Acknowledgement

We thank Evan Sparks and Josh Rosen for their many contributions to deepViz, a project that served as the basis for development of ML-o-scope. We also thank Mike Franklin, Ben Recht, and Maneesh Agrawala for their valuable feedback and guidance. This research is supported in part by NSF CISE Expeditions Award CCF-1139158, LBNL Award 7076018, and DARPA XData Award FA8750-12-2-0331, and gifts from Amazon Web Services, Google, SAP, The Thomas and Stacey Siebel Foundation, Apple, Inc., Cisco, Cloudera, EMC, Ericsson, Facebook, GameOnTalis, Guavus, HP, Huawei, Intel, Microsoft, NetApp, Pivotal, Splunk, Virdata, VMware, WANdisco and Yahoo!.

ML-o-scope: a diagnostic visualization system for deep machine learning pipelines

Daniel M. Bruckner
UC Berkeley
bruckner@eecs.berkeley.edu

Abstract

The recent success of deep learning is driving a trend towards structurally complex computer vision models that combine feature extraction with predictive elements into integrated pipelines. While some of these models have achieved breakthrough results in applications like object recognition, they are difficult to design and tune, impeding progress. We feel that visual analysis can be a powerful tool to aid iterative development of deep model pipelines. Building on feature evaluation work in the computer vision community, we introduce ML-O-SCOPE, an interactive visualization system for exploratory analysis of convolutional neural networks, a prominent type of pipelined model. We present ML-O-SCOPE's time-lapse engine that provides views into model dynamics during training, and evaluate the system as a support for tuning large scale object-classification pipelines.

1 Introduction

A new generation of pipelined machine learning models is achieving significantly higher performance than older approaches to computer vision applications. Thanks to the large scale of online activity data, and to novel ways, like crowd-sourcing, of collecting it, data sets of unprecedented size and depth are now available for modeling [5] [16]. At the same time, hardware advances and specialized software implementations that take advantage of acceleration [8] [7] and distribution [2] have enabled larger models to be trained on larger data sets. As a result, models are growing correspondingly with data sets in order to encode the richness of sample populations with the highest possible fidelity.

The growth of model complexity, however, is not simply in terms of sheer size, or number of model parameters, but rather in terms of how many distinct stages of processing a model composes. Successful large scale models combine series of pipelined operators into a coherent data flow. Such pipelined models treat an application from end-to-end, including raw input normalization, stages of feature extraction, and ultimately, prediction.

Artificial neural networks are a classic example of a pipeline, with each layer performing a function and the back-propagation algorithm providing a unified approach to training [13]. The momentum behind deep learning, or the application of many-layered convolutional neural networks to large scale learning problems, has proved a major driver of pipelined model complexity. The recent success of deep learning underscores the importance of large, composite models and the need for tools to manage their complexity [10] [19].

New tools are necessary, because large models present new challenges to designers. The fundamental challenge of working with pipelined models is to decide what operators to include, and in what order, to maximize predictive accuracy and avoid over-fit. Although individual operators have well defined functions, their combined effect can be difficult to predict and optimize. Moreover, each operator often has associated hyper-parameters that must be tuned for peak performance. Because upstream operators affect the input to downstream ones, decisions about components and their parameters cannot be made in isolation from one another: the optimization space is very large. In the case of large convolutional neural networks, these difficulties have made pipeline design impossible for all but a small number of expert practitioners with extensive experience in the field.

We propose visualization as a means to address the challenges of scale and complexity, and to make high-end machine learning pipelines approachable. Visualizations of different pipeline states can illustrate why particular configurations succeed or fail, and at what points particular designs break down. These visuals can allow non-experts to better explore and understand the internal dynamics of pipelined models, and gain insights into what works from just a few model instances.

In this way, the optimization space can be navigated quickly.

To demonstrate the utility of visualization applied to pipeline tuning, we have developed ML-O-SCOPE, an interactive visualization system for analysis of deep model pipelines. Given a set of model snapshots, saved during training, an input corpus, and adaptor code to query them, ML-O-SCOPE offers navigation, displays, and interactions that allow users to explore their model and its relation to the data.

While visualization may not suit all types of input data, we focus on computer vision applications where data is visual by nature. The modular composition of pipelines facilitates inspection of intermediate data, i.e., transformations of images as they pass through the model. In certain cases, operators themselves are visualizable. To take convolutional operators as an example, we can visualize their parameters as image filters. Recent work has shown that visualization of these reconstructed intermediate states can be an aid to model tuning [19], [17].

Automation has been proposed as an alternative approach to pipeline tuning [18], but only for modestly sized models. An automated tuning algorithm will select a series of parameter settings and train and evaluate a model for each setting, eventually returning the highest performing one. But with models' size complexity growing into the tens of millions or even the billions of parameters [2], it can take days or even weeks to train a single instance of a top-performing model. At this scale, training time is too great, and the search space too large, for automated tuning to have much impact without expert guidance in the context of a defined workflow.

ML-O-SCOPE targets an iterative workflow for development and refinement of pipelined models. At a given stage in the design process, a user trains a pipelined architecture and saves regular checkpoints of its state during the training process. The user then uses ML-O-SCOPE to inspect individual pipeline stages; analyze properties of the training process, like convergence rates; and diagnose weaknesses. These observations lead to revisions to the model architecture and further rounds of training, visualization, and assessment.

To evaluate its usefulness, we apply ML-O-SCOPE to several pipelines for visual object classification, trained on the CIFAR-10 and ILSVRC 2012 (ImageNet) data sets. We find that the system is a powerful tool for exploratory analysis of the tested models, and in practice, allows users to find, diagnose, and act on interesting properties of these complex models.

