Deep Convolutional Network for Facial Recognition with Security Application

Adam J. Carter

Abstract—A Deep Convolutional Network is presented for a facial recognition security application. The system is a version of GoogLeNet trained on four new classes of objects. Accuracy on self-gathered test data approached 90 percent. Factors influencing training, inference accuracy and inference rate are discussed.

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

1 Introduction

OBJECT detection and recognition has been a focus of machine learning algorithms since their inception. From the initial efforts of Yan LeCun to automatically read digits on pieces of mail, we gained LeNet. Further research at Google led to GoogLeNet, upon which this network is based. The objective of the project is to produce a control mechanism for an electronic deadbolt lock on the front door of a family residence. The control mechanism relies on facial recognition. One group of people were trained in as 'family' while other people (not trained in to the system) were used as 'decoys' to attempt to fool the system. Success is defined as detection of each individual (vs no person present) and categorization (proper name label applied).

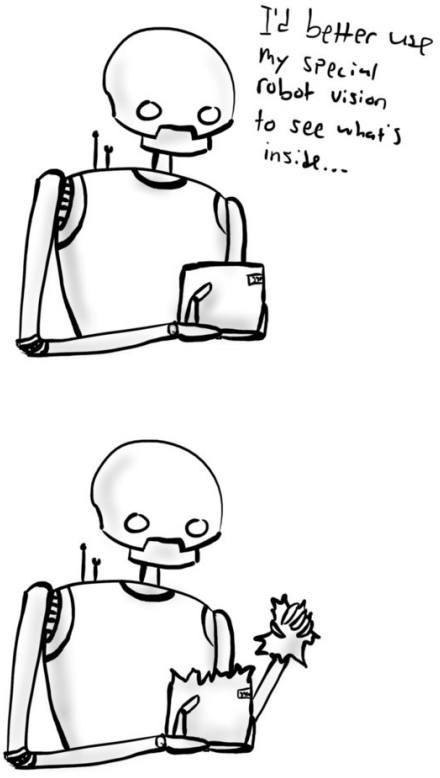

Fig. 1. Qille Deviant Art

2 Background / Formulation

The objective of this project is to train a neural network to identify people, with a security application as one possible use-case. Keeping the system hardware lightweight and affordable was an early objective. If any inference system is to be put to use in the real world and is meant to be cost-effective, then the computational complexity of the system becomes a central factor governing it's deployment.

Concerning computational complexity, all neural networks have one thing in common: the use of matrix multiplies to move information forward and back through the system. The total number of these multiplies or 'parameters' affects both training time and inference time. With this in mind, the value of a system can be assessed by reference to not only its classification accuracy but also inference time. Comparing two systems with equal classification accuracy for a given task, the 'winner' will be the one with fewer parameters.

The basic design of most deep-learning systems for computer vision are similar: one or more convolutional layers followed by max pooling or other interpolation technique before one or more fully-connected layers (or one-by-one convolutions). The basic approach to improving these systems has been to go deeper and wider with them. Although this approach does usually lead to gains in performance, it has drawbacks. Any change in depth comes with the penalty of adding parameters. In addition it increases the system's potential for over-fit which must be addressed with dropout or other parameter reducing techniques.

The 'Inception' architecture established by Google and used here in a version of GoogLeNet is designed specifically to reduce computational complexity through parameter reduction. GoogLeNet is a 22 layer deep model with a unique approach to convolutional filter sizes. To summarize, a variety of filter sizes are used to convolve over the same input data prior to concatenation of the resulting vector. This allows for a mixture of fine granular detail and large scale spatial relationships between the pixels to be gained. The

other and possibly more important trait conferred by 1x1 convolutions is that they can be used as a dimensionality reducing agent. This allows an engineer to selectively reduce parameters in the network and keep it lighter weight while expanding width and depth for greater inference capability. The high success rate of this architecture in a variety of use-cases is why it was chosen for this project.

3 Data Acquisition

Data collection for this task was done using a Logitech C270 webcam. Two datasets were gathered and used separately to train the network. The first consisted of 500 images in each class, the second of 1500 images per class and included minor data augmentation. A 'class' here means an individual person. The native resolution and output of the web-cam is RGB 720p with a 1 mega-pixel sensor. The images were reduced to 256x256 RGB in the python data gathering script. This size natively fits the expected input size of the GoogLeNet architecture. Matching input data size to anticipated network data size is a helpful and usually simple way to improve network training.

