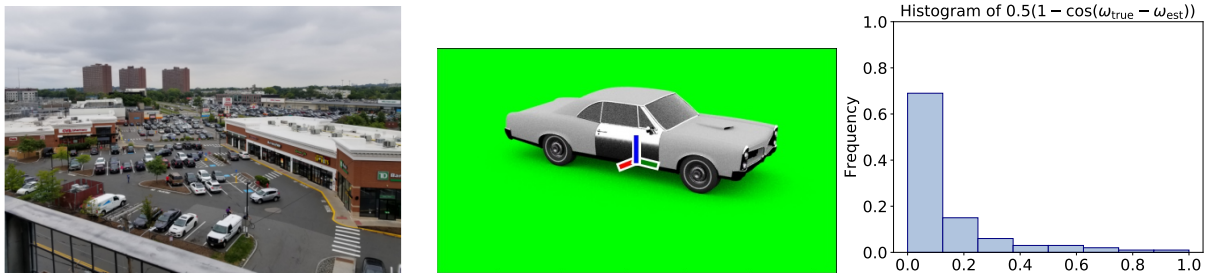
[Overview] Pose-estimation of 3D objects from a single image is often a key ingredient for semantic scene analysis. Yet, in many real-world scenarios, such as the one shown Fig. 1(a), pose estimation remains a challenge. In this project you will design a pose-regression neural network architecture, train it on synthetic data of the type shown in Fig. 1(b) and evaluate your method quantitatively, producing metrics such as the one in Fig. 1(c).



(a) (b) (c)

Figure 1: (a) Estimating 3D pose of objects (even of known classes) can be challenging under adverse imaging conditions such as occlusions and low resolution. (b) We could consider tackling this problem in stages, by first learning to estimate pose of objects in isolation and using synthetic data. For example, the Mitsuba [4] model of a Pontiac GTO 67 comes from [1]. (c) To quantitatively evaluate performance (accuracy) of our algorithm we would employ metrics that are specific to pose estimation, such as the one from [3] computed for a hypothetical validation set and visualized as a histogram with 8 bins.

[Starter code] See the course page: `https://courses.cs.washington.edu/courses/csep576/21au/`

Set up the data loaders and visualize a mini-batch [10pts] Fill in `setup_xforms_and_data_loaders()` following comments in the starter code. You can test your implementation using the corresponding unit test function named `ut_setup_xforms_and_data_loaders()` which is called from `run_all_unit_tests()`. When you've implemented the data loaders correctly, the corresponding `ut_` function will return `True`. Next fill in the details of `load_batch_of_data()` and verify that the corresponding unit test returns `True`.

Insert a synthetic object into the scene [20pts] Before performing quantitative evaluation, one may want to check visually that our predicted pose looks good. For applications like simultaneous localization and mapping (SLAM) one approach is to insert a synthetic pattern (e.g., camera icon with the correct pose) into the scene. In our case we'll overlay a three-axis pattern viewed from the direction estimated by our network on top of its input image, as in Fig. 1(b). You will find step-by-step instructions in the starter code `geometry.py`.

Define an error metric for angular errors [10pts] The starter code `pose_regress.py` contains hints on several loss functions you could try first; an example of an error metric is defined in `train_util.py` and shown in Fig. 1. Complete the implementation to turn the hints into one or two lines of code.

Complete the training code and run evaluations [30pts] Following the concepts you learned working on Project 3, implement SGD training via backpropagation. Evaluate this (baseline) model using metrics developed in the previous part of the assignment.

Extend the model and/or try new data [30pts] Extend the baseline model. Possible directions include (but are not limited to): (a) trying a different feature-extraction (encoding) architecture; (b) trying a different loss function and/or regression output space, e.g., $[\cos(t), \sin(t)]$ or full $3D$; and (c) predicting object masks as in Mask R-CNN. [2]. You are encouraged to try any publicly available, synthetic or real, dataset of your choice or collect/generate your own data.

[What to turn in] Upload your code and a report (PDF) that is concise but thorough. The report needs to include:

- motivation for your work (you can adapt it from the assignment preamble)
- hypotheses you wanted to test
- experiments you conducted
- lessons you learned

A significant part of your report should be devoted to qualitative and quantitative results. The report should show:

- status of the unit tests (for a correct implementation, they should all return `True`)
- quantitative results (training/test loss vs. epoch; angular-error histogram plots at the beginning and the end of training; any other metric you may wish to include)
- qualitative results (examples of visualizing the pose)

For the last part of the assignment, provide the qualitative and quantitative results in the same format as for your baseline implementation. Make sure to cite any papers and give credit to any open-source code that you use directly or indirectly!

References

[1] B. Bitterli. Rendering resources, 2016. https://benedikt-bitterli.me/resources/. 1

[2] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask R-CNN. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988, 2017. 1

[3] A. Mousavian, D. Anguelov, J. Flynn, and J. Kosecka. 3D bounding box estimation using deep learning and geometry. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017. 1

[4] M. Nimier-David, D. Vicini, T. Zeltner, and W. Jakob. Mitsuba 2: A retargetable forward and inverse renderer. *Transactions on Graphics (Proceedings of SIGGRAPH Asia)*, 38(6), Dec. 2019. 1