2 Related Work

Recent work by Zeiler and Fergus [19] demonstrates the use of visualization to analyze deep convolutional neural networks. They apply a technique called deconvolution [20] to construct illuminating visual representations of individual points in the model. In short, they highlight regions of a sample image—from simple patterns to complex objects like faces—that maximize the output of part of the model. While these visualizations are compelling, the authors find direct views of parameters better suited to model improvement. They use visualizations of filters from the model design of Krizhevsky, Sutskever, and Hinton [10] to adjust two hyper-parameters and consequently boost performance significantly. With ML-O-SCOPE, we aim to extend their work on visual parameter tuning with an interactive system and by adding the time dimension to analysis.

Visualization has been used to explicate convolutional neural networks since some of the earliest implementations. LeNet [12], a system for handwritten digit recognition, and one of the the first to achieve near-human accuracy, notably includes an interactive visualization system to display predictions and features extracted from input images. The visualizations allow direct and compelling demonstration of important properties of the system like invariance to translations and deformations of the input. While LeNet's visualizations provide evidence for the system's merits, they do not serve as design aids to practitioners.

Beyond neural networks, visualization has been used in computer vision more generally as a tool to aid in feature evaluation. In [17], Vondrick, Khosla, and Malisiewicz argue for the necessity of visual inspection of image features to understand models' failures. They use feature inversion algorithms, whereby image features are transformed back into the original, human-comprehensible image-space, thereby giving intuitive access to abstract features. Le, et al. [11] perform inverse optimization on a model trained by unsupervised learning to construct the optimal inputs for single parameters. In particular, they find single deep neurons trained to respond to faces (both human and feline) and bodies. ML-O-SCOPE incorporates similar feature visualizations together with an interactive interface to enable exploration and analysis of individual vision operators in the context of a complete pipeline.

Much recent work explores the growing design space of machine learning pipelines, and convolutional neural networks in particular. Jarrett, et al. [6] evaluate architectural variations of different hand-designed networks on several data sets. Others, like Yamins, Tax,

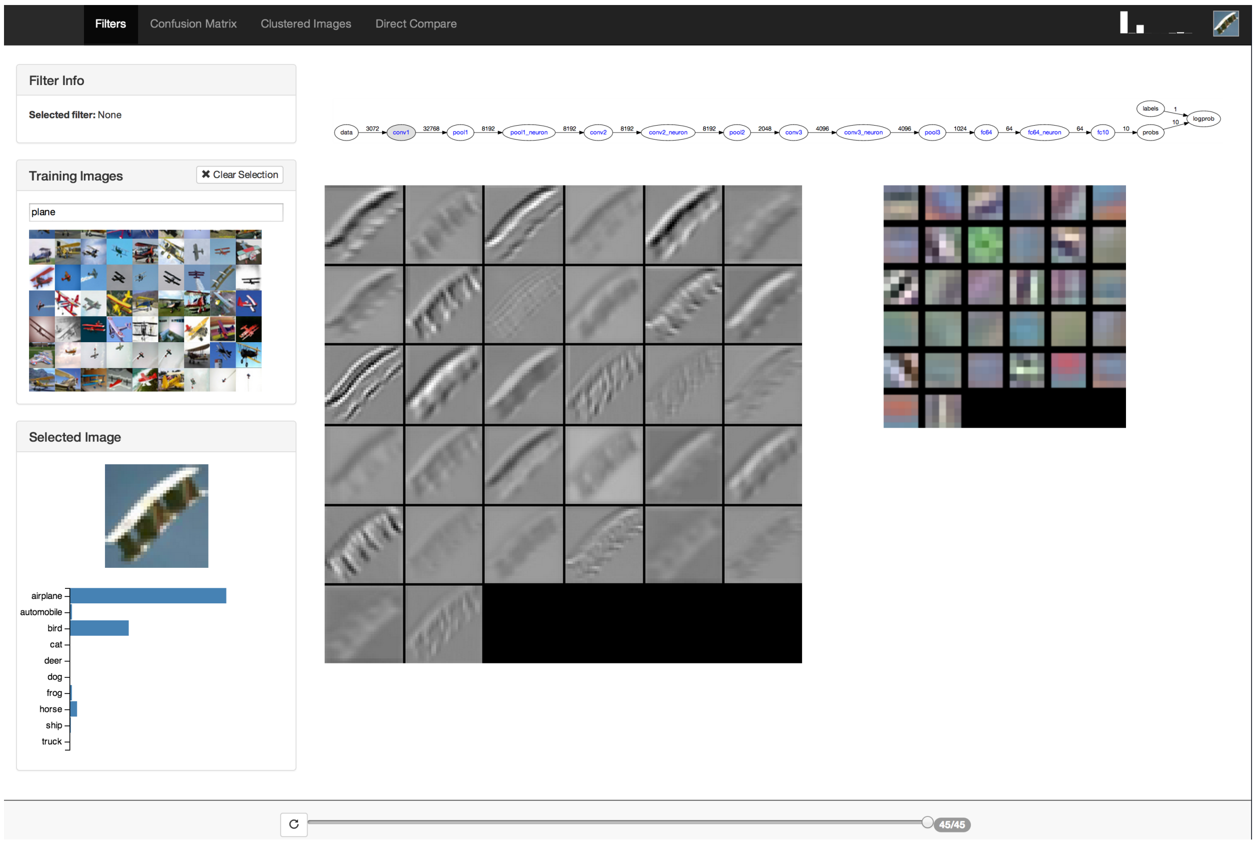


Figure 1. *The* ML-O-SCOPE *user interface.*

and Bergstra [18], use Bayesian methods to automatically search the parameter space of convolutional networks. ML-O-SCOPE intends to supplement such efforts by helping users develop heuristics to guide search and evaluation in this increasingly complex space.