Since the planned security application would involve cameras in static positions, training data was gathered by placing subjects in front of a static background. The ability of the system to differentiate similar looking humans comes down to it's ability to distinguish uniquely identifying characteristics of each subject. Similar items of clothing would be anticipated to cause confusion. Different hairstyles, skin tone/facial structure and overall body shape were anticipated to be good differentiators.

In order to produce robust representations of each subject it is best to present the system with examples of those subjects in a wide variety of states. Changes of clothing and physical poses were used to increase variety of representation and force the system to learn to ignore variable traits such as clothing while memorizing persistent traits such as facial structure and body appearance. Translational invariance (ability of the network to identify an object regardless of its position in the frame) is conferred by the convolutional step.

4 Results

Network performance was above 85 percent in each class. Inference time was subjectively reasonable. The following charts detail the training and validation curves and the confusion matrix shows performance on test images that were not included in either training or validation sets.

5 Discussion

The situations in which network performance was lower than anticipated provide the most interesting points for discussion. In the initial rendition of the training data (500 images per class) the worst performance was on Brady who's physical appearance is somewhat similar to his older brother Kieran's. When the network did mis-classify Brady, it most often did so by classifying him as Kieran. The differences in performance between the two data sets is remarkable. The second data set (1500 images per class) had stronger performance across all classes as anticipated.

All classifications

© Adam J. Carte

	Path	Top predictions							
1	/home/workspace/test_images/Adam_337.png	Adam	98.39%	Heath	1.59%	McKenzie	0.02%	no subject	0.0%
2	/home/workspace/test_images/Adam_467.png	Adam	97.91%	Heath	1.91%	McKenzie	0.18%	no subject	0.0%
3	/home/workspace/test_images/Adam_584.png	Adam	97.93%	Heath	1.88%	McKenzie	0.2%	no subject	0.0%
4	/home/workspace/test_images/Brady_218.png	Brady	52.85%	Kieran	47.03%	no subject	0.12%	McKenzie	0.0%
5	/home/workspace/test_images/Brady_316.png	Brady	83.7%	Kieran	15.58%	no subject	0.69%	McKenzie	0.02
6	/home/workspace/test_images/Brady_407.png	Kieran	63.09%	Brady	36.86%	no subject	0.05%	McKenzie	0.0%
7	/home/workspace/test_images/Heath_46.png	Heath	94.05%	Adam	5.6%	no subject	0.2%	McKenzie	0.15
8	/home/workspace/test_images/Heath_238.png	Heath	91.0%	Adam	8.98%	McKenzie	0.01%	no subject	0.0%
9	/home/workspace/test_images/Heath_504.png	Heath	94.95%	Adam	2.73%	no subject	2.22%	McKenzie	0.1%
10	/home/workspace/test_images/Kieran_225.png	Kieran	94.99%	Brady	5.0%	no subject	0.02%	McKenzie	0.0%
11	/home/workspace/test_images/Kieran_314.png	Kieran	94.06%	Brady	5.92%	no subject	0.02%	McKenzie	0.0%
12	/home/workspace/test_images/Kieran_52.png	Kieran	94.74%	Brady	5.25%	no subject	0.01%	McKenzie	0.0%
13	/home/workspace/test_images/McKenzie_70.png	McKenzie	85.43%	no subject	11.69%	Brady	1.45%	Heath	1.02
14	/home/workspace/test_images/McKenzie_193.png	McKenzie	80.0%	no subject	15.79%	Brady	2.76%	Heath	1.11
15	/home/workspace/test_images/McKenzie_296.png	McKenzie	79.88%	no subject	16.13%	Brady	2.41%	Heath	1.2%
16	/home/workspace/test_images/no_subject_83.png	no subject	78.6%	Brady	19.05%	Kieran	1.87%	Heath	0.3%
17	/home/workspace/test_images/no_subject_148.png	no subject	76.18%	Brady	21.08%	Kieran	2.29%	Heath	0.28
18	/home/workspace/test_images/no_subject_229.png	no subject	76.61%	Brady	20.77%	Kieran	2.17%	Heath	0.27

