DLCV HW2 REPORT

R11521701 程懷恩

Problem 1：Diffusion models

Code reference:

[1]
https://github.com/TeaPearce/Conditional_Diffusion_MNIST/blob/main/LICENSE

[2] **https://github.com/dome272/Diffusion-Models-pytorch**

1. **Follow the Github Example to draw your model architecture and describe your implementation details.**

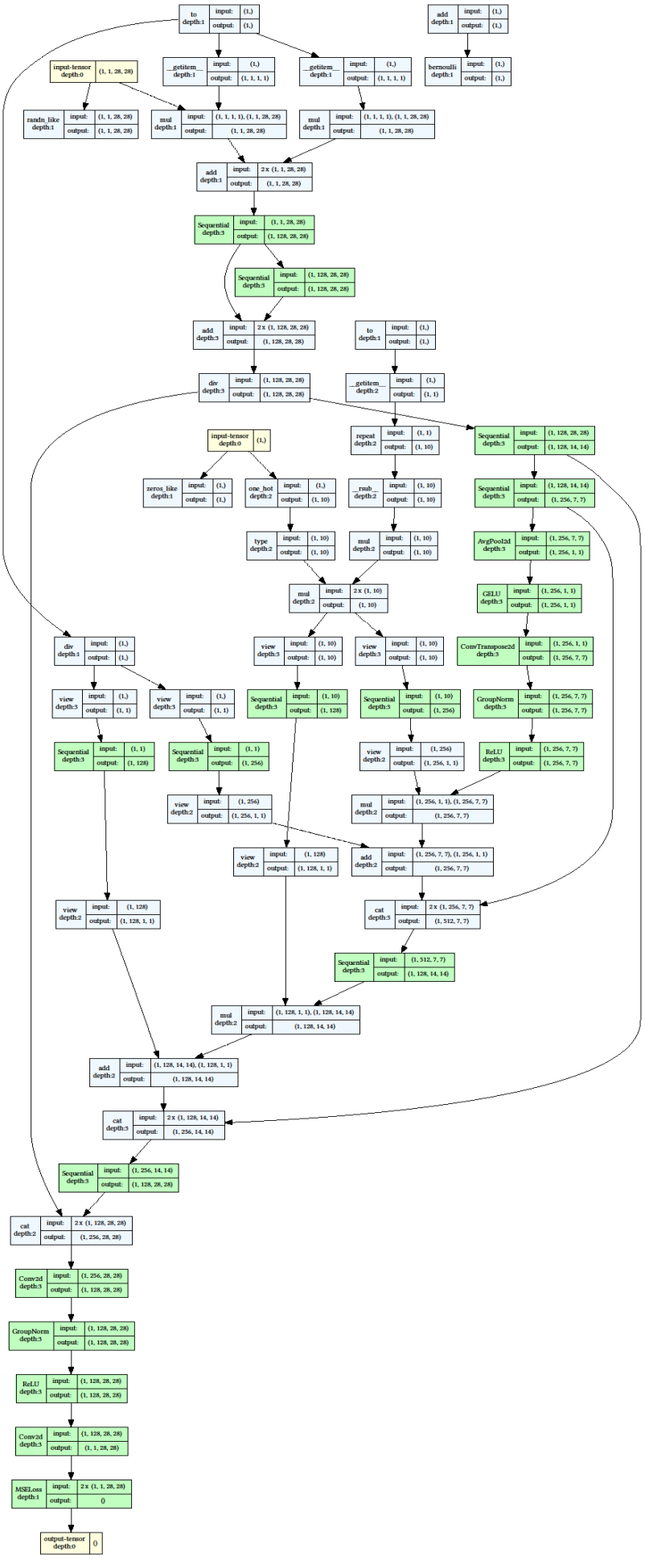
   CFG implementation:

```
# split predictions and compute weighting
eps = self.nn_model(x_i, c_i, t_is, context_mask)
eps1 = eps[:n_sample]
eps2 = eps[n_sample:]
eps = (1+guide_w)*eps1 - guide_w*eps2
```

   The noise predictions (eps) from the model are computed for the entire doubled batch. Then, these predictions are split into two: one corresponding to the classifier-guided samples (eps1) and the other corresponding to the context-free samples (eps2).

