Computer Vision Final Project

Ankeet Parikh
Department of Electrical and Computer Engineering, Rutgers University–New Brunswick, Piscataway, NJ
E-mail:ankeet.parikh@rutgers.edu

I. Introduction

This project consisted of several parts. The first was to perform 3-D reconstruction using two-dimensional images. The procedures involved included algorithms learned in class such as SIFT and Bundle Adjustment, in addition to more complicated optimization procedures. The photos were taken using an iPhone 6s camera, and reconstructed using openMVG, and the point cloud was plotted in MATLAB. The intrinsic matrix of the camera, denoted K, was determined using the openCV camera calibration toolbox.

The second portion of the project was to use neural networks for image classification. The first portion was to use images for four different classes and determine some important benchmarks of the network such as the confusion matrix, accuracy, precision, recall, etc. The second portion was to use an already trained network, and modify it to classify images as Cats vs. Dogs. Here, we use a procedure commonly known as "fine tuning", which removes the top layer of the neural net, replaces it with a new layer, and trains the neural net for whatever custom classification is required. Reusing chunks of a neural network not only saves on computation, but also leads to equally good results as construction from scratch.

II. Problem 1: 3-D Reconstruction

A. Results

This portion dealt with 3-dimensional reconstruction of an object using a series of images. In a more general sense, this procedure is called "Structure from Motion", which takes images that are related to each other by some movements (rotation and translation), and reconstructs a point cloud based off those images. The algorithms used are quite complicated, so we will explain only a few of the simple ones here. Before we do that, let's take a look at a few of the images taken and the point cloud that was constructed. We took several pictures of a house located at 542 George Street on the College Avenue campus:

Fig. 1: Social Work Annex Building, 542 George Street

Fig. 2: Additional Image of the same

In total, there were 22 images of the house taken, and they produce a point cloud that noticeably resembles the picture of the house. Using the computer vision toolbox in MATLAB, the following point clouds were constructed:

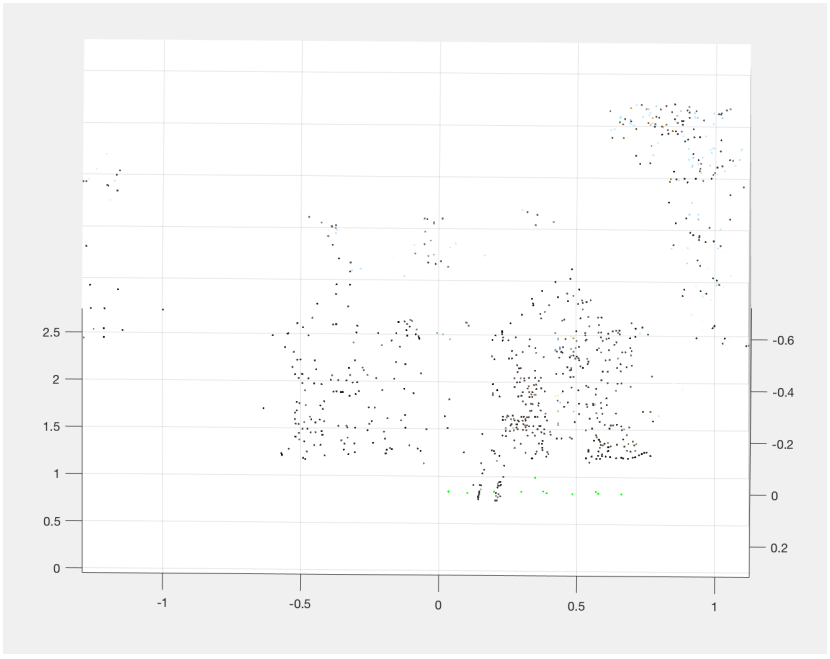

Fig. 3: Point cloud, front view

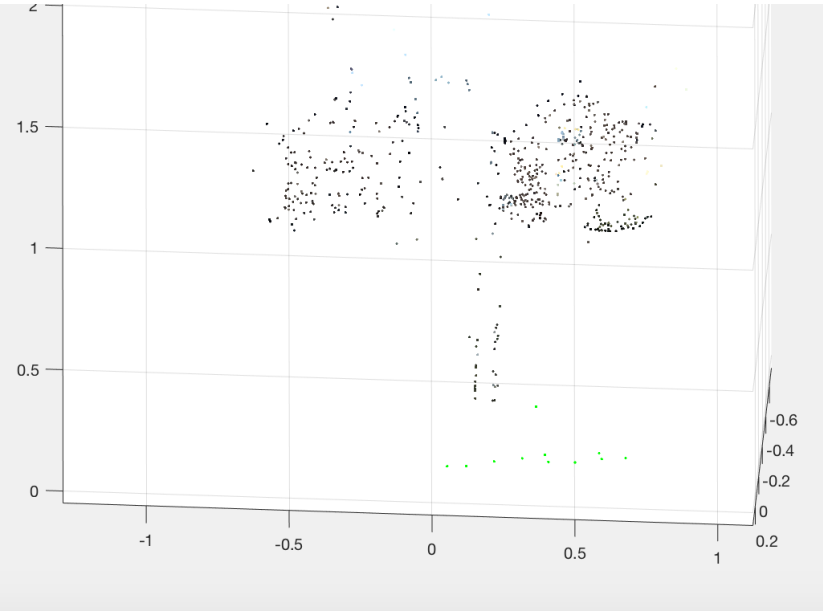

Fig. 4: Point cloud, front-top view, note the sidewalk

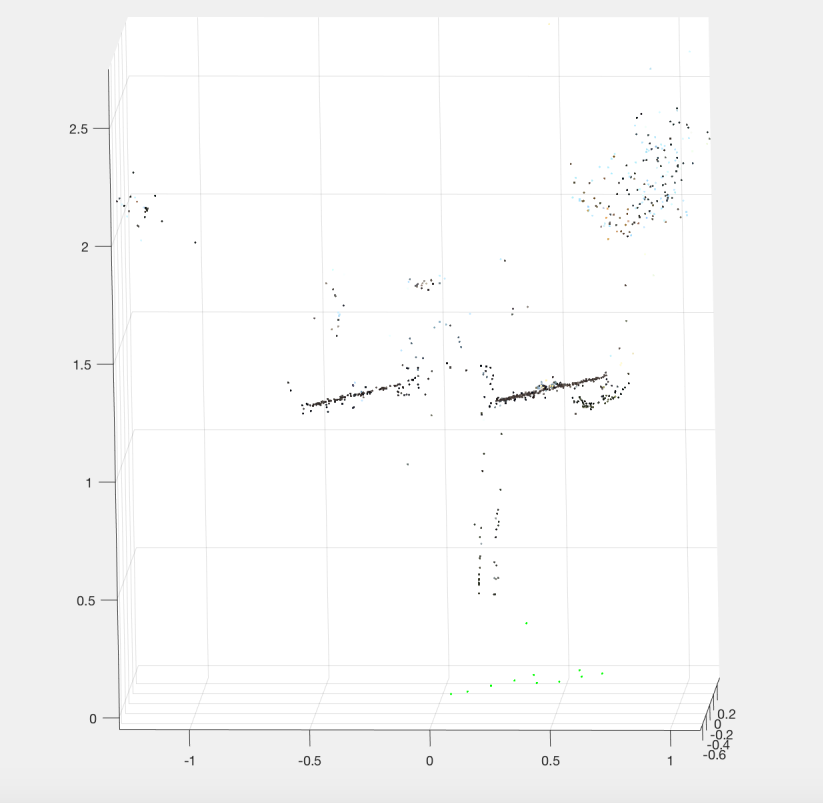

Fig. 5: Point cloud, top view, note the front face of the house and the tree in the top-right

B. Methodology

The code I used for this procedure was in the openMVG_Build directory as ”tutorial_demo.py”. The code submitted in Part1.py is almost exactly the same as the code provided, with some minor file path modifications. Let’s take a look at what the algorithm is doing.

1) Intrinsics Analysis: Here, we provide the matrix K and openMVG computes some important values it will need in subsequent operations. These include focal length in pixels and millimeters, and the sensor width size.

2) Computing Features: This portion extracts features such as edges, blobs, corners, ridges, curvatures, and other such nuances.

3) Computing Matches: Here, **SIFT** is used to compute point matches between images. SIFT computes the gradient around each pixel of an image, and considers the 16x16 window surrounding the image. The window is weighted by a Gaussian fall-off function, and then the keypoint descriptor is determined. Points are matched by comparing the similarity between their keypoint descriptors.

4) Sequential/Global Reconstruction: Here is where the rays are triangulated and the world coordinates are determined. The image formation pipeline has been described excessively in class so I will not describe it here.

In addition to the algorithms described above, the openMVG 3-D reconstruction uses procedures such as RANSAC, and Bundle Adjustment to compute a more accurate result. RANSAC is an algorithm that consists of several iterations of the same procedure. At each iteration, a minimal subset of the data points is chosen, and the model is determined using that subset. Using the computed model, a set of inliers is determined. Inliers are the data points that are deemed consistent with the model. The model with the largest number of inliers is then used for subsequent algorithms. Bundle adjustment seeks to minimize the reprojection error of the observed and predicted image points. This is done through using several images and computing minimum values of nonlinear functions.