3 Background

We begin by defining machine learning pipelines and exploring the design space of one particularly important example: deep convolutional neural networks. Briefly, we will highlight how major components and operators in these systems work, and how a whole pipeline is trained. While these networks can be trained for unsupervised learning tasks, we focus on the supervised case.

3.1 Pipelined Models

To apply machine learning to a problem usually requires two steps. The first is to identify and possibly engineer a good set of predictive features from raw data and the second is to train a model on these features. Formal machine learning focuses mainly on the second problem, and provides a variety of techniques and guidance to solve it. Feature extraction, however, remains an ad hoc process that depends greatly on the problem domain. For applications like computer vision where the best inputs to a classifier are not at all obvious—how does one get from a bitmap of pixels to a catalog of objects?—improving model performance consists almost entirely in improving the features fed in.

Pipelined models seek to couple feature extraction with prediction components so that they can be co-designed and optimized. A pipelined model is a series of operators that first preprocess raw data, then extract features, and finally use those features to make predictions. Because operators are modular and have uniform data flow interfaces, a pipeline framework allows easy experimentation with overall architecture. For example, one operator can be directly substituted for another, or a series of operators could be rearranged.

In designing ML-O-SCOPE, we focus on deep convolutional neural networks. As pipelines, this class of model architecture uses different compositions of convolutional and other image processing operators for feature extraction, followed by typical neural network classification structures. Such pipelines add the ability to train the feature extraction components together with the predictive components, via gradient back-propagation. In certain respects, this represents a way to automate, through learning, the difficult task of feature engineering.

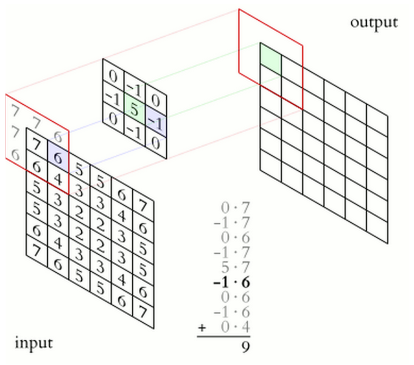
3.2 Convolutional Neural Networks

In general, an artificial neural network is a pipeline where operators are described as layers of so-called *neurons*. A neuron computes a function on inputs from the preceding layer and passes the result, sometimes called the neuron's *activation*, to outputs in the succeeding layer. Within each layer, all neurons compute the same function, but individual neurons may have distinct sets of inputs and outputs and may assign different weights to their inputs. Different types of layers are defined by the number and pattern of connections between neurons, and the functions they compute. Successions of *fully connected* layers, where neurons receive input from every output in the preceding layer, function as predictive units [15]. *Convolutional layers'* neurons are connected only to a local neighbors of outputs from a preceding layer in such a way that they compute the convolution of an input "image" with a filter. We describe convolution in greater detail below. Other types of layers may perform other types of data and image processing including contrast normalization and sampling.

As described above, a complete network architecture is a pipelined series of feature extraction layers, like convolutions and down-sampling, followed by predictive layers. When applied to object classification, the output of a pipeline will be a vector of probabilities predicting to which class an input image belongs. This output can be used by an optimization algorithm, like gradient descent, to update the pipeline and reduce error. Back-propagation is an algorithm that allows this optimization process to be applied to all the layer in the network, including those involved in feature extraction.

3.2.1 Convolution

Since many of the visualizations implemented in ML-oscope relate to convolutional operators, we give a brief review of convolution. Convolution applies a filter to an image to produce a new image. A filter is a $k \times k$ weight-matrix where k is an odd number (so that the matrix has a center pixel). Pixels in the output image are produced by placing the filter on top of the input image, with its center aligned at the corresponding pixel, and computing the dot product of the filter with the pixels below it.



(Michael Plotke / CC-BY-SA-3.0)

Figure 2. *Image convolution.*

In effect, the convolution moves the filter across the image and replaces each pixel with some filtered combination of its neighbors. In fact, convolutional transformations can perform various useful image processing functions, like emphasizing edges and computing gradients of hue and value. Moreover, deep successions of convolutions have been shown to produce image encodings that are favorable for classification, owing to emergent invariance to translation and deformation [1]. But exactly what is computed—and its usefulness for classification—depends on the filters used, and therefore success of a convolutional network depends crucially on choosing good filters.

3.3 Pipeline Design Space

Recent success in image classification has come from going deeper: composing pipelines with more convolutional layers and more filters per layer. By learning features instead of engineering them directly, back-propagation has given well-designed, well-tuned pipelines a major advantage in complex domains like vision.

But in a certain respect, the promise of automatically learned features is undercut by the imposition of a new challenge: pipelines are complicated entities that are difficult to design. The problem shifts from engineering good features to engineering a pipeline capable of learning good features.

The case of convolutional neural networks is illustrative of the difficulty of optimization. Although at a high level the design is straightforward—a sequence of convolutional operators followed by a classifier—many details need to be tuned. At the architectural level, the number of convolutional layers must be determined. Additional convolutions tend to improve model performance, but at some point the marginal return of another layer is outweighed by its added complexity. The number and position of non-convolutional operators—both for feature extraction, e.g., sampling and normalization, and for prediction—must also be decided.

More decisions are attendant on the level of individual operators. Convolutional operators have no shortage of hyper-parameters, including, the number of filters in them, the size of those filters, how those filters connect to filters in the layers before and after, and so on. Hyper-parameters of non-convolutional layers include sampling ratios, and fully-connected layer sizes. Learning parameters like gradient descent step size, regularization coefficients, and initial model weight distributions add yet more dimensions to the design space that must be tuned.

