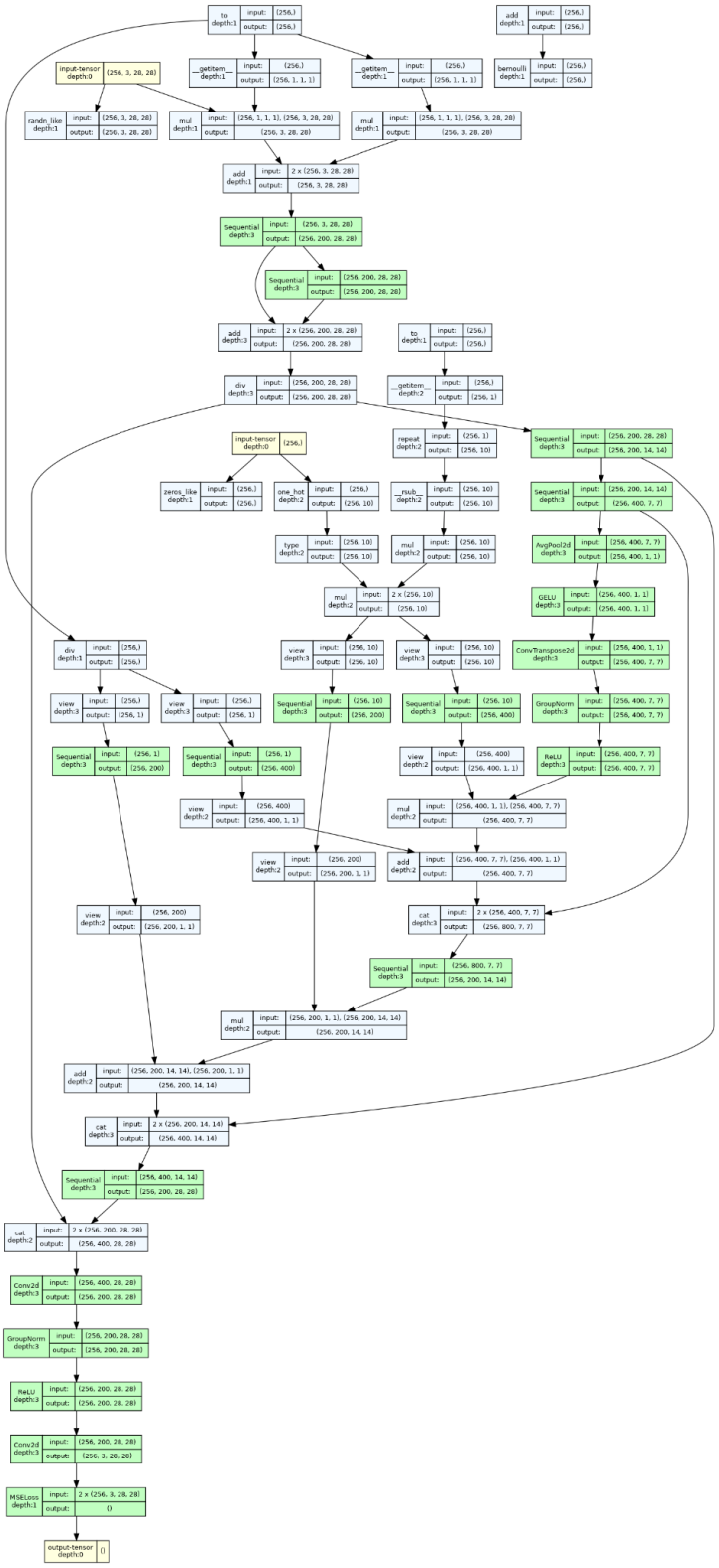
DLCV HW2

1.1

Model Architecture:



The model uses DDPM from the GitHub repository: https://github.com/TeaPearce/Conditional_Diffusion_MNIST, and the values set during training are also the same as those in the repository. This model is based on the U-Net architecture, which is a type of convolutional neural network (CNN) that is used for semantic segmentation. U-Net consists of a contracting path (encoder) that captures context, and an expansive path (decoder) that enables precise localization. Here, the U-Net model is adapted for denoising images and named `ContextUnet`. The model takes as input a noisy image `x`, a context label `c`, a timestep `t`, and a context mask `context_mask`. The output is a denoised version of the input image.

