CS231A: Computer Vision, From 3D Reconstruction to Recognition Homework #3

(Spring 2025) Due: **Tuesday, May 20**

1 Overview

This PSET will involve concepts from lectures 8, 10, 11, 12, and 13. It will involve 3D reconstruction, representation learning, as well as supervised and unsupervised monocular depth estimation. Although there are 4 problems so this PSET may look daunting, most of the problems require comparatively little work.

2 Submitting

Please put together a PDF with your answers for each problem, and submit it to the appropriate assignment on Gradescope. We recommend you to add these answers to the latex template files on our website, but you can also create a PDF in any other way you prefer. For the written report, in the case of problems that just involve implementing code you can include only the final output and in some cases a brief description if requested in the problem. There will be an additional coding assignment on Gradescope that has an autograder that is there to help you double check your code. Make sure you use the provided ".py" files to write your Python code. Submit to both the PDF and code assignment, as we will be grading the PDF submissions and using the coding assignment to check your code if needed.

For submitting to the autograder, **just submit p1.py**!

3 Space Carving (45 points)

Dense 3D reconstruction is a difficult problem, as tackling it from the Structure from Motion framework (as seen in the previous problem set) requires dense correspondences. Another solution to dense 3D reconstruction is space carving[1], which takes the idea of volume intersection and iteratively refines the estimated 3D structure. In this problem, you implement significant portions of the space carving framework. In the starter code, you will be modifying the `p1.py` file inside the `p1` directory.

(a) The first step in space carving is to generate the initial voxel grid that we will carve into. Complete the function `form_initial_voxels()`. **[5 points]**

(b) Now, the key step is to implement the carving for one camera. To carve, we need the camera frame and the silhouette associated with that camera. Then, we carve the silhouette from our voxel grid. Implement this carving process in `carve()`. **Briefly describe how you went about implementing this, and include the resulting image in your report.[10 points code + 5 points written]**

[1] http://www.cs.toronto.edu/~kyros/pubs/00.ijcv.carve.pdf

(c) The last step in the pipeline is to carve out multiple views. Submit the final output after all carvings have been completed, using `num_voxels` $\approx 6,000,000$. **What are some flaws with the reconstruction? [5 points]**

(d) Notice that the reconstruction is not really that exceptional. This is because a lot of space is wasted when we set the initial bounds of where we carve. Currently, we initialize the bounds of the voxel grid to be the locations of the cameras. However, we can do better than this by completing a quick carve on a much lower resolution voxel grid (we use `num_voxels` = 4000) to estimate how big the object is and retrieve tighter bounds. Complete the method `get_voxel_bounds()` and change the variable `estimate_better_bounds` in the `main()` function to `True`. Submit your new carving, which should be more detailed, and your code. **Are there any remaining issues with the construction? Why do you think they are still there? [8 points code + 2 points written]**

(e) Finally, let's have a fun experiment. Notice that in the first three steps, we used perfect silhouettes to carve our object. Look at the `estimate_silhouette()` function implemented and its output. Notice that this simple method does not produce a really accurate silhouette. However, when we run space carving on it, the result still looks decent!

(i) Why is this the case? **[2 points]**

(ii) What happens if you reduce the number of views? **[1 points]**

(iii) What if the estimated silhouettes weren't conservative, meaning that one or a few views had parts of the object missing? **[2 points]**