Finally, these various design decisions cannot in general be made in isolation from one another. Properties of one operator will affect the behavior of other downstream from it. Moreover, large-scale pipelined models are run in resource constrained settings. For example, the deep convolutional architecture of Krizhevsky et al. [10] is designed to saturate a specific model GPU. In this regime, decisions to allocate more resources to one operator, e.g., more filters in a convolutional layer, must trade decreased performance elsewhere in the model.

All of these factors can have a dramatic impact on model performance and complexity. By offering visual tools to analyze the effects of design decisions, ML-O-SCOPE enables users to explore the design space without blindly trying all possible permutations.

4 The ML-o-scope System

We implemented the ML-O-SCOPE system to investigate the usefulness of visual exploratory analysis applied to convolutional neural network pipeline optimization. ML-O-SCOPE is a light-weight web application that allows users to visually examine saved snapshots of a trained model. This section gives an overview of ML-O-SCOPE's system architecture and the visual and navigational features that aid model exploration and diagnosis.

4.1 Representing Models

Most features of ML-O-SCOPE are built upon a core abstract data model of convolutional neural networks. The back-end supports three classes of visualization: views of model parameters, views of features (data transformed by the model), and summary views. Parameter and feature views, as well as navigational features, access data and meta-data from saved model instances through this core abstraction. Summary views are supported by a separate pre-computed statistics database described below.

The data model for queries includes model checkpoints, layers, and model parameters. A model checkpoint is a complete instance of the model at some point during the training process, typically measured in epochs or iterations. Model instances contain a set of layers, defined by their architecture, and each layer contains some number of parameters. Often, parameters are grouped in a natural way, as is the case with filters in convolutional layers. ML-O-SCOPE stores meta-data about connected models that describe overall pipeline architecture and details about each layer.

Different implementations of convolutional pipelines store models in distinctive formats that may not align with ML-O-SCOPE's own representation. To handle this heterogeneity, ML-O-SCOPE provides an interface to register adaptor code. The adaptor abstraction consists of a core set of query primitives for accessing checkpoints, layers, and parameters, as well as meta-data about pipeline architecture. With adaptors, all connected model instances can be accessed through the same abstract data layer. We have implemented adaptors for models trained by `decaf`, `caffe`, and `cuda-convnet`, each of fewer than 100 lines of python.

4.2 Supporting Views and Navigation

Visualizations of model parameters can be built directly from the results of model queries. Visualizations of intermediate feature data, on the other hand, require the model to be evaluated on some input data. Our model adaptor interface allows us to import a model checkpoint into `decaf` [3], a python native implementation of convolutional neural networks, and evaluate it on demand.

Views of feature data further depend on access to a collection of image data from which to draw examples. Like model instances, data sets can be connected to ML-O-SCOPE via an adaptor interface. Data adaptors implement access to individual images in the data set, and can optionally provide meta-data about each image. Basic access allows users to find random images to test against the model. File names, keywords, and class label meta-data allows ML-O-SCOPE to give users a simple faceted search interface to the data set.

Large pipelined models can contain an overwhelming number of parameters, too many to visually inspect and analyze together at once. ML-O-SCOPE uses model meta-data to provide users with a navigational interface that allows reasonably sized chunks of the model to selected and viewed. Users can select which layer to look at, and can further select a subset of parameters (or filters, in the convolutional case) in that layer. In addition to these layer- and parameter-axes, ML-O-SCOPE provides navigation along the time-axis. By keeping track of model snapshots, the system can build parameter and feature views based on any checkpointed moment during training.

4.3 Statistics Engine

Several of our visualizations require summary statistics that are be computed over a complete data set. These summaries include prediction performance measures, e.g., counts for building confusion matrices, and analysis of output probability vectors including clustering and indexing. Since generating model output over a full data set can be time consuming, we provide an engine to compute statistics in batches and save them to a database. In the same way that views of feature data are built internally with `decaf`, the statistics engine can import a model instance through the standard interface and run it on a connected data set. This is sufficient for a small data set like CIFAR-10, but at ImageNet scale it often makes more sense to run data through each model with its native platform (e.g., the GPU accelerated `caffe` system). In this case, the statistics engine can take raw output of predictions and class probabilities directly. At run time, the statistics database is queried via the same web service that powers our filter visualizations.

Currently, the statistics engine calculates the following statistics over the corpus at each time step: the set of class probabilities by image; counts of model confusion, i.e., how often images from each class were predicted to belong to each other class; an index of images by their actual and predicted classes; and a set of clusters and the k-nearest neighbors of those clusters that are calculated from the class posterior probability vectors output by the model. Each of these statistics is used to drive one of the views described below.

4.4 Web Application

We provide access to the model query interface, visualization generation, and statistics engine via a RESTful interface backed by a flask (python) application. The client uses web requests to this interface to query for model state, feature data, meta-data, and summary information. Responses are returned as either bitmap image data (PNG), vector graphics (SVG), or JSON. The client is responsible for issuing requests and handling responses in order to display views and enable interactive exploration. Since exploratory analysis often involves issuing many related queries, the server makes heavy use of caching to reduce latency when the same object is requested multiple times.

ML-O-SCOPE allows all parameter and feature-space views to be animated, so that users can watch how they evolve over the course of training. To avoid flickering in the animation, which could contribute to change blindness and a [4] diminished experience, the front-end uses several optimizations to maintain a responsiveness. Images are pre-fetched by the browser and positioned off-screen until they have loaded completely. With this approach, frame updates are seamless and don't require a round-trip to the web service.