Fig. 2. Confusion Matrix: Performance on self-gathered data

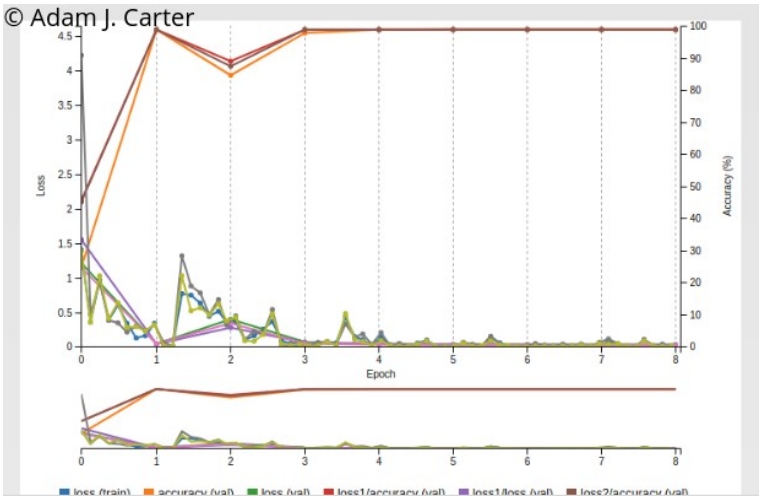

Fig. 3. P1 Data Training Curves

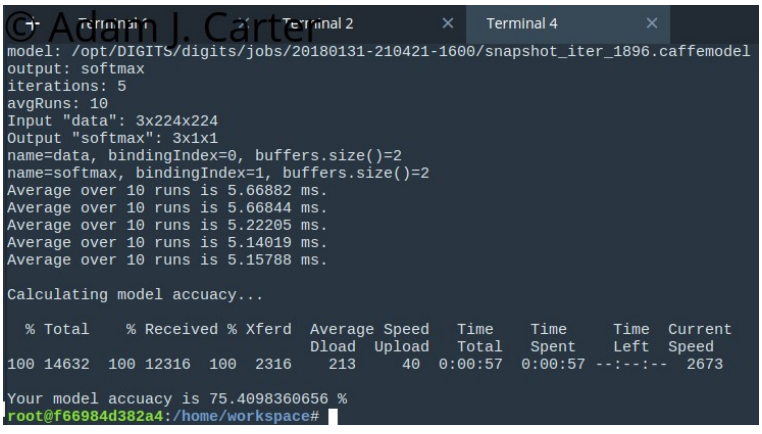

Fig. 4. Output of Evaluate Function

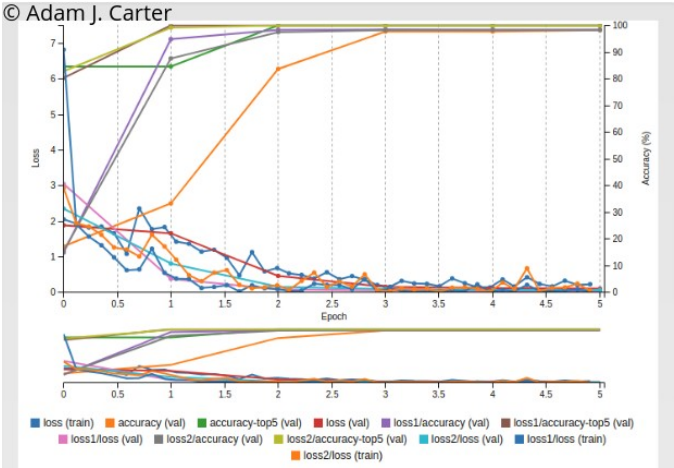

Fig. 5. Self Gathered Data Training Curves

For a security application inference time is not nearly as important as accuracy. It can certainly be argued that the consequences of a slow-to-act door release mechanism could be high in some situations. However, aside from that it seems that much worse outcomes would be expected if the slow mechanism made inference errors and let the wrong person in. In other words, if it failed at it's core security purpose. With this in mind, priority was placed on accuracy over inference time.

6 Conclusion / Future work

The preliminary results from this experiment would suggest that with further training on a larger and more diverse data set and possibly some tweaks to the algorithm, this project could be effective as an aid to a human-monitored security apparatus. That comes with some strong qualifications. The stated final accuracy of the system should be transparently kept in mind when assessing the overall efficacy of the system. A recognition system that scores 98 percent on test data will be expected to be wrong at least 2 percent of the time in the real world, if not significantly more. If that is an unacceptably high error rate for the specific application the end-user has in mind then this approach would fail. For a less strict and possibly human-monitored environment it may be highly effective as a 'security aid'. The take-home message is that after building and testing this project, both myself and my family did not feel comfortable relying on a security device that 'just might' let a stranger in the front door in the middle of the night. A person could always disable the recognition system at night, making the deadbolt into a 'normal' manual lock and then use the recognition system only as a novelty at other times. The project did provide a fascinating exploration of a deployable machine learning application. Methods to improve on this project include but are not limited to the following:

1) **Data Augmentation**.
Data augmentation involves transmogrifying the existing data in a way that challenges the network's representation of the object to make it more robust. Examples include 90 degree rotation (or other angular values) of the training inputs, shearing, stretching, color and contrast manipulations, and addition of noise to the data. Data augmentation could be applied to further enhance training data diversity.

