DLCV Hw2

R12943010 林孟平

Problem 1: Diffusion models

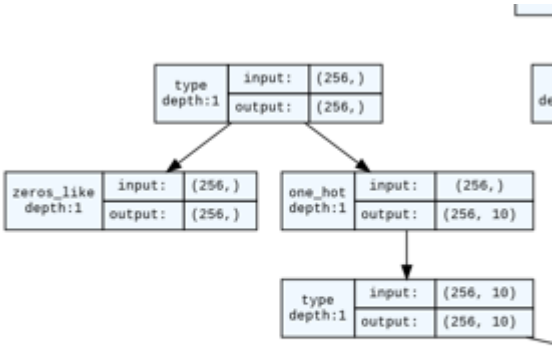
1. Follow the Github Example to draw your model architecture and describe your implementation details.

The model is adapted from this work:

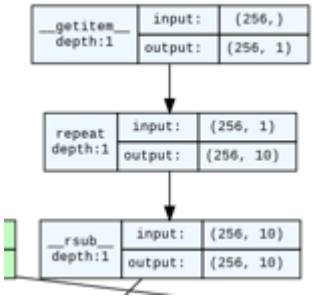
https://github.com/TeaPearce/Conditional_Diffusion_MNIST/tree/main

Since the size of model is too big and it cannot be clearly seen in the report, I attach the image of model and the source code for generating the model picture on Github. It seems that the draw_graph function only accepts one input tensor, we create input of c, t, and context_mask in the forward function, they can be found here:

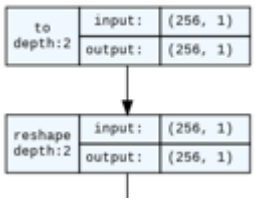
c:



context_mask:



t:



The idea of this model is similar to what we do in image segmentation (UNet). We down-sample and then up-sample to get the same size of output as the input image. The difference is that the model will predict noise being added corresponding to the current time-step. What worth mentioning is that: to generate conditional DDPM process, the model needs to accept time embedding and context(condition) embedding. To achieve this, we convert the input condition into one-hot vectors. The vectors of time embedding and condition embedding will then pass fully connected layer and thus have the same dimensions as the feature in the hidden layer. The input of up-sampling layer will thus be the combination of this feature of image, feature of time-step and feature of condition.

Hyperparameters:
Epochs: 100
Batch Size: 256
Total time-steps: 400
Beta Scheduler: Linear Scheduling with Beta1 = 1e-4, Beta2 = 2e-2
Optimizer: Adam with Learning Rate = 1e-4
Scheduler: Cosine Annealing with Tmax = 100, eta_min = 1e-6
Drop Probability: 0.1
Guide w: 2.0
Loss function: Cross Entropy Loss

During the training step, we minimize the Cross-entropy loss of the model between ground truth noise and the noise predicted by the model. At the validation stage, the model will generate 100 images (10 for each digit), and the accuracy of prediction by the classifier will be determined by the digit classifier. We can then select the model with the highest accuracy and then save it.

The implementation details of my reverse process are similar to (https://github.com/TeaPearce/Conditional_Diffusion_MNIST/blob/main/script.py)
Note that we use the idea of 'Classifier-Free Diffusion Guidance' (https://arxiv.org/abs/2207.12598) to force the output of the de-nosing more condition-dependent.)
During the reverse process stage, we will mix the conditional and un-conditional output at the reverse process stage. The parameters “Guild w” determines the weights of conditional and unconditional components. This is also the reason why we set the parameters “drop probability”. We can put some unconditional input in the training stage to make the type of images more various.

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.**

t = 400 t = 320 t = 240 t = 160 t = 80 t = 1

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

After finishing this part, I have a very clear understanding of conditional diffusion model, including how to do parameters scheduling, model selecting and condition embedding. I have tried many different ways (ex: put embedding at the down-sampling layer of UNet) to embed the condition, but the original method in (https://github.com/TeaPearce/Conditional_Diffusion_MNIST/blob/main/script.py) gives us the best result.

The Diffusion model can generate very clear images according to different conditions, especially when it is combined with diffusion guidance. What is even more surprising is: the accuracy of prediction by classifier can reach 100%, which really amazes me. However, the generating process of images is also time-consuming, and it takes me a lot of time to train and select a best model. To summarize, implementing diffusion model is an interesting experience and helps me find some insights of generating model in computer vision.

