qegvonkua

March 31, 2024

0.1 Selecting only digits 1 and 9

[26]:
```
import torch
from torchvision import datasets, transforms
from torch.utils.data import DataLoader, Subset

transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.,), (1.,))
])

train_dataset = datasets.MNIST(root='./data', train=True, download=True,␣
 ↪transform=transform)
test_dataset = datasets.MNIST(root='./data', train=False, download=True,␣
 ↪transform=transform)
train_indices = [i for i in range(len(train_dataset)) if train_dataset.
 ↪targets[i] in [1, 9]]
test_indices = [i for i in range(len(test_dataset)) if test_dataset.targets[i]␣
 ↪in [1, 9]]
filtered_train_dataset = Subset(train_dataset, train_indices)
filtered_test_dataset = Subset(test_dataset, test_indices)
batch_size = 64
train_mnistloader = DataLoader(filtered_train_dataset, batch_size=batch_size,␣
 ↪shuffle=True)
val_mnistloader = DataLoader(filtered_test_dataset, batch_size=batch_size,␣
 ↪shuffle=False)
device = "cuda" if torch.cuda.is_available() else "cpu"
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to
./data/MNIST/raw/train-images-idx3-ubyte.gz

100%|      | 9912422/9912422 [00:00<00:00, 206676731.03it/s]

Extracting ./data/MNIST/raw/train-images-idx3-ubyte.gz to ./data/MNIST/raw



Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to
./data/MNIST/raw/train-labels-idx1-ubyte.gz

100%|          | 28881/28881 [00:00<00:00, 44683029.81it/s]

Extracting ./data/MNIST/raw/train-labels-idx1-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to
./data/MNIST/raw/t10k-images-idx3-ubyte.gz

100%|          | 1648877/1648877 [00:00<00:00, 79284313.66it/s]

Extracting ./data/MNIST/raw/t10k-images-idx3-ubyte.gz to ./data/MNIST/raw



Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to
./data/MNIST/raw/t10k-labels-idx1-ubyte.gz

100%|          | 4542/4542 [00:00<00:00, 6331182.71it/s]

Extracting ./data/MNIST/raw/t10k-labels-idx1-ubyte.gz to ./data/MNIST/raw
```

[28]:
```
# for imgs,_ in train_mnistloader:
#     img = imgs[0]
#     print(img.shape,img[0])
#     break
```

0.2 Geomloss for Sinkhorn loss for optimal transport

[ ]:
```
%%capture
!pip install geomloss
```

0.3 Autoencoder with latent dimension mapping to normal distrbution and optimizing optimal transport loss wrt to Normal Distribution

Also applying reparametrization trick for end to end backpropagation.

[31]:
```
import torch
import torch.nn as nn
import torch.optim as optim
from geomloss import SamplesLoss
from torch.utils.data import DataLoader
from tqdm import tqdm

class VAE(nn.Module):
    def __init__(self, latent_dim):
```

```
        super(VAE, self).__init__()
        self.encoder = nn.Sequential(
            nn.Conv2d(1, 64, kernel_size=7, stride=2, padding=3),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),
            nn.Conv2d(64, 128, kernel_size=3, stride=2, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True),
            nn.Conv2d(128, 256, kernel_size=3, stride=2, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(inplace=True),
            nn.Flatten(),
            nn.Linear(4096, latent_dim * 2),
            nn.BatchNorm1d(latent_dim * 2)
        )

        self.decoder = nn.Sequential(
            nn.Linear(latent_dim, 256 * 7 * 7),
            nn.BatchNorm1d(256 * 7 * 7),
            nn.ReLU(inplace=True),
            nn.Unflatten(1, (256, 7, 7)),
            nn.ConvTranspose2d(256, 128, kernel_size=3, stride=2, padding=1, output_padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True),
            nn.ConvTranspose2d(128, 64, kernel_size=3, stride=2, padding=1, output_padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),
            nn.ConvTranspose2d(64, 1, kernel_size=3, stride=1, padding=1),
            nn.Sigmoid()
        )

    def encode(self, x):
        z = self.encoder(x)
        mu, log_var = torch.chunk(z, 2, dim=1)
        return mu, log_var

    def decode(self, z):
        return self.decoder(z)

    def reparameterize(self, mu, log_var):#####mapping to normal distribution but reparametrization trick for backpropagation
        std = torch.exp(0.5 * log_var)
        eps = torch.randn_like(std)
        return mu + eps * std
```

```
    def forward(self, x):
        mu, log_var = self.encode(x)
        z = self.reparameterize(mu, log_var)
        x_recon = self.decode(z)
        return x_recon, z


latent_dim = 64
learning_rate = 0.1
num_epochs = 15
batch_size = 64
beta = 0.15  ## weighting optimaltransport loss by beta

model = VAE(latent_dim).to(device)

reconstruction_loss = nn.BCELoss(reduction='sum')
optimaltransportloss_fn = SamplesLoss("sinkhorn", p=2, blur=0.05)##optimal␣
 ↪transport loss

optimizer = optim.Adam(model.parameters(), lr=learning_rate)

prev_loss = float('inf')
best_loss = float('inf')

losses = []
for epoch in range(num_epochs):
    train_loss = 0.0
    val_loss = 0.0

    # Validation loop
    model.eval()
    val_loader = tqdm(val_mnistloader, desc=f'Epoch {epoch + 1}/{num_epochs}␣
 ↪(val)', total=len(val_mnistloader))
    for batch_idx, (data,_) in enumerate(val_loader):
        data = data.to(device)
        with torch.no_grad():
            recon_batch, z = model(data)
            loss = reconstruction_loss(recon_batch, data)
            normal_samples = torch.randn_like(z)
            optimaltransportloss = optimaltransportloss_fn(z, normal_samples)
            loss += beta*optimaltransportloss
            val_loss += loss.item()
            val_loader.set_postfix(val_loss=val_loss / ((batch_idx + 1)))
        del data

    if val_loss / len(val_mnistloader) >= prev_loss:
        learning_rate *= 0.1
```

```
        optimizer = optim.Adam(model.parameters(), lr=learning_rate)
        print(f"Changing learning rate to {learning_rate}")
    prev_loss = val_loss/(len(val_mnistloader))

    # Training loop
    model.train()
    train_loader = tqdm(train_mnistloader, desc=f'Epoch {epoch + 1}/
↪{num_epochs} (train)', total=len(train_mnistloader))
    for batch_idx, (data,_) in enumerate(train_loader):
        data = data.to(device)
        optimizer.zero_grad()
        recon_batch, z = model(data)
        loss = reconstruction_loss(recon_batch, data)
        normal_samples = torch.randn_like(z)
        optimaltransportloss = optimaltransportloss_fn(z, normal_samples)
        loss += beta*optimaltransportloss
        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)
        optimizer.step()
        train_loss += loss.item()
        train_loader.set_postfix(train_loss=train_loss / ((batch_idx + 1)))
    del data

    losses.append(val_loss)
    if val_loss<best_loss:
        best_loss = val_loss
        torch.save(model.state_dict(),"model.pth")
        print("saved model to model.pth")
```

