CS298 Proposal

Detecting Cars in a Parking Lot using Deep Learning

Samuel Ordonia (samuel.ordonia@sjsu.edu)

Advisor: Dr. Chris Pollett

Committee Members: TBD

Abstract:

Deep learning has reemerged onto the scene of research because of higher computational power and enormous data sets. Specifically, in the field of computer vision, there have been several advances in detection accuracy following the renewed research efforts into deep learning. Since the introduction of the Convolutional Neural Net (CNN) in the late 1990s, several iterations and improvements on the CNN have been made in this field over the years afterwards. However, these improvements have mostly been in iconic images--those with one or two objects in them. This master's project focuses on classifying and locating several objects--namely, cars--using a custom CNN built on top of the darknet "You Only Look Once" (YOLO) fully convolutional net.

CS297 Results

- **Pytorch Binary Classifier**

  Build a digit classifier that will distinguish 0-4 digits from 5-9 ones. This is a modification of the MNIST digit classifier, which classifies images of digits 0-9 by matching them with their corresponding ground truth meaning with ~97% accuracy.

  The project showcases how Pytorch does the following.

  1. Declare tensors
  2. Define the neural net layers, including layer types, input and output channels, and activation functions
  3. Set up the optimizer with parameters like learning rate and momentum
  4. Partition training versus testing
  5. GPU integration

  The neural net is based on the AlexNet Convolutional Net that is used in the MNIST 0-9 classifier. The network is modified to output a binary result, instead of a [1, 10] one hot vector denoting the classified digit. It also is "simpler" than the AlexNet one, lacking the first of the dense layers, since feature sharing can simply happen at the end during binary classification in the fully connected output layer.

  Accuracy is kept at ~97% for the binary classifier. For training, three epochs were sufficient to saturate at the 97% accuracy value. Training and testing took under thirty seconds on an NVIDIA 1070 GPU using the CUDA framework.

- **Image Processing and Deep Learning Environment**

  ##### Environment

  Set up an environment with image processing and deep learning functionalities. It should be sufficiently sequestered from the main machine while also support easy integration with the GPU and CUDA framework. It should not install (esp not as root) anything to the host machine directly.

  This environment would contain the deep learning framework Pytorch and the open CV image processing and manipulation framework. Pytorch has been simple to set up with a virtual environment. But open CV proved to wreak havoc on the host machine's environment by overriding install paths and installing other folders in unexpected places.

  A solution was needed to set up these dependencies for object detection that could support all the functionalities of both frameworks, but leave the host machine untouched.

  The solution is to use Docker to contain the dependencies.

  ##### Image Processing Functions

  Some open CV image transformations were added and tested out in this environment. The following are the functionalities built.

  - Rotation
  - Translation
  - Resize

- Concatenation
- Affine

There is also some logic open CV supports to convert videos into a series of frame-by-frame images. If there is camera recording input, this can be converted to a series of images, then fed into the neural net. Video is the next level of training input for ML models, and Facebook has been a forerunner in that respect with its massive data stores of video content. It makes sense to have this functionality handy, especially for testing.

- **Custom Data Annotation for Images**

  Annotation of an image requires accurately identifying which objects are in each image as well as drawing bounding boxes around each one. The ground truth is saved to a text or markup file. Careful attention must be paid to maintaining the relationship between the output ground truth and the image annotated.

  The annotation tool is built. It supports YOLO or VOC format. Annotations moving forward can be done with either format, but should stay consistent.

  The much bigger challenge is how to scale image annotation. There simply is not enough time for research and manual annotation. It is good to have this tool available, but it is quite apparent that research can only be accomplished if outside resources support image annotation. There are whole teams, such as the Microsoft one behind the COCO dataset, that have devoted themselves to building such datasets. It is an unfortunate reality that a single researcher simply does not have the time or capacity to do the same.

  Fortunately, there are researchers willing to share their data. Two data sets have been identified (the PKLot one and the Hsieh one).

  Connected with one researcher about getting annotated parking lot images. There are a few thousand parking lot frames to train on. They are annotated, but the conversion will need to be made into whatever format YOLO takes. His work focused on image detection with drones. It is a promising stepping stone towards working on a fixed camera solution to object detection. There is also the PKLot dataset that can provide more training data. It will require some work to port the annotation format over to VOC or YOLO. It is apparently non-trivial to set up training for, but perhaps the effort would pay off with the additional training examples.

  #### Final Remarks

  The annotation tool is handy to have around, but it is best used in the context of a team of researchers and "annotationists" dedicated to generating ground truth parking lot images. For a single researcher to annotate even a hundred images is arduous and not scalable.

  There are thousands of annotated parking lot data to leverage. Once a neural network is set up for training, validation, and testing, efforts must be made to incorporate these images into training.

  Perhaps the final 100 or so images manually annotated by the researcher can be used for testing.

- **YOLO Detection**

  Set up YOLO neural network for object detection in an input image. The YOLO architecture has shown promise in real-time object detection at speeds fast enough to be applicable to dynamic environments.

  YOLO works quite well on iconic images. Even on non-iconic images, the model can detect objects in the background as well as upfront ones. But accuracy needs to improve for cards further back in the image.

  The time spent per image is under two seconds on an NVIDIA 1070 GPU. The majority of the time is lost in the lack of optimization in the Pytorch implementation of YOLO, which would improve with a preloaded model running on a server with gunicorn or django accepting image requests. The GPU does not seem to be fully utilized either, running at ~11% VRAM usage during detection.

