Loss: 0.0567
Epoch 211, Loss: 0.0420
Epoch 212, Loss: 0.0058
Epoch 213, Loss: 0.0511
Epoch 214, Loss: 0.0500
Epoch 215, Loss: 0.0346
Epoch 216, Loss: 0.0809
Epoch 217, Loss: 0.0229
Epoch 218, Loss: 0.0561
Epoch 219, Loss: 0.0406
Epoch 220, Loss: 0.0483
Epoch 221, Loss: 0.0169
Epoch 222, Loss: 0.0431
Epoch 223, Loss: 0.0319
Epoch 224, Loss: 0.0366
Epoch 225, Loss: 0.0339
Epoch 226, Loss: 0.0750
Epoch 227, Loss: 0.1153
Epoch 228, Loss: 0.0119
Epoch 229, Loss: 0.0126
Epoch 230, Loss: 0.0531
Epoch 231, Loss: 0.0180
Epoch 232, Loss: 0.0517
Epoch 233, Loss: 0.0560
Epoch 234, Loss: 0.0262
Epoch 235, Loss: 0.0548
Epoch 236, Loss: 0.0323
Epoch 237, Loss: 0.0250
Epoch 238, Loss: 0.0798
Epoch 239, Loss: 0.0341
Epoch 240, Loss: 0.0730
```

```
Epoch 241, Loss: 0.0238
Epoch 242, Loss: 0.0164
Epoch 243, Loss: 0.0100
Epoch 244, Loss: 0.0418
Epoch 245, Loss: 0.0181
Epoch 246, Loss: 0.0268
Epoch 247, Loss: 0.0190
Epoch 248, Loss: 0.0114
Epoch 249, Loss: 0.0351
Epoch 250, Loss: 0.0294
Epoch 251, Loss: 0.0150
Epoch 252, Loss: 0.0145
Epoch 253, Loss: 0.0194
Epoch 254, Loss: 0.0199
Epoch 255, Loss: 0.0760
Epoch 256, Loss: 0.0145
Epoch 257, Loss: 0.0272
Epoch 258, Loss: 0.0160
Epoch 259, Loss: 0.0109
Epoch 260, Loss: 0.0057
Epoch 261, Loss: 0.0822
Epoch 262, Loss: 0.0587
Epoch 263, Loss: 0.0793
Epoch 264, Loss: 0.0060
Epoch 265, Loss: 0.0951
Epoch 266, Loss: 0.0744
Epoch 267, Loss: 0.0080
Epoch 268, Loss: 0.0204
Epoch 269, Loss: 0.0440
Epoch 270, Loss: 0.0300
Epoch 271, Loss: 0.0381
Epoch 272, Loss: 0.0326
Epoch 273, Loss: 0.0490
Epoch 274, Loss: 0.0311
Epoch 275, Loss: 0.0148
Epoch 276, Loss: 0.0154
Epoch 277, Loss: 0.0164
Epoch 278, Loss: 0.0156
Epoch 279, Loss: 0.0348
Epoch 280, Loss: 0.0247
Epoch 281, Loss: 0.0260
Epoch 282, Loss: 0.0406
Epoch 283, Loss: 0.0122
Epoch 284, Loss: 0.0409
Epoch 285, Loss: 0.0073
Epoch 286, Loss: 0.0993
Epoch 287, Loss: 0.0192
Epoch 288, Loss: 0.0660
```

```
Epoch 289, Loss: 0.0592
Epoch 290, Loss: 0.0273
Epoch 291, Loss: 0.0400
Epoch 292, Loss: 0.0486
Epoch 293, Loss: 0.1019
Epoch 294, Loss: 0.0657
Epoch 295, Loss: 0.0427
Epoch 296, Loss: 0.0112
Epoch 297, Loss: 0.0483
Epoch 298, Loss: 0.0381
Epoch 299, Loss: 0.0332
Epoch 300, Loss: 0.0397
Epoch 301, Loss: 0.0189
Epoch 302, Loss: 0.1018
Epoch 303, Loss: 0.0444
Epoch 304, Loss: 0.0466
Epoch 305, Loss: 0.0830
Epoch 306, Loss: 0.0105
Epoch 307, Loss: 0.0126
Epoch 308, Loss: 0.0306
Epoch 309, Loss: 0.0245
Epoch 310, Loss: 0.0164
Epoch 311, Loss: 0.0374
Epoch 312, Loss: 0.0403
Epoch 313, Loss: 0.0619
Epoch 314, Loss: 0.0844
Epoch 315, Loss: 0.0254
Epoch 316, Loss: 0.0537
Epoch 317, Loss: 0.0327
Epoch 318, Loss: 0.0087
Epoch 319, Loss: 0.0165
Epoch 320, Loss: 0.0903
Epoch 321, Loss: 0.0436
Epoch 322, Loss: 0.0233
Epoch 323, Loss: 0.0103
Epoch 324, Loss: 0.0862
Epoch 325, Loss: 0.0743
Epoch 326, Loss: 0.0136
Epoch 327, Loss: 0.0739
Epoch 328, Loss: 0.0070
Epoch 329, Loss: 0.0276
Epoch 330, Loss: 0.0137
Epoch 331, Loss: 0.0149
Epoch 332, Loss: 0.0816
Epoch 333, Loss: 0.0323
Epoch 334, Loss: 0.0183
Epoch 335, Loss: 0.0339
Epoch 336, Loss: 0.0077
```

