

Image Inference

Forrest Edwards

Abstract—Two models were trained on the NVIDIA DIGITS platform and are presented here. The first is a classification model built to identify object packaging type from images of products traveling down a conveyor. The package classifier was trained using the AlexNet network on 7570 images. The package classification model achieved an accuracy of 75.41% with average classification times of 4.55ms. The second network presented is a detection model trained to detect 15 distinct Legos. The second network was trained on 100 images of these Legos using a DetectNet network modified for multiple classes. The detection network achieved 75.2% overall accuracy.

Index Terms—Robot, IEEEtran, Udacity, L^AT_EX, deep learning.

1 INTRODUCTION

Overcoming the challenges of using machines to identify objects has historically required some form of compromise. This compromise often consists of augmenting the object to be more readily identifiable or by simplifying the perceived environment to reduce the complexity of the task. To make objects more readily identifiable common solutions include augmenting the item with a tag such as a barcode, RFID, or magnetic text. Alternatively in applications such as PCB manufacturing the recognition task presented to these systems are highly controlled, defined and static such as detecting the presence of a component on an assembled circuit board. If a system could identify objects in images without the addition of an external tagging system, or in less structured environments it would have an advantage over these limited predecessors. For example, if a machine vision system could identify a product without application of a RFID tag the cost of those tags could be eliminated. In the case of a barcode the packaging engineers and artists could reuse the occupied space to display additional consumer facing content. In highly irregular environments, such as trash moving through a recycling plant a system that could segregate valuable recyclables from trash could increase the yield of recyclables from waste streams.

The two cases that follow are intended to demonstrate the feasibility of applying neural networks and machine learning techniques to classify the objects present in a series of images. In both cases this will be achieved without specialized imaging equipment, enforcing a overly constrained environment, or leveraging an external machine readable tagging mechanism. While some simplification of the data was necessary to manage the effort associated with the collection and annotation of the images, the cases presented provide a foundation for addressing the types of challenges mentioned previously.

2 BACKGROUND / FORMULATION

2.1 Network Model: Packaging Images

Two sets of images were used for building the classification models presented. The first set was provided by Udacity as

part of the project and consists of images of consumer product goods (CPG) taken while transported on a conveyor.

Both AlexNet and GoogLeNet were used to train the packaging classification model. AlexNet is an 8 level ConvNet consisting of 5 convolutional layers and 3 fully connected layers [1]. When it was introduced in 2012, it represented a step change in reducing error rates for the task of image classification [2]. It's performance has since been surpassed by other networks, but it represents a relatively simple network that is capable of achieving the required results of 75% accuracy. The parameters for training the AlexNet model are shown in Table 1.

TABLE 1
AlexNet Parameters

Parameter	Run 1	Run 2	Run 3
Epochs	10	10	50
Image Size	500x500	256x256	256x256
Solver	ADAM	ADAM	ADAM
Learning Rate	1e-4	1e-4	1e-4
Subtr. Mean	None	None	None
Solver Policy	Exp. Decay	Exp. Decay	Exp. Decay

GoogLeNet was also used to model the dataset because it is capable of delivering higher accuracy [3] than AlexNet but at 22 layers(over 100 when counting individual layers in the inception layers) [4] it is much more complex. GoogLeNet parameters are shown below in Table 2.

TABLE 2
GoogleNet Parameters

Parameter	Run 1
Epochs	10
Solver	ADAM
Learning Rate	1e-4
Subtr. Mean	None
Solver Policy	Exp. Decay

ADAM was chosen for the solver in all cases due faster convergence, lower CPU cost and adaptive learning rate that scales for the different layers of the network [5].

2.2 Network Model: Lego Images

The second image set was left at the discretion of the author and consists of images of 16 randomly arranged Lego pieces. The Lego pieces were readily available and offered distinct shapes and colors that are easily annotated. For the Lego dataset a modified Detectnet was used to train a detection model.