  YOLO proves itself to be a powerful starting point to develop a model that can detect several objects of different pose in an image. Detecting several cars in a parking lot will require updates to the model.

  #### Image Types

  Not all images are the same. There are differences such as time of day, number of objects per image, how many different types of objects are in an image, and even the image pixel density. Classical image detection would also apply processing techniques such as conversion to grayscale to augment image detection.

  There are two main distinctions that have been most prevalent in recent training sets such as VOC and COCO.

  - Number of classes across the dataset
  - How iconic the objects in images are captured

  #### Iconic

These images have the object front and center. Some examples of these include product shots found on Amazon, or portraits dedicated to a person or thing. These are also the easiest to annotate, since there is usually just one object.

It is also these types of images that YOLO has the easiest time accurately detecting. There was one case where an iconic image failed to classify toy cars accurately, mistaking them for cell phones. This is most likely because of the training set used for YOLO to teach it what car features are significant.

Non-iconic

These have more objects in a frame. "More" has been loosely applied as having greater than three objects in an image, and these objects often are different classes from each other.

YOLO has some success detecting objects of this type, even with objects in the background, instead of front and center. However, with the use case of a parking lot, YOLO has had a hard time detecting cars further back in the recesses.

- **YOLO Training**

  This involves training on the latest version of the YOLO CNN architecture. It runs on a Pytorch implementation of the model and interfaces with the GPU via the CUDA framework API.

  A model is only as good as the data it is trained on. And because the model will morph during the research development process, it will need to run through training data whenever it mutates. Therefore, it is crucial to the continued progress of this research topic to set up training for this model.

  Training works off the same code that is used to detect objects in images. The neural net expression is in the same application. However, much more additional work is completed to compile training data into a format the neural net can consume. The CNN is trained on the VOC data set. It contains a mixture of iconic and non-iconic images. The non-iconic images do not contain nearly the same density of objects as a parking lot full of cars, however.

  Training successfully occurs on the GPU because of the relative ease of using CUDA with Pytorch. In fact, training on the GPU needs to happen. Otherwise, there will not be much progress towards improving the model given time constraints and the necessity to see results sooner rather than later during the research development process.

Proposed Schedule

Week 1: 1/27-2/2	Data formatting for CNN consumption
Week 2: 2/3-2/9	Data formatting for CNN consumption
Week 3: 2/10-2/16	Data formatting for CNN consumption
Week 4: 2/27-2/23	[Deliverable 1] Data formatting for CNN consumption
Week 5: 2/24-3/2	[Deliverable 2] Test CNN (new data)
Week 6: 3/3-3/9	Modify CNN and train
Week 7: 3/10-3/16	Modify CNN and train
Week 8: 3/17-3/23	[Deliverable 3] Test CNN (modified model architecture)
Week 9: 3/24-3/30	Preprocess images
Week 10: 3/31-4/6	Preprocess images
Week 11: 4/7-4/13	Preprocess images
Week 12: 4/14-4/20	Train
Week 13: 4/21-4/27	[Deliverable 4] Test CNN (modified ground truth)
Week 14: 4/28-5/4	Summarize project
Week 15: 5/5-5/11	Summarize project
Week 16: 5/12-5/18	[Deliverable 5] Summarize project. Present to committee

Key Deliverables:

- Software
  - Compilation scripts for different training data sets
  - Pytorch implementation of the deep learning model, as well as training and detection operations
  - CUDA integration with the GPU during training and detection

    - Open CV logic for image processing
- Report
    - Training examples impact on accuracy
    - Role of image processing in accuracy

Innovations and Challenges

- The most innovative and challenging aspect of this project is the detection of dozens of objects--cars--in a single image. Most advances in computer vision have been identifying objects in iconic images, where only a few objects are the subject of an image. There have been efforts for years to identify a greater number of objects in a frame, but these approaches have usually involved region detection and scoring, requiring additional time to iterate or slide across an image. Moreover, these approaches have shown only marginal improvements in accuracy.
- Another challenging aspect of this project is collecting accurate training data and running the model against multiple epochs. This will require collaboration with other researchers who have collected similar data sets, personal annotation of data sets, and modifications to the training data via image processing prior to the CNN training. In particular, data augmentation is a practice of the scientific hypothesis itself. Empiricism may be the only approach to determining the ideal representation of ground truth for the model to recognize key features to detect objects within such an object-dense image.
- The third innovate and challenging aspect of this project is how to incorporate image processing prior to CNN detection. One application of image processing is the multiplication of training examples from the same set of original images. Another application would be custom dimensionality reduction even before the actual convolution layers. Image processing has shown promise in previous studies, but the use case never included so many objects per image.

References:

[J. Redmon 2018] "YOLOv3: An Incremental Improvement." Redmon et al. arXiv. 2018.

[J. Redmon 2016] "YOLO9000: Better, Faster, Stronger." Redmon et al. arXiv. 2016.

[R. Yusnita 2012] "Intelligent Parking Space Detection System Based on Image Processing." R. Yusnita et al. International Journal of Innovation, Management and Technology. 2012.

[LeCun 1998] "Object Recognition with Gradient-Based Learning." LeCun et al. AT&T Shannon Lab. 1998.

[D. Bullock 1993] "A NEURAL NETWORK FOR IMAGE-BASED VEHICLE DETECTION." D. Bullock et al. Pergamon Press Ltd. 1993.