Caffe: Convolutional Architecture for Fast Feature Embedding*

Yangqing Jia*, Evan Shelhamer*, Jeff Donahue, Sergey Karayev,
Jonathan Long, Ross Girshick, Sergio Guadarrama, Trevor Darrell
UC Berkeley EECS, Berkeley, CA 94702
{jiayq,shelhamer,jdonahue,sergeyk,jonlong,rbg,sguada,trevor}@eecs.berkeley.edu

ABSTRACT

Caffe provides multimedia scientists and practitioners with a clean and modifiable framework for state-of-the-art deep learning algorithms and a collection of reference models. The framework is a BSD-licensed C++ library with Python and MATLAB bindings for training and deploying general-purpose convolutional neural networks and other deep models efficiently on commodity architectures. Caffe fits industry and internet-scale media needs by CUDA GPU computation, processing over 40 million images a day on a single K40 or Titan GPU ($\approx$ 2.5 ms per image). By separating model representation from actual implementation, Caffe allows experimentation and seamless switching among platforms for ease of development and deployment from prototyping machines to cloud environments.

Caffe is maintained and developed by the Berkeley Vision and Learning Center (BVLC) with the help of an active community of contributors on GitHub. It powers ongoing research projects, large-scale industrial applications, and startup prototypes in vision, speech, and multimedia.

Categories and Subject Descriptors

I.5.1 [**Pattern Recognition**]: [Applications–Computer vision]; D.2.2 [**Software Engineering**]: [Design Tools and Techniques–Software libraries]; I.5.1 [**Pattern Recognition**]: [Models–Neural Nets]

General Terms

Algorithms, Design, Experimentation

Keywords

Open Source; Computer Vision; Neural Networks; Parallel Computation; Machine Learning

*Corresponding Authors. The work was done while Yangqing Jia was a graduate student at Berkeley. He is currently a research scientist at Google, 1600 Amphitheater Pkwy, Mountain View, CA 94043.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
MM'14, November 3–7, 2014, Orlando, Florida, USA.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-3063-3/14/11 ...$15.00.
http://dx.doi.org/10.1145/2647868.2654889.

1. INTRODUCTION

A key problem in multimedia data analysis is discovery of effective representations for sensory inputs—images, soundwaves, haptics, etc. While performance of conventional, handcrafted features has plateaued in recent years, new developments in deep compositional architectures have kept performance levels rising [8]. Deep models have outperformed hand-engineered feature representations in many domains, and made learning possible in domains where engineered features were lacking entirely.

We are particularly motivated by large-scale visual recognition, where a specific type of deep architecture has achieved a commanding lead on the state-of-the-art. These *Convolutional Neural Networks*, or CNNs, are discriminatively trained via back-propagation through layers of convolutional filters and other operations such as rectification and pooling. Following the early success of digit classification in the 90's, these models have recently surpassed all known methods for large-scale visual recognition, and have been adopted by industry heavyweights such as Google, Facebook, and Baidu for image understanding and search.

While deep neural networks have attracted enthusiastic interest within computer vision and beyond, replication of published results can involve months of work by a researcher or engineer. Sometimes researchers deem it worthwhile to release trained models along with the paper advertising their performance. But trained models alone are not sufficient for rapid research progress and emerging commercial applications, and few toolboxes offer truly off-the-shelf deployment of state-of-the-art models—and those that do are often not computationally efficient and thus unsuitable for commercial deployment.

To address such problems, we present Caffe, a fully open-source framework that affords clear access to deep architectures. The code is written in clean, efficient C++, with CUDA used for GPU computation, and nearly complete, well-supported bindings to Python/Numpy and MATLAB. Caffe adheres to software engineering best practices, providing unit tests for correctness and experimental rigor and speed for deployment. It is also well-suited for research use, due to the careful modularity of the code, and the clean separation of network definition (usually the novel part of deep learning research) from actual implementation.

In Caffe, multimedia scientists and practitioners have an orderly and extensible toolkit for state-of-the-art deep learning algorithms, with reference models provided out of the box. Fast CUDA code and GPU computation fit industry needs by achieving processing speeds of more than 40 mil-

Nicholson EX-1021, Page 1
Nicholson Manufacturing Ltd. v. Bid Group Technologies Ltd.
PGR2025-00081

Framework	License	Core language	Binding(s)	CPU	GPU	Open source	Training	Pretrained models	Development
Caffe	BSD	C++	Python, MATLAB	✓	✓	✓	✓	✓	distributed
cuda-convnet [7]	unspecified	C++	Python		✓	✓	✓		discontinued
Decaf [2]	BSD	Python		✓		✓	✓	✓	discontinued
OverFeat [9]	unspecified	Lua	C++,Python	✓				✓	centralized
Theano/Pylearn2 [4]	BSD	Python		✓	✓	✓	✓		distributed
Torch7 [1]	BSD	Lua		✓	✓	✓	✓		distributed

Table 1: Comparison of popular deep learning frameworks. *Core language* is the main library language, while *bindings* have an officially supported library interface for feature extraction, training, etc. *CPU* indicates availability of host-only computation, no GPU usage (e.g., for cluster deployment); *GPU* indicates the GPU computation capability essential for training modern CNNs.

lion images per day on a single K40 or Titan GPU. The same models can be run in CPU or GPU mode on a variety of hardware: Caffe separates the representation from the actual implementation, and seamless switching between heterogeneous platforms furthers development and deployment—Caffe can even be run in the cloud.

While Caffe was first designed for vision, it has been adopted and improved by users in speech recognition, robotics, neuroscience, and astronomy. We hope to see this trend continue so that further sciences and industries can take advantage of deep learning.

Caffe is maintained and developed by the BVLC with the active efforts of several graduate students, and welcomes open-source contributions at `http://github.com/BVLC/caffe`. We thank all of our contributors for their work!

2. HIGHLIGHTS OF CAFFE

Caffe provides a complete toolkit for training, testing, finetuning, and deploying models, with well-documented examples for all of these tasks. As such, it's an ideal starting point for researchers and other developers looking to jump into state-of-the-art machine learning. At the same time, it's likely the fastest available implementation of these algorithms, making it immediately useful for industrial deployment.

Modularity. The software is designed from the beginning to be as modular as possible, allowing easy extension to new data formats, network layers, and loss functions. Lots of layers and loss functions are already implemented, and plentiful examples show how these are composed into trainable recognition systems for various tasks.

Separation of representation and implementation. Caffe model definitions are written as config files using the Protocol Buffer language. Caffe supports network architectures in the form of arbitrary directed acyclic graphs. Upon instantiation, Caffe reserves exactly as much memory as needed for the network, and abstracts from its underlying location in host or GPU. Switching between a CPU and GPU implementation is exactly one function call.

Test coverage. Every single module in Caffe has a test, and no new code is accepted into the project without corresponding tests. This allows rapid improvements and refactoring of the codebase, and imparts a welcome feeling of peacefulness to the researchers using the code.

Python and MATLAB bindings. For rapid prototyping and interfacing with existing research code, Caffe provides Python and MATLAB bindings. Both languages may be used to construct networks and classify inputs. The Python bindings also expose the solver module for easy prototyping of new training procedures.

Pre-trained reference models. Caffe provides (for academic and non-commercial use—not BSD license) reference models for visual tasks, including the landmark "AlexNet" ImageNet model [8] with variations and the R-CNN detection model [3]. More are scheduled for release. We are strong proponents of reproducible research: we hope that a common software substrate will foster quick progress in the search over network architectures and applications.

2.1 Comparison to related software

We summarize the landscape of convolutional neural network software used in recent publications in Table 1. While our list is incomplete, we have included the toolkits that are most notable to the best of our knowledge. Caffe differs from other contemporary CNN frameworks in two major ways:

(1) The implementation is completely C++ based, which eases integration into existing C++ systems and interfaces common in industry. The CPU mode removes the barrier of specialized hardware for deployment and experiments once a model is trained.

(2) Reference models are provided off-the-shelf for quick experimentation with state-of-the-art results, without the need for costly re-learning. By finetuning for related tasks, such as those explored by [2], these models provide a warm-start to new research and applications. Crucially, we publish not only the trained models but also the recipes and code to reproduce them.

3. ARCHITECTURE

3.1 Data Storage

Caffe stores and communicates data in 4-dimensional arrays called *blobs*.

Blobs provide a unified memory interface, holding batches of images (or other data), parameters, or parameter updates. Blobs conceal the computational and mental overhead of mixed CPU/GPU operation by synchronizing from the CPU host to the GPU device as needed. In practice, one loads data from the disk to a blob in CPU code, calls a CUDA kernel to do GPU computation, and ferries the blob off to the next layer, ignoring low-level details while maintaining a high level of performance. Memory on the host and device is allocated on demand (lazily) for efficient memory usage.

Nicholson EX-1021, Page 2
Nicholson Manufacturing Ltd. v. Bid Group Technologies Ltd.
PGR2025-00081

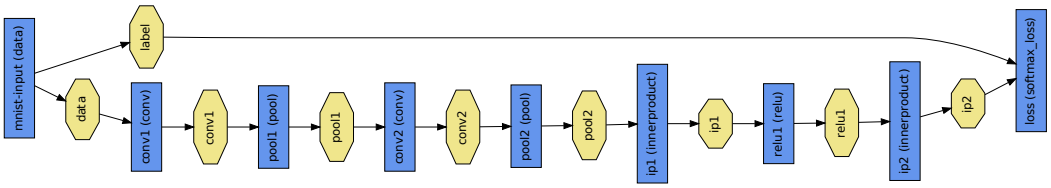

Figure 1: An MNIST digit classification example of a Caffe network, where blue boxes represent layers and yellow octagons represent data blobs produced by or fed into the layers.

Models are saved to disk as Google Protocol Buffers[1], which have several important features: minimal-size binary strings when serialized, efficient serialization, a human-readable text format compatible with the binary version, and efficient interface implementations in multiple languages, most notably C++ and Python.

Large-scale data is stored in LevelDB[2] databases. In our test program, LevelDB and Protocol Buffers provide a throughput of 150MB/s on commodity machines with minimal CPU impact. Thanks to layer-wise design (discussed below) and code modularity, we have recently added support for other data sources, including some contributed by the open source community.

3.2 Layers

A Caffe layer is the essence of a neural network layer: it takes one or more blobs as input, and yields one or more blobs as output. Layers have two key responsibilities for the operation of the network as a whole: a *forward pass* that takes the inputs and produces the outputs, and a *backward pass* that takes the gradient with respect to the output, and computes the gradients with respect to the parameters and to the inputs, which are in turn back-propagated to earlier layers.

Caffe provides a complete set of layer types including: convolution, pooling, inner products, nonlinearities like rectified linear and logistic, local response normalization, element-wise operations, and losses like softmax and hinge. These are all the types needed for state-of-the-art visual tasks. Coding custom layers requires minimal effort due to the compositional construction of networks.

3.3 Networks and Run Mode

Caffe does all the bookkeeping for any directed acyclic graph of layers, ensuring correctness of the forward and backward passes. Caffe models are end-to-end machine learning systems. A typical network begins with a data layer that loads from disk and ends with a loss layer that computes the objective for a task such as classification or reconstruction.

The network is run on CPU or GPU by setting a single switch. Layers come with corresponding CPU and GPU routines that produce identical results (with tests to prove it). The CPU/GPU switch is seamless and independent of the model definition.

3.4 Training A Network

Caffe trains models by the fast and standard stochastic gradient descent algorithm. Figure 1 shows a typical example of a Caffe network (for MNIST digit classification) during training: a data layer fetches the images and labels from disk, passes it through multiple layers such as convolution, pooling and rectified linear transforms, and feeds the final prediction into a classification loss layer that produces the loss and gradients which train the whole network. This example is found in the Caffe source code at `examples/lenet/lenet_train.prototxt`. Data are processed in mini-batches that pass through the network sequentially. Vital to training are learning rate decay schedules, momentum, and snapshots for stopping and resuming, all of which are implemented and documented.

Finetuning, the adaptation of an existing model to new architectures or data, is a standard method in Caffe. From a snapshot of an existing network and a model definition for the new network, Caffe finetunes the old model weights for the new task and initializes new weights as needed. This capability is essential for tasks such as knowledge transfer [2], object detection [3], and object retrieval [5].

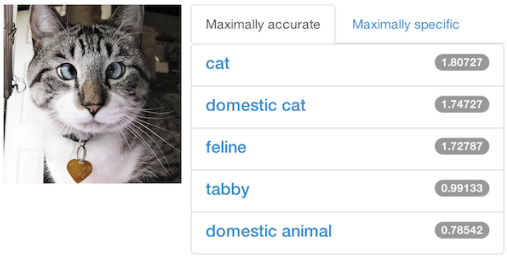

Figure 2: An example of the Caffe object classification demo. Try it out yourself online!

4. APPLICATIONS AND EXAMPLES

In its first six months since public release, Caffe has already been used in a large number of research projects at UC Berkeley and other universities, achieving state-of-the-art performance on a number of tasks. Members of Berkeley EECS have also collaborated with several industry partners such as Facebook [11] and Adobe [6], using Caffe or its direct precursor (Decaf) to obtain state-of-the-art results.

Object Classification Caffe has an online demo[3] showing state-of-the-art object classification on images provided by the users, including via mobile phone. The demo takes the image and tries to categorize it into one of the 1,000 ImageNet categories[4]. A typical classification result is shown in Figure 2.

Furthermore, we have successfully trained a model with all 10,000 categories of the full ImageNet dataset by finetuning this network. The resulting network has been applied to open vocabulary object retrieval [5].

[1] `https://code.google.com/p/protobuf/`
[2] `https://code.google.com/p/leveldb/`
[3] `http://demo.caffe.berkeleyvision.org/`
[4] `http://www.image-net.org/challenges/LSVRC/2013/`

Nicholson EX-1021, Page 3
Nicholson Manufacturing Ltd. v. Bid Group Technologies Ltd.
PGR2025-00081

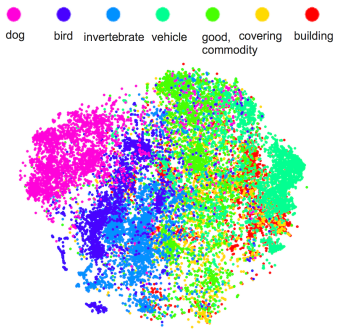

Figure 3: Features extracted from a deep network, visualized in a 2-dimensional space. Note the clear separation between categories, indicative of a successful embedding.

Learning Semantic Features In addition to end-to-end training, Caffe can also be used to extract semantic features from images using a pre-trained network. These features can be used "downstream" in other vision tasks with great success [2]. Figure 3 shows a two-dimensional embedding of all the ImageNet validation images, colored by a coarse category that they come from. The nice separation testifies to a successful semantic embedding.

Intriguingly, this learned feature is useful for a lot more than object categories. For example, Karayev et al. have shown promising results finding images of different styles such as "Vintage" and "Romantic" using Caffe features (Figure 4) [6].

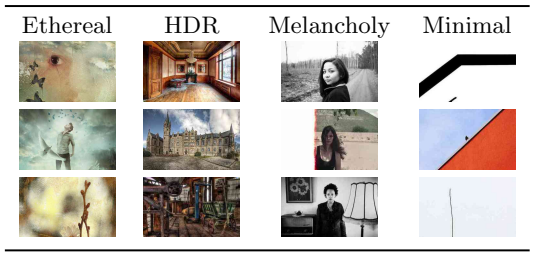

Figure 4: Top three most-confident positive predictions on the Flickr Style dataset, using a Caffe-trained classifier.