The noise addition process is part of the `DDPM` class, which is a implementation of denoising diffusion probabilistic models (DDPM). DDPM is a generative model that learns to generate samples from a data distribution by gradually adding noise to the data samples and then learning to reverse this noise process. In the `forward` method of the `DDPM` class, a random timestep `_ts` is sampled, and Gaussian noise is added to the input image `x` according to the chosen timestep. This noisy image `x_t` is then passed through the `ContextUnet` model to predict the added noise. The loss is computed as the mean squared error (MSE) between the predicted noise and the true added noise.

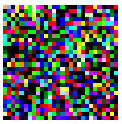
The U-Net model architecture in `ContextUnet` consists of an initial convolutional block `init_conv`, followed by two downsampling blocks `down1` and `down2`, a vectorization block `to_vec`, four embedding blocks `timeembed1`, `timeembed2`, `contextembed1`, and `contextembed2`, and three upsampling blocks `up0`, `up1`, and `up2`, followed by an output block `out`. The downsampling blocks use residual convolutional blocks and max pooling layers to reduce the spatial dimensions of the input image, while the upsampling blocks use transposed convolutional layers and residual convolutional blocks to increase the spatial dimensions of the image. The embedding blocks are used to embed the context label and timestep into the feature space. The output block uses convolutional layers to generate the final denoised image.

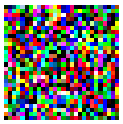
In the `forward` method of `ContextUnet`, the input image is passed through the initial convolutional block, followed by the downsampling blocks, and then vectorized. The context label is converted to a one-hot encoding and masked according to the `context_mask` parameter. The masked context label and timestep are then embedded using the embedding blocks. The embedded context and timestep are used to modulate the feature maps in the upsampling blocks. The final denoised image is generated by the output block.

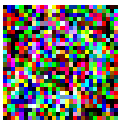
1.2

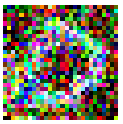
1.3

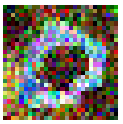
t = [400,1]


t = 400


t = 241


t = 161


t = 81


t = 41


t = 1

1.4

Denosing Diffusion Probabilistic Models (DDPM) are a type of generative model that have become very popular for their ability to produce high-quality samples. This model works by slowly transforming data samples into pure noise through a diffusion process, and then learning to reverse this process to generate new samples starting from noise. One particularly interesting extension of DDPM is the conditional diffusion model, which generates samples conditioned on certain attributes.

The implementation of the conditional diffusion model for generating images of handwritten digits based on their class label was a fascinating and educational experience. The model learns to incorporate the conditioning information, allowing us to have control over the content of the generated images. This is very powerful as it opens up the possibility of generating samples with specific desired characteristics. Over the course of training, I observed the quality of the generated images improve, and the model was able to produce diverse samples that were consistent with the conditioning information.

However, training the conditional DDPM was not without its challenges. There are several hyperparameters that need to be carefully tuned, such as the number of diffusion steps and the learning rate. These hyperparameters play a crucial role in determining the performance of the model, and getting them just right can be a bit of a trial and error process. Additionally, evaluating the performance of the model requires a separate classifier that is trained to classify the generated images. This involves an additional step of training and validation, and the accuracy score obtained gives us a quantitative measure of how well the DDPM is able to generate images that match the conditioning information.

In conclusion, working with DDPM and conditional diffusion models has been a valuable learning experience. It has given me a deep understanding of the DDPM framework and the potential it holds for generating high-quality samples. The process of training, tuning, and evaluating the model has also taught me about the intricacies and considerations involved in working with generative models. I am excited to explore more applications and extensions of DDPM in the future.

