Cornell University

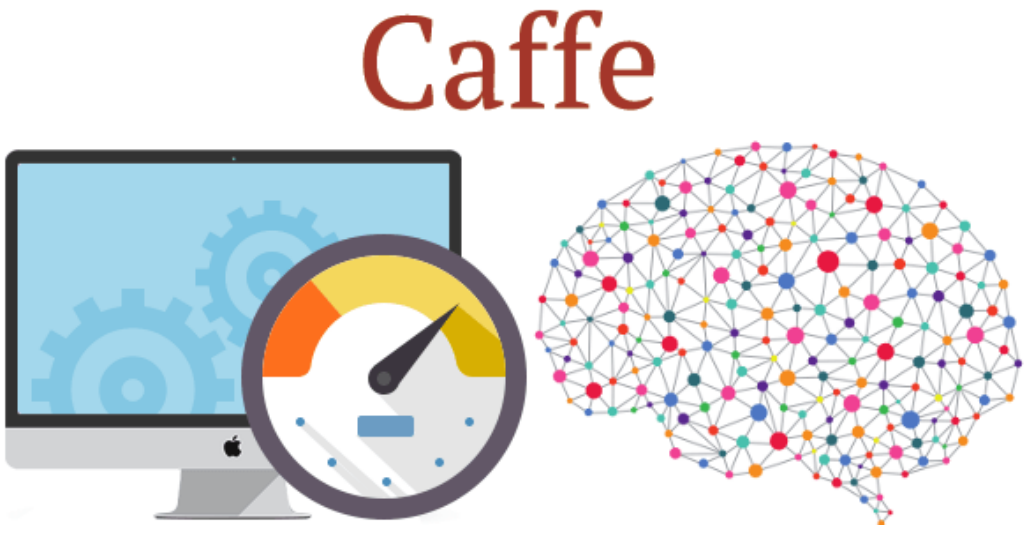
CS 4701: Practicum in Artificial Intelligence

Food Recognition with Deep Learning

Daryl Sew (dns55)
Cornell Computer Science '17
darylsew@gmail.com

Alice Zhang (gz74)
Cornell Computer Science '17
gz74@cornell.edu

Instructor, Bart Selman (bs54)



December 13, 2016

1 Problem Description

Consider the problem of creating an artificially intelligent grocery shopper who is able to make recipe substitutions and suggestions based on availability. Such an agent would be highly desirable to anyone who would rather do something else with their time than shop for groceries, and the different intelligent components that go into this agent would have widespread applications in other fields.

Recent advances in machine learning algorithms and computing capabilities have enabled deep neural network architectures to surpass the state of the art traditional machine learning classifiers in nearly every problem domain. In particular, Convolutional Neural Networks have surpassed traditional computer vision and machine learning techniques in many tasks, provided with large amounts of data.

Inspired in part by the textbook grocery shopper example and in part by IBM's work in recipe recommendation with Watson, we decided to build the perception system for food recognition, using deep learning methods in order to achieve the best possible performance. The ability to recognize a food item in an image has countless applications; for example, it is useful in building cooking robots, creating an artificially intelligent grocery shopper, automatically tracking shopper actions (i.e. Amazon Go), diet tracking programs, and many more.

2 High Level Design

2.1 Approach

Our first attempt to train our network on resulted in very poor ($< 5\%$) accuracy, so we decided to break the problem down into more manageable chunks.

We picked K Nearest Neighbors and Support Vector Machines as baseline classifiers so we would be able to meaningfully evaluate the Deep Convolutional Neural Network's performance. If we are using CNNs effectively, we should see much better results compared to the baseline. We used the LeNet network, as it is one of the most widely studied and used network architectures in the field, so we knew it would be fairly easy to work with.

We started with using the baseline classifiers to classify several types of hand drawn shapes with no feature extraction, mimicking the MNIST dataset. We simply used the grayscale pixel values of an image as the feature vector. We then trained the CNN on these, evaluated the performance, and compared the results.

After this, we tried using manual feature extraction via corner detection to improve the performance of KNN and SVMs, and tuning network parameters to improve the performance of CNNs.

Once we had completed these sanity checks, we moved on to testing with real datasets. We picked a dataset.

2.2 Software Architecture

2.2.1 `potato.py`

Loads model weights for a convolutional neural network and feeds forward test inputs through the network, evaluating its performance. The network we used was the off the

shelf LeNet.

2.2.2 `knn.py`

Evaluates the K Nearest Neighbors classifier on test inputs. Experiments with thresholding and corner detection as feature extractors. We chose to implement this ourselves as we thought it would be a valuable exercise. However we did not implement other classifiers ourselves due to time constraints.

2.2.3 `svm.py`

Evaluates the Support Vector Machine classifier on test inputs. Uses the Scikit-Learn multiclass SVM with a Radial Basis Function kernel, which allows the SVM to make nonlinear decision boundaries and thus classify data that is not linearly separable.

2.2.4 `train_test_split.py`

Splits data into train and test data, using a library function to convert it to the binary format that LeNet expects.

2.3 Data Collection Process

TODO

2.4 Theory

2.4.1 K Nearest Neighbors

how does it work

2.4.2 Support Vector Machine

how does it work

2.4.3 Convolutional Neural Network (LeNet)

how does it work

We created the following network visualization via Caffe's `draw_net.py`. TODO split up image

2.4.4 Cross Validation

talk about train test split

3 Evaluation

For evaluating performance, we decided to focus on multiclass precision-recall curves, as they are a fairly holistic method of evaluating classifier performance. The area under the curve is a simple metric to optimize, and the curve itself effectively characterizes classifier performance in different situations.

