ECE / CS 6501: Geometry of Data **Due Wed 11/6**

Homework 3: Manifold Learning with Autoencoders

Instructions: Submit a single Jupyter notebook (.ipynb) of your work to Canvas by 11:59pm on the due date. All code should be written in Python. **Be sure to show all the work involved in deriving your answers! If you just give a final answer without explanation, you may not receive credit for that question.**

You may discuss the concepts with your classmates, but write up the answers entirely on your own. Do not look at another student's answers, do not copy answers from the internet or other sources, and do not show your answers to anyone. **Remember to cite any sources you use, including any prompts given to AI chatbots.**

1. Download the code `Autoencoder.ipynb`. You will want to copy the code for the model definitions, but do not run the training! (unless you have a GPU and want to play with the models beyond the assignment) Download the pre-trained fully-connected layer models `fcAE<dim>.pth` and convolutional layer models `convAE<dim>.pth`, where `<dim>` is the code (z) dimension: 16, 32, or 128.

2. Plot a 10×10 grid of images from the **test** set. For each model, plot the same grid of these images reconstructed by that model. (Recall reconstructing an image means running it through the encoder followed by the decoder). Qualitatively compare the various models. What difference does convolutional vs. fully-connected layers make? What difference does the code dimension make?

3. Compute a PCA of the **training** images. Plot the same 10×10 images reconstructed from the first k principal component dimensions, where $k = 16, 32, 128$. How does PCA reconstruction compare qualitatively to the autoencoders? What does this tell you about the data manifold?

4. Pick two images from the **test** set that visually seem very different (e.g., from two different classes). Plot a sequence of images that are along the linear path between these two images. Next, for each autoencoder model, plot a sequence of images that are interpolated in the latent code space, followed by application of the decoder. Try all these plots again with two images that look visually similar. Describe what you see comparing all of the different interpolations.

5. For each model, answer the following questions:
   (a) What are the domains and ranges for each layer? (They are all mappings from $\mathbb{R}^a \to \mathbb{R}^b$, so this is asking what is the input dimension a and output dimension b for each layer.)
   (b) By looking only at the weights, is the encoder a submersion? Possible answers are "yes", "no", "can't tell from the weights alone". Explain why.
   (c) Again, looking only at the weights, is the decoder an immersion? Same possible answers as above. Explain why.

6. Write a function to compute the Jacobian matrix of the decoder of a given model. It should take as input the data point (image) at which to compute the Jacobian matrix. For each model, compute the Jacobian of the decoder at a particular encoded data point. Compute the SVD of the Jacobian and report the min and max singular values. Does it seem that the

Jacobian is full rank? What does this say about whether the decoder is an immersion at this point?

7. Pick an image as a base point. For each model, plot (as a grid of images) the tangent vectors corresponding to the latent coordinate axes. Also, include a plot of the tangent vectors corresponding to the principal components up to 128 dimensions. Describe what you are seeing. Do the latent dimensions make sense in terms of changes to the base image?

8. Use the same base image, and again repeat the following for all models and PCA. Check how close translations and rotations are to being represented in the tangent space, as follows: Generate tangent vectors induced by x and y translation and rotation. Compute the angles from these tangent vectors to the model's tangent space.

9. Now repeat the previous rotation and translation experiment, but using the model `convAE32_aug.pth`, which was trained with data augmentation of randomly rotated and translated versions of the images. Did data augmentation help the model learn rotations and translations (according to the tangent space measurements)?