Robotic Inference

Nagarjun Redla

Abstract—Optimizing neural networks for inference is a very important aspect for mobile robots. With recent advances in Deep Learning methods, it is a great advantage for mobile robots to leverage this technology to improve perception. There have been many image classification networks that have been proposed in recent years that have changed the face of the image processing community. In this paper we discuss the implementation of a Neural Network called 'GoogLeNet' [1], a LeNet based architecture developed by engineers at Google. We present this neural network's performance by training on two datasets, one provided by Udacity and another on our own dataset.

Index Terms—Robot, IEEEtran, Udacity, LaTeX, deep learning.

1 Introduction

Machine Learning methods can be split into two main phases: Training and inference. In the context of Deep Neural Networks, the training phase takes a lot of compute power and memory. This is because training involves calculating and keeping track of gradients for backpropagation that helps the network learn. Inference, on the other hand, only uses the graph generated after training for forward pass, and though still compute intensive, it does not calculate gradients or perform a backward pass on the network. This gives us the opportunity to optimize this phase. These optimizations allow us to run inference on devices with much lower compute power, like a mobile phone. This project deals with development for inference, and focuses more on optimizing the neural network for prediction rather than dive into the details of training. Nvidia's DIGITS tool is used for training, which removes the complexity of normally having to write up your own neural network layers or tweak pretrained networks to your preference. For the inference phase, Nvidia provides a tool called TensorRT which optimizes the final trained graph for accelerating inference speeds on low power CUDA GPUs, like the Jetson.

2 Background / Formulation

The initial idea for the project originated from a demo shown at the Lenovo stall at Intel's AI Developer Conference 2018 at San Francisco. The demo involved a Logitech C920 pointed at a conveyor belt which had soda cans on it. A neural network was used to detect the amount of damage the soda can went through. It was a pretty simple setup but had quite a bit of misclassification. The training set was only 60 images but they used data augmentation techniques like occlusion and distortion to generate more training examples. This project is based on a similar idea but to classify a soda can into three categories: good, damaged and defective. An example of a good soda can is one which is not crushed at all with the printed labels as they should be. A defective soda can is one whose printed labels are tampered with or are not clear. A damaged soda can is one that is visibly crushed or broken. The main challenge of this project was to make a neural network converge given the how similar the input dataset was. All cans chosen were of Diet Coke, and the defective and good cans look very similar in shape. The defective and good cans only differed in the printed pattern on the cans while the damaged pieces looked far different from the others since their shape is distorted. The model chosen was GoogLeNet which is a pre-supplied model in DIGITS, and is on the sweet spot in the speed vs. accuracy graph.

3 Data Acquisition

The dataset was collected using an Nvidia Jetson TX2 with its on-board camera. Previous attempts were made to use a Logitech C920 camera but that took a lot of downscaling before training and the data acquisition process was resource intensive using a HD camera that uses up processor bandwidth over USB. The Jetson's CSI camera offered a much more seamless interface for collecting data. The other hurdle was to collect enough cans for use. A Diet Coke is a low calorie drink but should not be consumed in excess by a single person. Also a bunch of empty cans lying around in the office or in a shared apartment will eventually get cleaned away when a colleague or a roommate takes out the trash if not informed beforehand. Another point to be noted (it will be addressed separately at the end of the report) is to collect photos of cans in the same position and surrounding lighting. It is difficult to know what a neural network will concentrate on as a feature beforehand and will take up time while troubleshooting if data acquisition was not done properly.

The pictures taken are in RGB format with a dimension of 500x500 pixels. There are a total of 252 images in 4 different classes: Good, Defective, Damaged and Nothing.

4 Results

4.1 Provided Dataset

The provided dataset includes objects from three categories: Bottle, Candy Box and Nothing. We train GoogLeNet to train on this dataset. It is split into train, test and validation sets with a 70:10:20 split. It contains 10094 RGB images. The

(a) Good soda can examples

(b) Defective soda cans

(c) Damaged soda cans

Fig. 1: Soda can dataset examples

evaluate output is given in figure 2 and results are given in figure 3.

```
root@1fb5e42dcf36:/home/workspace# evaluate
Do not run while you are processing data or training a model.

Please enter the Job ID: 20181204-060842-0683

Calculating average inference time over 10 samples...
deploy: /opt/DIGITS/digits/jobs/20181204-060842-0683/deploy.prototxt
model: /opt/DIGITS/digits/jobs/20181204-060842-0683/snapshot_iter_6630.caffemodel
output: softmax
iterations: 5
avgRuns: 10
Input "data": 3x224x224
Output "softmax": 3x1x1
name=data, bindingIndex=0, buffers.size()=2
name=softmax, bindingIndex=1, buffers.size()=2
Average over 10 runs is 5.47126 ms.
Average over 10 runs is 5.48036 ms.
Average over 10 runs is 5.47066 ms.
Average over 10 runs is 4.951 ms.
Average over 10 runs is 4.93424 ms.

Calculating model accuacy...

  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 14614  100 12298  100  2316    169     31  0:01:14  0:01:12  0:00:02  2793

Your model accuacy is 75.4098360656 %
```

Fig. 2: Provided Dataset evaluate output.

