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

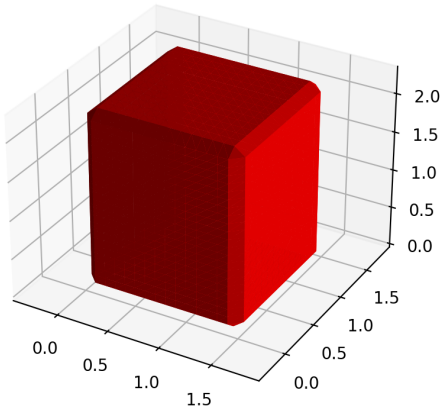
(Winter 2021) Due: **Friday, Februrary 25**

Vasu G. Patel: vgpatel1@stanford.edu
*worked in collaboration with Raghav Khandelwal

1 Space Carving (40 points)

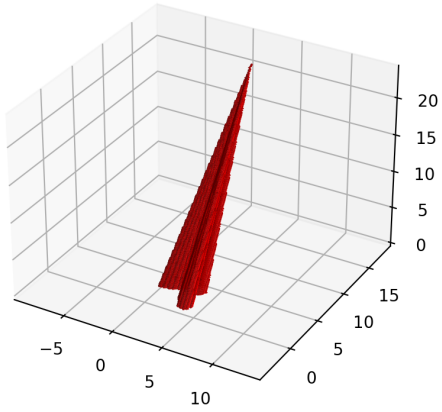
(a) The first step in space carving is to generate the initial voxel grid that we will carve into. Complete the function `form_initial_voxels()`. Submit an image of the generated voxel grid and a copy of your code. **[5 points]**

Answer 1(a):
Initial voxel grid generated is as follows:



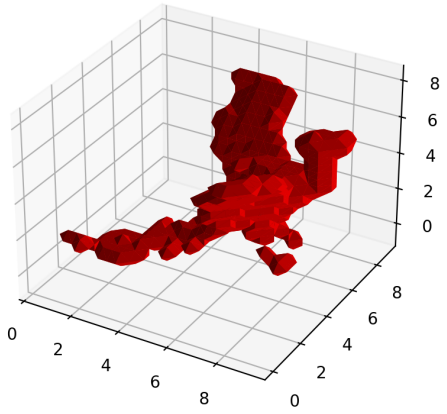
(b) Now, the key step is to implement the carving for one camera. To carve, we need the camera frame and the silhouette associated with that camera. Then, we carve the silhouette from our voxel grid. Implement this carving process in `carve()`. Submit your code and a picture of what it looks like after one iteration of the carving. **[15 points]**

Answer 1(b):
Carving from one camera is obtained as following:



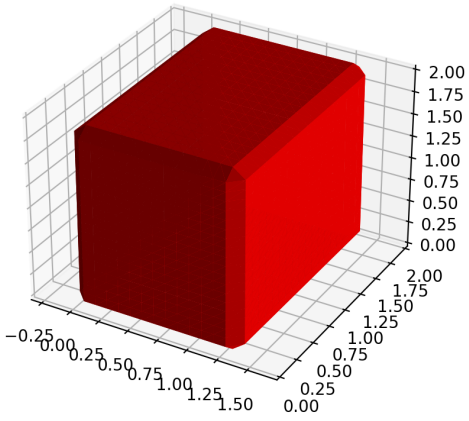
(c) The last step in the pipeline is to carve out multiple views. Submit the final output after all carvings have been completed, using `num_voxels` $\approx 6,000,000$. **[5 points]**

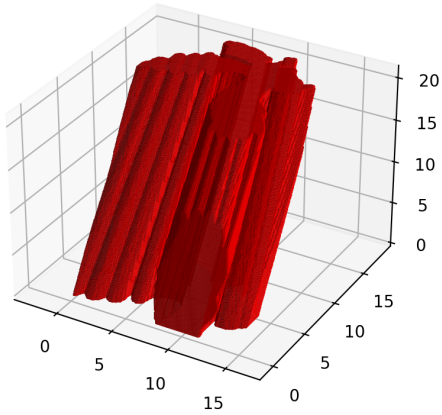
Answer 1(c):
Final output after all carving is done on 6,000,000 voxels is:

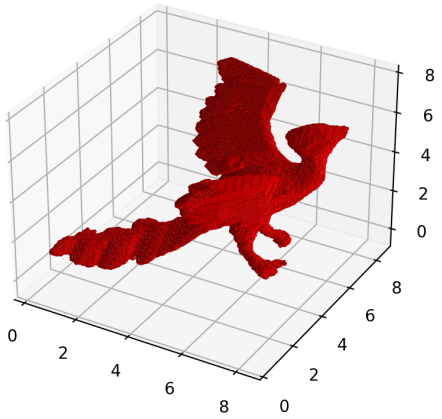


(d) Notice that the reconstruction is not really that exceptional. This is because a lot of space is wasted when we set the initial bounds of where we carve. Currently, we initialize the bounds of the voxel grid to be the locations of the cameras. However, we can do better than this by completing a quick 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. **[10 points]**

Answer 1(d):
After implementing `estimate_better_bounds`, new carved voxels are obtained as follows:







(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]**

Answer 1(e)_i:
The function use color specific heuristic to generate the silhouette of an object which compares the R channel of the image with G and B channels. Given the object is in red colors it satisfy (imageR¿imageB) or (/) (image R¿ image g) . The particular constraint depicts the general shape of the object pretty well

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

Answer 1(e)_ii:
If we reduce the number of views, we observe corresponding compromise in the final carved object. Since the consistency rule of space carving states that each voxel present in the final 3D reconstruction must be present in all the image silhoutte. Hence, if we reduce the number of views, final carved object will be coarser.