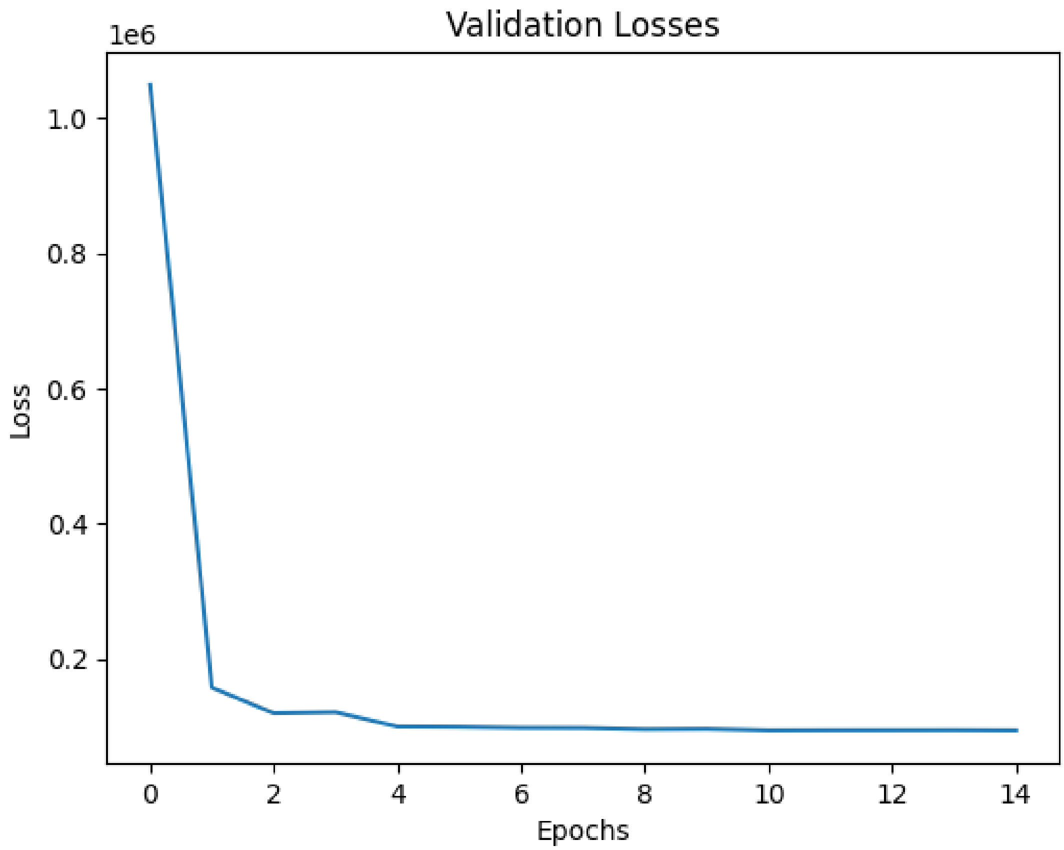
```
Epoch 1/15 (val): 100%|      | 34/34 [00:01<00:00, 23.46it/s,
val_loss=3.09e+4]
Epoch 1/15 (train): 100%|      | 199/199 [00:07<00:00, 26.63it/s,
train_loss=8.16e+3]

saved model to model.pth

Epoch 2/15 (val): 100%|      | 34/34 [00:01<00:00, 32.59it/s,
val_loss=4.62e+3]
Epoch 2/15 (train): 100%|      | 199/199 [00:07<00:00, 26.81it/s,
train_loss=4.01e+3]

saved model to model.pth

Epoch 3/15 (val): 100%|      | 34/34 [00:01<00:00, 32.20it/s,
val_loss=3.5e+3]
Epoch 3/15 (train): 100%|      | 199/199 [00:07<00:00, 26.14it/s,
train_loss=3.68e+3]

saved model to model.pth
```

```
Epoch 4/15 (val): 100%|     | 34/34 [00:01<00:00, 32.50it/s,
val_loss=3.55e+3]

Changing learning rate to 0.010000000000000002

Epoch 4/15 (train): 100%|     | 199/199 [00:07<00:00, 27.16it/s,
train_loss=3.19e+3]
Epoch 5/15 (val): 100%|     | 34/34 [00:01<00:00, 33.05it/s,
val_loss=2.93e+3]
Epoch 5/15 (train): 100%|     | 199/199 [00:07<00:00, 27.77it/s,
train_loss=3.12e+3]

saved model to model.pth

Epoch 6/15 (val): 100%|     | 34/34 [00:01<00:00, 33.11it/s,
val_loss=2.91e+3]
Epoch 6/15 (train): 100%|     | 199/199 [00:07<00:00, 27.32it/s,
train_loss=3.09e+3]

saved model to model.pth

Epoch 7/15 (val): 100%|     | 34/34 [00:01<00:00, 32.61it/s,
val_loss=2.87e+3]
Epoch 7/15 (train): 100%|     | 199/199 [00:07<00:00, 27.00it/s,
train_loss=3.05e+3]

saved model to model.pth

Epoch 8/15 (val): 100%|     | 34/34 [00:01<00:00, 32.90it/s,
val_loss=2.87e+3]
Epoch 8/15 (train): 100%|     | 199/199 [00:07<00:00, 27.04it/s,
train_loss=3.02e+3]

saved model to model.pth

Epoch 9/15 (val): 100%|     | 34/34 [00:01<00:00, 33.05it/s,
val_loss=2.81e+3]
Epoch 9/15 (train): 100%|     | 199/199 [00:07<00:00, 27.16it/s,
train_loss=3e+3]

saved model to model.pth

Epoch 10/15 (val): 100%|     | 34/34 [00:01<00:00, 32.98it/s,
val_loss=2.82e+3]

Changing learning rate to 0.0010000000000000002

Epoch 10/15 (train): 100%|     | 199/199 [00:07<00:00, 27.08it/s,
train_loss=2.93e+3]
Epoch 11/15 (val): 100%|     | 34/34 [00:01<00:00, 32.51it/s,
val_loss=2.76e+3]
Epoch 11/15 (train): 100%|     | 199/199 [00:07<00:00, 26.87it/s,
train_loss=2.94e+3]

saved model to model.pth
```

```
Epoch 12/15 (val): 100%|      | 34/34 [00:01<00:00, 32.34it/s,
val_loss=2.77e+3]

Changing learning rate to 0.00010000000000000003

Epoch 12/15 (train): 100%|      | 199/199 [00:07<00:00, 27.63it/s,
train_loss=2.93e+3]
Epoch 13/15 (val): 100%|      | 34/34 [00:01<00:00, 33.44it/s,
val_loss=2.77e+3]

Changing learning rate to 1.0000000000000004e-05

Epoch 13/15 (train): 100%|      | 199/199 [00:07<00:00, 27.57it/s,
train_loss=2.92e+3]
Epoch 14/15 (val): 100%|      | 34/34 [00:01<00:00, 33.50it/s,
val_loss=2.77e+3]

Changing learning rate to 1.0000000000000004e-06

Epoch 14/15 (train): 100%|      | 199/199 [00:07<00:00, 27.27it/s,
train_loss=2.92e+3]
Epoch 15/15 (val): 100%|      | 34/34 [00:01<00:00, 29.93it/s,
val_loss=2.76e+3]
Epoch 15/15 (train): 100%|      | 199/199 [00:07<00:00, 27.19it/s,
train_loss=2.92e+3]

saved model to model.pth
```