4.2 Soda Can Dataset

The dataset was initially trained for 30 epochs with satisfactory results. The dataset is split in 80:10:10 ratio of train, test and validation respectively. The epochs were increased to 50 with only a slight increase in accuracy, i.e. the model accuracy didn't change for epochs beyond 30 much. The final model also ran in real time using TensorRT on the Nvidia Jetson. There is a high probability of overfitting since the data is highly correlated/very similar across labels. The

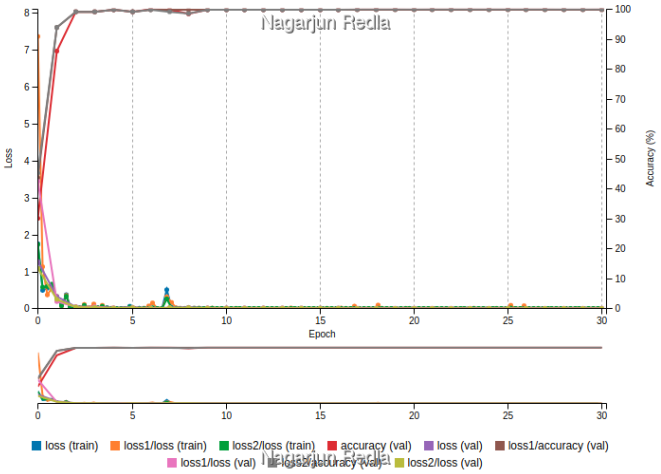

Fig. 3: Provided Dataset results for 30 epochs.

output graphs from DIGITS for the soda can model are shown in figures 4 and 5.

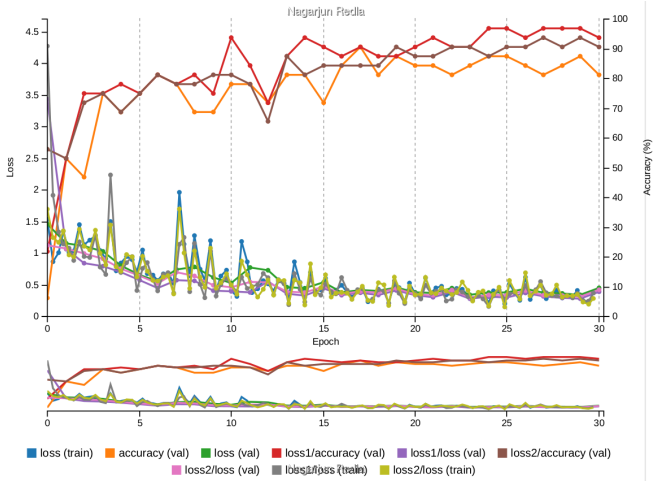

Fig. 4: Results for 30 epochs on soda can dataset.

5 Discussion

The model performed well on test images on DIGITS, but had varied results when deployed to the Jetson. Most of the shortcomings of this model were to do with data acquisition. Soda cans are highly reflective surfaces and the main difficulty was in classifying good and defective soda cans. A defective soda can is a good soda can with a tampered or improperly printed label. Unlike an untrained machine learning model, we humans know what to expect of the label. For a machine learning model trained from scratch it is harder to learn the actual printed label of the can. During data collection, there was a lot of reflection from the can, which included a human subject in a few pictures and other items from the room in the reflection. This makes it harder for the Machine Learning algorithm to understand the true label on an untampered good soda can. Individual images tested on DIGITS showed neurons that were activated and had similar neurons being activated for both defective and good images, since the network

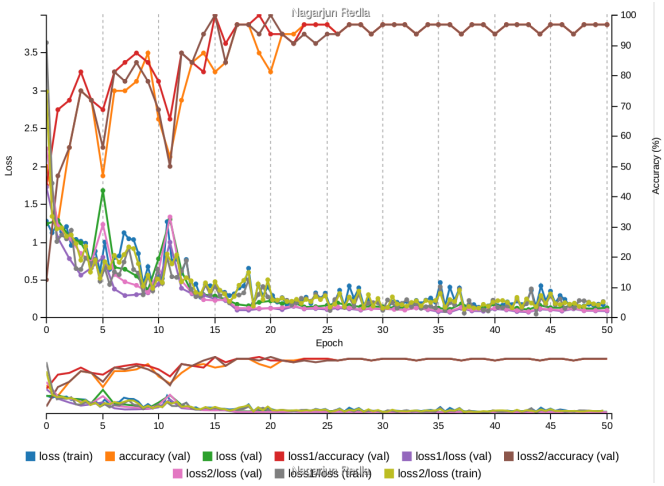

Fig. 5: Results for 50 epochs on soda can dataset.

was picking up perturbations in reflections as defects with the soda can's printed label. Deploying on the Jetson and trying live camera inference made these problems much more distinguishable, with a small change in the reflection on the can making the model jump between classifying it as good or defective. The second issue was soda can placement during data acquisition. Unlike the conveyor belt model given in the example dataset, the soda cans were manually placed in front of the camera, sometimes not in the same place (a marked place and a solid background would have helped). Because of this difference in placement and non-uniform surroundings, the model could not push for higher confidence values. The final issue was diagnosed during live camera inference where objects farther away were being classified as damaged. Since damaged soda cans differed from the good and defective ones mainly in shape, the neural network associated the smaller size of good or defective cans to them being damaged. All of these shortcomings can be mitigated with better, consistent data acquisition methods.

6 Conclusion / Future work

This idea of detecting damaged and defective soda cans could be used by soda makers and by soda can recycling plants. This could reduce time and overhead created by defective manufacturing/recycling processes. This was the final idea chosen after trying multiple different methods that weren't easy to execute given constraints of the provided VM. A previous idea was a detection model using DetectNet (as explained on DIGITS' documentation) [2] which kept running out of space on the GPU. Detection networks on DIGITS also requiresthe input to be in KITTI format, which isn't the default option for common labeling tools. The soda can idea was chosen to offset complexity from training to concentrate more on inference. Now that a simple configuration has been executed, the next step would be try out a detection network

References

[1] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," 2015.
[2] Nvidia, "Detectnet for digits,"