```
Epoch 337, Loss: 0.0163
Epoch 338, Loss: 0.0993
Epoch 339, Loss: 0.0213
Epoch 340, Loss: 0.1236
Epoch 341, Loss: 0.0825
Epoch 342, Loss: 0.0264
Epoch 343, Loss: 0.0290
Epoch 344, Loss: 0.0425
Epoch 345, Loss: 0.0211
Epoch 346, Loss: 0.0045
Epoch 347, Loss: 0.0262
Epoch 348, Loss: 0.0206
Epoch 349, Loss: 0.0237
Epoch 350, Loss: 0.0524
Epoch 351, Loss: 0.0229
Epoch 352, Loss: 0.0561
Epoch 353, Loss: 0.0145
Epoch 354, Loss: 0.0842
Epoch 355, Loss: 0.0080
Epoch 356, Loss: 0.0287
Epoch 357, Loss: 0.0486
Epoch 358, Loss: 0.0154
Epoch 359, Loss: 0.0269
Epoch 360, Loss: 0.0624
Epoch 361, Loss: 0.1025
Epoch 362, Loss: 0.0438
Epoch 363, Loss: 0.0393
Epoch 364, Loss: 0.0526
Epoch 365, Loss: 0.0043
Epoch 366, Loss: 0.0371
Epoch 367, Loss: 0.0307
Epoch 368, Loss: 0.0149
Epoch 369, Loss: 0.0787
Epoch 370, Loss: 0.0981
Epoch 371, Loss: 0.0346
Epoch 372, Loss: 0.0491
Epoch 373, Loss: 0.1224
Epoch 374, Loss: 0.0077
Epoch 375, Loss: 0.0135
Epoch 376, Loss: 0.0066
Epoch 377, Loss: 0.0198
Epoch 378, Loss: 0.0222
Epoch 379, Loss: 0.0179
Epoch 380, Loss: 0.0296
Epoch 381, Loss: 0.0147
Epoch 382, Loss: 0.0664
Epoch 383, Loss: 0.0153
Epoch 384, Loss: 0.0812
```

```
Epoch 385, Loss: 0.0848
Epoch 386, Loss: 0.0174
Epoch 387, Loss: 0.0176
Epoch 388, Loss: 0.0392
Epoch 389, Loss: 0.0173
Epoch 390, Loss: 0.0292
Epoch 391, Loss: 0.0154
Epoch 392, Loss: 0.0370
Epoch 393, Loss: 0.0240
Epoch 394, Loss: 0.0528
Epoch 395, Loss: 0.0381
Epoch 396, Loss: 0.0628
Epoch 397, Loss: 0.0416
Epoch 398, Loss: 0.0091
Epoch 399, Loss: 0.0779
Epoch 400, Loss: 0.0347
Epoch 401, Loss: 0.1012
Epoch 402, Loss: 0.0932
Epoch 403, Loss: 0.0818
Epoch 404, Loss: 0.0101
Epoch 405, Loss: 0.0586
Epoch 406, Loss: 0.0868
Epoch 407, Loss: 0.0270
Epoch 408, Loss: 0.0282
Epoch 409, Loss: 0.0562
Epoch 410, Loss: 0.0480
Epoch 411, Loss: 0.0265
Epoch 412, Loss: 0.0320
Epoch 413, Loss: 0.0527
Epoch 414, Loss: 0.0429
Epoch 415, Loss: 0.0373
Epoch 416, Loss: 0.0746
Epoch 417, Loss: 0.0363
Epoch 418, Loss: 0.0310
Epoch 419, Loss: 0.0765
Epoch 420, Loss: 0.0367
Epoch 421, Loss: 0.0636
Epoch 422, Loss: 0.0402
Epoch 423, Loss: 0.0478
Epoch 424, Loss: 0.0543
Epoch 425, Loss: 0.0499
Epoch 426, Loss: 0.0899
Epoch 427, Loss: 0.0294
Epoch 428, Loss: 0.0321
Epoch 429, Loss: 0.0048
Epoch 430, Loss: 0.0424
Epoch 431, Loss: 0.0577
Epoch 432, Loss: 0.0489
```