[32]:

```
import matplotlib.pyplot as plt

plt.plot(losses)
plt.title("Validation Losses")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.show()
```



[33]:
```
try:
    del model
    print("deleted model")
except:
    None
torch.cuda.empty_cache()
```

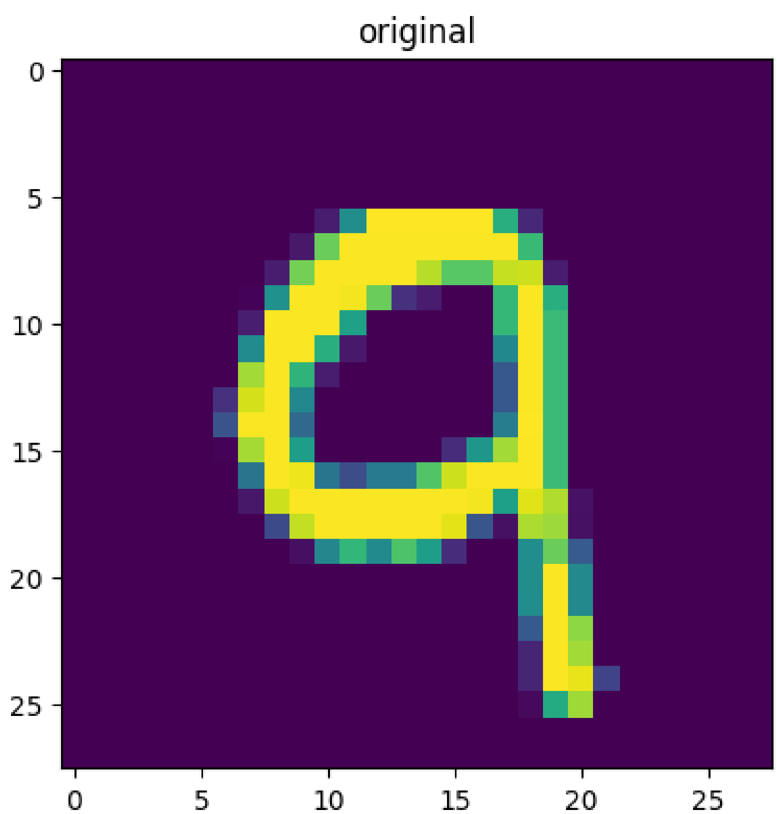
```
deleted model
```

[42]:
```
import random
import matplotlib.pyplot as plt
model = VAE(latent_dim).to(device)
model.load_state_dict(torch.load("model.pth"))
model.eval()
r = random.randint(0,1024)
dataset = filtered_test_dataset
imgorig = dataset[r][0]
imgrecons = model(imgorig.unsqueeze(0).to(device))[0]
recon_batch, z = model(imgorig.unsqueeze(0).to(device))
loss = reconstruction_loss(recon_batch, imgorig.unsqueeze(0).to(device))
```

```
print(loss)
normal_samples = torch.randn_like(z)
optimaltransportloss = 0.15*optimaltransportloss_fn(z, normal_samples)
loss += optimaltransportloss
print(loss)
# print(np.array(imgrecons.cpu()).shape)
pltorig = imgorig.permute(1,2,0).cpu().numpy()
pltrecons = imgrecons.permute(0,2,3,1).detach().cpu().numpy()[0]
plt.title("original")
plt.imshow(pltorig)
```

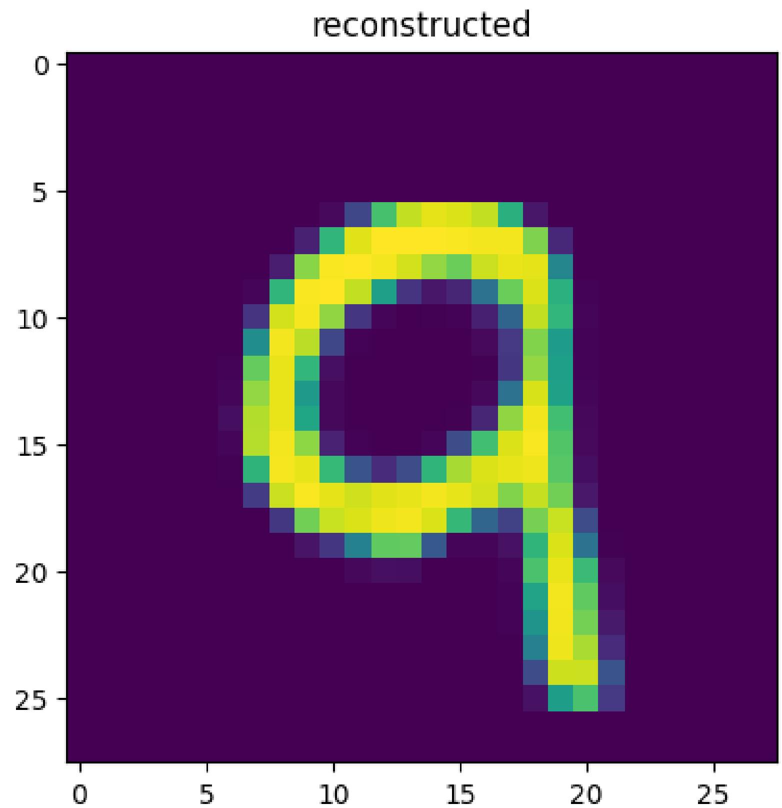
```
tensor(62.6666, device='cuda:0', grad_fn=<BinaryCrossEntropyBackward0>)
tensor(85.6434, device='cuda:0', grad_fn=<AddBackward0>)
```