Detectnet was leveraged for several reasons. The primary reason for choosing it was that it can detect multiple classes and output bounding boxes to indicate where a detected object class is located. This is more practically applicable to robotic applications. Secondly, since DetectNet is based on GoogLeNet [6] a pre-trained GoogLeNet model can be used to seed the training weights. This reduces the number of epochs required to obtain a functioning model by starting with weights that are initialized from previous training data. Since CovNets are at their core feature detectors these starting weights, although not optimized for the Lego use case, have a lower initial cross-entropy than weights that are initialized completely at random. For the Lego dataset the pretrained model `bvlc_googlenet.caffemodel` [7] was used to initialize network weights at the start of training. As in the classification model, ADAM was again used as the solver due to the advantages mentioned previously. The starting point for the DetectNet network was the prototxt model found in the caffe example code [7] for a 2 class detector. To adapt this model for application to the 15 class Lego detector the model was modified as follows [8]:

- The 15 custom classes and their labels were added to the `train_transform`, `val_transform`, `python cluster`, and `cluster_gt` layer as separate objects
- 15 additional Score class / mAP class layers were added to output the scores for each object class
- Parameters for the input, clustering and mAP layers were updated to reflect the correct image dimensions of 1024x1344

Once the modifications were complete, the DetectNet model was trained in Digits with the parameters show in Table 3.

TABLE 3
Lego Network Training Parameters

Parameter	Value
Pad Image	2688x2048
Resize Image	1344x1024
Custom Classes	15
Epochs	660
Snapshot Interval	20
Validation Interval	20
Random Seed	None
Batch Size	3
Batch Accum.	5
Solver	ADAM
Base Learning Rate	.0001
Subtract Mean	None
Crop Size	None
Policy	Exponential Decay
Gamma	0.99

3 DATA ACQUISITION

3.1 Provided Dataset: Packaging Images

3.1.1 Image Capture

Acquisition of source images for the provided packaging dataset was semi-automated by mounting the Jetson TX2 above a conveyor as shown in Figure 1. The stated goal being to train a model to perform real-time sorting of three classes: (Bottle, Candy box, Nothing). A total of 10,094 images were included in the dataset. The distribution of images by class as well as the relative percentage of the set used for training vs. validation is shown in Table 4

TABLE 4
Packaging Classification Data Set

Parameter	Value
Bottle Images	4568
Candy Box Images	2495
Empty Conveyor	3031
Used for Training	75%
Used for Validation	25%
Testing Set Size	60

TABLE 5
Packaging Image Parameters

Parameter	Value
X-Resolution	500
Y-Resolution	500
Image format	PNG
Color / Gray	Color 24 bit

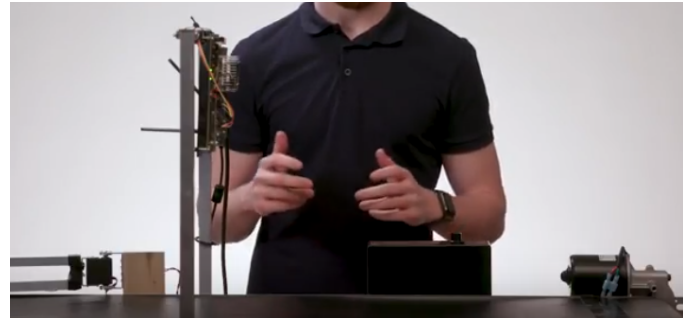


Fig. 1. Udacity conveyor for image capture.

3.1.2 Image Annotation

Images for the classification model did not require detailed annotation other than ensuring they were segregated into groups based on their class. Images were provided in the directory hierarchy expected by Digits which is shown in Figure 2.