We are not in a position where false positives or false negatives have different desirabilities, either, so no modification or special analyses are required.

3.1 Hand Drawn Data

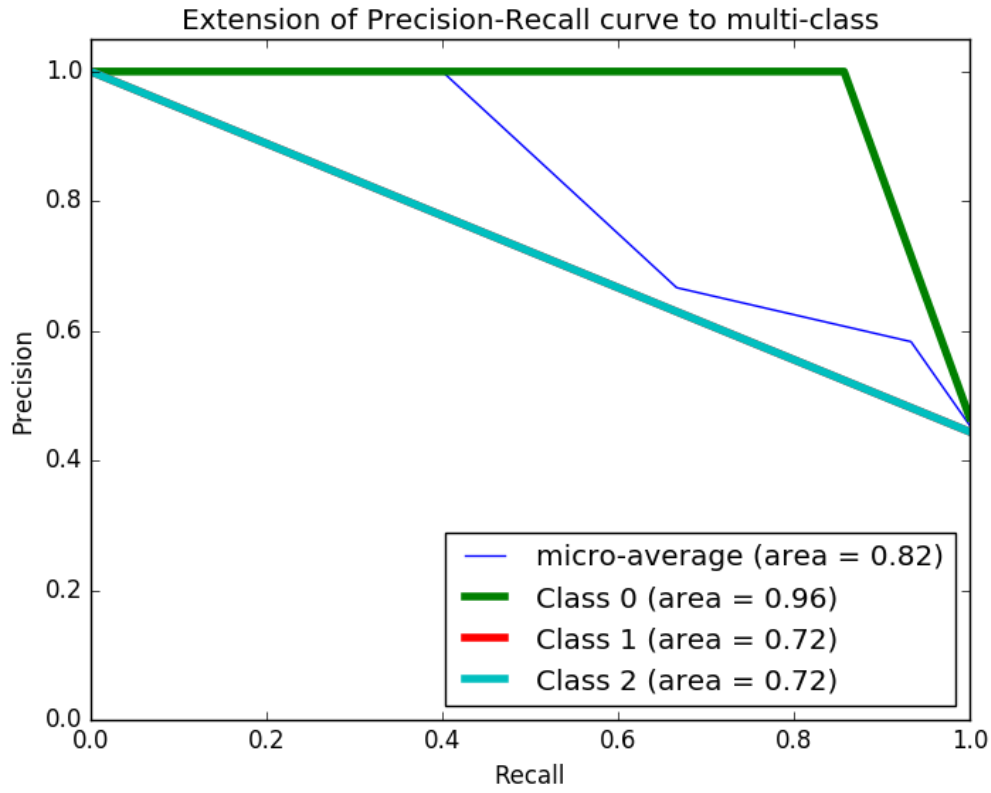
The following are sample images from the three classes of hand drawn images we evaluated. The network scales the inputs to 28x28, so we present the images after scaling below. Resizing was done via the UNIX `convert` command line tool.

The left image is a potato. The right image is a triangle. The lower image is a five point star.

3.1.1 K Nearest Neighbors

3.1.2 Support Vector Machine

(trained on 20 16x16 images; if time redo on 40 28x28 images. there seems to be a bug rn) classes correct: [4.0, 3.0, 0.0] total: [5, 3, 7] accuracy: [0.8 1. 0.]



3.1.3 Convolutional Neural Network

3.2 Food Dataset

3.2.1 K Nearest Neighbors

3.2.2 Support Vector Machine

3.2.3 Convolutional Neural Network

4 Conclusion

5 Future Work

6 Appendix

Here we present an annotated listing of all of the code we wrote for this project. The code is also available on GitHub.

7 Permissions

Our code is under the Apache license; see GitHub for more details.

If you found our work helpful in your research, we would appreciate it if you could consider listing us as authors. Feel free to contact us.

8 Credits

Thanks to the Cornell Graphics and Vision lab for donating a tiny amount of GPU power for us to train our network. We did not have access to Nvidia GPUs, so we would have had to spend orders of magnitude more time training on CPUs otherwise.

Thanks to gkielian for the script we used to convert our PNG images into the custom binary format the LeNet expects.

References

[1] Jia, Yangqing, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. *Caffe: Convolutional Architecture for Fast Feature Embedding.* arXiv preprint arXiv:1408.5093, 2014.

[2] Pedregosa, F. and Varoquaux, G. and Gramfort, A. and Michel, V. and Thirion, B. and Grisel, O. and Blondel, M. and Prettenhofer, P. and Weiss, R. and Dubourg, V. and Vanderplas, J. and Passos, A. and Cournapeau, D. and Brucher, M. and Perrot, M. and Duchesnay, E. *Scikit-learn: Machine Learning in Python.* Journal of Machine Learning Research 12.2825-2830, 2011.

[3] Y. LeCun, L. D. Jackel, B. Boser, J. S. Denker, H. P. Graf, I. Guyon, D. Henderson, R. E. Howard, and W. Hubbard. *Handwritten digit recognition: Applications of neural net chips and automatic learning.* IEEE Communication, pages 41-46, November 1989. invited paper.