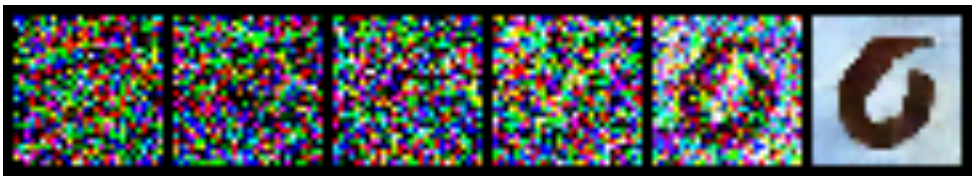
```
x_i = (
    self.oneover_sqrta[i] * (x_i - eps * self.mab_over_sqrtmab[i])
    + self.sqrt_beta_t[i] * z
)
```

   Then, image at each timestep (x_i) is updated using the diffusion process, and the noise term (eps) that's weighted with the guidance. Moreover, at the beginning of each epoch, the learning rate is adjusted according to a linear decay schedule to help the model converge more smoothly. In summary, the DDPM architecture consists of a U-Net model trained to reverse the diffusion process, generating clean images from noise. The process is guided by a learned schedule of noise levels, and the model's performance is improved with techniques like CFG and batch normalization.


to
depth:1
input: (1,)
output: (1,)
add
depth:1
input: (1,)
output: (1,)
input-tensor
depth:0
(1, 1, 28, 28)
__getitem__
depth:1
input: (1,)
output: (1, 1, 1, 1)
__getitem__
depth:1
input: (1,)
output: (1, 1, 1, 1)
bernoulli
depth:1
input: (1,)
output: (1,)
randn_like
depth:1
input: (1, 1, 28, 28)
output: (1, 1, 28, 28)
mul
depth:1
input: (1, 1, 1, 1), (1, 1, 28, 28)
output: (1, 1, 28, 28)
mul
depth:1
input: (1, 1, 1, 1), (1, 1, 28, 28)
output: (1, 1, 28, 28)
add
depth:1
input: 2 x (1, 1, 28, 28)
output: (1, 1, 28, 28)
Sequential
depth:3
input: (1, 1, 28, 28)
output: (1, 128, 28, 28)
Sequential
depth:3
input: (1, 128, 28, 28)
output: (1, 128, 28, 28)
add
depth:3
input: 2 x (1, 128, 28, 28)
output: (1, 128, 28, 28)
to
depth:1
input: (1,)
output: (1,)
div
depth:3
input: (1, 128, 28, 28)
output: (1, 128, 28, 28)
__getitem__
depth:2
input: (1,)
output: (1, 1)
input-tensor
depth:0
(1,)
repeat
depth:2
input: (1, 1)
output: (1, 10)
Sequential
depth:3
input: (1, 128, 28, 28)
output: (1, 128, 14, 14)
zeros_like
depth:1
input: (1,)
output: (1,)
one_hot
depth:2
input: (1,)
output: (1, 10)
rsub
depth:2
input: (1, 10)
output: (1, 10)
Sequential
depth:3
input: (1, 128, 14, 14)
output: (1, 256, 7, 7)
type
depth:2
input: (1, 10)
output: (1, 10)
mul
depth:2
input: (1, 10)
output: (1, 10)
AvgPool2d
depth:3
input: (1, 256, 7, 7)
output: (1, 256, 1, 1)
mul
depth:2
input: 2 x (1, 10)
output: (1, 10)
GELU
depth:3
input: (1, 256, 1, 1)
output: (1, 256, 1, 1)
div
depth:1
input: (1,)
output: (1,)
view
depth:3
input: (1, 10)
output: (1, 10)
view
depth:3
input: (1, 10)
output: (1, 10)
ConvTranspose2d
depth:3
input: (1, 256, 1, 1)
output: (1, 256, 7, 7)
view
depth:3
input: (1,)
output: (1, 1)
view
depth:3
input: (1,)
output: (1, 1)
Sequential
depth:3
input: (1, 10)
output: (1, 128)
Sequential
depth:3
input: (1, 10)
output: (1, 256)
GroupNorm
depth:3
input: (1, 256, 7, 7)
output: (1, 256, 7, 7)
Sequential
depth:3
input: (1, 1)
output: (1, 128)
Sequential
depth:3
input: (1, 1)
output: (1, 256)
view
depth:2
input: (1, 256)
output: (1, 256, 1, 1)
ReLU
depth:3
input: (1, 256, 7, 7)
output: (1, 256, 7, 7)
view
depth:2
input: (1, 256)
output: (1, 256, 1, 1)
mul
depth:2
input: (1, 256, 1, 1), (1, 256, 7, 7)
output: (1, 256, 7, 7)
view
depth:2
input: (1, 128)
output: (1, 128, 1, 1)
add
depth:2
input: (1, 256, 7, 7), (1, 256, 1, 1)
output: (1, 256, 7, 7)
view
depth:2
input: (1, 128)
output: (1, 128, 1, 1)
cat
depth:3
input: 2 x (1, 256, 7, 7)
output: (1, 512, 7, 7)
Sequential
depth:3
input: (1, 512, 7, 7)
output: (1, 128, 14, 14)
mul
depth:2
input: (1, 128, 1, 1), (1, 128, 14, 14)
output: (1, 128, 14, 14)
add
depth:2
input: (1, 128, 14, 14), (1, 128, 1, 1)
output: (1, 128, 14, 14)
cat
depth:3
input: 2 x (1, 128, 14, 14)
output: (1, 256, 14, 14)
Sequential
depth:3
input: (1, 256, 14, 14)
output: (1, 128, 28, 28)
cat
depth:2
input: 2 x (1, 128, 28, 28)
output: (1, 256, 28, 28)
Conv2d
depth:3
input: (1, 256, 28, 28)
output: (1, 128, 28, 28)
GroupNorm
depth:3
input: (1, 128, 28, 28)
output: (1, 128, 28, 28)
ReLU
depth:3
input: (1, 128, 28, 28)
output: (1, 128, 28, 28)
Conv2d
depth:3
input: (1, 128, 28, 28)
output: (1, 1, 28, 28)
MSELoss
depth:1
input: 2 x (1, 1, 28, 28)
output: 0
output-tensor
depth:0
0

2. **Please show 10 generated images for each digit (0-9) in your report. You can put all 100 outputs in one image with columns indicating different noise inputs and rows indicating different digits.**

3. **visualize total six images in the reverse process of the first "0" in your grid in (2) with different time steps.**



(From t=1000 sample back to t=0)

4. **Please discuss what you've observed and learned from implementing conditional diffusion model.**

   Firstly, I refer to [2] to implement the unconditional diffusion model from scratch, then keep going on the conditional model using EMA (Exponential Moving Average) and CFG (Classifier Free Guidance) to supervise and improve the performance. Moreover, in conditional ddpm training, I adopted conditional UNet which has been extended to handle conditioning on discrete class labels or conditions through an embedding layer. The embedding layer is designed to transform a discrete class label y into a dense vector that will be added to the positional encoding t in the forward method.

   During the training process, I created sampling GIFs for different epochs to understand the denoising progress. According to my observations by visualizing

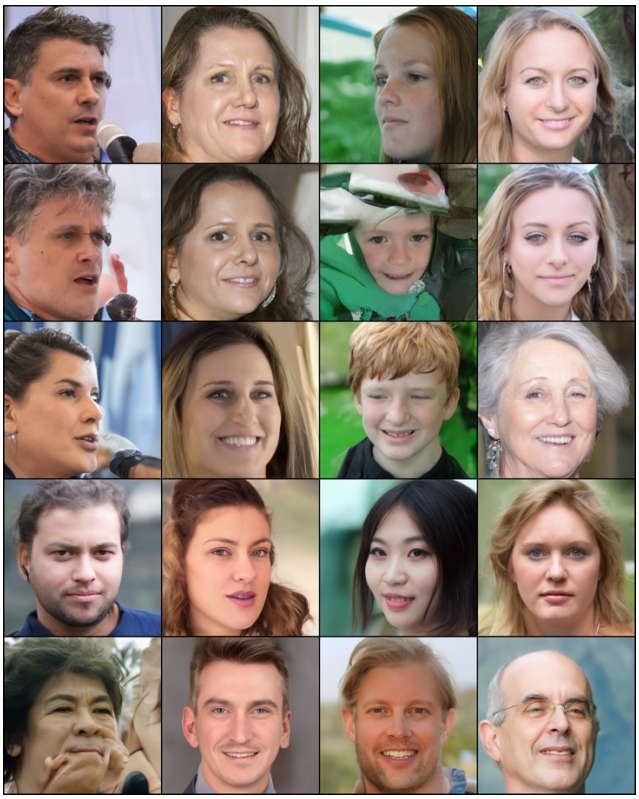
more than six images in the reverse process of the first “0”, the effectiveness of denoising is not linear but accelerates as noise decreases. Additionally, I incorporated Classifier Free Guidance during training to linearly enhance the conditioning capability.