3.2 Custom Dataset: Legos

3.2.1 Image Capture

Source images the custom dataset were acquired using the on-board camera of the Jetson TX2 developer board. Due to challenges encountered executing the supplied Python and

```

P1_data/
├── Bottle/
│   ├── Bottle_1.png
│   └── Bottle_2.png
├── Candy_box/
│   ├── Candy_box_1.png
│   └── Candy_box_2.png
└── Nothing/
    ├── Nothing_1.png
    └── Nothing_2.png

```

Fig. 2. Image classification data folder structure.

C++ capture scripts the argus_camera application was used as the image capture application. Images were captured with the settings shown in Table 6.

TABLE 6
Lego Image Aquisition Parameters

Parameter	Value
X-Resolution	2592
Y-Resolution	1944
Image format	JPEG
Stabilization	off
De-Noise mode	fast
De-Noise strength	-1.00
Edge Enhance mode	fast
Edge Enhance strength	-1
AE antibanding mode	off
Auto White Balancing mode	off

To capture the images a rudimentary studio was constructed to capture the images as shown in Figures 3, 4 and 5. A total of 200 images were captured using the following process.

- 1) Legos stored in a glass bowl were dumped out on the stage, consisting of an 8.5" x 11" sheet of paper.
- 2) Legos were minimally arranged so that they did not occlude one another, and fit within the frame of the camera.
- 3) An image was captured through the argus_camera application.
- 4) The paper stage was then rotated 180 degrees and a second image was captured.
- 5) The Legos were returned to the bowl to start the process over again.

Acquiring the images in this fashion increased the rate at which images could be taken since two images were produced with each cycle. While similar augmentation steps could have been taken post-capture, as is done in the DetectNet augmentation layer, this manual augmentation has the advantage of being able to introduce additional lighting and shadow variations to the features. In Figure 6 one can see the additional variation in augmentation this process provides over a purely digital flip or rotation applied to the image.

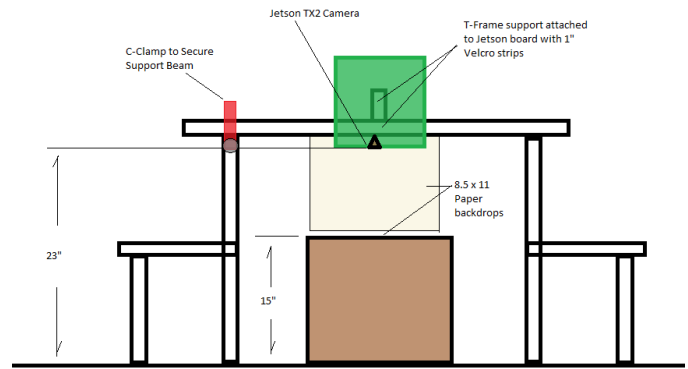


Fig. 3. Lego studio diagram.



Fig. 4. Lego studio.

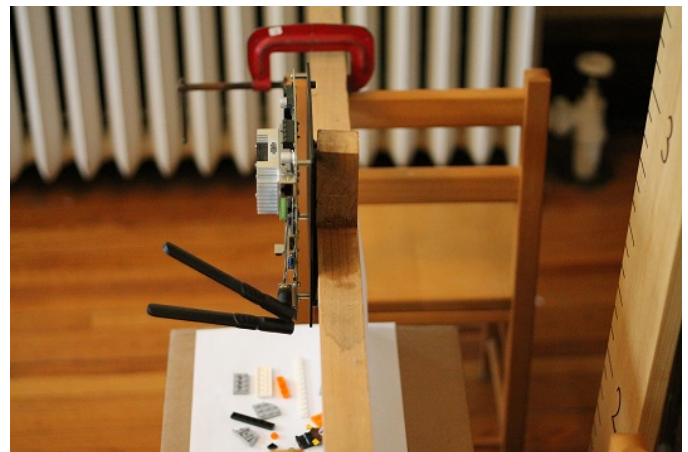


Fig. 5. Lego studio detail



Fig. 6. Physical data augmentation through stage rotation.