2.1

DDIM from top to bottom of the eta value is getting bigger and bigger, which means the randomness is getting bigger and bigger, and when the eta equals to 1, it is DDPM. you can see that the top and bottom rows look completely different, which means that the weight of random noises is getting bigger and bigger, which leads to the generation of two different faces in the end.

2.2

Through my observation of the generated images, I can conclude that the slerp function maintains facial coherence in all intermediate stages. Slerp manages to do so by effectively traversing the shortest path on the data's manifold.

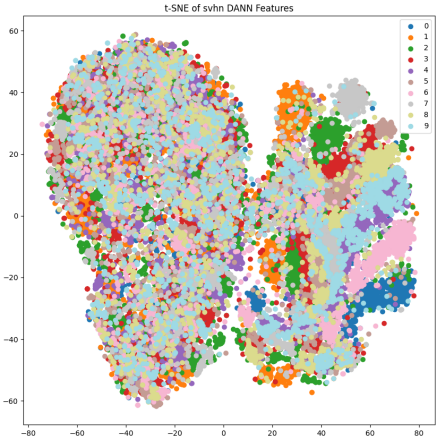
I think if we use linear interpolation instead, it would simply take a direct path between the start and end points in the feature space, potentially leading to unrealistic combinations of features that do not lie on the manifold of natural faces. Given the structure of human facial features and the way they vary, a straight-line interpolation in the latent space might result in unnatural facial constructions.

In conclusion, I think that the use of spherical linear interpolation in the context of DDIM is effective for generating intermediate face images that are realistic and display a coherent transition between two different individuals. This is because slerp accounts for the spherical geometry of the latent space where these facial representations reside.

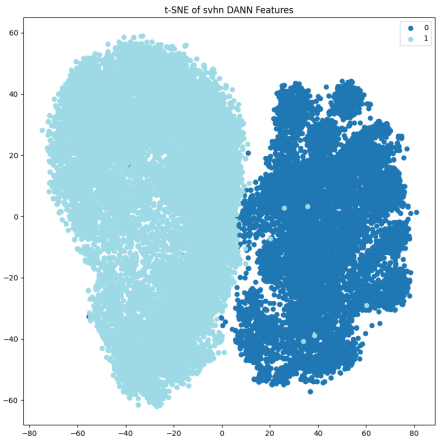
3.1

	MNIST-M → SVHN	MNIST-M → USPS
Trained on source	37.74%	75.25%
Adaptation (DANN)	42.06%	82.51%
Trained on target	98.40%	90.87%

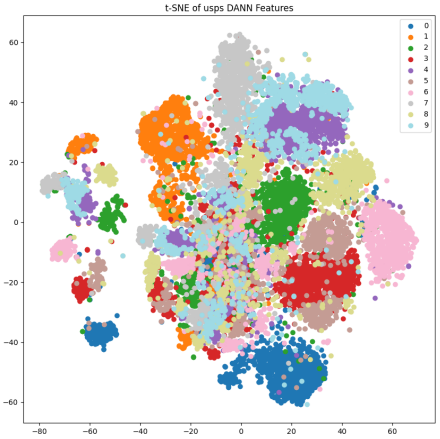
3.2



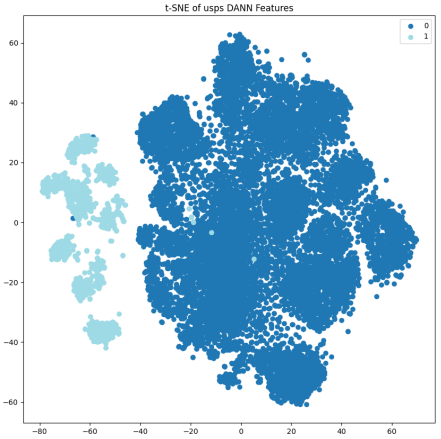
TSNE(svhn class)



TSNE(svhn domain)



