Programming Assignment 3: Convolutional Neural Networks, Image Formation, Camera Calibration, and Stereo

CS 4670 Spring 2020

Assigned:	Wednesday, April 29
Due:	Tuesday, May 12
Files to Submit:	`student.py`, `Writeup.pdf`, `model_weights.pth`, Kaggle submission

This project can be done **individually or in pairs**. You may use up to **two** slip days.

Overview

This assignment is split into **two parts**!. There is a written component similar to homework 1 and a programming part. The written component asks short answer questions about image formation, camera calibration, and stereo as well as some small questions related to the programming component. The programming component has you explore convolutional neural networks (CNNs) in the context of image classification. To create the CNN, you will be using the popular ML framework, Pytorch. This writeup will first detail the programming component followed by the written component.

Deliverables

You should submit the following files:

1. `student.py`: contains all of your work for the programming component. Make sure your work filled in all of the TODOs detailed in `PA3.ipynb`.

2. `Writeup.pdf`: contains answers to several questions found in `PA3.ipynb` as well as the questions about image formation, camera calibration, and stereo found in this writeup. This must be submitted on GradeScope.

3. `model_weights.pth`: contains weights to your network submitted to Kaggle competition. Must be < 5 MB.

4. A submission on Kaggle. Use `Kaggle_submission.ipynb` to create the .csv file containing the outputs of your model to submit to Kaggle.

Programming: Convolution Neural Networks

The following are your tasks:

1. Build and train a baseline model
2. Implement data augmentation techniques
3. Build your own model to improve upon the baseline
4. Generate adversarial examples to explore potential shortcomings of CNNs

Initial setup

Download the files for this project from CMS and put them in a single folder. There should be the following project files on CMS:

- `PA3.ipynb`: Jupyter notebook with instructions and visualizations
- `student.py`: Python file for student code
- `data.zip`: Folder of image data
- `models.zip`: Folder holding weights for pretrained baseline model
- `data_augmentation_sample.png` Example output plot for data augmentation portion
- `Kaggle_Submission.ipynb`: Jupyter notebook with instructions for creating the csv output to submit for Kaggle.

Download PyTorch (Below commands assume you are in Python 3.6)

Mac: `conda install pytorch torchvision -c pytorch`

Windows users: `conda install pytorch-cpu torchvision-cpu -c pytorch`

Additionallly, if you do not have conda, you may checkout https://pytorch.org/ for download instructions. If you have any questions about the initial setup, please check Piazza to see if someone else has run into the same issue. If not, feel free to post a new question!

Opening the assignment

Here is the workflow you should use any time you are working on this project:

1. In a terminal window, activate the `cs4670` environment.
2. Run `jupyter notebook` in the directory containing the project files. This should open up a GUI in your browser.
3. Click on `PA3.ipynb`, which will open up the Jupyter Notebook that we will be using for this project. This file contains detailed explanations about the project and the various TODOs. If you haven't used a Jupyter Notebook before, or if you'd like a refresher, check out this quick video tutorial!
4. All the code that you will write should go into `student.py` – this is the file that will be submitted and graded (in addition to the writeup)! The cells in the notebook simply import your code and run it, and you should not need to change any of the code in the notebook itself (although you are free to do so if you would like to modify the visualizations). Moreover, every cell that imports your code will automatically re-import your latest `student.py` code, so you can just save changes to `student.py` and re-run any cell.
5. Note that the notebook will lose its variables when it's restarted. Whenever you start the Jupyter Notebook server afresh, we recommend re-running all the cells up till the one you are currently working on. This will ensure that all the variables up till that point are re-loaded into the notebook.

Testing and Debugging

In the notebook

We have added cells within `PA3.ipynb` that help you visualize your results so far – these should give you a good idea of whether you're on the right track or not! You can also print variables in the notebook cells in case you'd like to look at them for debugging purposes.

Kaggle competition

There will be a Kaggle competition for this project. You will train your own models and make predictions on unseen test data (which you will not have labels for). You will get to see your model accuracy on the leaderboard. There will be an instructor model submission, which you will attempt to beat to get full points. More details about the competition will be released soon!

A note about grading

You may have noticed that there are no automated tests for this project. That's because it's inherently hard to have test cases with exact numerical comparisons for neural networks. Instead, your grade is based on the following:

- Quality of your images for data augmentation in the writeup
- Quality of adversarial images: must get at least 85 % misclassification rate for full points
- Quality of your network: must beat instructor model on Kaggle competition for full points, with your model size < 5MB
- Quality of answers and explanations in writeup

If you have any questions, please ask on Piazza or in office hours. We are here to help!

Optional Resources

The following are optional resources you may use if you would like a refresher on PyTorch or using GPU-accelerated training for your models. These are completely optional and will not be necessary for the assignment.

PyTorch

The examples on this site can give you a quick tutorial on how tensors work with PyTorch and how to use PyTorch in general. There are examples of how to build general models on the website, but note that the content will be different from the model we are asking you to build. The concepts that you will need to know to create the model for the assignment can be found in lecture slides.