3.2.2 Image Annotation

Image annotation was performed using the LabelMe [9] tool (<http://labelme2.csail.mit.edu>). Of the 200 images acquired 125 were annotated. Boundaries were defined as polygons outlining the contours of each individual Lego. This resulted in a total of 2000 polygons to capture all 16 objects present in each image. Figure 7 shows a sample completed image annotation using the LabelMe tool. Bounding boxes were also an available, and more expedient, option but since LabelMe collections may be used for future research by others, it was decided that polygons would provide higher quality annotation data for any future consumer of this image set.

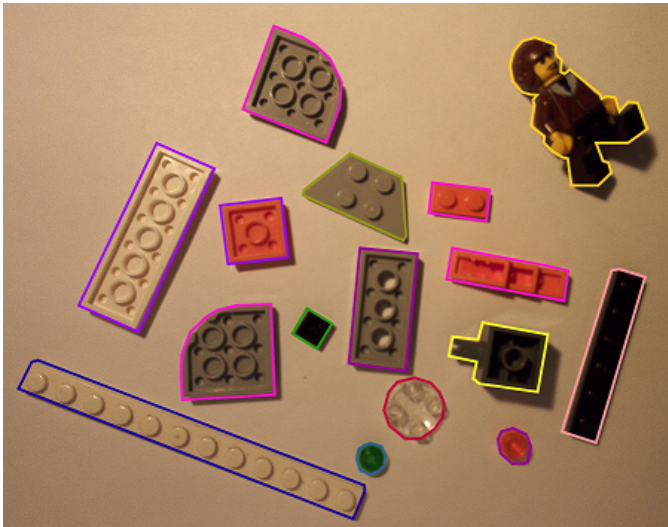


Fig. 7. Sample of LabelMe annotated image.

The generated LabelMe annotations are not immediately ready for ingestion by the Digits workspace. LabelMe annotation files are XML based and consist of a nested list of annotated objects each with multiple X-Y pairs that identify the bounding polygon for each object. Digits uses the KITTI format for annotation. To address this a Python script (<https://github.com/froohoo/poly2BB>) was created to convert LabelMe XML files to the KITTI formatted files expected by the Digits workspace.

4 RESULTS

4.1 Packaging Classification Model

The trained packaging classification model were successful in that it met the minimum goal of 75%. Accuracy reported by the evaluation script was 75.4% for all models. These identical results for all 4 models despite variations in the training image size, epochs, and network was unexpected. Classification speed was met for all models as well vs. the goal of 10ms. Forward propagation for all 4 models were found to be in the 4 - 6ms range. Its also notable that all 4 models converged on 100% validation accuracy after 5 epochs. Summary of accuracy and training times are shown in Table 7

TABLE 7
Packaging Classification Model Results

Model	Training Time	Accuracy	Classification Time
AlexNet-1	0H 6m 36s	75.41%	4.43284ms
AlexNet-2	0H 5m 31s	75.41%	4.24133ms
AlexNet-3	0H 5m 11s	75.41%	4.55820ms
GoogLeNet-1	0H 17m 25s	75.41%	5.29794ms

4.2 Lego Detection Model

The Lego detection model was able to successfully identify 9 of the 15 classes at varying rates depending on the class. Total training time for the model was approximately 8 hours. No tool was provided for evaluating the user generated data sets so instead the 'Evaluate one image' functionality in Digits was leveraged to perform a small scale validation. The test set was composed of 11 images that were not part of the training or validation set. Table 8 shows the overall detection results as well as details for each individual class in the test set.