[42]: <matplotlib.image.AxesImage at 0x7ba9d027f9a0>



[44]:
```
plt.title("reconstructed")
plt.imshow(pltrecons)
```

[44]: <matplotlib.image.AxesImage at 0x7ba9d07f56c0>

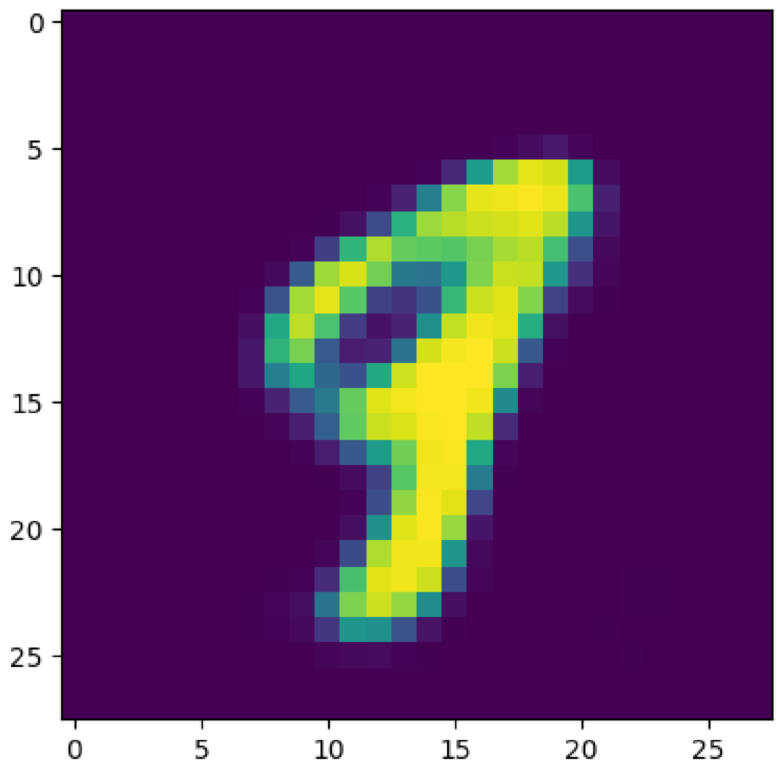


0.4 Sampling from normal distribution randomly

[52]:
```
import random
model = VAE(latent_dim).to(device)
model.load_state_dict(torch.load("model.pth"))
model.eval()

imgorig = torch.randn((1,latent_dim))##normal distribution

imgrecons = model.decoder(imgorig.to(device))[0]
imgrecons = imgrecons.permute(1,2,0).detach().cpu()
print((imgrecons).shape)
plt.imshow(imgrecons)
```

```
torch.Size([28, 28, 1])
```

[52]:
```
<matplotlib.image.AxesImage at 0x7ba9ae553df0>
```



0.5 Image generation on quarks/gluon dataset

[53]:
```
!pip install gdown
import gdown
import gdown
import zipfile
import os
url = 'https://drive.google.com/uc?id=1WO2K-SfU2dntGU4Bb3IYBp9Rh7rtTYEr'
output_path = 'large_file.hdf5'
gdown.download(url, output_path, quiet=False)
import matplotlib.pyplot as plt
import numpy as np
import h5py
with h5py.File('large_file.hdf5', 'r') as file:
    train_imgs = np.array(file['X_jets'][:4096])
    test_imgs = np.array(file['X_jets'][4096:4096+1024])
    train_labels = np.array(file['y'][:4096])
    train_labels = np.array(file['y'][4096:4096+1024])
    print(train_imgs[0].shape)
```

```
Requirement already satisfied: gdown in /opt/conda/lib/python3.10/site-packages
(5.1.0)
```

```
Requirement already satisfied: beautifulsoup4 in /opt/conda/lib/python3.10/site-
packages (from gdown) (4.12.2)
Requirement already satisfied: filelock in /opt/conda/lib/python3.10/site-
packages (from gdown) (3.13.1)
Requirement already satisfied: requests[socks] in
/opt/conda/lib/python3.10/site-packages (from gdown) (2.31.0)
Requirement already satisfied: tqdm in /opt/conda/lib/python3.10/site-packages
(from gdown) (4.66.2)
Requirement already satisfied: soupsieve>1.2 in /opt/conda/lib/python3.10/site-
packages (from beautifulsoup4->gdown) (2.5)
Requirement already satisfied: charset-normalizer<4,>=2 in
/opt/conda/lib/python3.10/site-packages (from requests[socks]->gdown) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /opt/conda/lib/python3.10/site-
packages (from requests[socks]->gdown) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/opt/conda/lib/python3.10/site-packages (from requests[socks]->gdown) (1.26.18)
Requirement already satisfied: certifi>=2017.4.17 in
/opt/conda/lib/python3.10/site-packages (from requests[socks]->gdown) (2024.2.2)
Requirement already satisfied: PySocks!=1.5.7,>=1.5.6 in
/opt/conda/lib/python3.10/site-packages (from requests[socks]->gdown) (1.7.1)
```

```
Downloading…
From (original):
https://drive.google.com/uc?id=1WO2K-SfU2dntGU4Bb3IYBp9Rh7rtTYEr
From (redirected): https://drive.google.com/uc?id=1WO2K-SfU2dntGU4Bb3IYBp9Rh7rtT
YEr&confirm=t&uuid=0a82e0c8-9171-48ca-88b6-2afb6707ded7
To: /kaggle/working/large_file.hdf5
100%|      | 701M/701M [00:03<00:00, 208MB/s]
```

```
(125, 125, 3)
```

[54]:
```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import torchvision.transforms.v2 as transforms
class Data(torch.utils.data.Dataset):
    def __init__(self,imgs):
        super().__init__()
        self.transform = transforms.Compose([
            transforms.ToTensor(),
            #transforms.Normalize([0.,0.,0.],[1.,1.,1.]),
        ])
        self.imgs = imgs
    def __len__(self):
        return len(self.imgs)
    def __getitem__(self,idx):
        img = self.transform(self.imgs[idx])
```

```
        img2 = torch.zeros((3,128,128)).to(img.dtype)
        img2[:,:125,:125] = img
        return img2

train_loader = torch.utils.data.DataLoader(Data(train_imgs), batch_size=64)
val_loader = torch.utils.data.DataLoader(Data(test_imgs), batch_size=64)
```