III. Problem 2: Neural Net Classification

A. Imagenet Classification

This portion of the project was identical to question 3 on Homework 7. We used ResNet50 to classify images using classes in the ImageNet database. ResNet50, short for "Residual Network 50" is a neural network consisting of fifty layers. The principle behind ResNet is that **shortcut connections** which skip one or more layers help improve the accuracy of the network. Here is a building block of ResNet that shows a shortcut connection:

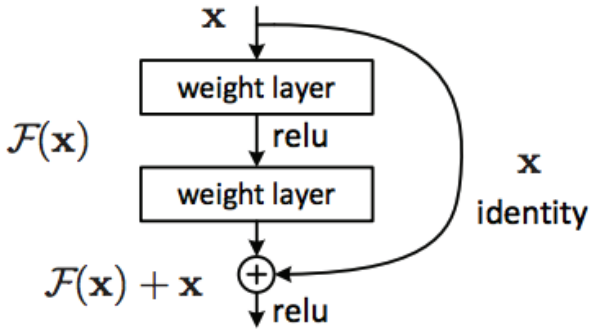

Fig. 6: Building Block of Resnet

In addition to shortcut connections, ResNet uses downsampling between layers, max pooling, specific stride values, and a softmax activation function. Here are the results:

		Actual Class			
		Menu	Palace	Lynx	Pizza
Predicted Class	Menu	3	-	-	-
	Palace	-	3	-	-
	Lynx	-	-	5	-
	Pizza	-	-	-	3
	Scoreboard	1	-	-	-
	Crate	1	-	-	-
	Church	-	1	-	-
	Castle	-	1	-	-
	Cauliflower	-	-	-	1
	Band-Aid	-	-	-	1

With values:

$$TP = 14$$

$$FP = 0$$

$$FN = 6$$

$$TN = 0$$

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{14}{20} = 70.0\%$$

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{14}{14} = 100\%$$

$$\text{Recall} = \frac{TP}{TP + FN} = \frac{14}{20} = 70\%$$

B. Cats vs. Dogs

This portion of the project involved fine tuning a network to classify images as cats vs dogs. Fine tuning is a procedure where the bottom layers of a network are frozen, and a new top layer is added. The network is trained (though the weights of the top layers are modified) as usual. Here, we froze 172 layers, replaced the top layer, and then left the rest to train. The network we use for this procedure is called InceptionV3. InceptionV3 is a network that is specifically trained on the ImageNet database. Let's take a look at how it works.

InceptionV3 is a network that consists of **inception modules** connected in series to each other. Each module will do several convolutions (3x3, 5x5, etc.) of the previous layer, then concatenate the convolutions together and pass them on to the subsequent module. Why is this useful? The basic idea is that at each layer, you cannot know for sure that doing a 3x3 convolution is a better idea than doing a 5x5 convolution. You could do all the convolutions and then let the model decide which one is best. In order to avoid significant computation at each layer, we can do something called **Dimensionality Reduction** which reduces the number of large convolutions that need to be performed at each layer.

For training the neural net, we used 500 images of cats and 500 images of dogs. The validation directory had the same size files. The test directory also consisted of 500 cats and 500 dogs. Here is the confusion matrix:

		Actual Class	
		Cat	Dog
Predicted	Cat	494	27
Class	Dog	6	473

Here are the relevant values:

$$TP = 494 + 473 = 967$$

$$FN = 6 + 27 = 33$$

Because there are only two classes, a false positive would technically be the same thing as a false negative. For example, if I were to give a picture of a dog, and the classifier said that it was a cat, then we have two things that happen: a false positive because we said it was a cat when it was not, a false negative because it said it was not a dog when it really was. Similarly, a true negative is the same thing as a true positive. Thus, we could change the problem from "Cats vs Dogs" to "Cats vs Not Cats and Dogs vs Not Dogs"

Lets do Cats vs Not Cats:

$$TP = 494$$

$$FP = 27$$

$$FN = 6$$

$$TN = 473$$

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{967}{1000} = 96.7\%$$

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{494}{494 + 27} = 94.8\%$$

$$\text{Recall} = \frac{TP}{TP + FN} = \frac{494}{494 + 6} = 98.8\%$$

Now Dogs vs Not Dogs:

$$TP = 473$$

$$FP = 6$$

$$FN = 27$$

$$TN = 473$$

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{967}{1000} = 96.7\%$$

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{473}{473 + 6} = 98.7\%$$

$$\text{Recall} = \frac{TP}{TP + FN} = \frac{473}{473 + 27} = 94.6\%$$

IV. Open Source Code Usage

This project used some opensource code, which I will cite here. For Part 1, we used the tutorial code that was in the openMVG_Build directory. You can see the code here:

https://github.com/openMVG/openMVG/blob/master/src/software/SfM/tutorial_demo.py.in

For Part 2 Imagenet Classification, I followed the tutorial here, and used the open source code to perform the procedures required:

https://www.pyimagesearch.com/2016/08/10/imagenet-classification-with-python-and-keras/

For Part 2 CatsvDogs, I used code from this tutorial, and the code is provided here:

https://github.com/DeepLearningSandbox/DeepLearningSandbox/tree/master/transfer_learning Additionally for part 2, I used some code from this blog by the author of keras:

https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

References

[1] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *Inception Modules: explained and implemented*. Addison-Wesley, Reading, Massachusetts, 1993.

[2] Mulch, Tommy. (2016, September 25) *Inception Modules: explained and implemented*. Retrieved from https://hacktilldawn.com/2016/09/25/inception-modules-explained-and-implemented/

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun: Deep Residual Learning for Image Recognition. CVPR 2016: 770-778