4 Representation Learning (20 points)

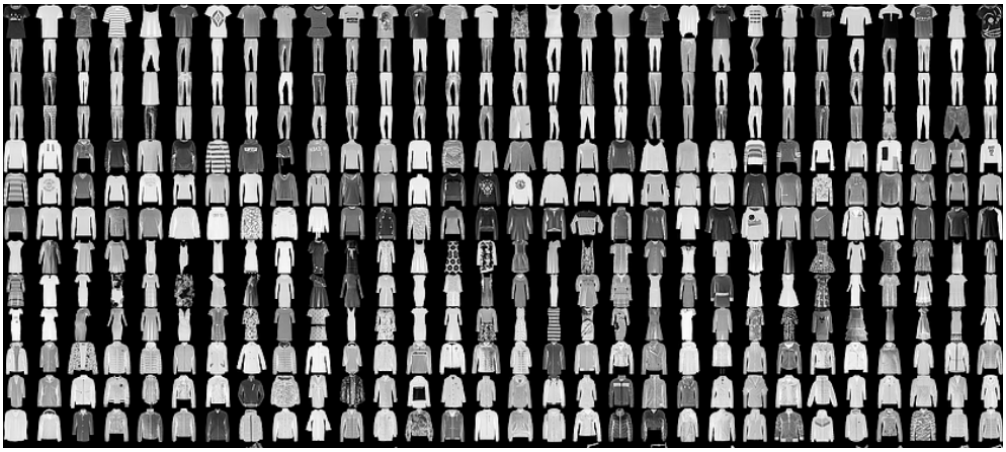

Figure 1: The Fashion MNIST dataset

In this problem, you'll implement a method for image representation learning for clasifying clothing items from the Fashion MNIST dataset.

Unlike the previous problems, for this one you will have to work with Google Colaboratory. In Google Drive, follow these steps to make sure you have the ability to work on this problem:

a. Click the wheel in the top right corner and select Settings.

b. Click on the Manage Apps tab.

c. At the top, select Connect more apps which should bring up a GSuite Marketplace window.

d. Search for Colab then click Add.

Now, upload "p2/RepresentationLearning.ipynb" and the contents inside 'p3/code' to a location of your choosing on Drive. Then, navigate to this folder and open the file "RepresentationLearning.ipynb" with Colaboratory. The rest of the instructions are provided in that document. Note that there is no autograder for this problem, as you should be able to confirm whether your implementation works from the images and plots. Include the following in your writeup:

a. Once you finish the section "Fashion MNIST Data Preparation", include the 3 by 3 grid visualization of the Fashion MNIST data [**3 point**].

b. Once you finish the section "Training for Fashion MNIST Class Prediction", include the two graphs of training progress over 10 epochs, as well as the test errors [**5 points**].

c. Once you finish the section "Representation Learning via Rotation Classification", include the 3 by 3 grid visualization and training plot, as well as the test errors [**7 points**].

d. Once you finish the section "Fine-Tuning for Fashion MNIST classification", include all 3 sets of graphs from this section, as well as the test errors. **Why do you think learning to un-rotate images works as a method of representation learning** [**5 points**]?

Fun fact: the method you just implemented is from the ICLR2018 paper "Unsupervised Representation Learning by Predicting Image Rotations".

5 Supervised Monocular Depth Estimation (15 points)

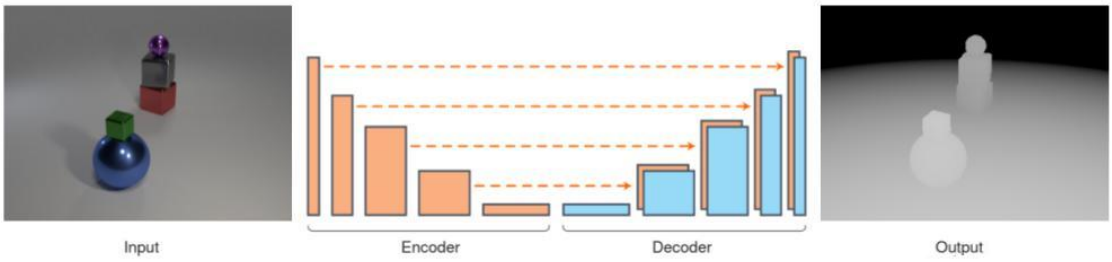

Figure 2: The task this problem will involve

Now that you've had some experience with representation learning on a small dataset, we'll move on to the more complex task of training a larger model on the task of monocular depth estimation.

We will be implementing the approach taken in "High Quality Monocular Depth Estimation via Transfer Learning", which showed that taking a big model pre-trained on classifying objects from the ImageNet dataset helps a lot with training that model to perform monocular depth estimation on the NYU Depth v2 dataset. We created a smaller and simpler dataset that is a variation on the CLEVR dataset for this problem that we call CLEVR-D, so we'll actually not be using the pre-trained features, but otherwise the approach is the same.

As with the previous problem, you'll be working with Colaboratory, so once again begin by uploading the contents of 'p3/code' to a location of your choosing on Google Drive. Then, open the file "MonocularDepthEstimation.ipynb" with Colaboratory. The rest of the instructions are provided in that document. Note that there is no autograder for this problem, as you should be able to confirm whether your implementation works from the images and plots. Include the following in your writeup:

a. Once you finish the section "Checking out the data", include the grid visualization of the CLEVR-D data in this report **[5 points]**.

b. Once you finish the section "Training the model", include a screenshot of your train and test losses from Tensorboard as well as the final outputs of the network **[10 points]**.

Extra Credit As described at the bottom of the Colab notebook, you may optionally try to do representation learning by using an autoencoder, and see if that helps with training for monocular depth prediction. If you decide to do this, include a several sentence summary of how you went about it as well as sentence or two about the results, plus plots from Tensorboard and visualizations of the network in action **[20 points]**.

6 Unsupervised Monocular Depth Estimation (20 points)

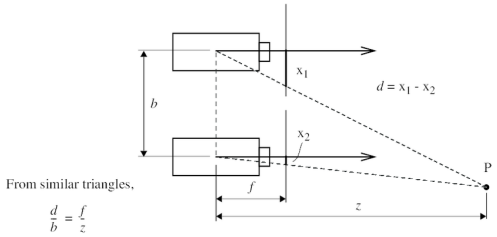

Figure 3

In this problem, we will take a step further to train monocular depth estimation networks without ground-truth training data. Although neural networks perform best with large-scale training datasets, it is often challenging to collect ground-truth data for every problem domain. For example, the Microsoft Kinect, one of the most popular depth cameras, uses an infrared sensor that does not work outdoors, making it particularly difficult to train monocular depth estimation networks for outdoor scenes.

Instead, we will leverage the stereo computer vision concepts we studied in this course to train monocular depth estimation networks without ground-truth data. Specifically, we will train a network to predict disparity. As shown in Figure 3, disparity (d) is inversely proportional to depth (z), which still serves our purpose. Given a pair of rectified left and right images as input, we can synthesize the right image by shifting the left image rightward by the disparity, and vice versa. We will use this property to synthesize both the left and right images and enforce similarity to the original left and right images.

Figure 4 summarizes how unsupervised monocular depth estimation works. This method is based on the paper *"Unsupervised Monocular Depth Estimation with Left-Right Consistency"*. The network takes the left view of the stereo image, img_l, as input and outputs two disparity maps: $disp_l$ (the disparity map of the left view that maps the right image to the left) and $disp_r$ (the disparity map of the right view that maps the left image to the right). Although the network only takes the left image as input, it is trained to predict the disparity for both views. This design allows monocular depth prediction (i.e., without requiring stereo images as input) and enforces cycle consistency between the left and right views.

Assuming the input images are rectified, we can generate the left and right images from the predicted disparities. Specifically, using the left disparity $disp_l$, we synthesize the left image and disparity as follows: $img'_l = \text{generate\_image\_left}(img_r, disp_l)$ and $disp'_l = \text{generate\_image\_left}(disp_r, disp_l)$. Similarly, using the right disparity $disp_r$, we synthesize the right image and disparity as follows: $img'_r = \text{generate\_image\_right}(img_l, disp_r)$ and

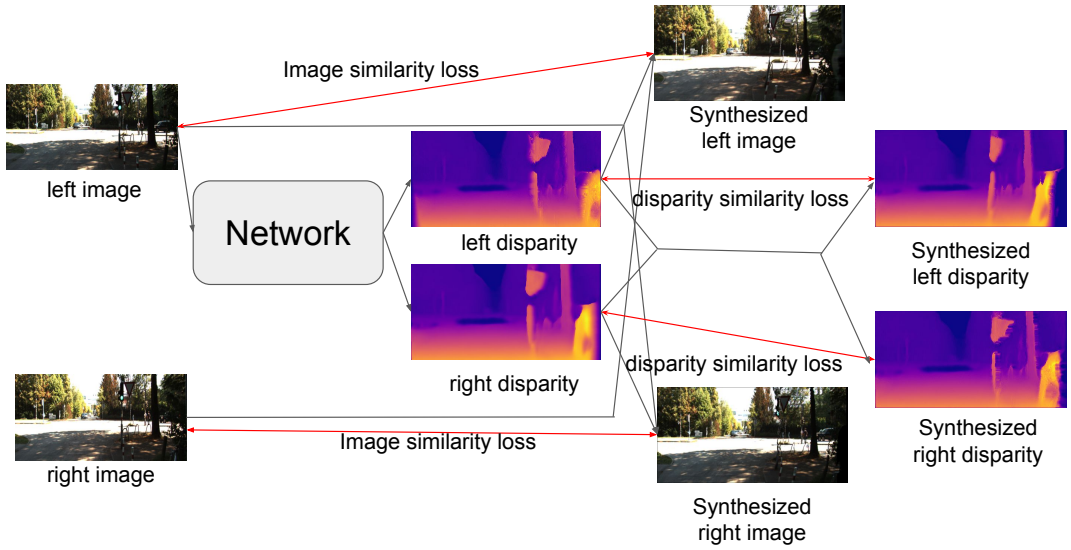

Figure 4

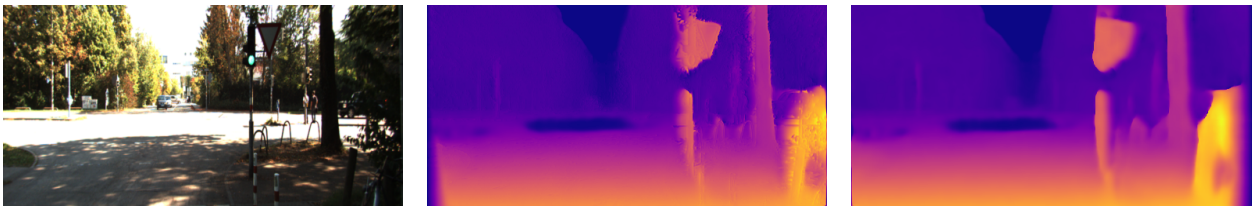

Figure 5: Input and output (left/right disparity) of trained monocular depth estimation networks.

$disp'_r = \text{generate\_image\_right}(disp_l, disp_r)$. You will implement the functions `generate_image_left` and `generate_image_right` in this problem set.

To predict a reasonable disparity that can shift the left image to the right and vice versa, we compare the synthesized images with the real images:

$$L_{img} = \text{compare}_i(img'_l, img_l) + \text{compare}_i(img'_r, img_r)$$

For completeness, compare_i uses L1 and SSIM.

To enforce cycle consistency between the left and right disparities, we compare the synthesized disparities with the predicted disparities:

$$L_{disp} = \text{compare}_d(disp'_l, disp_l) + \text{compare}_d(disp'_r, disp_r)$$

For completeness, compare_d uses L1.

Please fill in the code in `p4/problems.py` as outlined below. To run the code, install PyTorch and torchvision, and then test it locally with `python problems.py`. Alternatively, you can use the Jupyter notebook `UnsupervisedMonocularDepthEstimation.ipynb` on Google Colab, as you did in the previous two problems. Note that we did not require you to train the model yourself from scratch in this problem, as it is too computationally intensive.

a. Before we begin, implement a data augmentation function for stereo images that randomly flips the given image horizontally. Data augmentation plays a crucial role in improving the

generalization of neural networks. One of the most common augmentations for 2D images is random horizontal flipping. However, because we use pairs of rectified stereo images, extra care is needed to maintain the stereo relationship after flipping. Please implement this function and include the images generated by this code in your report (input images are not required). **[5 points]**

b. Implement the function `bilinear_sampler`, which shifts the image horizontally according to the disparity. The key idea of unsupervised monocular depth estimation is that we can generate the left image from the right (and vice versa) by horizontally sampling the rectified images using the disparity. Please implement this function and include the generated images in your report (input images are not required). **[5 points]**

c. Implement the functions `generate_image_right` and `generate_image_left`, which generate the right view from the left image using the disparity and vice versa. These will be simple one-liners that apply `bilinear_sampler`. Please include the generated images in your report (input images are not required). **[5 points]**

d. In Figure 5, we visualize the outputs of the networks trained with the losses you implemented. You may notice some boundary artifacts on the left side of the left disparity and the right side of the right disparity. Briefly explain why these artifacts may exist. **[5 points]**