Problem 2：DDIM

Code reference:

[1] https://github.com/fungtion/DANN_py3/blob/master

1. **Please generate face images of noise 00.pt ~ 03.pt with different eta in one grid. Report and explain your observation in this experiment.**



With higher eta values, one would expect the images to become less typical and potentially more diverse. Therefore, in my grid, although the realism of the image occasionally falters, the batch of images still maintains a high level of realism and coherence even as eta increases. This suggests that the DDIM model is robust and capable of handling increased randomness without significant degradation in image quality.

2. **Please generate the face images of the interpolation of noise 00.pt ~ 01.pt. The interpolation formula is spherical linear interpolation, which is also known as slerp.**

   Below are the images generated spherical linear interpolation with different alpha.

   Then, I tried the linear interpolation, each of the interpolations doesn't look like an image that falls between the two.

   - The transition appears as a blend or mix between the two faces.
   - Intermediate images can seem washed out or less vivid, as the pixel values are averaged directly.
   - Features such as edges might become less sharp, as the intermediate steps don't necessarily represent a realistic image.
   - The transition maintains a constant velocity in the pixel space, meaning the change from one image to the next is uniform.

Problem 3：DANN

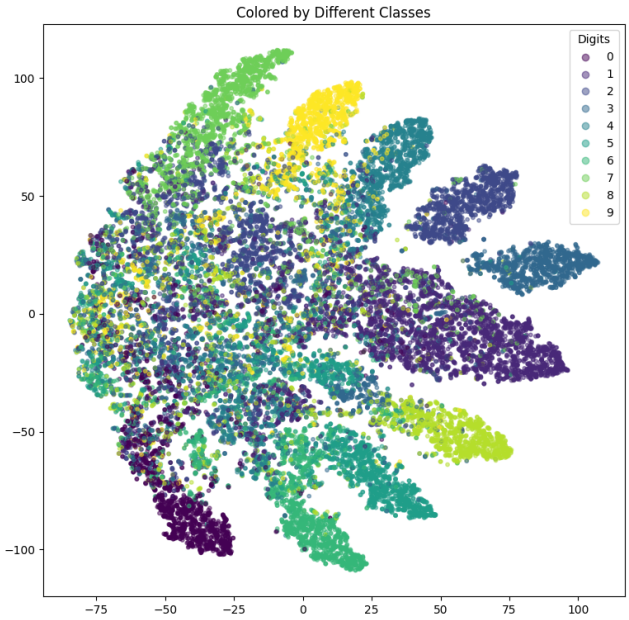
Code reference:

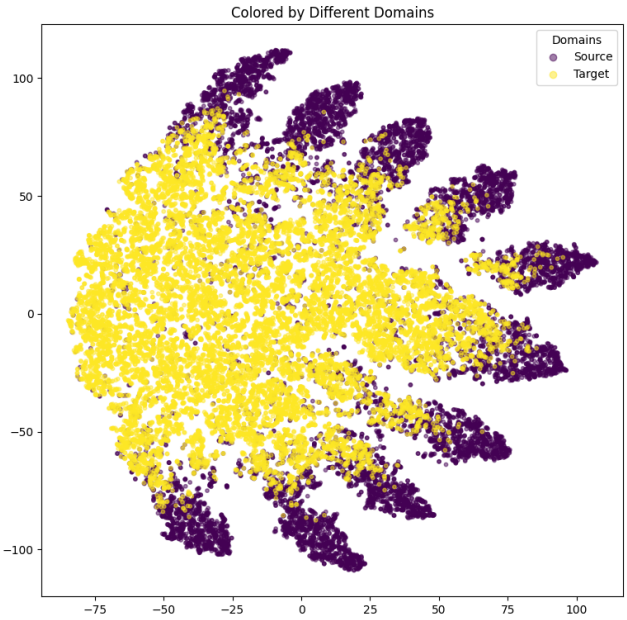
1. **Please create and fill the table with the following format in your report:**

	MNIST→SVHN	MNIST→USPS
Trained on source	25.74%	76.08%
DANN	45.61%	81.05%
Trained on target	91.85%	98.79%