2) **Additional Data Gathering**.
More data is generally a good thing when it comes to training neural networks. Improvement was observed when moving from the 500 image-per-class dataset to the 1500 image/class set.

3) **Physical deployment of a locking mechanism**.
Due to time and financial constraints, a working model of the proposed dead-bolt system was not achieved. Given more time and money, one could be attempted. The following are links to resources used by the author to plan this deployment.

http://www.jpuderer.net/2015/08/reverse-engineering-my-electronic-door.html

http://www.somersetrecon.com/blog/2016/6/7/electronic-safe-lock-analysis-part-1-teardown

7 Inference Development

In the process of building this model the author had the opportunity to utilize a workflow built by Nvidia to expedite use of pre-trained neural networks for deployment. As a demonstration of these principles a network was trained up to 75 percent accuracy with an inference time of around 5 ms on a set of training data consisting of 256x256 RGB color images of food containers. An unmodified version of GoogLeNet was used for this purpose. In doing so, a number of valuable lessons were learned.

1) **Choose the right network for the job.**
Using LeNet as an example, it was demonstrated that matching input data type to the network's anticipated data type is helpful for training. LeNet takes greyscale images of size 28x28 pixels. If you feed it color images that are 256x256 it will perform more poorly than if you give it the proper anticipated input. A machine learning engineer must treat network outputs similarly, matching anticipated output of the network to the end-user's desired output. In general, there can either be a single output node (the value of which corresponds to a predicted class), or an equal number of output nodes and classes. In the latter case the output node with the highest activation represents the predicted class. In either case, this information will need to be translated into a useful paradigm for the end user based on their preferences.

2) **Choose or generate the right data (and data amount) for the job.**
One approach to improving prediction accuracy is to train for more iterations. More epochs can be run on the same training data to get marginal improvements in accuracy. There is a limit to the effectiveness of this approach however and additional training data is often needed. A good engineer will be able to inspect network

results, find areas of weakness, correlate that to the data set and find ways to supplement the data to improve training. Data augmentation, as discussed above, is also useful for this purpose.

3) **Modify the network architecture.**

In situations where additional or more varied data is not available or helpful, the network architecture itself can be modified in an attempt to improve learning. The specific ways in which a network can be modified are as numerous as the individual parts of the network. Points of manipulation include but are not limited to the following:

Number of layers (depth), number of neurons per layer (width), method of weight and bias initialization, activation function choice, method of gradient descent used during backpropagation (and tools such as momentum), use of dropout during training, use of 1x1 convolutions in CNN's etc. Currently we do not have a reliable method of predicting the size a network needs to be to solve a given problem. Generally proper fit is found by paring down an overly large network until overfit is reduced and learning is optimized.

4) **Balance the accuracy/inference time trade-off.**

Currently the round trip from a robot to the cloud and back is too slow for most practical robotic applications. The processing required to utilize a neural network in robotics is best done 'at the edge' or at the robot's location in other words. This means local hardware like the TX2 or a cellphone will be used to deploy most applications, making computational complexity (and thus power consumption esp on batteries) a limiting factor. To investigate this, Canziani et al compared a number of networks with regard to their accuracy, operations count, parameters count, inference time, memory usage and power consumption. Results provided the following insights (taken directly from the Udacity curriculum):

- **Power consumption is independent of batch size and architecture.** 'When full resources utilisation is reached, generally with larger batch sizes, all networks consume roughly an additional 11.8W.'
- **Accuracy and inference time are in a hyperbolic relationship.** 'A little increment in accuracy costs a lot of computational time.'
- **Energy constraint is an upper bound on the maximum achievable accuracy and model complexity.** 'If energy consumption is one of our concerns, for example for battery-powered devices, one can simply choose the slowest architecture which satisfies the application minimum requirements.'
- **The number of operations is a reliable estimate of the inference time.**

There are also options available in the event that network architecture changes are not leading to desired accuracy improvements (from the Udacity curriculum):

- Redeploy to a higher performance hardware platform with improved operation execution time
- Upgrade the software platform with an optimized release that affects inference time
- Increase the accuracies by customizing the data to a smaller number of relevant classifications

[1]

REFERENCES

[1] L. Lamport, *LATEX: a document preparation system: user's guide and reference manual*. Addison-wesley, 1994.

