Robotics and Computer Vision Final Project Paper

Manish Kewalramani

December 2019

1 Problem 1

This semester we have learned algorithms for 3D reconstruction with two images. However, many computer vision applications work better when they use multiple images. For this first problem of the project we will use structure from motion (SFM) to create a 3D point cloud of an object from multiple images of it. The 30 images used for reconstruction can be found in the submission directory Problem1Data/InputImages. To do the reconstruction, the provided library OpenMVG was used. The library was downloaded onto my google drive for use in google colab. It can be found here: https://openmvg.readthedocs.io/en/latest/software/SfM/SfM/ The code for doing so is seen here:

```
import os
from google.colab import drive

drive.mount('/content/drive', force_remount=True)
os.chdir('/content/drive/My Drive/data')
!sudo apt-get install libpng-dev libjpeg-dev libtiff-dev libxxf86vm1 libxxf86vm-dev libxi-dev libxrandr-dev
!sudo apt-get install graphviz
!git clone --recursive https://github.com/openMVG/openMVG.git
!mkdir openMVG_Build && cd openMVG_Build
!cmake -DCMAKE_BUILD_TYPE=RELEASE ./openMVG/src/
!cmake --build . --target install
!make test
!ctest --output-on-failure -j
```

Figure 1: My code for installing OpenMVG

OpenMVG will take the input images and run them through its pipeline, which is broken into the following steps:

1. Intrinsics analysis
   This step converts the input images into JSON files that the OpenMVG algorithm can use for processing. If OpenMVG can tell what kind of camera you used to take the images, it will obtain the cameras intrinsic matrix and store it in this object.

2. Compute features
   This step computes the features of key points and creates description vectors to describe them. The default method for getting the features is to use SIFT, and I used the default.

3. Compute matches
   This step uses the feature vectors of the keypoints calculated in the previous step to find matches between keypoints. Essentially if a keypoint in one image has the same feature vector as a keypoint in a different image they are matched to be the same point in the 3D reconstruction.

4. Do Global reconstruction
   This step uses the matches found in the previous step to create the 3D point cloud. The positions of the matching points in each image are used to find the positions of other key points relative to the matching points. In this step the algorithm performs the bundle adjustment in order to make sure that the point cloud is globally optimal.

5. Colorize Structure
   This is the last step that I used. It uses the original images to color the points in the pointcloud.

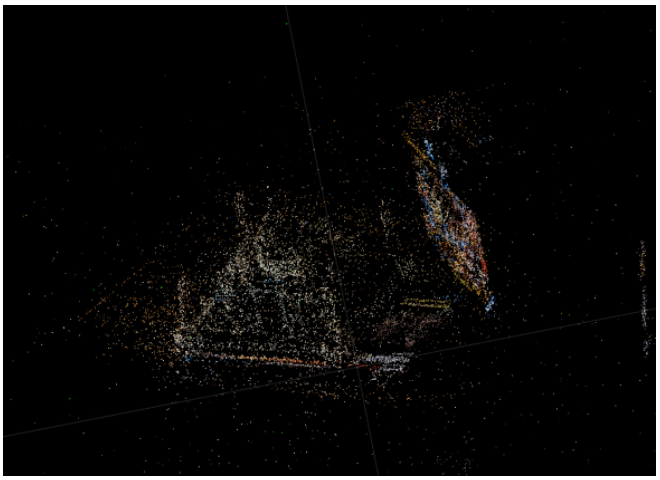
The code for the pipeline can be seen here:

```
print ("1. Intrinsics analysis")
pIntrisics = subprocess.Popen( [os.path.join(OPENMVG_SFM_BIN, "openMVG_main_SfMInit_ImageListing"),  "-i", input_dir, "-o", matches_dir, "-d", camera_file_params , "-f", "3500"] )
pIntrisics.wait()

print ("2. Compute features")
pFeatures = subprocess.Popen( [os.path.join(OPENMVG_SFM_BIN, "openMVG_main_ComputeFeatures"),  "-i", matches_dir+"/sfm_data.json", "-o", matches_dir, "-m", "SIFT"] )
pFeatures.wait()

print ("3. Compute matches")
pMatches = subprocess.Popen( [os.path.join(OPENMVG_SFM_BIN, "openMVG_main_ComputeMatches"),  "-i", matches_dir+"/sfm_data.json", "-o", matches_dir, "-g", "e"] )
pMatches.wait()

# Create the reconstruction if not present
if not os.path.exists(reconstruction_dir):
    os.mkdir(reconstruction_dir)

print ("4. Do Global reconstruction")
pRecons = subprocess.Popen( [os.path.join(OPENMVG_SFM_BIN, "openMVG_main_GlobalSfM"),  "-i", matches_dir+"/sfm_data.json", "-m", matches_dir, "-o", reconstruction_dir] )
pRecons.wait()

print ("5. Colorize Structure")
pRecons = subprocess.Popen( [os.path.join(OPENMVG_SFM_BIN, "openMVG_main_ComputeSfM_DataColor"),  "-i", reconstruction_dir+"/sfm_data.bin", "-o", os.path.join(reconstruction_dir,"colorized.ply")] )
pRecons.wait()
```

Figure 2: Code for running SFM pipeline

One input image can be seen here, next to the 3D point cloud reconstruction.

(a) The object to be reconstructed



(b) The 3D point cloud of the object

Figure 3: One input image and the output point cloud

2 Problem 2

For this problem I fine tuned a 5 class classifier using Pytorch. The images used were a training set consisting of 50 images each from the classes 'airplane', 'car', 'cat', 'dog', and 'flower' and an evaluation set containing 30 images each from the same classes. The images were taken from a much larger dataset called 'Natural Images' on Kaggle, which can be found here: https://www.kaggle.com/prasunroy/natural-images . The network used was ResNet 18, an 18 layer deep convolutional neural network that is notable for its use of skip connections within the architecture to make back propogation and gradient calculation more efficient. This network has been trained on the imagenet dataset, meaning it was trained on over a million images to identify amongst 1000 classes. For this problem I will finetune the network to classify among only five classes with only 250 images of training data. The finetuning was done using the pytorch tutorial linked in the assignment, http://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html.

The classifier was trained with 10 epochs, and the loss function used was the Pytorch library's Cross Entropy Loss, a variation of logistic loss. The equation for the loss function is:

$$\text{loss}(x, class) = -\log\left(\frac{\exp(x[class])}{\sum_j \exp(x[j])}\right) = -x[class] + \log\left(\sum_j \exp(x[j])\right)$$

The training loss of the classifier after each Epoch is plotted as:

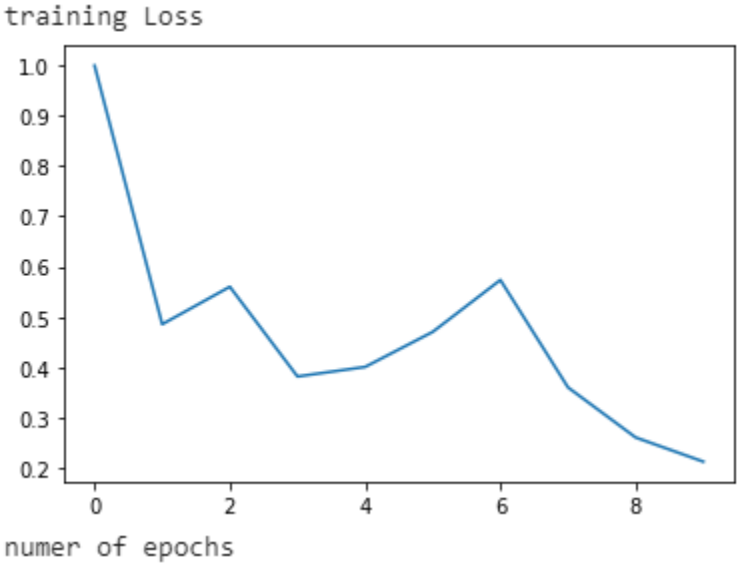


Figure 4: Training Loss for classifier

We can see from the figure that the loss decreased over the course of the epochs, as expected. The training ran in 1 minute and 21 seconds. After the training finished running it on the evaluation data produced this confusion matrix:

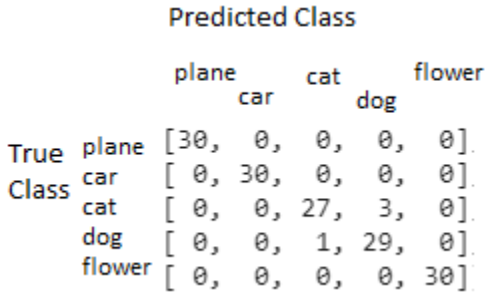


Figure 5: Confusion matrix after training the classifier

From the confusion matrix I evaluate this table of precision, recall, and accuracy:

Class	TP	TN	FP	FN	Accuracy	Precision	Recall
Airplane	30	120	0	0	1	1	1
Car	30	120	0	0	1	1	1
cat	27	117	1	3	0.972973	0.964286	0.9
dog	29	119	3	1	0.973684	0.90625	0.966667
flower	30	120	0	0	1	1	1

Figure 6: Precision/Recall/Accuracy Table

If we examine the accuracy, each classifier seems to work over 97% of the time, which is phenomenal. The precision and recall numbers are also very good, although a little lower for cats and dogs for some reason. If we look at the confusion matrix, we can see clear evidence of a hunch that you might have had from looking at the table, as we see that the classfier seems to be mistaking some cats for dogs and has mistaken a dog for a cat. This would suggest that we would need a larger dataset of dog and cat images to stop it from confusing these two classes. The training was redone with batch normalization and drop out regularization. Batch normalization normalizes the activation layers to speed up the learning process. Drop out regularization zeroes out some of the nodes in the network at random in order to stop the problem of overfitting. Both of the batch normalized training and the dropout training finished in 52 seconds, roughly 64% of the time it took the training to happen without either process. The highest accuracy of an epoch on the evaluation set was comparable for all three processes. The data is summarized in the following table:

	Training time	Best epoch Accuracy
Initial training	1m 21 s	0.98
Batch Normalization	52s	0.98
Dropout Regularization	52s	0.966667

Figure 7: Comparison of training methods