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

Answer 1(e)_iii:
If one or few views had parts of the object missing, then following the consistency rule of space carving, we won't be able to generate full 3D reconstruction of the object and there will be a compromise in the details of the object.

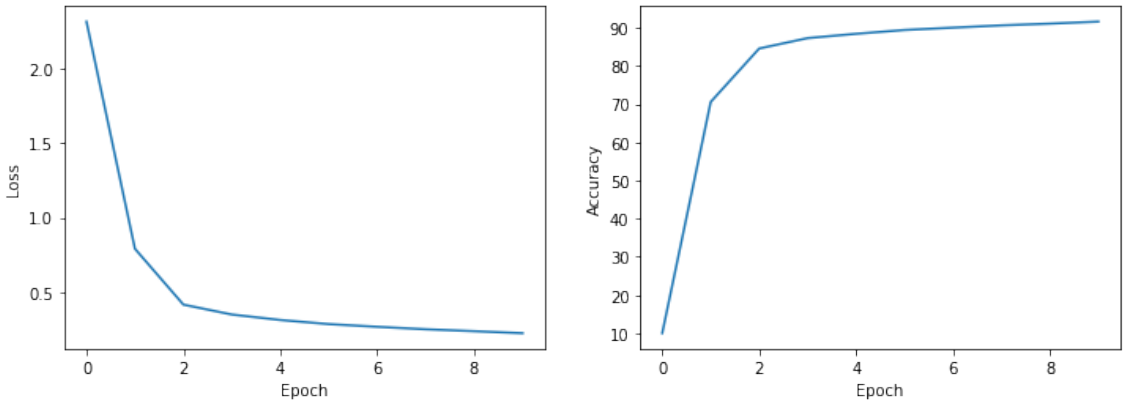
2 Representation Learning (10 points)

a. Once you finish the section "Fashion MNIST Data Preparation", include the 3 by 3 grid visualization of the Fashion MNIST data **[1 point]**.

Answer 2(a):

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 **[4 points]**.

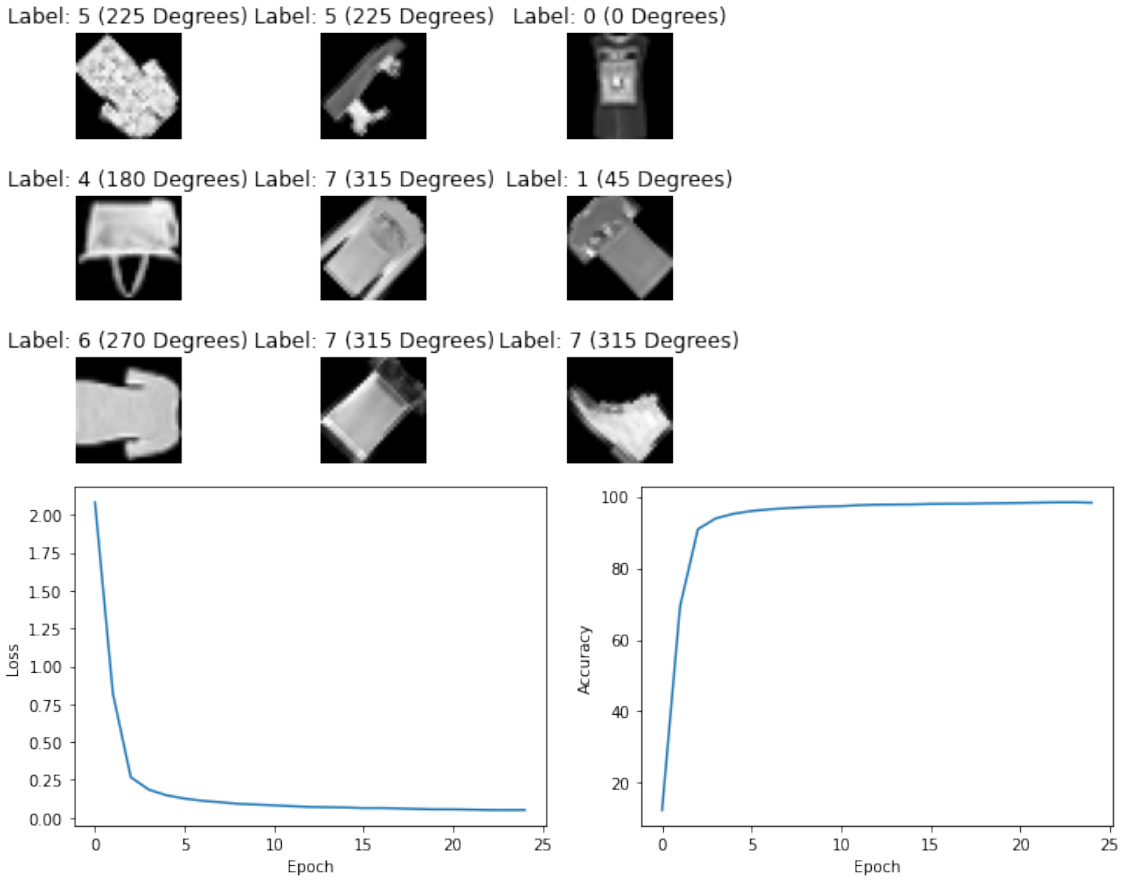
Answer 2(b):
Training loss and accuracy graphs for 10 epochs is as following:



Testing error reported after 10 epochs is 90.01%

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 error [**4 points**].