5 DISCUSSION

5.1 Packaging Classification Model

Several intriguing results presented themselves after evaluating the packaging detection models. One unexpected result was the model convergence for all models occurs after only 5 epochs. This is likely due to the large set of source images. Were this a smaller dataset, more epochs would have been needed to pass the same number of images through the network. Another somewhat surprising result of the packaging model training was the identical accuracy results for all the models despite variation in image size, epochs, and training network chosen. One possible explanation is that this is related to the methodology and test images used by the evaluation script as it is the only obvious commonality. It would be a worthwhile future endeavor to understand the evaluate scripts results in more detail and understand the source of this anomaly. The difference in training times between the AlexNet trained model and GoogLeNet model were also shown to be significant with the GoogLeNet model taking 3x longer to train than the AlexNet. This is attributable to 3x as many layers present in the GoogLeNet model when compared to AlexNet.

While the 75.4% accuracy met the project goal, it would not be acceptable as the sole solution for identifying products in this application. Laser line, linear CCD, RFID and 2D CCD decoder systems are all capable of over 99% read rates of 1D and 2D barcode symbologies and would be better choices. However, since these systems can not decode barcodes that are damaged, missing, or facing away from the reading device these inference models could provide an excellent secondary detection system with the potential to reduce error rates by 75% of the overall combined system. With respect to speed, the model was more than fast enough to classify in excess of 100 images per second, which is much faster than items are typically moved loosely on conveyor belts (typ. 2-5 per second).

5.2 Lego Detection Model

Overall, the results provided by the multi-class lego detection model were acceptable at 75.2%. The detailed results indicate however that the model average can not be extrapolated to the individual classes. With one exception, the model accuracy for the individual Lego's was either 100% or 0%. This is likely a result of both under-training the model and having a relatively small annotated data set. As can be seen in Figure 8 as the model trains it learns different classes at different stages. This behavior would likely be somewhat arbitrary, but the use of a pretrained model ensures that at initialization the weights are pre-positioned to learn some classes faster than others. For the poor accuracy classes in the 50-300px size range more images or iterations would likely improve these results. For the larger Legos some modifications to the filter size, stride and clustering parameters may also be required for optimal detection as one of the test images shows a large Lego with 3 bounding boxes overlayed; an indication the model has detected the class in 3 separate but overlapping regions, when only one large Lego was present.

6 CONCLUSION / FUTURE WORK

The two trained models presented here both achieved accuracies of 75% successfully and demonstrate the feasibility of applying CovNet's to the task of image classification and object detection. While not commercially viable as stand-alone solutions, additional tuning of hyperparameters and further training on a GPU would lead to increases in accuracy. It would also be worth pursuing training these models for a less ideal environment with more background noise and evaluating their performance vs. the relatively clean/sterile backgrounds of these two data sets.

REFERENCES

- [1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, pp. 84–90, May 2017.
- [2] H. Gao, "A Walk-through of AlexNet," Aug. 2017.
- [3] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," pp. 1–9, IEEE, June 2015.
- [4] A. Deshpande, "The 9 Deep Learning Papers You Need To Know About (Understanding CNNs Part 3)."
- [5] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv:1412.6980 [cs]*, Dec. 2014. arXiv: 1412.6980.
- [6] "DetectNet: Deep Neural Network for Object Detection in DIGITS," Aug. 2016.
- [7] "caffe: Caffe: a fast open framework for deep learning," June 2018. original-date: 2013-09-12T18:39:48Z.
- [8] M. Tomislav, "Training a Custom Multiclass Object Detection Model Using Nvidia Digits."
- [9] B. C. Russell, A. Torralba, K. P. Murphy, and W. T. Freeman, "LabelMe: A Database and Web-Based Tool for Image Annotation," *International Journal of Computer Vision*, vol. 77, pp. 157–173, May 2008.