Problem 2: DDIM

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.**

We can see that when eta = 0, the images being generated are always the same (same as the GT images). When eta is small (<0.5), there is some randomness on generated images, and the images will still look like the GT images to some extent, such as the location of the face, the background or people's expression. When eta is high (>0.5), the randomness goes up, and we can clearly see the difference between generated images and GT images.

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. What will happen if we simply use linear interpolation? Explain and report your observation.

Slerp:

α = {0.0, 0.1, 0.2, …, 1.0} from the left to the right, respectively.

When α is small, the image looks like the man (00.pt). As α increase, the image looks more like the woman(01.pt). Moreover, we can see that background also changes from blue to white according to different α. When we mix two noise together, I feel that the spherical linear interpolation gives us a very "linear" change.

Linear:

α = {0.0, 0.1, 0.2, …, 1.0} from the left to the right, respectively.

When we use simple linear interpolation, we can see that the model generates a person looks like neither the men nor the women, and the person has the pink hairs (or surrounded by the pink background?), which is very weird. Maybe it is not suitable to use simple linear interpolation to mix noises together when we generate images.

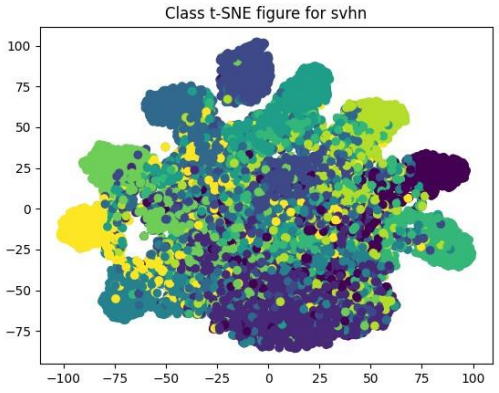
Problem 3: DANN

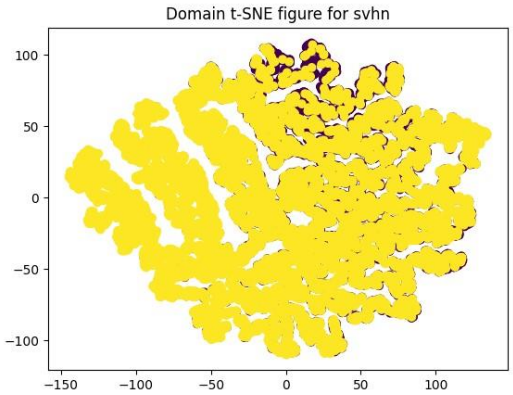
1. **Please create and fill the table with the following format in your report:**

	MNIST-M → SVHN	MNIST-M → USPS
Trained on source	0.3371	0.7392
Adaptation (DANN)	0.4443	0.7890
Trained on target	0.9097	0.9879

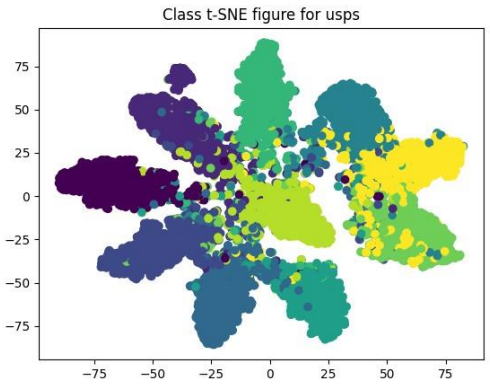
2. **Please visualize the latent space (output of CNN layers) of DANN by mapping the *validation* images to 2D space with t-SNE. For each scenario, you need to plot two figures which are colored by digit class (0-9) and by domain, respectively.**

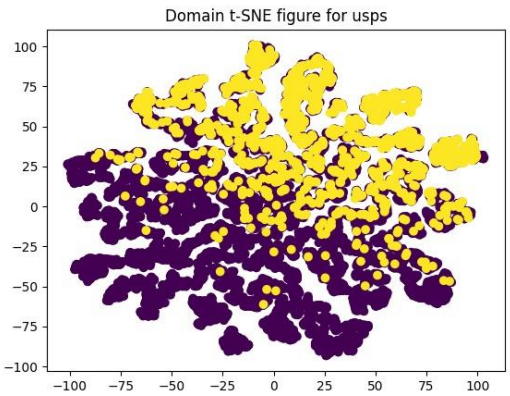
SVHN:





USPS:





3. Please describe the implementation details of your model and discuss what you've observed and learned from implementing DANN.

Both models are adapted from this work:
https://github.com/NaJaeMin92/pytorch_DANN

MNIST-M → SVHN model:

```
NN_SVHN(
  (extractor): Sequential(
    (0): Conv2d(3, 48, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): BatchNorm2d(48, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (4): Conv2d(48, 64, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (5): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): Dropout(p=0.5, inplace=False)
    (7): ReLU()
    (8): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (9): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (10): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (11): ReLU()
  )
  (classifier): Sequential(
    (0): Linear(in_features=3136, out_features=1024, bias=True)
    (1): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): Linear(in_features=1024, out_features=128, bias=True)
    (4): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU()
    (6): Linear(in_features=128, out_features=10, bias=True)
  )
  (discriminator): Sequential(
    (0): Linear(in_features=3136, out_features=256, bias=True)
    (1): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): Linear(in_features=256, out_features=2, bias=True)
  )
)
```

MNIST-M → USPS model:

```
NN_USPS(
  (extractor): Sequential(
    (0): Conv2d(3, 32, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (4): Conv2d(32, 64, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (5): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): ReLU()
    (7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (8): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (10): ReLU()
  )
  (classifier): Sequential(
    (0): Linear(in_features=6272, out_features=1024, bias=True)
    (1): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): Dropout(p=0.2, inplace=False)
    (3): ReLU()
    (4): Linear(in_features=1024, out_features=128, bias=True)
    (5): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): Dropout(p=0.2, inplace=False)
    (7): ReLU()
    (8): Linear(in_features=128, out_features=10, bias=True)
  )
  (discriminator): Sequential(
    (0): Linear(in_features=6272, out_features=256, bias=True)
    (1): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): Dropout(p=0.2, inplace=False)
    (3): ReLU()
    (4): Linear(in_features=256, out_features=2, bias=True)
  )
)
```

Hyperparameters (MNIST-M → SVHN):

Epochs: 200

Batch Size: 1024

Optimizer: Adam with Learning Rate = 2e-3

Loss function: Cross Entropy Loss for both class and domain

Scheduler:

The scheduler given in the paper of DANN (https://arxiv.org/abs/1505.07818):

$$\mu_p = \frac{\mu_0}{(1 + \alpha \cdot p)^\beta}$$

Here we set beta = 0.75 and alpha = 10, where p = (current iteration of training / total iteration in training).

Lambda:

We use lambda given in the paper of DANN.

$$\lambda_p = \frac{2}{1 + \exp(-\gamma \cdot p)} - 1$$

Here we set gamma = 10.

Hyperparameters (MNIST-M → USPS):

Epochs: 200

Batch Size: 1024

Optimizer: Adam with Learning Rate = 2e-4

Loss function: Cross Entropy Loss for both class and domain

Scheduler:

The scheduler given in the paper of DANN (https://arxiv.org/abs/1505.07818):

$$\mu_p = \frac{\mu_0}{(1 + \alpha \cdot p)^\beta}$$

Here we set beta = 0.75 and alpha = 10, where p = (current iteration of training / total iteration in training).

Lambda:

We use lambda given in the paper of DANN.

$$\lambda_p = \frac{2}{1 + \exp(-\gamma \cdot p)} - 1$$

Here we set gamma = 10.

In training, I select the model with the best validation accuracy on the target dataset. At the beginning of this part, I modify the UNet from problem1 and start training, the result is that the model seems to overfit. Therefore, I use the relatively small model from (https://github.com/NaJaeMin92/pytorch_DANN) in this part. The overfit problem still seems to occur on MNIST-M → SVHN, so use a smaller feature extractor in MNIST-M → SVHN and add a Dropout layer. I also convert USPS dataset to RGB before training, and thus we can use 3-channels model on both datasets.

From the accuracy reported in question1, we can clearly see that the accuracy really depends on the setting of training images:

Target Acc > DANN Acc > Source Acc

It is quite intuitive that the more images (or labels) the model see on the target domain, the better performance it gives.

The accuracy also differs on two datasets:

USPS Acc > SVHN Acc

Although images in USPS are Black and white originally, the distribution between MNISTM and USPS seems to be more likely (from my naked eye) than MNISTM and SVHN. I guess that is the reason why the accuracy of USPS is better than SVHN in all setting. The Class t-SNE figures for two datasets also justify this point of view.