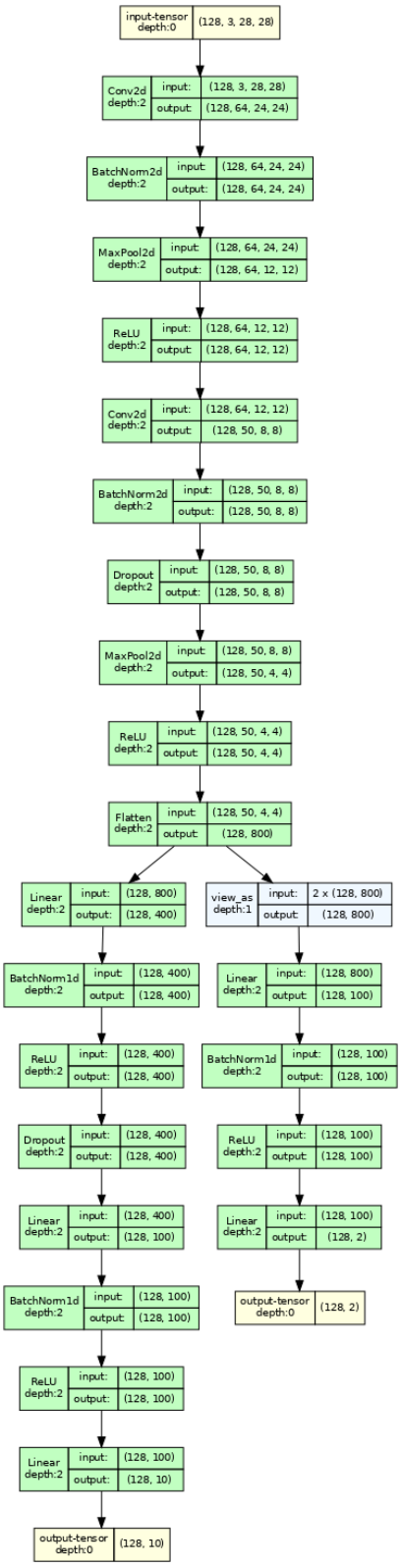
TSNE(usps class)



TSNE(usps domain)

3.3

Reference github repository: https://github.com/fungtion/DANN/tree/master
I've built the following model with reference to the above repository's modeling framework and training methodology



The implementation of the Domain-Adversarial Neural Network (DANN) model consisted of defining the architecture and the training loop for the model. The architecture of the DANN model is comprised of three primary components: the feature extractor, the class classifier, and the domain classifier. The feature extractor consists of convolutional layers that process the input image and extract relevant features. The class classifier is made up of fully connected layers that take the features extracted by the feature extractor and predict the class label of the input image. The domain classifier is another set of fully connected layers that predict the domain of the input image (source or target).

A key component of the DANN architecture is the gradient reversal layer, which is placed between the feature extractor and the domain classifier. The gradient reversal layer multiplies the gradient by a negative scalar during backpropagation, effectively reversing the gradient. This encourages the feature extractor to learn features that are useful for class prediction while being invariant to the domain.

In the training loop, the model processes batches of source and target data simultaneously. For each batch, the source images are passed through the model to obtain class and domain predictions, and the corresponding loss is calculated. The target images are also passed through the model to obtain domain predictions, and the loss is calculated. The total loss is then backpropagated to update the model parameters. A crucial hyperparameter in this process is the alpha parameter, which controls the weight of the gradient reversal layer and therefore the trade-off between class and domain prediction.

Through the implementation of DANN, I observed the importance of carefully tuning hyperparameters such as the learning rate, batch size, and number of epochs to ensure that the model converges to a good solution. I also learned about the challenges of domain adaptation, where the source and target data come from different distributions. In this case, not only the MNIST-M and SVHN datasets, but also MNIST-M and USPS have different distributions, which makes it difficult to learn a model that performs well on both domains. However, by using the DANN approach, I was able to learn features that are invariant to the domain and achieve good performance on the target domain. This experience has shown me the potential of DANN in addressing the challenges of domain adaptation and the importance of domain-invariant feature learning in machine learning.