Object Detection Most notably, Caffe has enabled us to obtain **by far** the best performance on object detection, evaluated on the hardest academic datasets: the PASCAL VOC 2007-2012 and the ImageNet 2013 Detection challenge [3].

Girshick et al. [3] have combined Caffe together with techniques such as Selective Search [10] to effectively perform simultaneous localization and recognition in natural images. Figure 5 shows a sketch of their approach.

Beginners' Guides To help users get started with installing, using, and modifying Caffe, we have provided instructions and tutorials on the Caffe webpage. The tutorials range from small demos (MNIST digit recognition) to serious deployments (end-to-end learning on ImageNet).

Although these tutorials serve as effective documentation of the functionality of Caffe, the Caffe source code additionally provides detailed inline documentation on all modules.

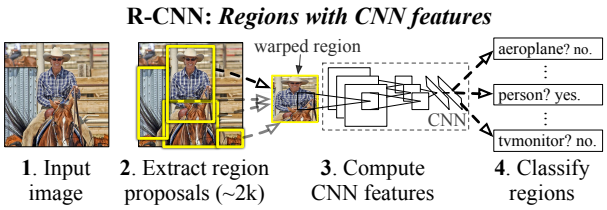

Figure 5: The R-CNN pipeline that uses Caffe for object detection.

This documentation will be exposed in a standalone web interface in the near future.

5. AVAILABILITY

Source code is published BSD-licensed on GitHub.[5] Project details, step-wise tutorials, and pre-trained models are on the homepage.[6] Development is done in Linux and OS X, and users have reported Windows builds. A public Caffe Amazon EC2 instance is coming soon.

6. ACKNOWLEDGEMENTS

We would like to thank NVIDIA for GPU donation, the BVLC sponsors (`http://bvlc.eecs.berkeley.edu/`), and our open source community.

7. REFERENCES

[1] R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A MATLAB-like environment for machine learning. In *BigLearn, NIPS Workshop*, 2011.

[2] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *ICML*, 2014.

[3] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014.

[4] I. Goodfellow, D. Warde-Farley, P. Lamblin, V. Dumoulin, M. Mirza, R. Pascanu, J. Bergstra, F. Bastien, and Y. Bengio. Pylearn2: a machine learning research library. *arXiv preprint 1308.4214*, 2013.

[5] S. Guadarrama, E. Rodner, K. Saenko, N. Zhang, R. Farrell, J. Donahue, and T. Darrell. Open-vocabulary object retrieval. In *RSS*, 2014.

[6] S. Karayev, M. Trentacoste, H. Han, A. Agarwala, T. Darrell, A. Hertzmann, and H. Winnemoeller. Recognizing image style. *arXiv preprint 1311.3715*, 2013.

[7] A. Krizhevsky. cuda-convnet. https://code.google.com/p/cuda-convnet/, 2012.

[8] A. Krizhevsky, I. Sutskever, and G. Hinton. ImageNet classification with deep convolutional neural networks. In *NIPS*, 2012.

[9] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. In *ICLR*, 2014.

[10] J. Uijlings, K. van de Sande, T. Gevers, and A. Smeulders. Selective search for object recognition. *IJCV*, 2013.

[11] N. Zhang, M. Paluri, M. Ranzato, T. Darrell, and L. Bourdev. Panda: Pose aligned networks for deep attribute modeling. In *CVPR*, 2014.

[5] `https://github.com/BVLC/caffe/`
[6] `http://caffe.berkeleyvision.org/`

Nicholson EX-1021, Page 4
Nicholson Manufacturing Ltd. v. Bid Group Technologies Ltd.
PGR2025-00081