5 Visual Analysis

We present the views supported by ML-O-SCOPE to help model builders understand their convolutional neural networks. The main display lets users interactively explore different components of a network and view its internal structure directly and via features extracted from sample data. Additional summary views are available in subsidiary displays. These provide supporting information to help the user assess hypotheses about the cause of certain types of errors and understand the interaction between classes. All these visualizations help users to understand how the model changes over the course of training. That is, they provide a mode of exploration and comparison across time steps. We feel that this is an important and differentiating characteristic of our work that may enable new insights into the model training process.

5.1 Main View: Filters and Features

Our primary display is a time-lapse view of model development for a particular network layer. Visualizations of both the layer's constituent parameters, and of the features produced by the layer, given an input image, are available in this display (see Figure 1). At the bottom of the window are timeline controls to support checkpoint selection and animation.

All of our views update in response to the current timeline value. Animated views allow users to see how the model evolves over the course of training and to observe

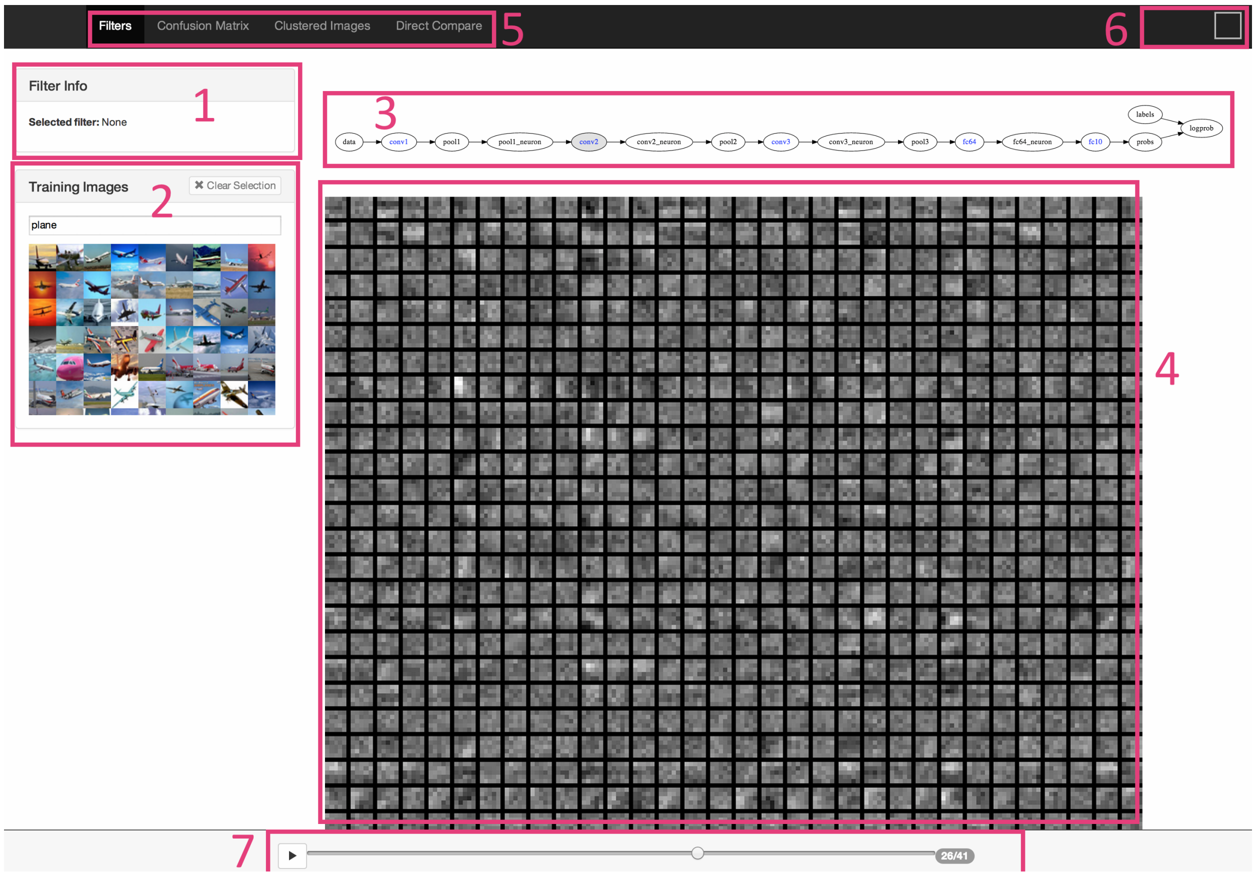


Figure 3. ML-o-scope *primary display. (1) Filter details; (2) image selector; (3) network overview and navigation; (4) filter visualization; (5) visualization selector; (6) selection helper; (7) animation progress slider.*

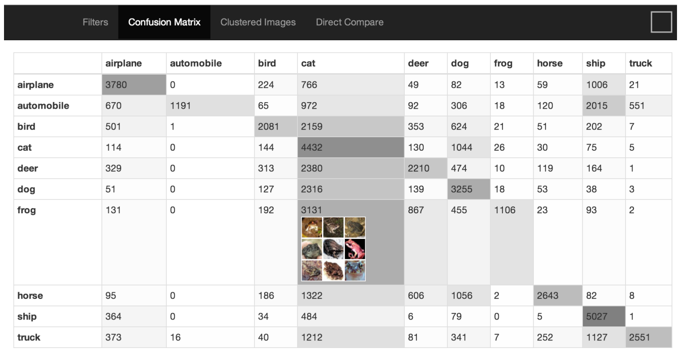
how structure emerges. To take one example, the filters shown at right in Figure 1, from the first convolutional layer of a model, progress into Gabor filters [14], a well studied type of convolutional feature extractor. They can be interpreted as deformation-invariant edge detectors [1], an effect that we can see in the visualized feature data in the figure, at center. It is important to note that the model converges on these filters automatically as part of the training process.