CUDA Acceleration

If you have a CUDA-enabled NVIDIA GPU (you can test this by running *torch.cuda.is_available()* in the notebook and have CUDA installed, you can potentially speed up the training of your model. In order to use CUDA acceleration, you need to modify PA3.ipynb. The following list outlines some of the things you may need to change to get CUDA acceleration working.

- Set torch.backends.cudnn.benchmark to True
- set the dataloader pin_memory parameter to True.
- Move the network to the GPU after it is initialized. For example, you can call net.to('cuda').

- Move your loss function to the GPU.
- Move the inputs and labels to the GPU (remember to do this for both the training and validation stage)
- The labels may need to be moved back to the CPU in the validation phase when converting back to NumPy.

Assuming no other issues arise, you should be able to see a speedup in training time. Using a desktop with an I7-6700K and GTX 980TI, we had a training time for the baseline model of $\sim$ 480s using the CPU and $\sim$ 280s using the GPU through cuda acceleration. Depending on your gpu or configuration, you may not experience as dramatic of a speedup or may even slow down due to inefficient memory accesses.

Google Colab

If you don't have a GPU but would still like to use CUDA acceleration to speed up training, you can use Google Colab. Google Colab is rather similar to Jupyter Notebook in format and will allow you to speed up model training since Google has dedicated GPUs that you can use. The free version allows you to use a NVIDIA Tesla K80 GPU for 12 hours at a time.

To get started, head over to colab.research.google.com and upload the PA3.ipynb notebook. From here, you need to upload student.py, data.zip, and models.zip. There are several places where you can load the data.

1. **Google Colab local storage**. Since our dataset is relatively small, you can unzip the model and data folders directly into the Google Colab local storage. The benefit is that you will not deal with delays with memory accesses as you would with google drive. The downside is that it takes a bit to upload the data and everytime you restart your runtime or close your runtime, the files are lost and you will need to re-upload the models, data, and student.py to the local storage.
2. **Google Drive Mounting**. You can mount your google drive to Google Colab and store your models and data on google drive. The benefit is that you won't need to constantly re-upload the models and data. However, you will have quite a large amount of delay between retrieving files from drive to use when training. This will probably increase training time.
3. **Google Cloud Storage** You can also store your data and models in Google Cloud Storage. The access time for GCS is faster than google drive. This is a paid service but you get $300 free credit to use.

Once you've decided how to store the data, if you want to use GPU acceleration, switch to the GPU by going to Edit$\rightarrow$ Notebook Settings and select **GPU** from the menu. Finally, follow the steps in the **CUDA Acceleration** to modify the notebook to make use of the GPU.

It took $\sim$ 290s to train the baseline model without hardware acceleration on Google Colab and with CUDA acceleration, it took $\sim$ 180s.

Written: Image formation, Camera calibration, and Stereo

1 TODOs from the Programming Component

Please answer the questions in the jupyter notebook that were labeled TODO [writeup] here. They are copied here for your convenience.

1. Why is it important to have separate train and validation sets? What might happen if you *don't* have a validation set?

2. Include the plots generated from training the baseline model from scratch (i.e., without loading in pretrained weights). What do you notice about the train vs. validation performance? What might this mean about the model?

3. Include the plot from below with your images for data augmentation in the writeup. For a sample of correct behavior, look at the provided `data_augmentation_sample.png`. Note that you won't match it exactly since many values are randomized, but you can still see if your code is doing what you expect.

4. Include the plots showing your model's loss and accuracy history. Discuss your decisions for your model architecture, as well as `get_student_settings` if you changed the parameters from the baseline. What worked and what didn't? What might be some reasons why?

5. Include 3 adversarial image plots from the cells below in your writeup: one with epsilon=0.005, one with `epsilon=0.02`, and one with `epsilon=0.15`. As you increase epsilon, what happens to the misclassification rate? Why?

6. If you have any feedback on the programming component as a whole, please include! It is still a rather new assignment so We would love to hear your thoughts on it and how we can make it better next time around. Thanks!

2 Image formation and Camera Calibration

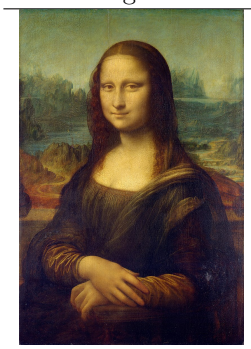
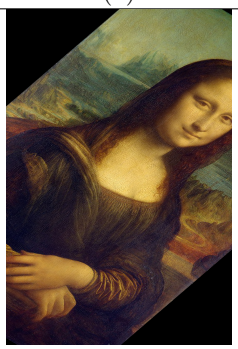
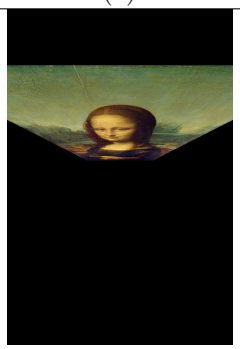
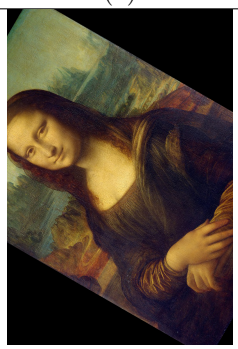
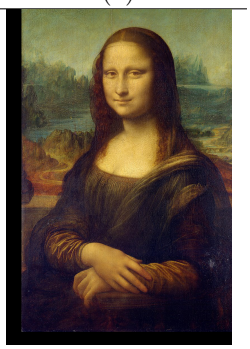
1. Given below are four geometric transformations that can be applied to an image, and the result of each transformation when applied to the Mona Lisa. However, the results are all jumbled up. Map each transformation to the corresponding results. Transformations:

   (a) $H = \begin{bmatrix} R & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix}$, $\det(R) = 1$.

