Assignment2: Report

Ronak Dedhiya Dinesh (ronakdedhiya@iisc.ac.in; SR No: 06-18-05-19-52-21-1-20116)
Prashanth N Bhat(prashanthbn@iisc.ac.in; SR No: 04-03-06-19-52-21-1-20082)

September 28, 2024

1 Problem Statement

1. **Diffusion Models and Family**: Construct a generative model on the CelebA and Bitmoji datasets using (a) Noise prediction formulation, (b) score-based formulation (Denoising Score Matching). Use a UNET with sinusoidal embedding for the time variable in both cases. Experiment with different time steps for langevian sampling, generate 1000 data samples, compute the FID and compare with results obtained with GAN-based models training in A1. Plot the denoising steps for 10 steps for 10 generated images in a grid for both noise prediction and score-based formulations. Finally, choose 10 random categories from the Celeb-A dataset and construct a classifier-guided diffusion model. Plot a 10x10 grid of generated images with each row corresponding to a category of the dataset.
2. **Domain Adaptation**: For these experiments, we use two sets of datasets:
   (a) USPS and MNIST
   (b) Clipart and Real-world categories of Office-Home dataset

   The former is the case where both the datasets contain images of handwritten digits and the latter contains images of 65 categories from four domains (of which you are supposed to work on only Clipart and Real-world categories). The following are the tasks:
   (a) First build a Resnet-50 based classifiers for both the pairs of datasets. You should train the classifier on one of the domains in both the cases and report the test accuracy on the other (on which the classifier is not trained).
   (b) Implement Domain-Adversarial Training of Neural Networks (DANN) where you modify the base Resent-50 with the gradient reversal layer. Report the results with gradient reversal layer at three different stages of the base classifier.

2 Diffusion Models and Family

2.1 Introduction

We implement a noise-based formulation of the diffusion model. It involves two processes:

- **Forward Process:**
  - The forward process adds noise to the data $x_0 \sim q(x_0)$, for T time steps.

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{\alpha_t}x_{t-1}, (1-\alpha_t)I) \tag{1}$$

$$q(x_{1:T}|x_0) = \prod_{t=1}^{T} q(x_t|x_{t-1}) \tag{2}$$

    where $\alpha_1, ..., \alpha_t$ is the variance schedule.
  - We can sample x_t at any time step t with

$$q(x_t|x0) = \mathcal{N}(x_t; \sqrt{\tilde{\alpha}_t}, (1-\tilde{\alpha}_t)I) \tag{3}$$

$$\tilde{\alpha_t} = \prod_{s=1}^{t} \alpha_s \tag{4}$$

- **Reverse Process:**

– The reverse process removes noise starting at $p(x_T) = \mathcal{N}(x_T; 0, I)$ for T time steps.

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)) \tag{5}$$

$$p_\theta(x_{0:T}) = p_\theta(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t) \tag{6}$$

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}}(x_t - \frac{1-\alpha_t}{\sqrt{1-\tilde{\alpha}_t}}\epsilon_\theta(x_t, t)) \tag{7}$$

θ are the parameters we train.

- **Loss:**
  - ELBO on negative log-likelihood is optimized, and we obtain the below loss:

$$L = \mathbb{E}_{x_0,\epsilon}[\frac{(1-\alpha_t)^2}{2\sigma^2\alpha_t(1-\tilde{\alpha_t})}||\epsilon - \epsilon_\theta(\sqrt{\tilde{\alpha}_t}x_0 + \sqrt{1-\tilde{\alpha}_t}\epsilon, t)||^2] \tag{8}$$

$$L_{simple}(\theta) = \mathbb{E}_{t,x_0,\epsilon}[||\epsilon - \epsilon_\theta(\sqrt{\tilde{\alpha}_t}x_0 + \sqrt{1-\tilde{\alpha}_t}\epsilon, t)||^2] \tag{9}$$

2.2 Implementation

Modified UNet Model is implemented, which includes:

- **Swish Activation Function:**
  - With ReLU, the consistent problem is that its derivative is 0 for half of the values of the input x.