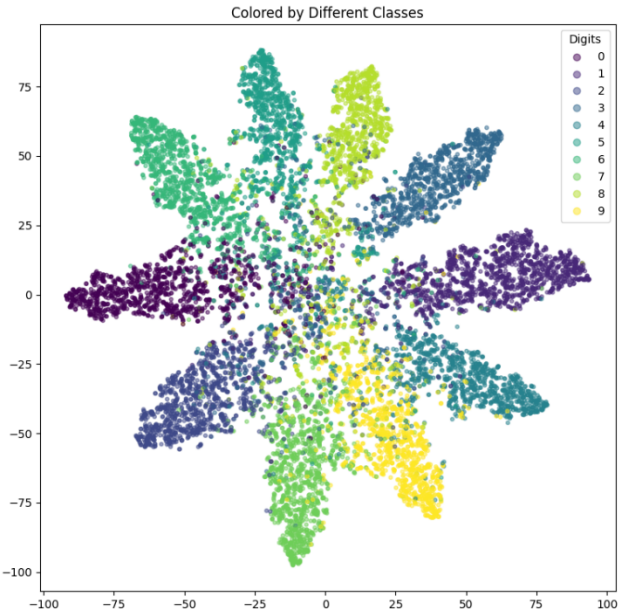
2. **Please visualize the latent space (output of CNN layers) of DANN by mapping the validation images to 2D space with t-SNE. For each scenario, you need to plot two figures which are colored by digit class (0-9) and by domain, respectively.**

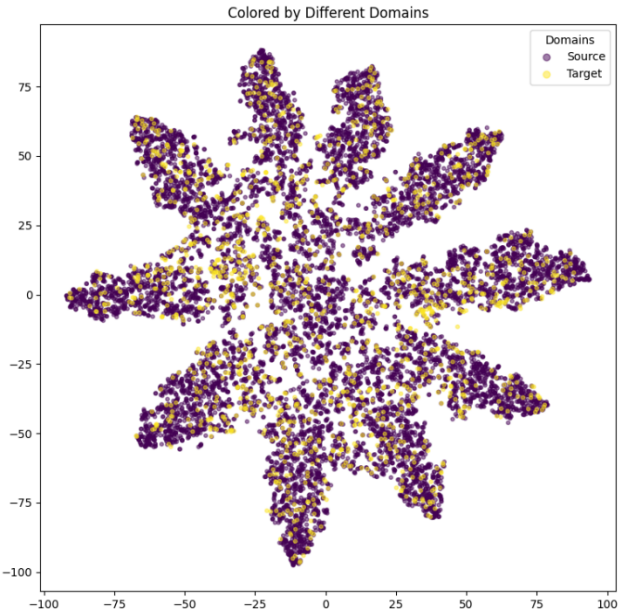
SVHN


Colored by Different Classes
Digits
0
1
2
3
4
5
6
7
8
9


Colored by Different Domains
Domains
Source
Target

USPS


Colored by Different Classes
Digits
0
1
2
3
4
5
6
7
8
9


Colored by Different Domains
Domains
Source
Target

3. **Please describe the implementation details of your model and discuss what you've observed and learned from implementing DANN.**
   There's a noticeable improvement when using DANN compared to only training on the source. For SVHN, the accuracy increased from 25.74% to 45.61%, and for USPS, it increased from 76.08% to 81.05%. This indicates that DANN helps the model generalize better to the target domain. Despite the improvement, there's still a significant gap compared to training directly on the target domain (91.85% for SVHN and 98.79% for USPS).

   Moreover, according to the t-SNE figure, It appears that for SVHN, the domain separation is less clear than for USPS, which might be why the DANN performance on USPS is closer to the performance when trained on the target.

   In conclusion, DANN's adversarial approach is effective for domain adaptation. However, fine-tuning the balance between domain discrimination and classification accuracy is crucial.

Reference:

1. 擴散模型之 DDIM https://zhuanlan.zhihu.com/p/565698027
2. 基於對抗的遷移學習方法：DANN 域對抗網路 https://zhuanlan.zhihu.com/p/73947456
3. Ganin, Yaroslav, et al. "Domain-adversarial training of neural networks." The Journal of Machine Learning Research 17.1 (2016): 2096-2030.
4. Goodfellow, Ian, et al. "Generative adversarial nets." Advances in neural information processing systems . 2014.
5. Yanwen Zhang, https://medium.com/deep-learning-domain-adaptation-on-image-segmentat/introduction-2b44dd49ea05
6. How To Train a Conditional Diffusion Model From Scratch https://wandb.ai/capecape/train_sd/reports/How-To-Train-a-Conditional-Diffusion-Model-From-Scratch--VmlldzoyNzIzNTQ1?fbclid=IwAR33nqbcT2t2tvB-AQ99NnoU_PesxF4qZKlZYrwrWRTj__a9Rgb1r53eCMM

Collaborator:

Thanks to R11521613, R11942180, Lin, and Huang for the discussion, which helped complete the homework.