   (b) $H = \begin{bmatrix} I & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix}$.

   (c) $H = \begin{bmatrix} R & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix}$, $\det(R) \neq 1$.

   (d) $H = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & c \\ 0 & d & e \end{bmatrix}$, $c, d, e \neq 0$.

Original	(1)	(2)	(3)	(4)
				

2. The camera projection matrix P maps a 3D point with homogenous coordinates $\vec{\mathbf{x}}_w$ to a 2D pixel with homogenous coordinates $\vec{\mathbf{x}}_{img}$. State if each statement below is true or false. Explain your answer in each case.

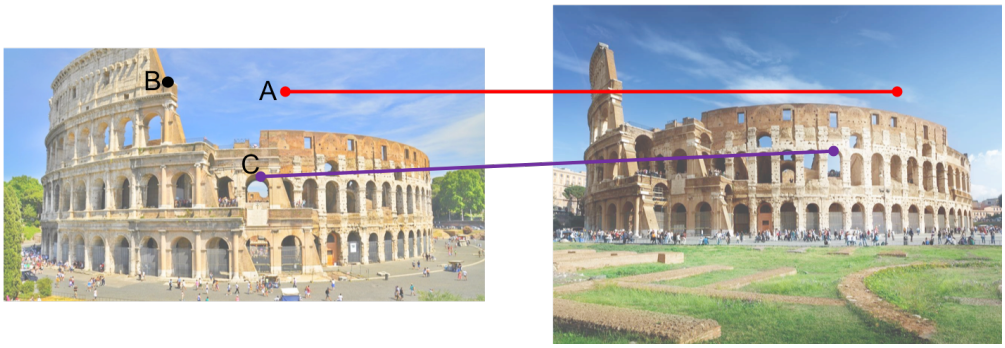
   (a) We can estimate the camera projection matrix if we have the image location of at least 4 non-coplanar points.

   (b) The camera projection matrix P and αP, for $\alpha \neq 0$ represent the same camera.

   (c) To reconstruct the $3D$ coordinates of a point, we need its image location in at least 3 cameras with known projection matrices.

3 Stereo

1. Consider the following two images of the same scene taken from two nearby cameras. Sakshi has written code to identify pairs of corresponding pixels, but she is getting a few incorrect correspondences, shown in the figure below (no good correspondence was found for point B: the difference between the descriptors was too much).



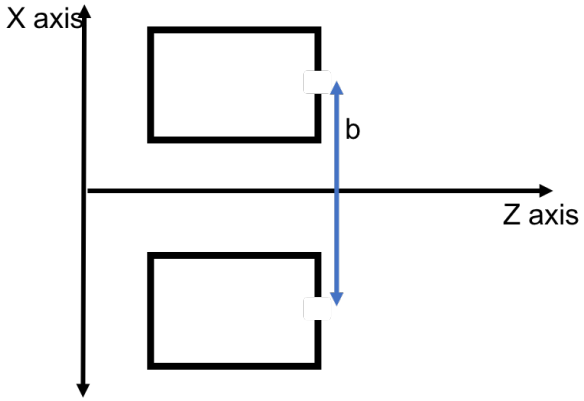
She is considering the following three modifications to her system. Each modification is designed to correct exactly one of the three errors. Match the modification with the error it will correct. Very briefly explain why the modification will correct the error.

(a) Increase the threshold on the corner detector so that it identifies fewer corners.

(b) Increase the size of the patch used for the SIFT descriptor.

(c) Reduce the number of orientation bins.

2. Consider a pair of rectified cameras, i.e., cameras pointing in the same direction, the Z-axis, translated by a distance b along the X axis. The world coordinate system is centered on the first camera.



(a) Given a pair of corresponding pixels (x, y) and (x', y') in the two images that correspond to the same 3D point (X, Y, Z), the disparity δ is the difference in the X coordinates $|x' - x|$. How is the disparity related to depth? How is it related to the distance between the cameras b?

(b) Consider two 3D points at depths Z and $Z + dZ$ whose depth differs by an infinitesimal amount dZ. How does the *difference between their disparities* $d\delta$ vary with Z? Pick one of the following options and explain your choice.

i. It is (approximately) directly proportional to Z.

ii. It is (approximately) directly proportional to Z^2.

iii. It is (approximately) inversely proportional to Z.

iv. It is (approximately) inversely proportional to Z^2.

(c) Disparity is often rounded up to the nearest integer. This rounding up introduces an error. Suppose the disparity for a particular pixel is 3 ± 0.5 units, and $b = 35$ units. What is the minimum and maximum depth for this pixel?