Additional elements of the main interface are highlighted in Figure 3. The top of the page displays an interactive graph representation of the model's pipelined architecture. Users can interact with the graph to navigate the network and display meta-data. Details about the currently selected layer are provided in the upper-left corner of the display.

A search interface for the image training corpus allows users to find and select images to pass to the model and view. Selected images are displayed in the feature space of the currently selected layer of the network, so users can visualize the output of each operator. The sidebar displays a histogram of the model's predicted classes for the selected image.

5.2 Summary Views

5.2.1 Confusion Matrix



	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	3780	0	224	766	49	82	13	59	1006	21
automobile	670	1191	65	972	92	306	18	120	2015	551
bird	501	1	2081	2159	353	624	21	51	202	7
cat	114	0	144	4432	130	1044	26	30	75	5
deer	329	0	313	2380	2210	474	10	119	164	1
dog	51	0	127	2316	139	3255	18	53	38	3
frog	131	0	192	3131	867	455	1106	23	93	2
horse	95	0	186	1322	606	1056	2	2643	82	8
ship	364	0	34	484	6	79	0	5	5027	1
truck	373	16	40	1212	81	341	7	252	1127	2551

Figure 4. *Confusion matrix display.*

The confusion matrix view, shown in Figure 4, helps users to diagnose "hot-spots" of misclassified images in their model. The matrix's rows correspond to true image classes and its columns correspond to the model's predicted classes. Each cell displays the number of images from one class predicted to be in another (or, on the diagonal, the same) class, for example, the number of dog images predicted to contain cats. Shading is used to emphasize cells with large counts so users can quickly perceive troublesome classes. A perfect classifier would produce a diagonal confusion matrix with zeros everywhere but on the main diagonal, so off-diagonal shading represents problems.

When the user mouses over an individual cell, the cell expands to show a sample of images that fall into it. If the misclassified images share common visual structure, the user may choose to give special treatment to this structure in a future version of their model. For example, if dark pictures tend to be misclassified, the user might choose to normalize input images before feeding them into the network.

Like the main filter display, the confusion matrix view is linked to the timeline slider to show how the model evolves over time.

5.3 Clustered Images

To further aid in the diagnosis of classification errors, the clustered images view displays a set of sample images clustered by their similarity in the raw pipeline output, normally a vector of predicted class probabilities. We cluster using K-Means with a Euclidean distance metric. For each cluster, we display the closest images to the cluster center. If a user wants to understand the possible causes of a set of misclassified images, they can inspect these clusters for anomalies like, say, a group of images of far-away airplanes that look like birds. The user may then adjust the parameters of their model to better handle this case, for example, by increasing the resolution of filters at an early layer.

Again, the time slider appears in this view to enable the user to see how these clusters evolve as the output of the fully connected layer changes at each model checkpoint. To our knowledge, this is a novel approach for diagnosing classification issues in the context of convolutional neural networks.

5.4 Direct Comparison

In the direct comparison view, shown in Figure 5, ML-O-SCOPE provides one more way to analyze changes in the model made over the course of training. Users select two points in time during training and then can display visualizations of those two snapshots side-by-side. As in the main display, users select which parts of the model to view, and whether to view model parameters directly or via extracted feature data. While the time-step animations of the main display allow a user to explore the incremental evolution of the model, this view highlights major cumulative changes across distant steps.

We can already see some utility and insights with this view. For example, filters that are initialized with high-variance weights tend to retain high variance weights in the final model. The filters in the fifth row and last row of the first convolutional layer all start with high variance and remain high variance at the end. This information can be used to inform approaches to model initialization and regularization.

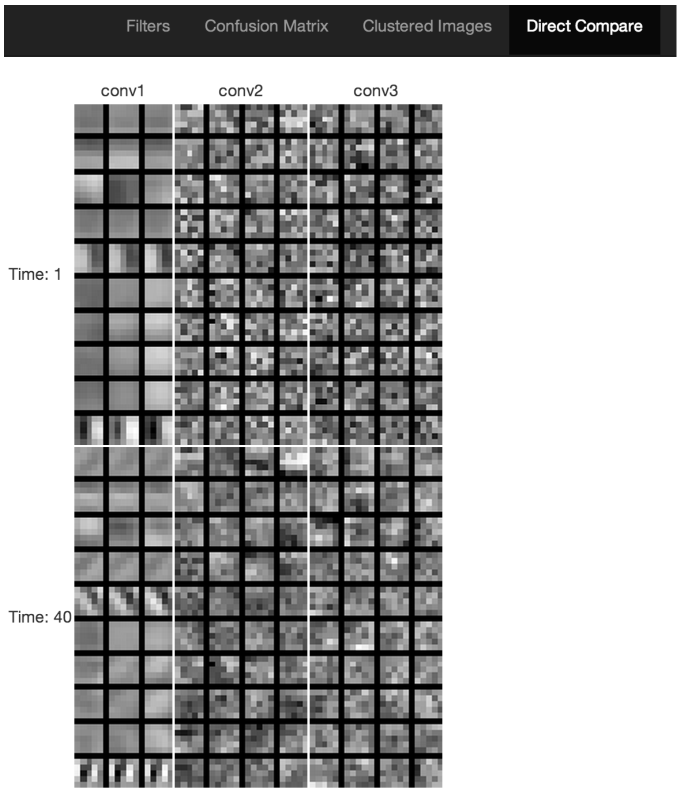


Figure 5. *Direct comparison display.*

6 Evaluation

To measure the usefulness of ML-O-SCOPE, we have instrumented it to connect to models trained by any of several systems, including Krizhevsky's `cudaconvnet` [8], and Jia's `caffe` [7] and `decaf` [3]. Both `cudaconvnet` and `caffe` use hardware acceleration to allow the training of an ImageNet scale model in a few days on a single machine with a recent generation GPU. `cudaconvnet` required slight modification to save intermediate model snapshots during training.