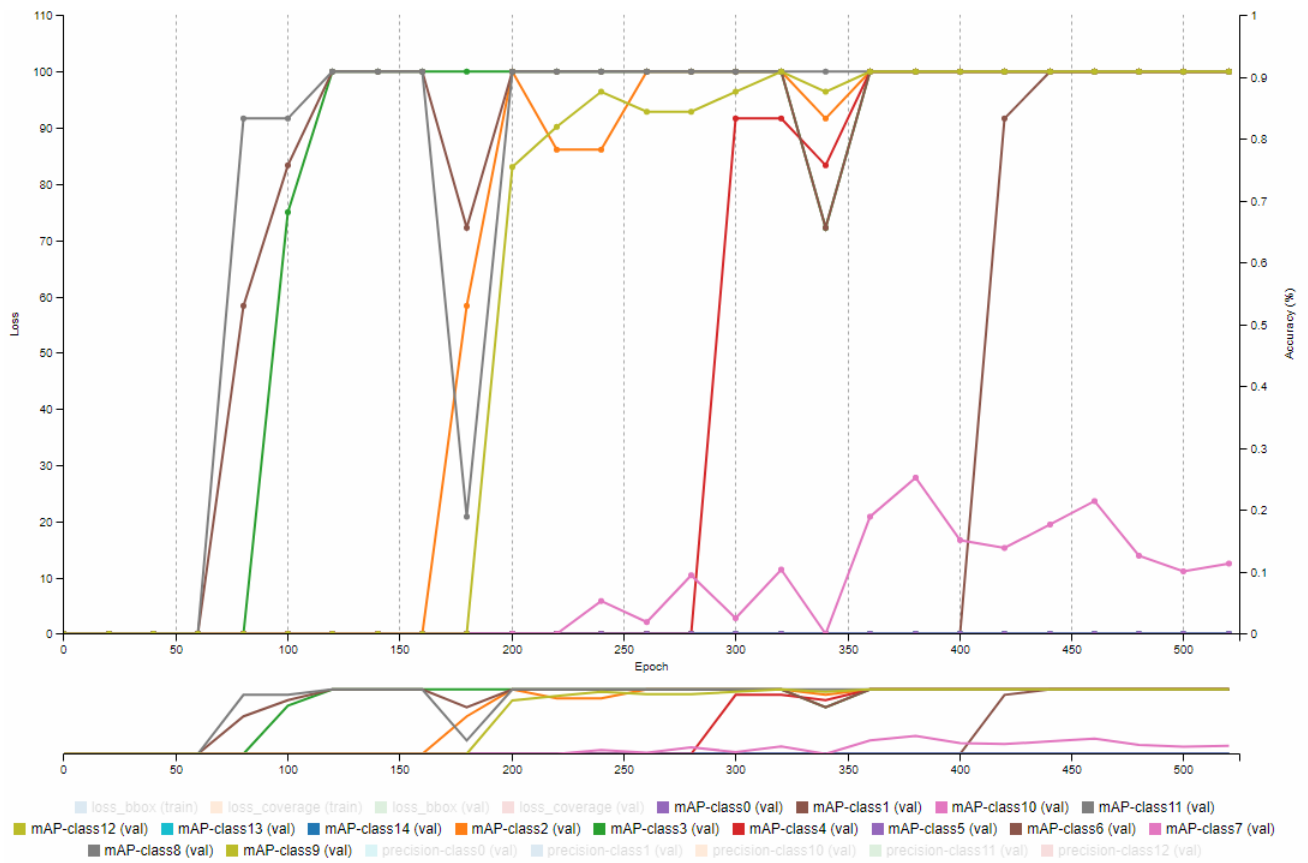


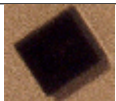











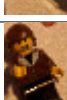


Fig. 8. Iteration vs. mAP for detected classes

TABLE 8
Lego Detection Model Results

Class	True Pos.	False Pos.	Miss	Accuracy
All Classes	101	0	75	75.2%
	11	0	0	100%
	11	0	0	100%
	11	0	0	100%
	11	0	0	100%
	11	0	0	100%
	0	0	11	0%
	11	0	0	100%
	2	0	9	18.2%
	11	0	0	100%
	22	0	0	100%
	0	0	11	0%
	0	0	11	0%
	0	0	11	0%
	0	0	11	0%
	0	0	11	0%