[55]:
```
device = "cuda" if torch.cuda.is_available() else "cpu"
```

[57]:
```
from geomloss import SamplesLoss
from torch.utils.data import DataLoader
from tqdm import tqdm

class VAE(nn.Module):
    def __init__(self, latent_dim):
        super(VAE, self).__init__()
        self.encoder = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),
            nn.Conv2d(64, 128, kernel_size=3, stride=2, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True),
            nn.Conv2d(128, 256, kernel_size=3, stride=2, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(inplace=True),
            nn.Flatten(),
            nn.Linear(65536, latent_dim * 2),
            nn.BatchNorm1d(latent_dim * 2)
        )

        self.decoder = nn.Sequential(
            nn.Linear(latent_dim, 256 * 8 * 8),
            nn.BatchNorm1d(256 * 8 * 8),
            nn.ReLU(inplace=True),
            nn.Unflatten(1, (256, 8, 8)),
            nn.ConvTranspose2d(256, 128, kernel_size=3, stride=2, padding=1,␣
 ↪output_padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True),
            nn.ConvTranspose2d(128, 64, kernel_size=3, stride=2, padding=1,␣
 ↪output_padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),
            nn.ConvTranspose2d(64, 32, kernel_size=3, stride=2, padding=1,␣
 ↪output_padding=1),
            nn.BatchNorm2d(32),
```

```
            nn.ReLU(inplace=True),
            nn.ConvTranspose2d(32, 3, kernel_size=3, stride=2, padding=1,␣
↪output_padding=1),
            nn.Sigmoid()
        )

    def encode(self, x):
        z = self.encoder(x)
        mu, log_var = torch.chunk(z, 2, dim=1)
        return mu, log_var

    def decode(self, z):
        return self.decoder(z)

    def reparameterize(self, mu, log_var):
        std = torch.exp(0.5 * log_var)
        eps = torch.randn_like(std)
        return mu + eps * std

    def forward(self, x):
        mu, log_var = self.encode(x)
        z = self.reparameterize(mu, log_var)
        x_recon = self.decode(z)
        return x_recon, z

latent_dim = 128
learning_rate = 0.1
num_epochs = 15
batch_size = 64

model = VAE(latent_dim).to(device)
reconstruction_loss = nn.MSELoss(reduction='sum')

optimaltransportloss_fn = SamplesLoss("sinkhorn", p=2, blur=0.05)

optimizer = optim.Adam(model.parameters(), lr=learning_rate)

prev_loss = float('inf')
best_loss = float('inf')
losses = []
for epoch in range(num_epochs):
    train_loss = 0.0
    val_loss = 0.0

    # Validation loop
    model.eval()
    val_loader = tqdm(val_loader, desc=f'Epoch {epoch + 1}/{num_epochs} (val)')
```

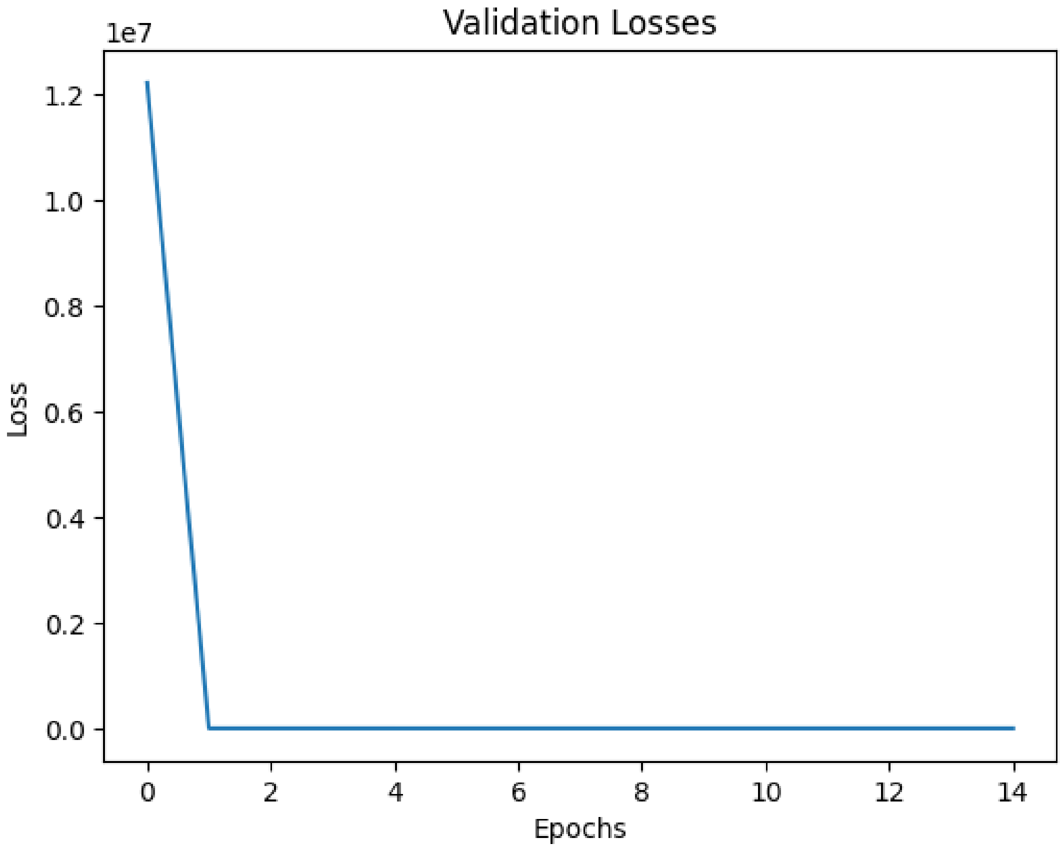
```
    for batch_idx, data in enumerate(val_loader):
        data = data.to(device)
        with torch.no_grad():
            recon_batch, z = model(data)
            loss = reconstruction_loss(recon_batch, data)
            normal_samples = torch.randn_like(z)
            optimaltransportloss = optimaltransportloss_fn(z, normal_samples)
            loss += 0.15*optimaltransportloss
            val_loss += loss.item()
            val_loader.set_postfix(val_loss=val_loss / ((batch_idx + 1)))
        del data

    if val_loss / len(val_loader) >= prev_loss:
        learning_rate *= 0.1
        optimizer = optim.Adam(model.parameters(), lr=learning_rate)
        print(f"Changing learning rate to {learning_rate}")
    prev_loss = val_loss/(len(val_loader))

    # Training loop
    model.train()
    train_loader = tqdm(train_loader, desc=f'Epoch {epoch + 1}/{num_epochs}␣
↪(train)')
    for batch_idx, data in enumerate(train_loader):
        data = data.to(device)
        optimizer.zero_grad()
        recon_batch, z = model(data)
        loss = reconstruction_loss(recon_batch, data)
        normal_samples = torch.randn_like(z)
        optimaltransportloss = optimaltransportloss_fn(z, normal_samples)
        loss += 0.15*optimaltransportloss
        loss.backward()
        optimizer.step()
        train_loss += loss.item()
        train_loader.set_postfix(train_loss=train_loss / ((batch_idx + 1)))
    del data

    losses.append(val_loss)

    if val_loss<best_loss:
        best_loss = val_loss
        torch.save(model.state_dict(),"model.pth")
        print("saved model to model.pth")
```

```
Epoch 1/15 (val): 100%|      | 16/16 [00:00<00:00, 26.06it/s,
val_loss=7.64e+5]
Epoch 1/15 (train): 100%|      | 64/64 [00:03<00:00, 16.17it/s,
train_loss=1.47e+4]
```

```
saved model to model.pth

Epoch 2/15 (val): 100%|     | 16/16 [00:00<00:00, 24.86it/s, val_loss=39.5]
Epoch 2/15 (train): 100%|     | 64/64 [00:04<00:00, 15.96it/s,
train_loss=82.4]

saved model to model.pth

Epoch 3/15 (val): 100%|     | 16/16 [00:00<00:00, 25.28it/s, val_loss=46.4]

Changing learning rate to 0.010000000000000002

Epoch 3/15 (train): 100%|     | 64/64 [00:03<00:00, 16.36it/s,
train_loss=50.3]
Epoch 4/15 (val): 100%|     | 16/16 [00:00<00:00, 25.47it/s, val_loss=32.3]
Epoch 4/15 (train): 100%|     | 64/64 [00:03<00:00, 16.41it/s,
train_loss=34]

saved model to model.pth

Epoch 5/15 (val): 100%|     | 16/16 [00:00<00:00, 25.02it/s, val_loss=28.4]
Epoch 5/15 (train): 100%|     | 64/64 [00:03<00:00, 16.56it/s,
train_loss=30.7]

saved model to model.pth

Epoch 6/15 (val): 100%|     | 16/16 [00:00<00:00, 25.22it/s, val_loss=26.6]
Epoch 6/15 (train): 100%|     | 64/64 [00:03<00:00, 16.43it/s,
train_loss=29.5]

saved model to model.pth

Epoch 7/15 (val): 100%|     | 16/16 [00:00<00:00, 25.76it/s, val_loss=25.9]
Epoch 7/15 (train): 100%|     | 64/64 [00:03<00:00, 16.51it/s,
train_loss=28.7]

saved model to model.pth

Epoch 8/15 (val): 100%|     | 16/16 [00:00<00:00, 25.61it/s, val_loss=25.1]
Epoch 8/15 (train): 100%|     | 64/64 [00:03<00:00, 16.19it/s,
train_loss=28.2]

saved model to model.pth

Epoch 9/15 (val): 100%|     | 16/16 [00:00<00:00, 25.10it/s, val_loss=24.7]
Epoch 9/15 (train): 100%|     | 64/64 [00:03<00:00, 16.61it/s,
train_loss=27.9]

saved model to model.pth

Epoch 10/15 (val): 100%|     | 16/16 [00:00<00:00, 25.05it/s,
val_loss=24.5]
Epoch 10/15 (train): 100%|     | 64/64 [00:03<00:00, 16.40it/s,
train_loss=27.4]

saved model to model.pth
```

```
Epoch 11/15 (val): 100%|      | 16/16 [00:00<00:00, 25.56it/s,
val_loss=23.9]
Epoch 11/15 (train): 100%|      | 64/64 [00:03<00:00, 16.53it/s,
train_loss=27]

saved model to model.pth

Epoch 12/15 (val): 100%|      | 16/16 [00:00<00:00, 25.26it/s,
val_loss=23.7]
Epoch 12/15 (train): 100%|      | 64/64 [00:03<00:00, 16.52it/s,
train_loss=26.6]

saved model to model.pth

Epoch 13/15 (val): 100%|      | 16/16 [00:00<00:00, 26.09it/s,
val_loss=23.5]
Epoch 13/15 (train): 100%|      | 64/64 [00:03<00:00, 16.42it/s,
train_loss=26.4]

saved model to model.pth

Epoch 14/15 (val): 100%|      | 16/16 [00:00<00:00, 25.70it/s, val_loss=24]

Changing learning rate to 0.0010000000000000002

Epoch 14/15 (train): 100%|      | 64/64 [00:03<00:00, 16.21it/s,
train_loss=26.5]
Epoch 15/15 (val): 100%|      | 16/16 [00:00<00:00, 25.49it/s,
val_loss=23.5]
Epoch 15/15 (train): 100%|      | 64/64 [00:03<00:00, 16.06it/s,
train_loss=25.8]
```

[58]:
```python
import matplotlib.pyplot as plt

plt.plot(losses)
plt.title("Validation Losses")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.show()
```