We used these systems to train models on two data sets. CIFAR-10 [9] is a modestly sized collection of images depicting ten classes of objects. It includes 60,000 images, each 32 by 32 pixels, drawn from the 80 Million Tiny Images data set [16] which consists of "in the wild" images scraped from the web. Despite its small size, CIFAR-10's origins make it a rich and challenging data set for object classification.

ML-o-scope has also been instrumented for ImageNet 2012 [5] data, and models trained on it. This data, from the ILSVRC 2012 challenge, consists of over one million full size images from web sources like Flickr.

6.1 Exploratory Analysis

ML-o-scope has proved useful for understanding the performance of CIFAR-10 and ImageNet models. The following use case illustrates the power of interactive exploratory analysis applied to model pipelines.

With `cudaconvnet`, we trained on CIFAR-10 a convolutional neural network architecture reported to achieve good performance with relatively little training [8]. The architecture consists of three stages of convolution and down-sampling, followed by a fully-connected network layer. Here the convolution and down-sampling operators represent the feature extraction component of the pipeline, and the fully-connected layer acts as a "universal classifier" [15]. This pipeline takes about ten epochs, or passes over the data, to train to convergence. We took checkpoints of the model before and after each epoch, and loaded these checkpoints into ML-o-scope for analysis.

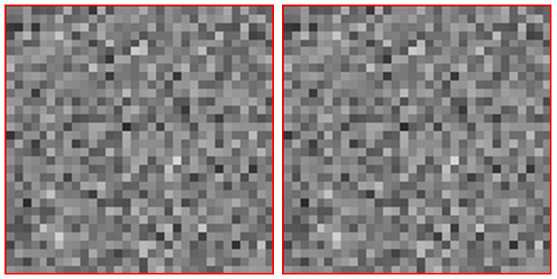


Figure 6. *Weights in the penultimate fully connected layer of a CIFAR-10 model, as initialized (left) and after 8 epochs of training. Lighter pixels correspond to higher weights.*

In exploring the development of model parameters in each stage of the pipeline, we observed that the visualization of the fully-connected operator remained static as training progressed. Since we expect learning to change the values of model parameters—which had been initialized randomly—we also expected to see the visualizations change. Suspecting a bug in our implementation, we queried the model checkpoint files directly and found that, indeed, the parameters of the fully connected layer remained essentially static during training.

This observation inspired follow-up experiments. We trained a slight variation of the original architecture where the fully connected component was replaced with multiplication by a random matrix. This is equivalent to the original architecture with learning disabled in that layer. The modified architecture had no loss of predictive accuracy on the CIFAR-10 test set (both architectures achieve about 25-26% error after 10 epochs), despite having fewer than 6% as many learned model parameters as the original. In principle, identifying non-learned components of a pipelined architecture like this could be exploited by software implementations to reduce training time per iteration.

It should be noted that this observation was specific to the model design and data set. For example, the corresponding fully connected layers in the popular Krizhevsky, et. al ImageNet architecture learn significantly throughout training. This variability across data sets and domains emphasizes the need for a tool like ML-o-scope to explore and diagnose new models trained for new applications.

6.2 User Study

ML-o-scope is intended as a tool to help machine learning practitioners design and tune optimal pipelined models in less time. We propose to directly measure its applicability to this problem domain through a user study comparing the system against the current alternatives. The target user is an analyst or data scientist who lacks intimate knowledge of model design and composition—including of particular types of pipelines like deep convolutional neural networks—but who has practical familiarity with machine learning models and how to construct them from data.

Our experimental design includes a small group ($N = 20$) of target users in two subgroups, namely, academic (students) and professional. To begin, participants are given a brief introduction to the basics of pipelined models, and of convolutional neural networks in particular, including a short guide with suggestions and best practices for tuning them. Participants should already be familiar with standard concepts like cross-validation, learning rates, regularization, and so on.

After the introduction, participants are given a series of model-tuning tasks to complete. For each task, the user is provided with a base architecture of a convolutional neural network pipeline appropriate for the CIFAR-10 data set, and a set of sub-optimal base parameters. In addition, users are given an interface where they can modify parameter settings, and a mechanism to train and evaluate a model given their current settings. The goal of the task is to minimize model error against the CIFAR-10 test set within a fixed budget of model iterations. To avoid overwhelming users, the task is limited to tuning a selected set of hyper-parameters (e.g., filter sizes and filter counts per layer), and the pipeline architectures remain fixed.

For some of the tasks, participants have access to ML-O-SCOPE to review the models trained at each iteration. Half of the participants have ML-O-SCOPE for the first half of tasks, and the other half for the second half only. In addition to the regular introduction, participants receive a brief introduction to ML-O-SCOPE before the tuning task where they are allowed to use it.

As a benchmark, we measure the performance of an expert designer of convolutional neural networks on the same tuning task, without the use of ML-O-SCOPE. The two primary metrics are: first, participants' models test accuracies after the budget of tuning iterations is expended; and second, the number of tuning iterations it takes participants' models to achieve near-expert level test accuracy, as determined by the expert designers results. Participants' performance distributions, according to both metrics, are compared for tasks completed with and without the aid of ML-O-SCOPE. An important secondary metric is the time taken to complete each task. We expect that participants will achieve higher accuracy with fewer iterations when using ML-O-SCOPE, likely at the cost of taking more time per iteration.