```
Epoch 433, Loss: 0.0352
Epoch 434, Loss: 0.0468
Epoch 435, Loss: 0.0229
Epoch 436, Loss: 0.1057
Epoch 437, Loss: 0.0378
Epoch 438, Loss: 0.0349
Epoch 439, Loss: 0.0732
Epoch 440, Loss: 0.0707
Epoch 441, Loss: 0.0454
Epoch 442, Loss: 0.0238
Epoch 443, Loss: 0.0145
Epoch 444, Loss: 0.0837
Epoch 445, Loss: 0.0193
Epoch 446, Loss: 0.0492
Epoch 447, Loss: 0.0248
Epoch 448, Loss: 0.0416
Epoch 449, Loss: 0.0220
Epoch 450, Loss: 0.0361
Epoch 451, Loss: 0.0092
Epoch 452, Loss: 0.0160
Epoch 453, Loss: 0.0323
Epoch 454, Loss: 0.0573
Epoch 455, Loss: 0.0222
Epoch 456, Loss: 0.0252
Epoch 457, Loss: 0.0207
Epoch 458, Loss: 0.1419
Epoch 459, Loss: 0.0099
Epoch 460, Loss: 0.0179
Epoch 461, Loss: 0.0419
Epoch 462, Loss: 0.1039
Epoch 463, Loss: 0.0545
Epoch 464, Loss: 0.0136
Epoch 465, Loss: 0.0601
Epoch 466, Loss: 0.0158
Epoch 467, Loss: 0.0221
Epoch 468, Loss: 0.0084
Epoch 469, Loss: 0.0183
Epoch 470, Loss: 0.0296
Epoch 471, Loss: 0.0145
Epoch 472, Loss: 0.0365
Epoch 473, Loss: 0.0270
Epoch 474, Loss: 0.0267
Epoch 475, Loss: 0.0368
Epoch 476, Loss: 0.0303
Epoch 477, Loss: 0.1073
Epoch 478, Loss: 0.0160
Epoch 479, Loss: 0.0467
Epoch 480, Loss: 0.0147
```

```
Epoch 481, Loss: 0.0276
Epoch 482, Loss: 0.0133
Epoch 483, Loss: 0.0640
Epoch 484, Loss: 0.0372
Epoch 485, Loss: 0.0144
Epoch 486, Loss: 0.0318
Epoch 487, Loss: 0.0248
Epoch 488, Loss: 0.0477
Epoch 489, Loss: 0.0365
Epoch 490, Loss: 0.0051
Epoch 491, Loss: 0.0107
Epoch 492, Loss: 0.0265
Epoch 493, Loss: 0.0219
Epoch 494, Loss: 0.0718
Epoch 495, Loss: 0.0240
Epoch 496, Loss: 0.1008
Epoch 497, Loss: 0.0521
Epoch 498, Loss: 0.0430
Epoch 499, Loss: 0.0042
Epoch 500, Loss: 0.0181
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-25-384a0c5bcca2> in <cell line: 41>()
     39
     40
---> 41 samp_images = diffusion_helper.reverse_diffuse(model, x0s.shape[0])
     42 save_images(samp_images, os.path.join(res_folder, f"{epoch+1}.png"))
     43

TypeError: DiffusionHelper.reverse_diffuse() missing 2 required positional␣
 ↪arguments: 'labels' and 'guidance_strength'
```

[30]:
```
model.load_state_dict(torch.load(os.path.join(check_point_dir,"500_ckpt.pt")))
```

[30]: <All keys matched successfully>

[31]:
```
label_i = [0, 1, 2, 3, 4]
labels_list = ["Daisy", "Dandelion", "Rose", "Sunflower", "Tulip"]
```

[33]:
```
SAMPLE_COUNT = 16

for lb_i in label_i:
  current_count = 0
  lbl_images = []
  for images, labels in dataloader:
    if len(lbl_images) == SAMPLE_COUNT:
```

```
        break
    for idx, lbl in enumerate(labels):
      if lbl.item() == lb_i:
        if len(lbl_images) == SAMPLE_COUNT:
          break
        lbl_images.append(inverse_transform(images[idx]))

  save_images(lbl_images, os.path.join(res_folder,␣
↪f"{lb_i}_{labels_list[lb_i]}_images.png"))
  samp_images = diffusion_helper.reverse_diffuse(model, SAMPLE_COUNT, torch.
↪full((SAMPLE_COUNT,), lb_i, device=device).long(), 3)
  save_images(samp_images, os.path.join(res_folder,␣
↪f"{labels_list[lb_i]}_final_results.png"))
```

[34]:
```
samp_images = diffusion_helper.reverse_diffuse(model, SAMPLE_COUNT, None, 3)
save_images(samp_images, os.path.join(res_folder, f"all_samp_final_results.
↪png"))
```