Answer 2(c):

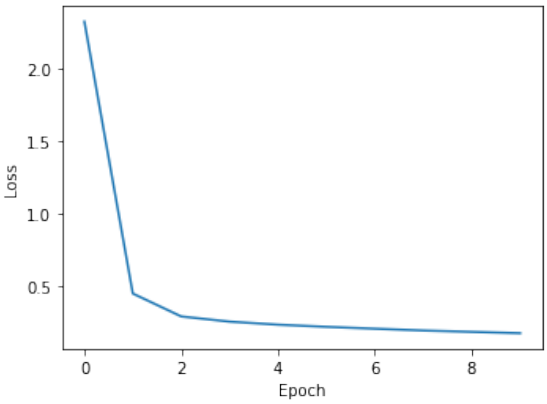


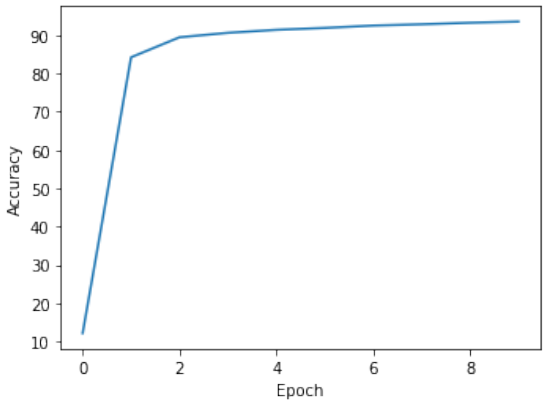
Testing accuracy for representation learning via rotation classification is 98.28%.

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 [**1 point**].

Answer 2(d):

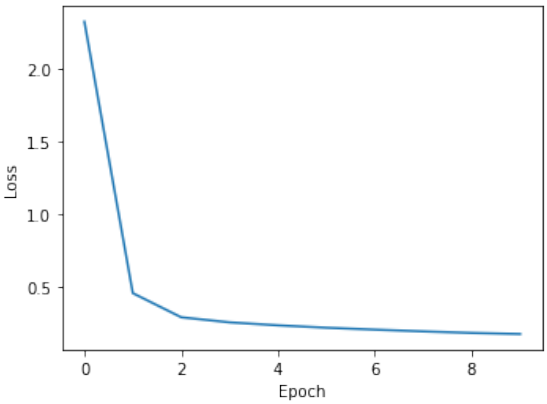
(A): Training and testing accuracy for original data-set with 10 epochs is:

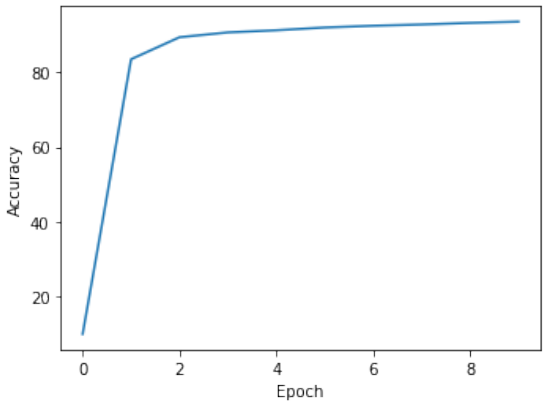




Testing accuracy is 91.31%.

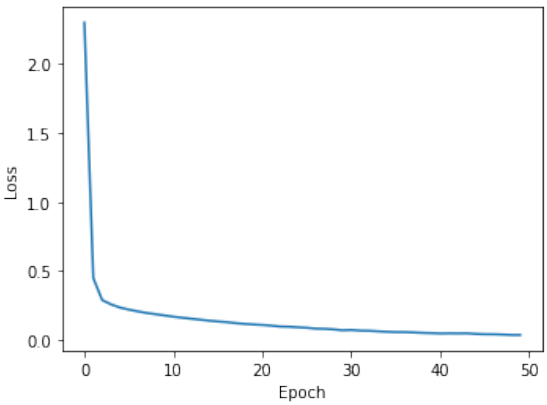
(B): Training and testing accuracy for small data-set with 10 epochs is:

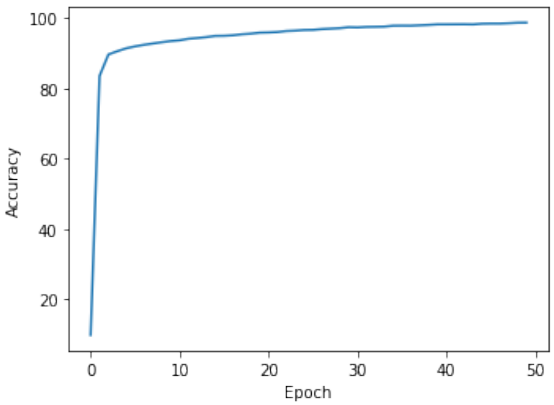




Testing accuracy is 91.37%.

(C): Training and testing accuracy for original data-set with 50 epochs is:



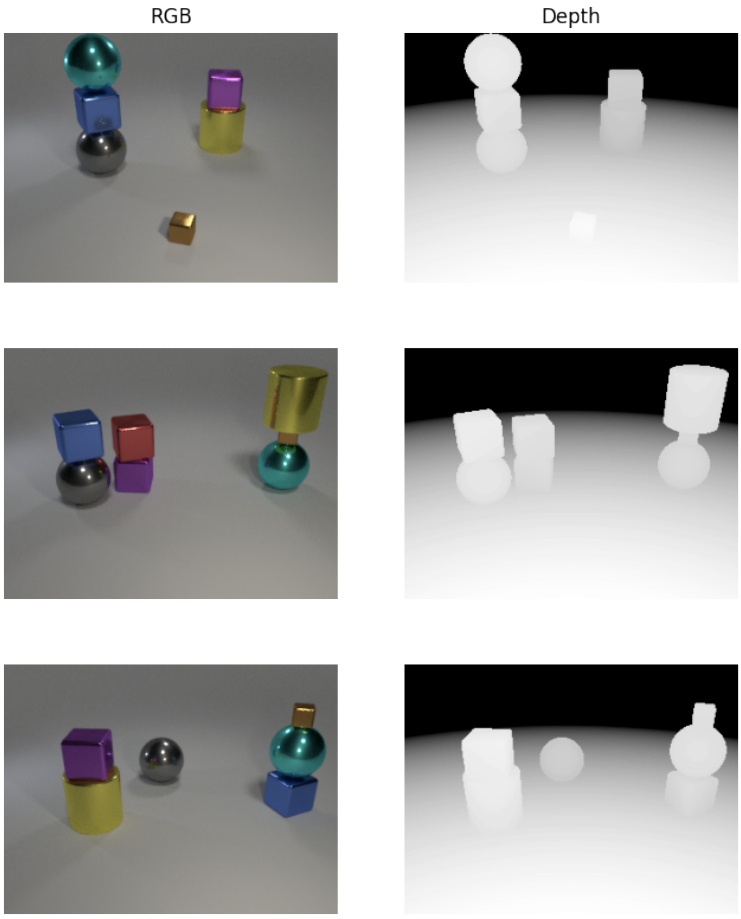


Testing accuracy is 90.69%.

3 Monocular Depth Estimation (10 points)

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

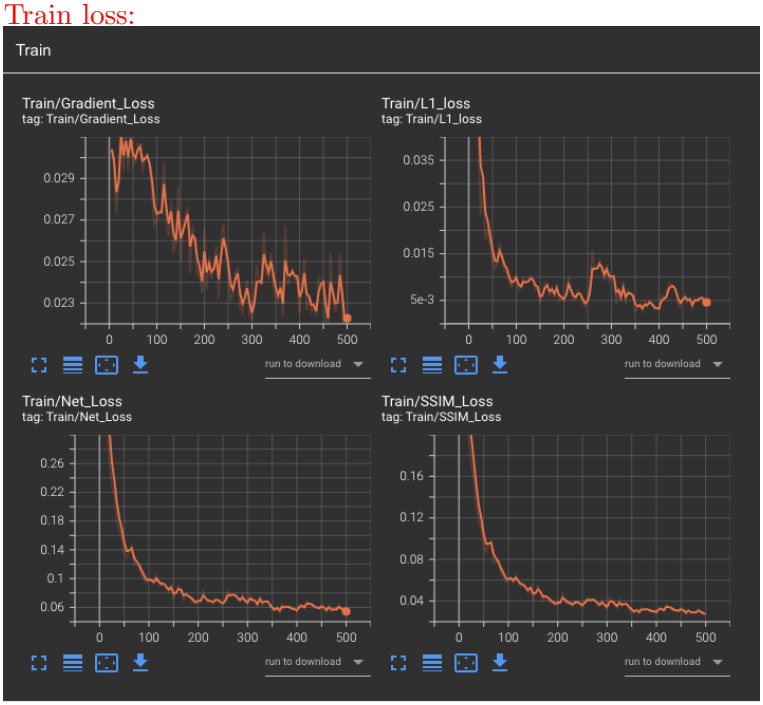
Answer 3(a):



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 [**5 points**].

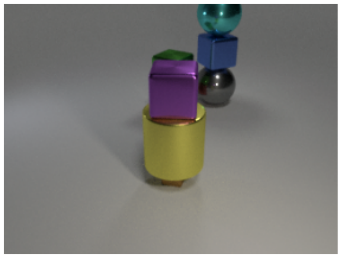
Answer 3(b):

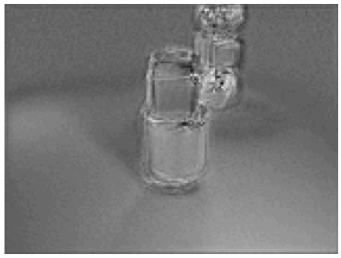
Train loss:

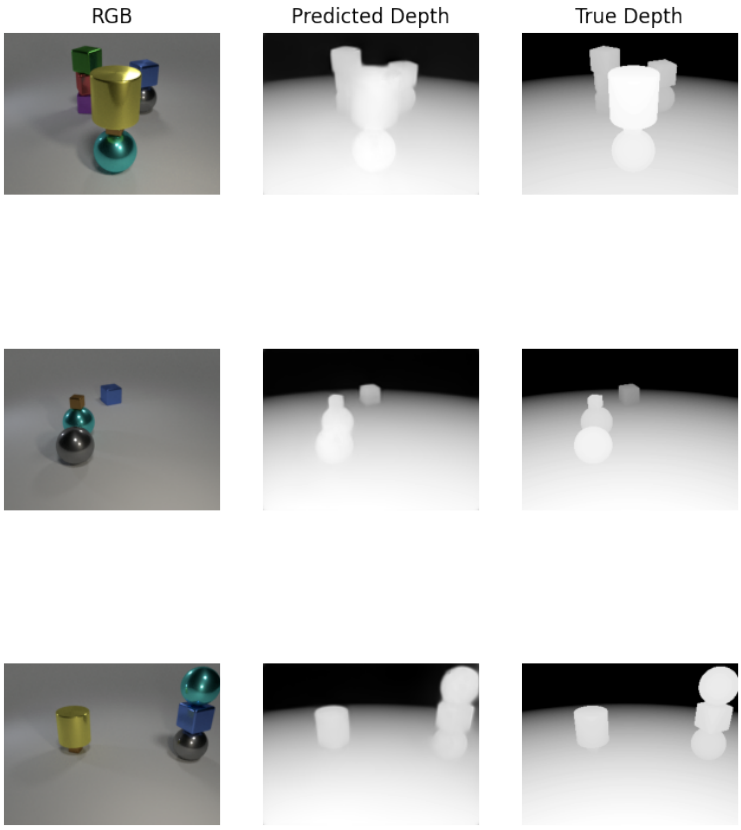


Test loss:

Network output:





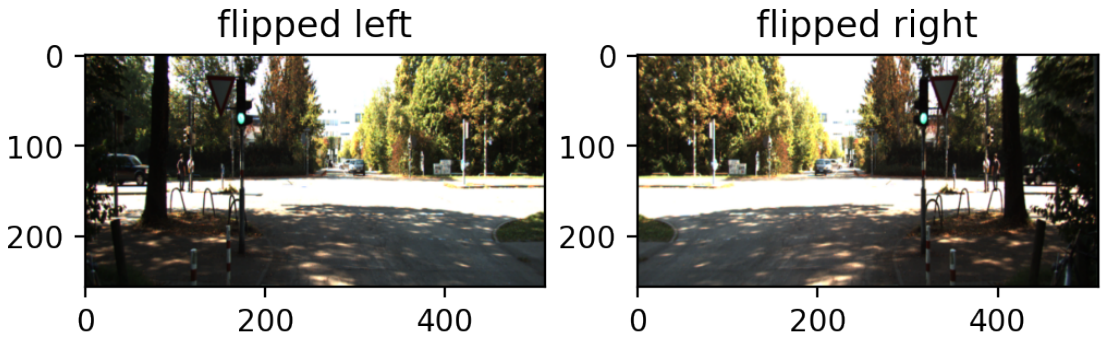


Extra Credit As described at the bottom of the Colab notebook, you may optionally try to do representation learning by using an autoencoder, and see if that helps with training for monocular depth prediction. If you decide to do this, include a several sentence summary of how you went about it as well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

4 Unsupervised monocular depth estimation (20 points)

a. Before we get started, we would like you to implement a data augmentation function for stereo images that randomly flips the given image horizontally. In neural networks, data augmentation takes a crucial role in better generalization of the problem. One of the most common data augmentation when using 2D images as input is to randomly flip the image horizontally. One interesting difference in our problem setup is that we take a pair of rectified stereo images as input. In order to maintain the stereo relationship after the horizontal flip, it requires a special attention. Please fill in the code to implement the data augmentation function. In your report include the images generated by this part of the code (no need to include the input images). [**5 points**]

Answer 4(a):

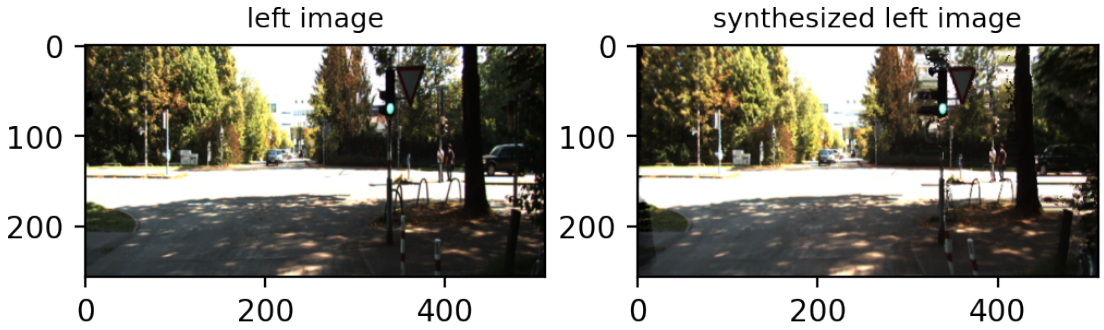


b. Implement a function bilinear_sampler which shifts the given horizontally given the disparity. The core idea of unsupervised monocular depth estimation is that we can generate left image from right and vice versa by sampling rectified images horizontally using the disparity. We will ask you to implement a function that simply samples image with horizontal displacement as given by the input disparity. In your report include the images generated by this part of the code (no need to include the input images). [**5 points**]

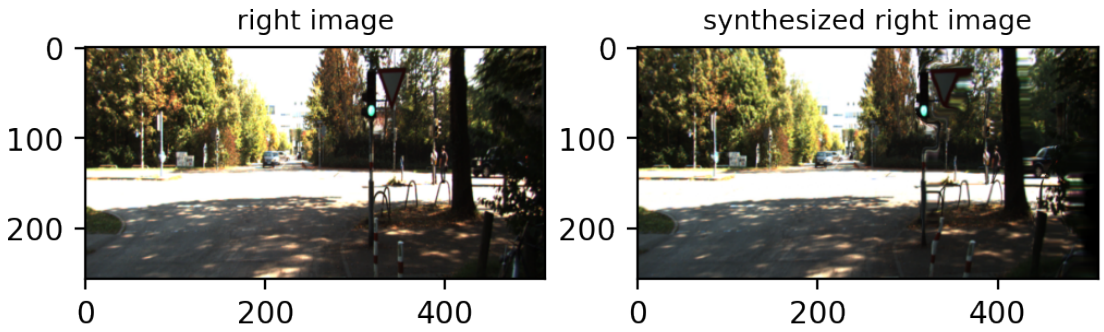
Answer 4(b): Applying bilinear samping on the disparity and originial image is:

c. Implement functions generate_image_right and generate_image_left which generates right view of the image from left image using the disparity and vice versa. This will be a simple one-liner that applies bilinear_sampler. In your report include the images generated by this part of the code (no need to include the input images). [**5 points**)]