We further compare against automated tuning techniques' performance on the same tuning tasks. Two autotuning implementations—one applying the Bayesian techniques of Yamins, Tax, and Bergstra [18], the other using random search of the parameter space—are run on each user task, with the algorithmic search space set to only those parameters users are asked to optimize. For each implementation, we measure the number of iterations to reach near-expert performance, as defined above, and contrast the results with user performance with and without ML-O-SCOPE. We argue that the number of iterations is of greater importance than the time per iteration because large scale pipeline tuning time is typically dominated by the time it takes to train each model revision, and this quantity is independent of the tuning method. We expect that human performance dominates algorithmic performance measured in number of iterations.

Beyond measuring the overall effectiveness of ML-O-SCOPE, we would like to study the contributions of individual system features and visualizations. To determine these effects, participants are asked to complete a brief survey after finishing all tuning tasks. The survey asks users to explain why they made specific changes to hyper-parameters during the tuning process, what techniques they found successful, and where they had difficulty. We expect this qualitative feedback to give insight into the uses and usefulness of specific visualizations.

7 Future Directions

To date, ML-O-SCOPE has been engineered for a specific type of pipelined model, namely convolutional neural networks for visual object classification. The principles behind its design, however, are applicable to a wider domain of both models and applications. We see the most immediate promise from supporting more general vision pipelines, for example, visualization support for standard features like HOG and SIFT. These extensions would enable diagnostics of a more open pipeline design space not directly tied to the neural network paradigm.

In addition, the system provides a solid platform to explore other applications of visualization to pipelined models. For example, implementing new pipeline operators for feature extraction from image data is a difficult undertaking, and visualization can help with development and debugging of new code. Adapting ML-O-SCOPE for code diagnostics could be a powerful extension to the system.

8 Conclusion

We have presented ML-O-SCOPE, a visualization tool aimed at helping experts understand and diagnose issues with convolutional neural networks. The tool allows users to explore various aspects of structurally complex pipelined models—from understanding the development of convolutional structure, to better understanding common types of misclassification—and demonstrates the applicability of visualization to the challenges of optimizing complex object-recognition pipelines.

We thank Evan Sparks and Josh Rosen for their many contributions to `deepViz`, a project that served as the basis for development of ML-O-SCOPE. We also thank Mike Franklin, Ben Recht, and Maneesh Agrawala for their valuable feedback and guidance. This research is supported in part by NSF CISE Expeditions Award CCF-1139158, LBNL Award 7076018, and DARPA

XData Award FA8750-12-2-0331, and gifts from Amazon Web Services, Google, SAP, The Thomas and Stacey Siebel Foundation, Apple, Inc., Cisco, Cloudera, EMC, Ericsson, Facebook, GameOnTalis, Guavus, HP, Huawei, Intel, Microsoft, NetApp, Pivotal, Splunk, Virdata, VMware, WANdisco and Yahoo!.

References

[1] Bruna, J., and Mallat, S. Invariant scattering convolution networks. *arXiv preprint arXiv:1203.1513* (2012).

[2] Dean, J., Corrado, G. S., Monga, R., Chen, K., Devin, M., Le, Q. V., Mao, M. Z., Ranzato, M., Senior, A., Tucker, P., Yang, K., and Ng, A. Y. Large scale distributed deep networks. In *NIPS* (2012).

[3] Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., and Darrell, T. Decaf: A deep convolutional activation feature for generic visual recognition. *arXiv preprint arXiv:1310.1531* (2013).

[4] Healey, C. G. Perception in visualization. *Retrieved February 10* (2007), 2008.

[5] ImageNet. `http://www.image-net.org/`, 2013.

[6] Jarrett, K., Kavukcuoglu, K., Ranzato, M., and LeCun, Y. What is the best multi-stage architecture for object recognition? In *Computer Vision, 2009 IEEE 12th International Conference on* (2009), IEEE, pp. 2146–2153.

[7] Jia, Y. Caffe: An open source convolutional architecture for fast feature embedding. `http://caffe.berkeleyvision.org/`, 2013.

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

[9] Krizhevsky, A., and Hinton, G. Learning multiple layers of features from tiny images. *Computer Science Department, University of Toronto, Tech. Rep* (2009).

[10] Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural-networks. In *NIPS* (2012), vol. 1, p. 4.

[11] Le, Q. V., Ranzato, M., Monga, R., Devin, M., Chen, K., Corrado, G. S., Dean, J., and Ng, A. Y. Building high-level features using large scale unsupervised learning. *arXiv preprint arXiv:1113.6209* (2011).

[12] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86*, 11 (1998), 2278–2324.

[13] LeCun, Y. A., Bottou, L., Orr, G. B., and Müller, K.-R. Efficient backprop. In *Neural networks: Tricks of the trade*. Springer, 2012, pp. 9–48.

[14] Movellan, J. R. Tutorial on gabor filters.

[15] Simard, P., Steinkraus, D., and Platt, J. C. Best practices for convolutional neural networks applied to visual document analysis. In *ICDAR* (2003), vol. 3, pp. 958–962.

[16] Torralba, A., Fergus, R., and Freeman, W. T. 80 million tiny images: A large data set for nonparametric object and scene recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 30*, 11 (2008), 1958–1970.

[17] Vondrick, C., Khosla, A., and Malisiewicz, T. HOGgles: Visualizing Object Detection Features. *... Vision (ICCV)* (2013).

[18] Yamins, D., Tax, D., and Bergstra, J. S. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)* (2013), pp. 115–123.

[19] Zeiler, M. D., and Fergus, R. Visualizing and understanding convolutional neural networks. *arXiv preprint arXiv:1311.2901* (2013).

[20] Zeiler, M. D., Taylor, G. W., and Fergus, R. Adaptive deconvolutional networks for mid and high level feature learning. In *2011 IEEE International Conference on Computer Vision (ICCV)* (2011), IEEE, pp. 2018–2025.