Going Deeper with Convolutions
Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich
Google Inc. University of North Carolina, Chapel Hill University of Michigan, Ann Arbor Magic Leap Inc.
szegedy,jiayq,sermanet,dragomir,dumitru,vanhoucke@google.com wliu@cs.unc.edu, reedscott@umich.edu, arabinovich@magicleap.com

An Analysis of Deep Neural Network Models for Practical Applications
Alfredo Canziani and Eugenio Culurciello
Weldon School of Biomedical Engineering Purdue University
canziani,euge@purdue.edu
Adam Paszke Faculty of Mathematics, Informatics and Mechanics University of Warsaw
a.paszke@students.mimuw.edu.pl

8 EXAMPLES OF SELF GATHERED DATA POINTS

Fig. 6. Adam

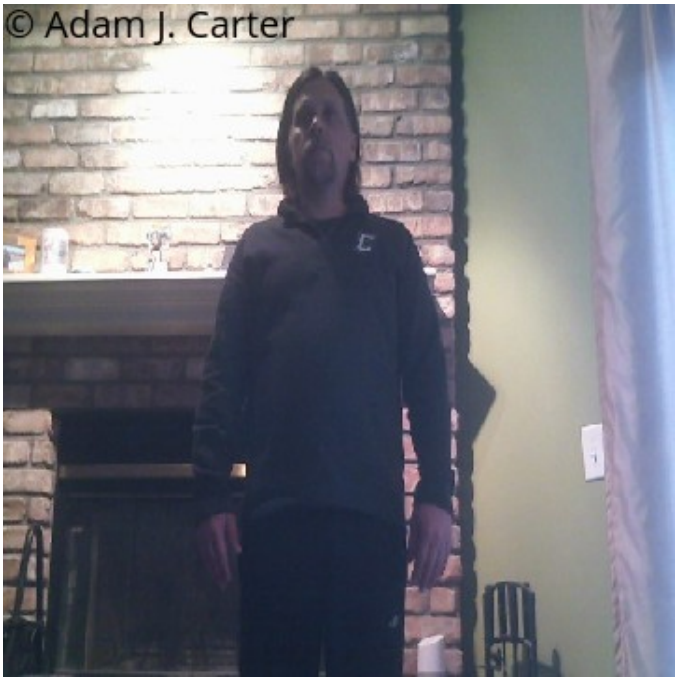

Fig. 7. Heath

Fig. 9. Brady

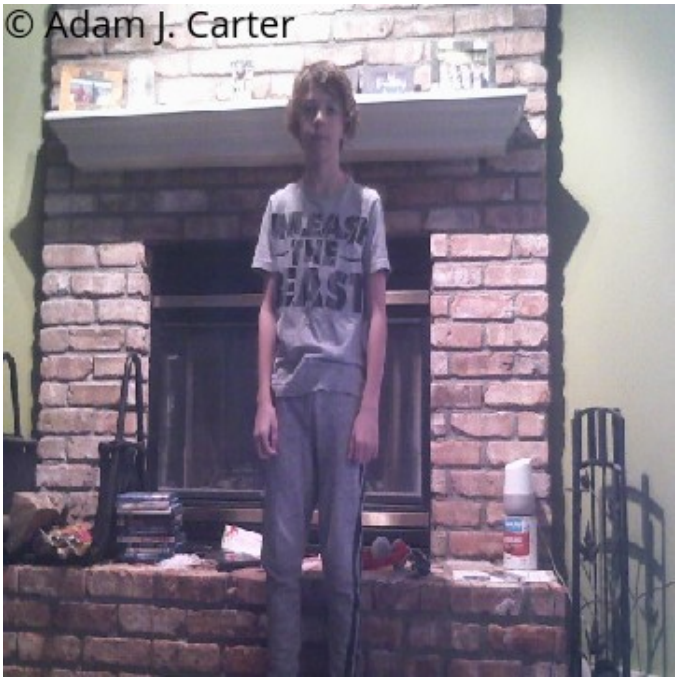

Fig. 8. Kieran