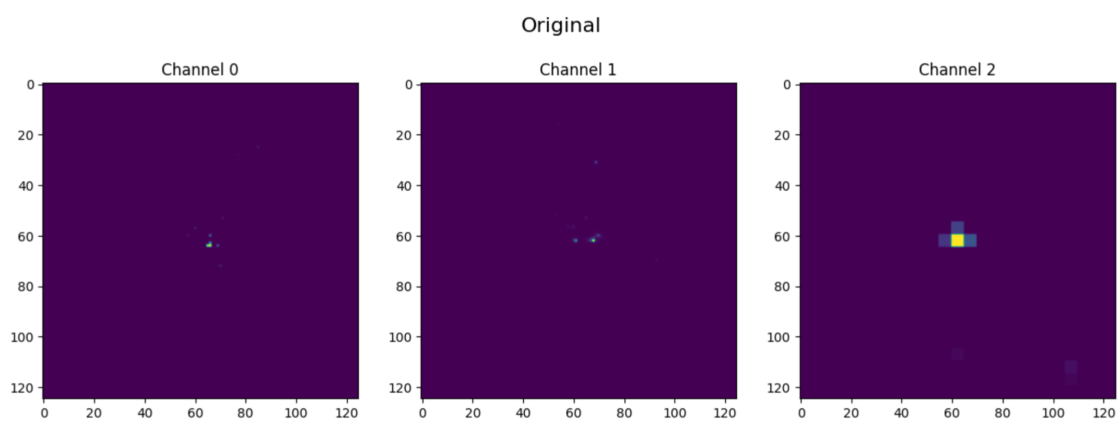
[59]:
```
try:
    del model
    print("deleted model")
except:
    None
torch.cuda.empty_cache()
```

```
deleted model
```

[64]:
```
import random
model = VAE(latent_dim).to(device)
model.load_state_dict(torch.load("model.pth"))
model.eval()
r = random.randint(0,1024)
dataset = Data(test_imgs)
imgorig = dataset[r]
imgrecons = model(imgorig.unsqueeze(0).to(device))[0]
recon_batch, z = model(imgorig.unsqueeze(0).to(device))
loss = reconstruction_loss(recon_batch, imgorig.unsqueeze(0).to(device))
print(loss)
```

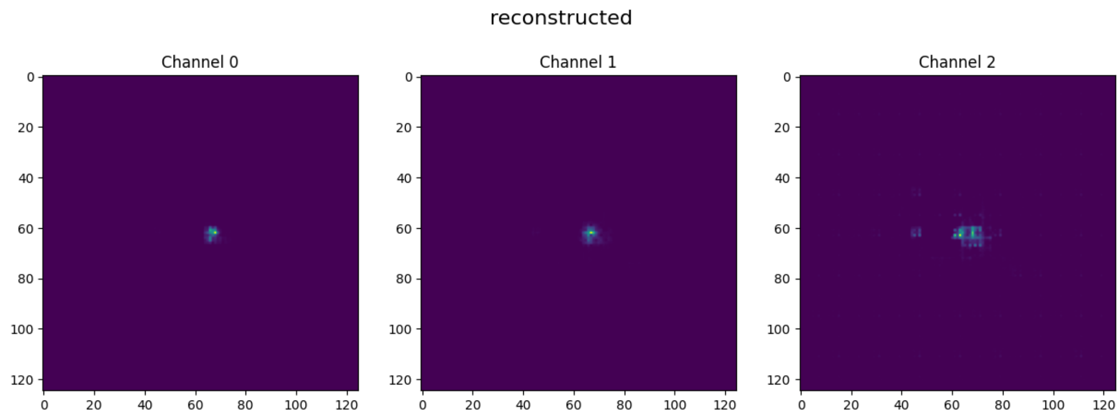
```
normal_samples = torch.randn_like(z)
optimaltransportloss = 0.15*optimaltransportloss_fn(z, normal_samples)
loss += optimaltransportloss
print(loss)
# print(np.array(imgrecons.cpu()).shape)
pltorig = imgorig.permute(1,2,0).cpu().numpy()[:125,:125,:]
pltrecons = imgrecons.permute(0,2,3,1).detach().cpu().numpy()[0][:125,:125,:]
import matplotlib.pyplot as plt

fig, axs = plt.subplots(1, 3, figsize=(15, 5))

axs[0].imshow(pltorig[:,:,0])
axs[0].set_title('Channel 0')

axs[1].imshow(pltorig[:,:,1])
axs[1].set_title('Channel 1')

axs[2].imshow(pltorig[:,:,2])
axs[2].set_title('Channel 2')

plt.suptitle('Original', fontsize=16)  # Set a common title for all subplots in␣
↪the x-axis direction
plt.savefig('pltrecons')  # Save the figure as "pltrecons.png"
plt.show()
```

```
tensor(0.1367, device='cuda:0', grad_fn=<MseLossBackward0>)
tensor(10.0243, device='cuda:0', grad_fn=<AddBackward0>)
```



[65]:
```
fig, axs = plt.subplots(1, 3, figsize=(15, 5))

axs[0].imshow(pltrecons[:,:,0])
axs[0].set_title('Channel 0')
```

```python
axs[1].imshow(pltrecons[:,:,1])
axs[1].set_title('Channel 1')

axs[2].imshow(pltrecons[:,:,2])
axs[2].set_title('Channel 2')

plt.suptitle('reconstructed', fontsize=16)  # Set a common title for all␣
↪subplots in the x-axis direction
plt.show()
```



[ ]:

```python

```