EEE7331-01
SPECIAL TOPICS IN DEEP LEARNING

2025 FALL

Assignment

Due Date: 1st of November (Saturday) before midnight to LearnUs.

INSTRUCTIONS:

1. This question paper consists of 9 pages.
2. Attempt ALL questions.
3. This is an individual-based assignment. Discussions among students are welcome. However, any attempt at cheating will be penalized.
4. Attach your code as an appendix. You may show code snippets to aid your explanation.
5. Upload the solution in a single PDF file.
6. Use English in the report.

Question 1 (35%):

In this question, you will implement a Variational Autoencoder (VAE) using the STL-10 dataset (https://cs.stanford.edu/~acoates/stl10/). You are expected to use PyTorch (or another deep learning framework) to complete this assignment.

Part 1: Building and Understanding a VAE

1. Define the VAE Model

Design and implement a VAE model for the STL-10 dataset. Your model should consist of:

- An encoder network that maps images to a latent space, producing both mean and variance vectors.
- A decoder network that reconstructs images from the latent vectors.
- Implement the encoder and decoder architectures using convolutional layers. Provide the specific details of the layers (# of conv. layers, # of channels in each layer, dimension of latent vector, etc) you have designed in a **table form**.
- Implement the reparameterization trick.
- Define the VAE loss function, which combines the reconstruction loss (binary cross-entropy or mean squared error) and the KL-divergence between the learned latent distribution and the unit Gaussian prior.

2. Training and Image Generation

Train the VAE on the STL-10 for at least 50 epochs and monitor the training loss over time.

- Train the model and plot the reconstruction loss, KL-divergence, and total loss over the epochs.
- After training, sample new latent vectors from a standard Gaussian distribution and decode them into images.
- Generate and show 10 random images by sampling from the latent space.

Discuss the quality of the generated images and whether they resemble real STL-10 data.

Part 2: Latent Space Interpolation and Style Transfer

1. Latent Space Interpolation

After training, the latent space should capture meaningful features of the STL-10 dataset. To explore this space, you will perform latent space interpolation.

- Select two random images from different classes in the STL-10.
- Encode these images to obtain their corresponding latent vectors $\mathbf{z}_1$ and $\mathbf{z}_2$.
- Perform linear interpolation between these latent vectors:

$$\mathbf{z}_{\text{interpolated}} = (1-\alpha)\mathbf{z}_1 + \alpha\mathbf{z}_2 \text{ where } \alpha \text{ in } [0, 1]$$

- Decode the $\mathbf{z}_{\text{interpolated}}$ to generate images.

- Show the images generated along the interpolation path for $\alpha \in [0, 0.25, 0.5, 0.75, 1]$. Two images for each α.

a) Describe the qualitative differences in the generated images along the interpolation path.

b) What does this tell you about the continuity and structure of the learned latent space?

2. Style Transfer Using Latent Vector Arithmetic

The goal is to transfer "style" features, like background color or texture, from Image B to Image A.

- Select two images, A and B, from **different classes** in STL-10.
- Feed both images into the encoder of your trained VAE to obtain their corresponding latent vectors, $\mathbf{z}_A$ and $\mathbf{z}_B$. These latent vectors represent the high-level features of the images, such as shape, color, and texture.
- Compute the mean latent vectors $\mathbf{z}_{mean}$ by averaging the latent vectors of many samples of the **same class**. This can help isolate class-specific attributes in latent space and give you a rough estimate of class-level characteristics (style or structure).
- Perform latent vector arithmetic to transfer the style (texture, background color, etc.) from Image B to Image A. The latent vector manipulation for style transfer can be defined as:

$$\mathbf{z}_{new} = \mathbf{z}_A + \lambda(\mathbf{z}_B - \mathbf{z}_{mean})$$

  where λ is a scalar controlling how much of the style of Image B you want to apply to Image A. The expression $(\mathbf{z}_B - \mathbf{z}_{mean})$ isolates the style of Image B.

- Also try $\mathbf{z'}_{new} = \mathbf{z}_A + \alpha(\mathbf{z}_B - \mathbf{z}_A)$ where $\alpha \in [0,1]$. This method interpolates between the two latent vectors to blend the styles.
- Once you have $\mathbf{z}_{new}$ and $\mathbf{z'}_{new}$, pass them through the decoder to generate a new image.

a) Try different hyperparameters λ and α (0.0, 0.2, ...,1.0) to control the strength of the style transfer. By varying these hyperparameters, plot two versions of each hyperparameter of the new image.

b) Examine how the VAE captures features such as shape, texture, and color as the interpolation progresses.

c) Describe the changes you observe along the interpolation path.

d) Does the background, object shape, or other visual features gradually morph between the two images? What image properties seem to be preserved or smoothly transitioned?

Question 2: StyleGAN Inversion (30%)

In this question, you will follow the provided tutorial (EEE7331_GAN_Inversion.ipynb) to invert face images into StyleGAN latent space. The tutorial works with ***Google Colaboratory*** *by default.*

Since StyleGAN has many version issues, running the ipynb notebook in Google Colaboratory is ***highly*** *recommended. However, you can also run this notebook in your local environment if you meet the requirements for the stylegan2-PyTorch repository.*

To open the .ipynb file on Google Colaboratory, upload the provided folder in your Google Drive, then download Google Colaboratory from Google Workspace Marketplace. With this, you will be able to open the .ipynb file.

Part 1: Effect of Latent Space on Image Inversion

In the tutorial, we invert the **Obama.jpg** using a simple MSE loss. Perform the same experiment for all 20 target images and fill in the table below.

You must report the average and the standard deviation for the three metrics. Additionally, report all the hyperparameters that you used. This may differ from the tutorial, but you must use **identical hyperparameters** across different latent spaces.

Latent Space	Loss Function	L2 Distance (↓)*	LPIPS** (↓)	Cosine Distance (↓)
z	MSE	mean (±std)		
w	MSE			
w^+	MSE			

*(↓): Lower value means better

**: https://richzhang.github.io/PerceptualSimilarity/

Hyperparameters:
- iteration: 1,000
- optimizer (lr): Adam (0.01)
- lr_scheduler: NaN
- etc.

a) Show five images generated from each latent space, for a total of 15 images.

b) Quantitatively analyze the achieved results. Which latent space “seems” better overall? Do the metrics agree with your perception? Which seems to be the best metric?

Part 2: Effect on Loss Function to Image Inversion

Instead of using only MSE loss, devise your own loss function for optimization. You can use the LPIPS and Cosine Distance that is already given in the tutorial as follows:

$$\mathcal{L} = \mathcal{L}_{MSE} + w_1\mathcal{L}_{LPIPS} + w_2\mathcal{L}_{cos}$$

a) If you do so, report the weighting factor for each loss function and provide a brief explanation. If you create your own function, also give a short explanation of that function.

b) Report the results in the table below. Note that for a fair comparison, the hyperparameters should remain the same as those used for the MSE experiment.

c) For each latent space, generate five images under the new loss function you design, totaling 15 images.

Latent Space	Loss Function	L2 Distance (↓)	LPIPS (↓)	Cosine Distance (↓)
z	MSE	mean (±std)		
z	Your Function			
w	MSE			
w	Your Function			
w^+	MSE			
w^+	Your Function			

Part 3: Crossover-based Style Transfer (Resolution Split).

- Use the inversion pipeline and metrics (MSE, L2, LPIPS) above. Keep all hyperparameters identical across runs (iterations, optimizer, LR, scheduler). Work in w^+ (18 style vectors, one per layer) so you can swap subsets of layers.
- Choose five pairs of content image I_c and style image I_s from the provided set or your own photos. Invert both to obtain w_c^+ and w_s^+ using your established loss in Part 2.
- Suppose **Split A (content-coarse, style-fine):** layers 1–9 from w_c^+ (resolutions 4–64), layers 10–18 from w_s^+ and **Split B (style-coarse, content-fine):** layers 1–9 from w_s^+, layers 10–18 from w_c^+. This is the "reverse" configuration to study coarse vs. fine attribute transfer.

- **Synthesize crossover images:** Construct $\widetilde{w}_A^+ = [w_c^{(1\to9)} || w_s^{(10\to18)}]$ and $\widetilde{w}_B^+ = [w_s^{(1\to9)} || w_c^{(10\to18)}]$ by concatenating the chosen layer subsets; generate $\tilde{I}_A = G(\widetilde{w}_A^+)$ and $\tilde{I}_B = G(\widetilde{w}_B^+)$. Do not fine-tune after crossover; the point is to isolate layer-wise effects exactly as taught.

a) For each pair (I_c ,I_s) and for each split (A/B), compute L2 distance and LPIPS for each configuration below:

	Configuration	L2 Distance (↓)	LPIPS (↓)
Identity Retention	I_c and $\tilde{I}_A$	mean (±std)	-
	I_c and $\tilde{I}_B$		-
Perceptual/texture change	I_S and $\tilde{I}_A$	-	
	I_S and $\tilde{I}_B$	-	

i. Using the **L2(**I_c , $\tilde{I}_A$) and **L2(**I_c and $\tilde{I}_B$) columns, which split achieves a lower mean (±std) error to the content image across pairs (preserve identity better)? Identify outliers and justify whether the winner is consistent or pair-dependent.

ii. Compare **LPIPS(**I_s, $\tilde{I}_A$) and **LPIPS(**I_s, $\tilde{I}_B$), which split is perceptually closer to the style image on average (transfers style more aggressively)? Quantify the effect using the gap Δ_{style} = **LPIPS(**I_s, $\tilde{I}_A$) - **LPIPS(**I_s, $\tilde{I}_B$) (or the reverse, but be consistent) and interpret the sign and magnitude.

b) Provide a (2x4) panel per pair: [I_c ,I_s, $\tilde{I}_A$, $\tilde{I}_B$] and their zoomed-in crops (hair/skin/eyes) to show coarse versus fine differences, illustrating that earlier layers control coarse attributes and later layers control fine details. Briefly annotate observed attribute transfers.

References:

[1] R. Abdal, Y. Qin, and P. Wonka, “Image2StyleGAN: How to Embed Images Into the StyleGAN Latent Space?,” in 2019 CVPR.

[2] R. Abdal, Y. Qin, and P. Wonka, “Image2StyleGAN++: How to Edit the Embedded Images?,” in 2020 CVPR.

Question 3: Simple Dinosaur Diffusion Model

The goal of this problem is to implement and visualize the DDPM forward (noising) and reverse (denoising) processes on a 2D dataset of dinosaurs.

Part 1: Forward (Noising) Process

- Load the TSV `DatasaurusDozen.tsv`. Standardize to zero mean and unit std. Scatter plot the standardized points (with axes off).
- Choose a linear noise schedule with T=50, $\beta_t \in [10^{-4}, 2\times10^{-2}]$.
- Compute $\alpha_t = 1 - \beta_t$, $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$.

a) Plot the **$\bar{\alpha}$ curve** over timesteps (0,…,T).

b) Forward noising: for each selected timestep t∈{1,6,12,25,50}, sample

$$x_t = \sqrt{\bar{\alpha}_t}\, x_0 + \sqrt{1-\bar{\alpha}_t}\, \varepsilon, \quad \varepsilon \sim \mathcal{N}(0, I),$$

scatter-plot the result, and arrange those plots in a 1×5 panel.

c) Explain how changing (beta_min, beta_max) in the linear noise schedule affects the rate information is destroyed.

Part 2(a): Reverse (Denoising) process *without* a time stamp

- Implement `DenoisingMLP ()` with:
  - **MLP** with position input [x_1 x_2] (2 dimensions), 3 hidden layers, width 64, ReLU, output dimension 2.
  - Identity input embedding (use raw 2D) and **zero** time embedding (no time input).
- Implement `train()` that trains to predict noise ε with MSE:

$$\mathcal{L} = \mathbb{E}_{x_0,t,\epsilon}\left[\|\epsilon - \epsilon_\theta(x_t, t)\|_2^2\right]$$

where

$$x_t = \sqrt{\bar{\alpha}_t}\, x_0 + \sqrt{1-\bar{\alpha}_t}\, \varepsilon$$

Hint:

✓ Clip gradients as in the printout to stabilize early training.
✓ Always plot ᾱ; schedules are where many “mysteries” hide.

Plot smooth test loss vs. epochs

• Implement ancestral **sampler**:

$$\hat{\mu}_t = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\,\epsilon_\theta(x_t, t)\right), \quad x_{t-1} = \hat{\mu}_t + \sigma_t z,\ \sigma_t = \sqrt{1-\alpha_t},\ z \sim \mathcal{N}(0, I)$$

d) Sample 1000 points; plot final scatter.

e) Diagnose limitations when the model **does not** see *t*.

Part 2(b): Reverse (Denoising) process with a time stamp

- Now, concatenate **linear time embedding** $\tilde{t} = (t-1)/(T-1)$, t~Unif$\{1, T\}$ with the position input [x_1 x_2] to predict ϵ_θ.
- Retrain the model with new input. Use the same schedule, optimizer, and training budget as your baseline for a fair comparison.

f) Sample 1000 points, plot final scatter.

g) Compare samples and loss with Part 2(a); does time embedding help? Why?

Part 2(c): Add time & position Fourier features (full conditioning)

- Use **Fourier** features [1] to encode both position (*L*=64) and time (*L*=32) input with fixed random projections **B,** since we know the distribution underlying our data is like an image - a high-frequency signal in a low-dimensional (2D) space.

  The Fourier features encoding is defined as:

  $$\text{FourierEncoding}(\mathbf{x}) = [\cos(2\pi\mathbf{B}\mathbf{x}), \sin(2\pi\mathbf{B}\mathbf{x})]^T$$

  where **B** is a random *L* x *D* Gaussian matrix, *D* is the feature dimension (*D*=2 for position and *D*=1 for time), and *L* is the dimension of the random frequency feature space.

- Retrain the model with new input.

Hint: Keep the Fourier projection matrices **fixed** after initialization.

h) Sample 1000 points, plot final scatter.
i) Discuss the improvements (visual fidelity and loss).
j) Discuss the role of Fourier features in fitting high-frequency functions.

[1] Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains. NeurIPS 2020.