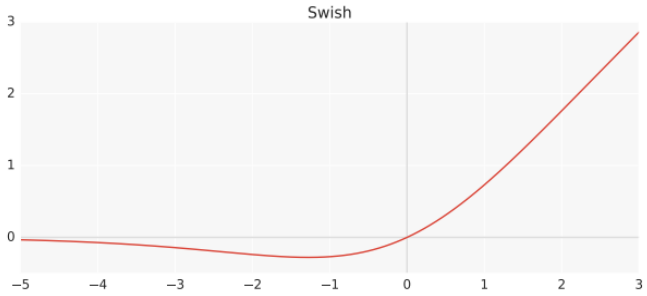
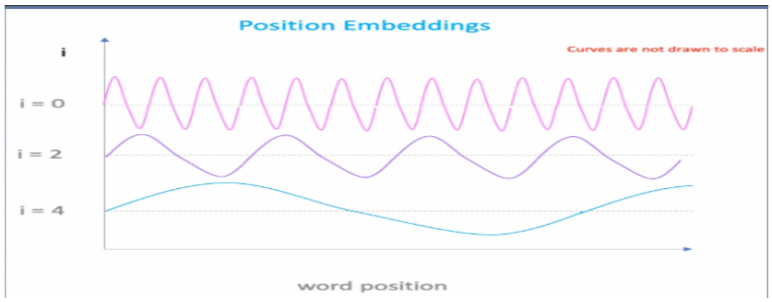


Figure 1: Swish Activation

Swish is similar to Relu and has a smooth, non-monotonic function that consistently matches or outperforms ReLU.

- **Time Step Embedding:**
  - We want to provide time-step information to the model along with the input image data. However, labeling the time step index as 1,2,3...so on can lead to a large number and may hurt the generalization of our model.
  - The proposed sinusoidal embedding represents the time as not a single number but a d-dimensional vector as illustrated in Fig. 2
  - The frequencies are decreasing along the vector dimension. Imagine the positional embedding as a vector containing pairs of sines and cosines for each frequency. It is similar to a binary representation of the number where the LSB bit alternate on every number and the second lowest bit is rotating on every two numbers. Here Sinusoidal functions are equivalent to alternating bits and there is a decrease in frequency along the vector dimension.
- **Residual Block:**
  - A residual block has two convolution layers with group normalization. We use two such residual blocks to process each resolution.

$$\overrightarrow{p_t}^{(i)} = f(t)^{(i)} := \begin{cases} \sin(\omega_k . t), & \text{if } i = 2k \\ \cos(\omega_k . t), & \text{if } i = 2k+1 \end{cases} \qquad \omega_k = \frac{1}{10000^{2k/d}}$$



$$\overrightarrow{p_t} = \begin{bmatrix} \sin(\omega_1 . t) \\ \cos(\omega_1 . t) \\ \sin(\omega_2 . t) \\ \cos(\omega_2 . t) \\ \vdots \\ \sin(\omega_{d/2} . t) \\ \cos(\omega_{d/2} . t) \end{bmatrix}_{d \times 1}$$

Figure 2: Time/Positional Embedding

  - Group Normalization is a normalization layer that divides channels into groups and normalizes the features within each group. GN does not exploit the batch dimension, and its computation is independent of batch sizes. In the case where the group size is 1, it is equivalent to Instance Normalization. See Fig. 3 for a diagrammatic illustration.

- **Attention Block:**
  - Multi-head Attention is a module for attention mechanisms that run through an attention mechanism several times in parallel. The independent attention outputs are then concatenated and linearly transformed into the expected dimension as shown in Fig. 3
  - Scaled dot-product attention is an attention mechanism where the dot products are scaled down by $\sqrt{d_k}$. Formally we have a query, a key, and a value and calculate the attention as:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{10}$$

- **UNET:**
  - U-Net model is used to predict noise $\epsilon_\theta(x_t, t)$. U-Net model is constructed with a lot of modification as shown in Fig. 3

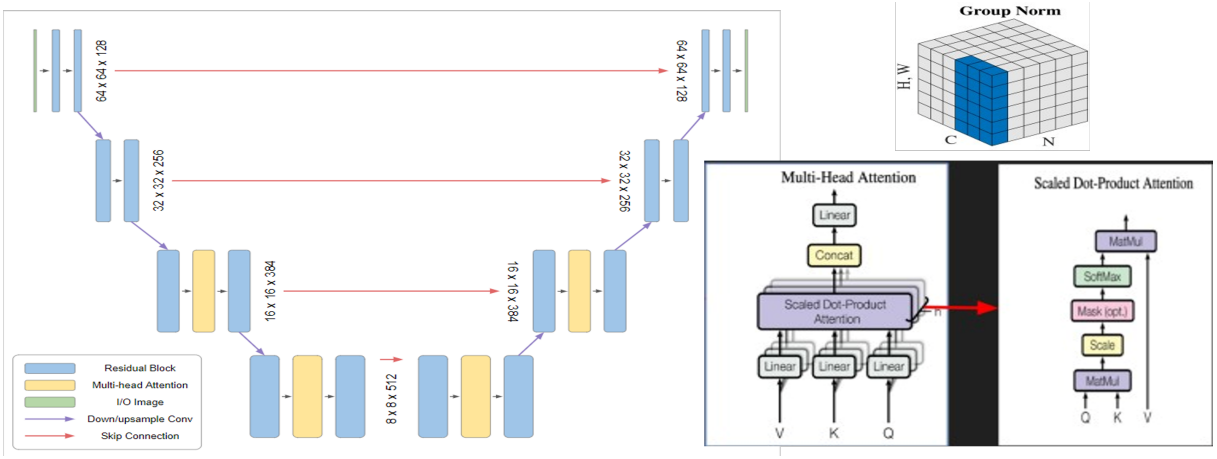


Figure 3: Modified UNET Model with Multi-head attention & Group normalization

  - U-Net processes a given image by progressively lowering (halving) the feature map resolution and then increasing the resolution. There is a pass-through connection at each resolution. This implementation contains a bunch of modifications to the original U-Net (residual blocks, multi-head attention) and also adds time-step embeddings t.

2.3 Results

- **FID Score on celebA and Bitmoji Dataset**
  - DDPM model is trained with T = 1000 and obtains FID scores of 50.84 for the celebA dataset and FID score of 58.34 for the Bitmoji dataset. In our previous exercise on training with the GAN model, we obtained an FID score of 155.14 on the bitmoji dataset and we clearly see the outperformance with a simple DDPM model with just 10 epochs against the GAN model.

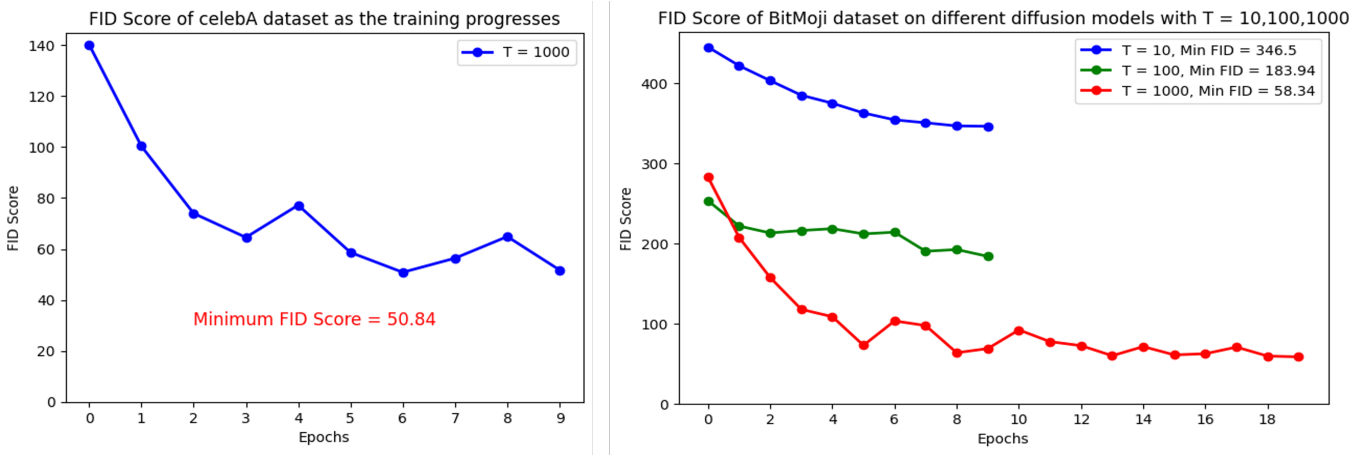


Figure 4: (Left) FID score on celebA dataset with T = 1000, (Right) FID score on Bitmoji dataset with T = 10,100,1000 as training progresses

  - As training progresses, the FID score decreases. However, we didn't find much change in FID score after 10 epochs. Additional hyperparameter tuning or learning rate scheduling is required to further reduce the loss as training progresses. We saw models are unable to learn when T = 10, 100 is used.

- **Samples Generated from trained diffusion models:**
  - We generate the samples from the trained diffusion model using the reverse process starting with the random noise. Fig. 5 shows generated output from diffusion model trained with T = 1000. We obtain satisfactory generated samples with the models trained only with 10 epochs.
  - We saw larger the T, the more times it takes to generate the samples.

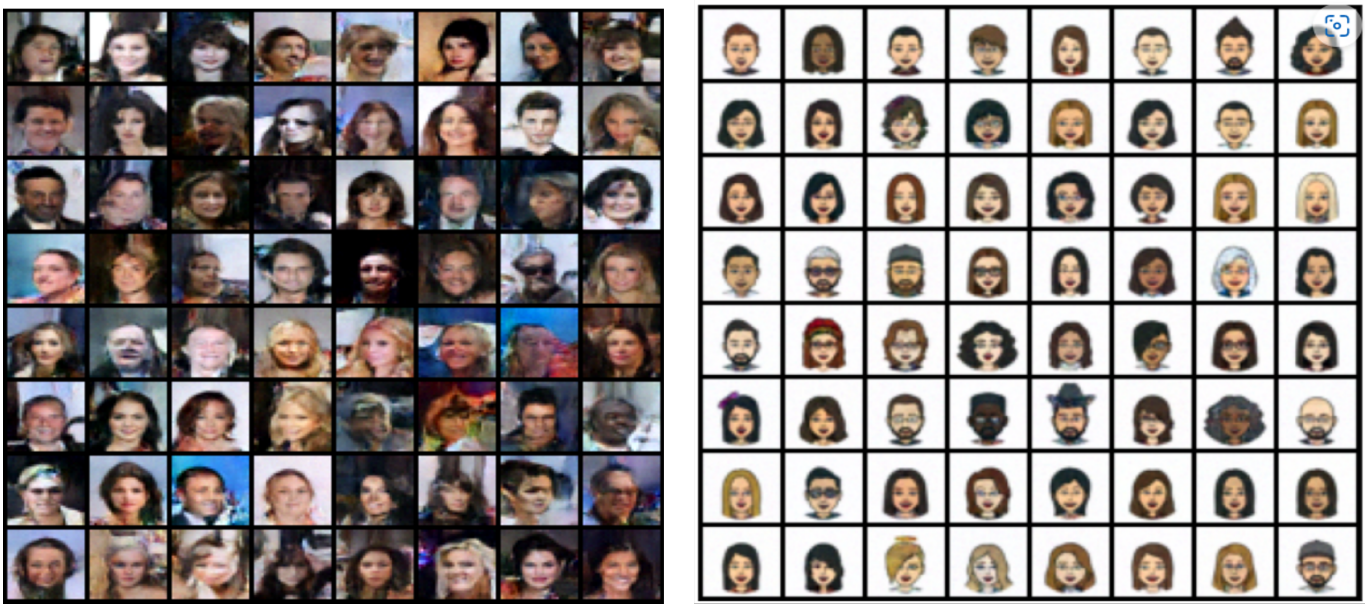


Figure 5: Generated samples from diffusion model trained on celebA (Left) and Bitmoji (Right)

- **Denoising step for celebA DDPM Model**
  - Images are generated using the sampling process with T = 1000. Instead of considering the last 10 steps, we considered intermediate steps from t = 0 to 1000 and plotted the denoising steps as shown in Fig. 6

Figure 6: Denosing step for celebA DDPM Model

- **Classifier Guided Diffusion model**
  - Classifier guided diffusion model is simple extension to DDPM model wherein label information is embedded with the time embedding and passed as additional information to generate the class conditional output.
  - We selected 10 classes = Bald, Black Hair, Blond Hair, Brown Hair, Eyeglasses, Male, Smiling, Mustache, No Beard, Young.

Figure 7: Class conditional Generated samples

- The labels for these 10 classes have values -1 or 1 for every class. We pass it to label embedding which is 2 layer MLP network and output from label embedding is simply added with time embedding passed on to the UNet network.
- while sampling, we kept only one class active(i.e value = 1) and rest other as (value = -1). We did this similarly for all the 10 classes and obtained the class conditional generated samples.
- Images generated are not of high quality, but it is attempting to generate images based on active class. For example, Images generated with black, blond, and brown hair. Also, it produces all male images for the Male category and produces people with eyeglasses for the eyeglasses category. It sufferers in creating a bald image or creating images with a mustache(bcoz here Male category was set -1 i.e., Female).

3 Domain Adaptation

The implementation is based on the paper titled *Domain-Adversarial Training of Neural Networks* by *Ganin et al*[1]. The paper introduces a new layer called *Gradient Reversal Layer*. This layer helps in *Domain Adaptation* where a *Deep Neural Network (DNN)* trained on classification of one domain can classify a similar domain whose distribution is close to the original domain.

3.1 Dataset Preparation

- For each set of datasets, one dataset is selected to be the source, and the remaining to be the target.
- For the MNIST and USPS set, USPS is set as the source, and MNIST as the target.
- For all the datasets, the labels are converted to one-hot encoding.
- In addition to the original label, an additional label is assigned to each dataset:
  - For the MNIST and USPS, assign 0 to MNIST and 1 to USPS dataset.
- For the MNIST and USPS:
  - The USPS dataset is of size $16x16x1$ meaning $16x16$ in *Grayscale*.
  - So, $16x16$ matrix is copied to each channel converting it to a $16x16x3$ image.
  - MNIST dataset is also imported in a $32x32x1$ format to match the USPS dataset.
  - Use $sklearn.transform$ to convert images to $32x32x3$ format.

3.2 Model architecture

The model architecture used for both datasets is depicted in Figure 8.

- The input image size is $32x32x3$
- The right branch is the base classifier with a resnet50 base.
  - The output layer has 10 neurons for the 10 digits for MNIST-USPS dataset.
- The left branch is connected to different points with a $GradientReversalLayer$ denoted by $GradReverse$
  - The output layer has only 1 neuron with a sigmoid activation for domain classification.

3.2.1 Gradient Reversal Layer

The Gradient Reversal layer is implemented as a $Tensorflow$ class in the following manner:

```
# Gradient reversal layer
@tf.custom_gradient
def grad_reverse(x):
    y = tf.identity(x)
    def custom_grad(dy):
        return -dy * lambda
    return y, custom_grad


class GradReverse(tf.keras.layers.Layer):
```

```
def __init__(self):
    super().__init__()
def call(self, x):
    return grad_reverse(x)
```

Where lambda is a hyperparameter. We got the best results with lambda = 0.001

3.2.2 Connection of Gradient Reversal Layer to main classifier

1. **Stage 1**: Gradient Reversal Layer is connected to output of resnet50.
2. **Stage 2**: Gradient Reversal Layer is connected to output of 1^{st} dense layer of label classifier.
3. **Stage 3**: Gradient Reversal Layer is connected to output of label classifier.

3.3 Results

The results for each of the experiments are depicted in Figure 8. They indicate the accuracy in predicting labels for source and target dataset.

- For MNIST-USPS dataset:
  - Significant improvement is seen when Gradient Reversal layer is attached right at the output of resnet50 model.
  - Other locations do not see any improvement in prediction accuracy.

3.4 Other Notes

- The model structure was plotted using Netron

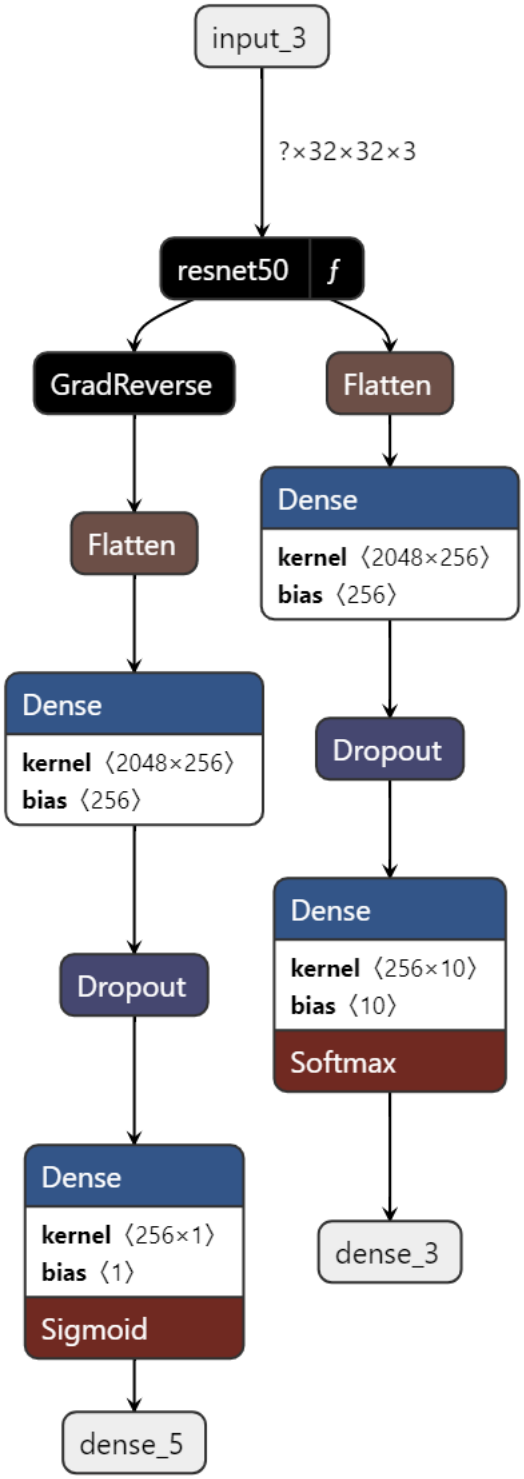


Figure 8: DANN architecture

		Source domain	Target domain
MNIST/USPS	No Gradient Reversal	95.20%	53.57%
	Gradient Reversal Stage 1	99.62%	73.33%
	Gradient Reversal Stage 2	76.42%	35.39%
	Gradient Reversal Stage 3	98.36%	55.76%

Figure 9: DANN results

References

[1] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. 2015.