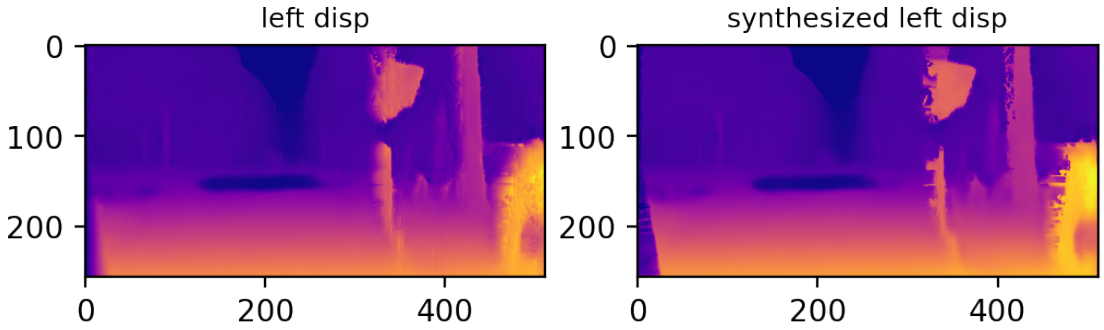
Answer 4(c):
Regenerated Left Image



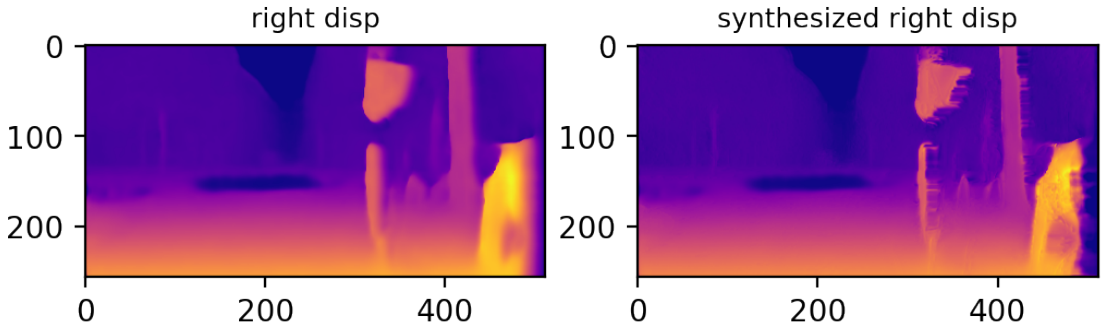
Regenerated Right Image



Synthesized left disparity image:



Synthesized right disparity image:



d. In Figure **??**, we visualize output of the networks trained with the losses you have implemented. You may notice that there are some boundary artifacts on the left side of the left disparity and right side of the right disparity. Briefly explain why. [**5 points**]

Answer 4(d): Some of the artefacts found in the left side of left synthesized disparity image and right side of right synthesized disparity image is due to occlusion. From the theory of baseline tradeof we know that camera separated by a given baseline distance tends to have occlusion and cannot capture some parts of object as seen from other camera. Hence, due to this occlusion condition, we see artefacts in the generated disparity images.

5 Tracking with Optical and Scene flow (20 points)

a. Detect and track the $N = 200$ best point features using Luca-Kanade point feature optical flow algorithm on the consecutive images, initialized with Tomasi-Shi corner features. Use the opencv implementation (look at this tutorial for more information: `https://docs.opencv.org/3.4/d4/dee/tutorial_optical_flow.html`). For the termination criteria use the number of iterations and the computed pixel residual from *Bouget*, 2001. Use only 2 pyramid

levels and use minimum eigen values as an error measure. What is the lowest quality level of the features detected in the first frame? Provide a visualization of the flow for block sizes 15×15 and 75×75. Explain the differences observed. Is it always better a larger or a smaller value? What is the trade-off? **[5 points]**

Answer 5(a):
For 75x75 window lowest quality level for frame 1 is as follows:
maX: [0.30919123]
min: [0.00209253]

For 15x15 window lowest quality level for frame 1 is as follows:
maX: [2.5227904]
min: [0.01361994]

Image for 75×75:

Image for 15×15:

For small window size of 15x15 we observe accurate reconstruction but it suffers from noisy reconstruction, however for larger window size of 75x75 suffers less noise and also has less accuracy. Dpending on the application, a good trade off for the size of the window is to choose a window size somewhere in between largest and smallest size, where we can achieve desired level of accuracy and noise.

b. Use the registered depth maps and the camera intrinsics provided in the starter code to compute the sparse scene flow (i.e. 3D positions of the tracked features between frames). (Note that depth maps contain NaN values. Features that have NaN depth value in any of the frames should be excluded in the result.). **[5 points]**

Answer 5(b):
I have used following approach for this question:
Input: From 5(a) we have pts (a list of frames containing (200, 2) points)
Step 1: I looped over each frame and within each frame, I found the depth for each (x, y) points from depth_all. If depth was `nan`, I would remove it from list of 2D points
Step 2: Found the 3D point using K_inverse and 2D point for all points not having a `nan` depth values.

c. Now let's switch to another scenario of a person opening a book. We have the same format of data as in part(a); the ten RGB frames and the corresponding depth files are available in the `book` folder. Run the algorithm you implemented for part(a) in this new setting and provide a visualization of the flow for block sizes 75×75. How does the result look qualitatively, compared to the visualization from part(a)? How many features are we tracking in this new scenario and how does it compare to part(a)? Give a brief explanation why it is the case.[**5 points**]

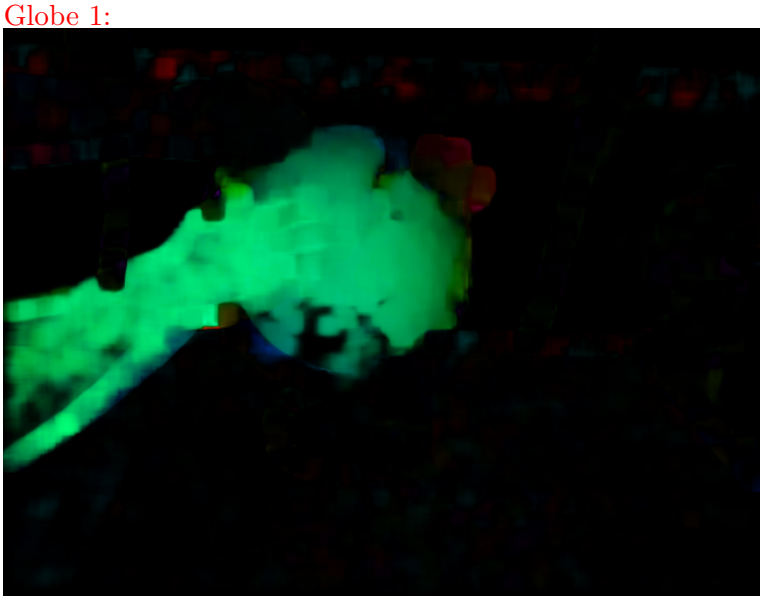
Answer 5(c):
For opening book, total 192 features are being tracked.
Opening book image's optical flow is bit less accurate/sparse as compared to one obtained for globe. Reason for noisy optical flow in the case of opening book is less texture of book as compared to the texture of globe. Since globe has high texture, feature points can be tracked better than the book image where the moving pixels are mainly of same texture (red color).

d. Next, we will be comparing dense optical flow tracking results from two different methods: Gunnar Farnebeck's algorithm, and FlowNet 2.0. First, run the provided p5.py script to get the dense optical flow for two frames from the set used in part b, and two frames from the Flying Chairs dataset. The resulting dense optical flow images should be saved to a file titled farnebeck-(imagename).png; include this in your report. Describe any artifacts you find

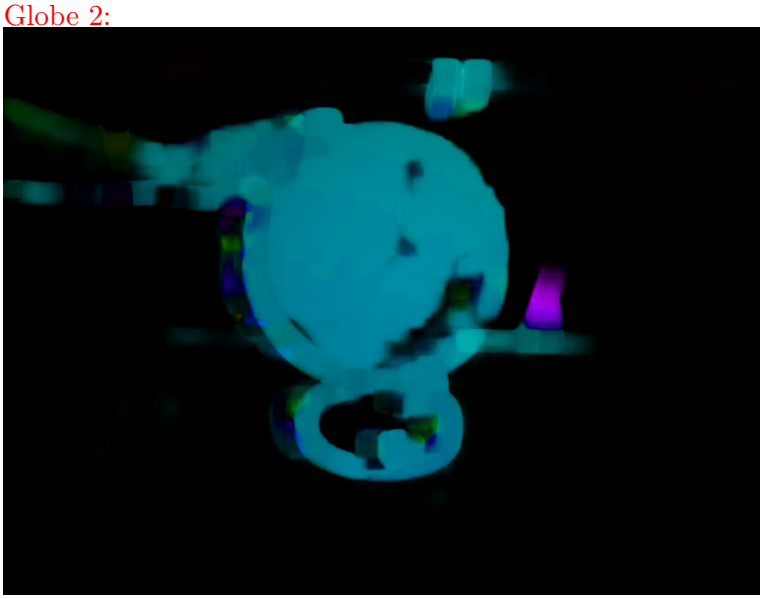
between farnebeck-globe.png and farnebeck-globe2.png. Does one type of image appear have a better optical flow result? What differences between the image pairs might have caused one optical flow result to appear clearer than the other?[**2 points**]

Answer 5(d):
Artefact observed in globe 2 is more accurate than globe 1: we clearly observe a globe and hand in farneback 2 (globe 2) than in franeback 1 (globe 1).
We observe better optical flow for globe 2 as compared to globe 1, because globe 2 is observed to have higher motion (high translation motion) as compare to the motion of globe 1 (rotating about its axis). Since optical flows are produced better for more displacement/motion.

Globe 1:



Globe 2:

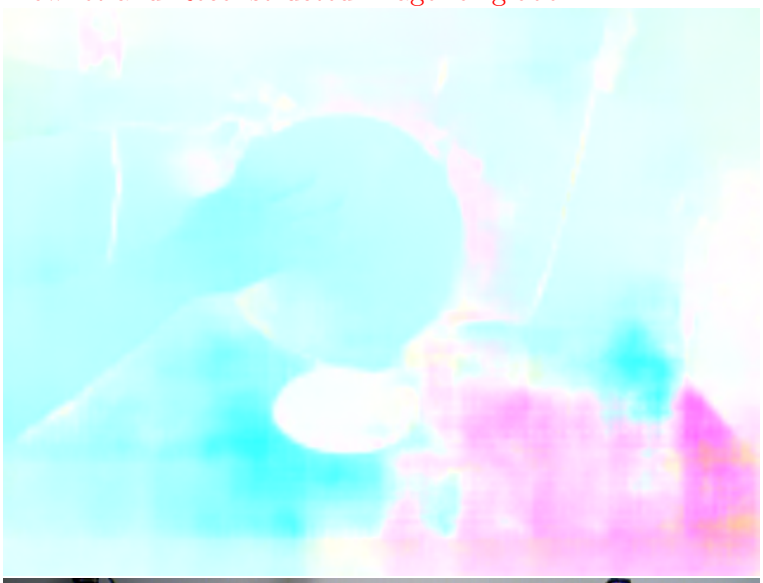


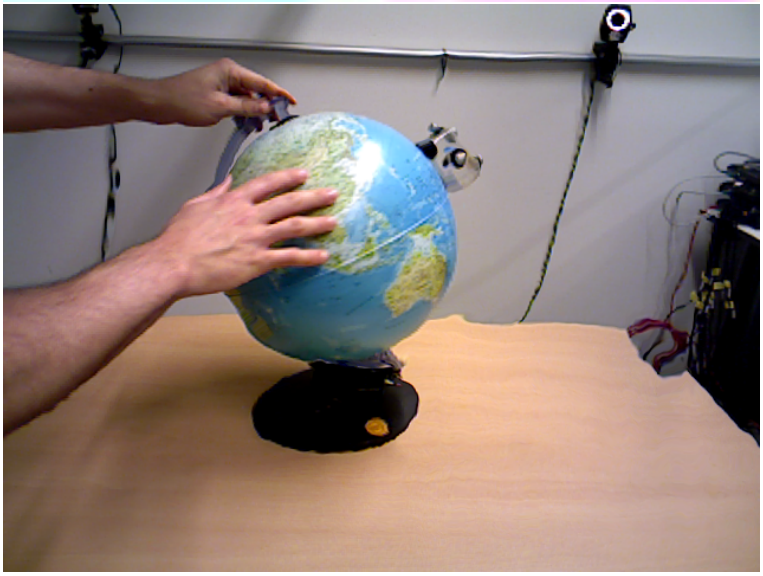
e. Next, go to Google Colab and upload the included `flownet.ipynb` notebook. Once loading the notebook, follow the included instructions to start a GPU runtime, and upload the respective images from the `p5_data` folder used in the last step (`p5_data/globe2/rgb01.png`, `p5_data/globe2/rgb02.png`, `p5_data/globe1/rgb04.png`, `p5_data/globe1/rgb06.png`, `p5_data/chairs/frame_chairs_1.png`,

`p5_data/chairs/frame_chairs_2.png`) to the notebook files tab. Run the cells in the notebook to produce the optical flow between the pairs of images, and include these flow images (and the reconstructed images) in your report. Note any qualitative differences you see between the optical flow results for the different pairs of images. Did one of the optical flow results and reconstructions from a specific pair of the images look less noisier than the others? Describe why you think this might be the case, and if there are any ways in which the performance on the other pairs of images could be improved with the FlowNet model. **[3 points]**

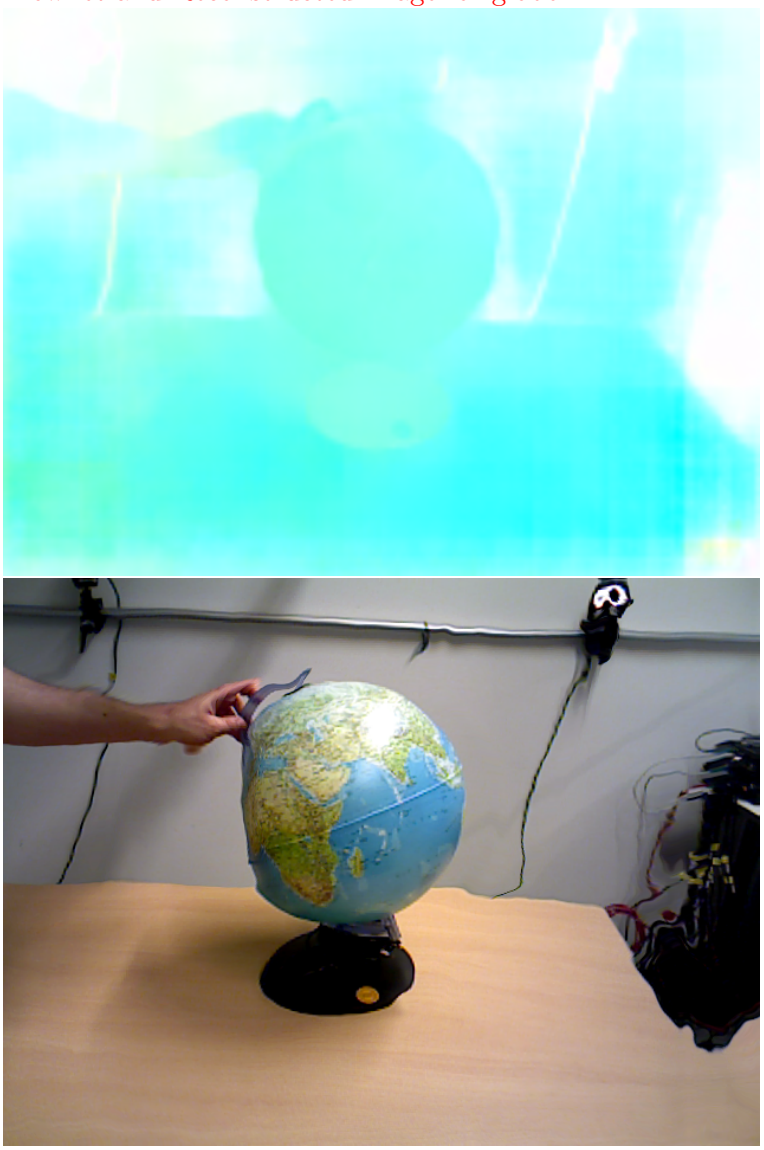
Answer 5(e):
Below are the flownet and reconstructed images of globe and floating chair respectively. We observe globe 1 reconstructed image has the least noise as compared to globe 2 and floating chair. Reason is floating chair represents a motion field which is practically not possible. Hence the model trains on the dataset which does not represent the distribution of actual dataset on which the optical flow will be tested.
We can improve the performance of the FlowNet model using the train dataset which comes from the similar distribution as test dataset, doing so will ensure high model performance.

Flownet and Reconstructed image for globe 1:





Flownet and Reconstructed image for globe 2:



Flownet and Reconstructed floating chair image: