Flexible Transfer Learning via Framework Conversion

Ankit Kumar
Stanford University
Palo Alto, California
ankitk@stanford.edu

Abstract

Recent results in Computer Vision have highlighted the effectiveness of Transfer Learning. Current Transfer Learning techniques are good first steps, but there is much work to be done in how to best Transfer Learn. To do this, researchers need both the flexibility to implement novel ideas, and pre-trained models to test their ideas with. No framework provides both. In this paper, we develop and release a library to convert trained models in Caffe to an equivalent model in the flexible Theano, and then show example usage with a novel Transfer Learning architecture.

1. Introduction

The fast and powerful Caffe[2] has an incredibly useful Model Zoo, where researchers can post trained models along with their architectures and hyperparameter settings. Caffe also allows for fine-tuning of pre-trained models and Net Surgery in order to adapt a pre-trained models to new datasets. This makes simple Transfer Leaning experiments easy: users can easily download a model, re-initialize and train new softmax layers, and then finetune through the architecture of the model.

There is a need for flexibility, so that researchers can use Transfer Learning along with novel architectures. In particular, this is necessary to advance research into how to best Transfer Learn; in addition, many varied tasks benefit from Transfer Learning, and these tasks often require more than the typical Convolutional Neural Network layers (Recurrent Layers, etc).

Our approach to solving this problem is to create a way to automatically convert Caffe models into Theano, an extremely flexible library used for mathematical operations. In this way, we can marry the excellent Model Zoo of Caffe with the flexibility of Theano, to allow researchers to quickly prototype and implement novel architectures, with the ability to use Transfer Learning.

We then explore the problem of how to best transfer learned representations to new datasets. This, too, is an important question, as previous methods have taken the simple approach of cutting off the last softmax layers and re-initializing softmax layers for the new dataset. We propose a novel architecture involving gates that adaptively learn which layers are best for representing which input images.

2. Caffe to Theano Conversion

2.1. Previous Work

2.1.1 Caffe

Caffe itself does offer ways of developing novel architectures and layers. However, this is not the goal of Caffe, and as a result, it is very hard to do, and it is not an efficient use of researcher's time. In order to implement a custom layer, one must[1]

- Find appropriate files and create a class declaration
- Implement the layer's CPU code in a .cpp file, abiding by Caffe's (poorly documented) conventions
- Implement the layer's GPU code in a .cu file, abiding by Caffe's (poorly documented) conventions
- Add the layer to the .proto file, including needed parameters
- Add the layer to the appropriate .cpp
- Write tests for the layer and use the gradient check to confirm that the Forward/Backward passes are working

The above steps are only for traditional, feedforward layers; implementing something like a Recurrent layer is even harder. Furthermore, Caffe is in constant development, and if something changes in the way Caffe

[1]https://github.com/BVLC/caffe/issues/684

handles any of the above steps, all code must change with it.

The cumbersome steps that one must go through to implement a new layer makes it hard for researchers to quickly get results on new ideas. Instead of trying something involving a novel layer, they might try to fit what they try to Caffe's framework, and not the other way around.

2.1.2 Theano

Theano is a python library that allows users to "define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays efficiently." [2] Among it's features is an excellent symbolic differentiation engine, allowing users to define models in a symbolic way with the confidence that the gradients will be computed correctly. Because of this, it is easy to very rapidly prototype model ideas, because the gradients do not need to be derived and computed by hand.

In addition, Theano compiles symbolic mathematical expressions into C and GPU code automatically, so developers do not need to know the ins and outs of C or CUDA in order to get the extreme performance advantages of those implementations.

2.1.3 Lasagne

Lasagne[3] is a thin wrapper around Theano that makes it easy to define layers that take Theano symbolic expressions and output other Theano symbolic expressions. By stacking these layers, one can easily define a large, complex model in Theano's symbolic language. Lasagne's framework of using layers as atomic units is very similar to Caffe's conception of a layer, and hence is a good candidate for converting Caffe models.

2.2. Approach

We take a conversion approach, so that we can leverage both the Caffe Model Zoo and the community that surrounds it, as well as the flexibility of Theano. In particular, our module has two main functions. One takes a caffe .prototxt file that defines a model architecture and returns an equivalent architecture in Lasagne; the other takes a caffe .caffemodel file that holds a trained caffemodel, as well as a Lasagne model that has the same architecture as the caffe model, and sets the Lasagne model with the parameters stored in the trained caffemodel – see Figure 1.

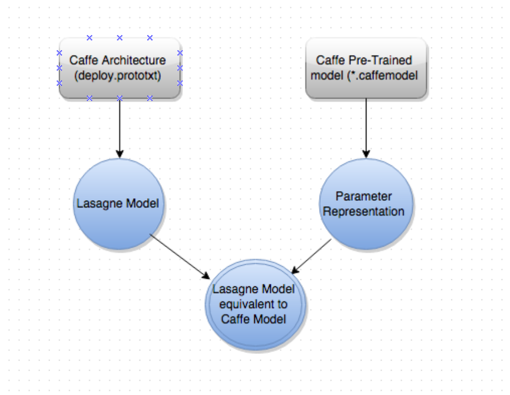

Caffe Architecture (deploy.prototxt)
Caffe Pre-Trained model (*.caffemodel
Lasagne Model
Parameter Representation
Lasagne Model equivalent to Caffe Model

Figure 1. Our conversion pipeline

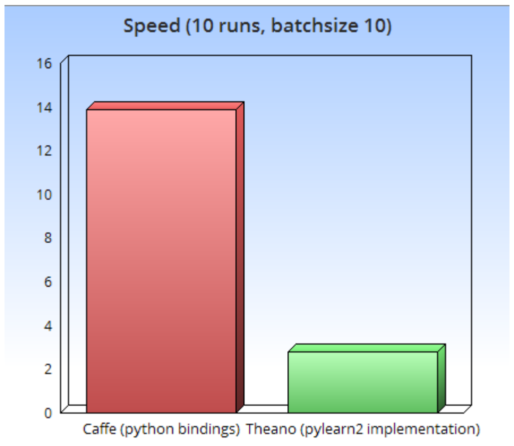


Figure 2. Speed comparison, Caffe's python bindings vs Theano

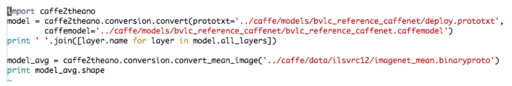
2.3. Results

We successfully developed the repository mentioned and implemented scripts to test it. Following the installation instructions in the repository, the tests pass from a clean installation. In addition, the Theano model's forward pass is 6x faster than Caffe's python bindings. See Figure 2 for a speed comparison. The tests were done by adding the time of 10 runs. For Caffe, we called Net.forward(**{net.inputs[0]:mat}), a function call in the Caffe examples. For Theano, we compiled a forward function and called it on the same mat.

2.3.1 Example Usage

The repository makes it easy to convert Caffe models by exposing a .convert method that accepts an architecture prototxt file and a caffemodel file, and returns a Lasagne Model. In addition, it is easy to convert an image mean

[2]http://deeplearning.net/software/theano/
[3]https://github.com/benanne/Lasagne

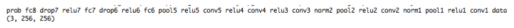


```
import caffe2theano
model = caffe2theano.conversion.convert(prototxt='../caffe/models/bvlc_reference_caffenet/deploy.prototxt',
        caffemodel='../caffe/models/bvlc_reference_caffenet/bvlc_reference_caffenet.caffemodel')
print ' '.join([layer.name for layer in model.all_layers])

model_avg = caffe2theano.conversion.convert_mean_image('../caffe/data/ilsvrc12/imagenet_mean.binaryproto')
print model_avg.shape
~
```

Figure 3. Example code snippet

binaryproto file into a numpy image representation by the .convert_mean_image function. The above code results in the following output:



```
prob fc8 drop7 relu7 fc7 drop6 relu6 fc6 pool5 relu5 conv5 relu4 conv4 relu3 conv3 norm2 pool2 relu2 conv2 norm1 pool1 relu1 conv1 data
(3, 256, 256)
```

Figure 4. Output of code snippet

as intended. See the repository [4] for more. Note that the repository is still in active development.

2.3.2 Difficulties

There were a number of difficulties we encountered when developing the repository. First and foremore, Caffe supports many architectures that are not really in use anymore, such as LRN layers and grouped convolutional layers. Many caffemodels use these layers. In order to convert these caffemodels into equivalent Theano models, we had to learn how these layers worked and implement them ourselves. They are in the "caffe_layers" directory in the repository.

In addition, in order to make use of the excellet pylearn2 cuda-convnet wrappers, we had to go into the pylearn2 code and change certain functions in order for them to accept the "group" argument. See the repository for instructions.

2.4. Future Steps

There are many more features we would like to add to the conversion repository, and the repository is currently under active development working on these features. Ideas for future improvements include:

- Comprehensive documentation and a setup.py installation.
- A solver class that works on the Lasagne Models, which would allow for automatic conversion of solver.prototxts. A prototype of this class is already implemented.
- A dataset class that would deal with loading data on the CPU/GPU and providing it to the solver. A prototype of this class is already implemented.
- Extending the Lasagne Model class to allow for slicing and adding Lasagne Models for intuitive net surgery.

[4]https://github.com/kitofans/caffe-theano-conversion

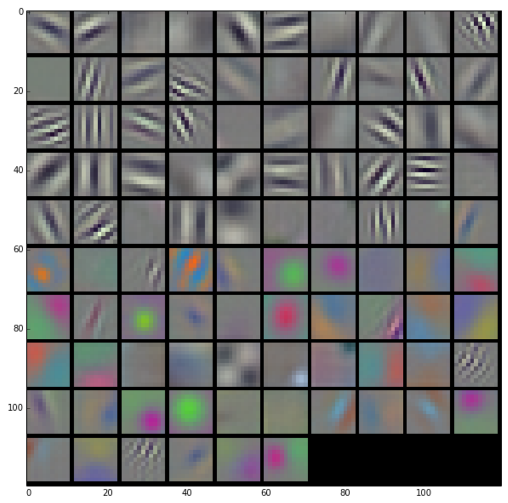


Figure 5. Filters learned by a Convolutional Neural Network. Note the abstract quality of them.

- Supporting snapshots of solvers and models.
- Supporting conversions from other frameworks such as Torch7.

Long-term, we would like to develop this repository into a way to convert models between different frameworks so that researchers who are comfortable with one framework can still get the benefits of work done on other frameworks. Caffe and Theano are a good place to start because of their popularity, easy of use, and interface.

3. A Novel Transfer Learning Architecture

We now describe our experiments in using the above-mentioned repository for Transfer Learning experiments. The task is Transfer Learning for Image Classification. We take a pre-trained model on the Caffe Model Zoo and explore how to best transfer the learned representation to a new, smaller dataset.

3.1. Previous Work

Convolutional Neural Networks, recently made popular by Krizhevsky's performance on the ImageNet2012 Challenge, take as input raw pixel values of images and propagate those images through alternating layers of Convolutional and Pooling layers, until finally going through Fully Connected layers ending in a linear classifier. They are successful because through the deep architecture, the model learns a feature representation of the image, and then uses that feature representation to classify the image at the end.

Convolutional Neural Networks are desirable for Transfer Learning, then, because that learned feature representation ought to be a useful place to start for any image, not just the ones that the CNN was originally trained. Visualizing the filters of a trained CNN gives good evidence for this idea, as the filters often look for simple things like lines in different orientations. Such high-level, abstract features ought to be good features for any image.

Intuition would also say that the features that a CNN learns should get worse, in terms of generalization to other images, as one gets nearer to the end of the CNN. This is because near the end of the CNN architecture, the immediate goal of the model is to classify the images in one of n classes, not to capture general, high-level features of the image. Empirical evidence so far has no backed this intuition, however, as most results state that the performance of Transfer Learning from different layers of the original CNN more or less increases as you get deeper through the CNN.

Recently there has been some work in benchmarking Transfer Learning methods for various datasets. Razavian et al famously [4] applied the weights of the popular OverFeat model to many diverse datasets and reported astonishing results. Their method was simple; they chopped off the softmax layer of the OverFeat network and trained a linear SVM on the features extracted from the first fully connected layer. The report specifically that the performance of their SVM does better as the layer at which they extract features gets deeper.

Yosinski et al[5] took a more academic approach and rigorously evaluated transfer learning by splitting their dataset into two sets and training classifiers on each set independently before transferring their weights to the other set. They find that chopping off the last layers of the original network and fine-tuning a new network gives best performance.

Importantly, most of the previous work on transfer learning does not attempt to devise new architectures to exploit the learned representation; rather, they use the same architecture as before, and experiment with the layer at which the original network is "chopped", or different hyperparametes and settings on how the new network is trained or how the weights are fine-tuned.

3.2. Our Approach

We take the approach of exploring novel architectures for Transfer Learning that do more than just use the same architecture as the original model and tuning parameters in different ways. Our model, which we call a Gated Transfer Network (GTN), is based on the intuition that different layers of the original network are better than others at capturing meaningful semantic information of different images. This is because the original network was trained on a very different distribution of images, and thus some images in the new dataset might be best represented by different layers in the original network, as later layers become more and more original-dataset-specific. Images in the new dataset that happen to look similar to images in the original dataset might be best represented by later layers; other images might be best represented by earlier layers.

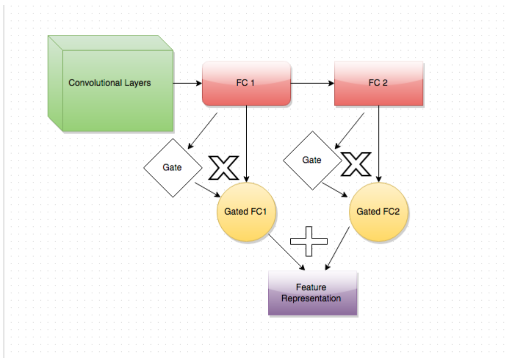


Figure 6. GTN model. Not shown: p(z).

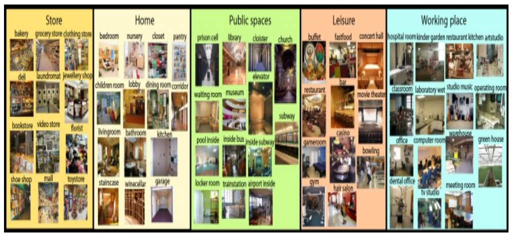


Figure 7. Example MIT-67 images. Compare to ImageNet.

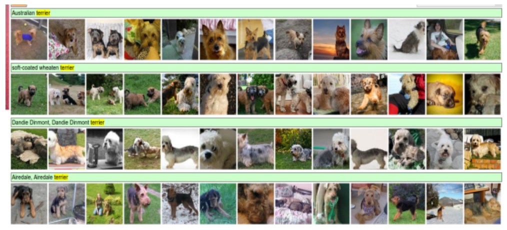

Australian terrier
soft-coated wheaten terrier
Dandie Dinmont, Dandie Dinmont terrier
Airedale, Airedale terrier

Figure 8. Example ImageNet images. Note prominent objects in the center of the image; different than MIT-67

Figure 5 shows the GTN model. For every layer of interest (in our experiments, these are each of the fully-connected layers after the convolutional layers), we connect the output of that layer to the Gate layer. The gate layer then has an nxd input, where n is the number of layers of ineterest and d is the dimensionality of the output of those layers. In our experiments, n is 2 and d is 4096.

The Gate layer then computes gates for each of the n

inputs:

$$z_i = f(W_z^i x_i + b_z) \quad (1)$$
$$r_i = p(z)_i \quad (2)$$
$$g_i = r_i \odot x_i \quad (3)$$

Where W_z^i is the ith gate matrix in R^{dxd}, b_z is the d-length bias, f is a nonlinearity z is the matrix of z_i's in R^{nxd}, p is a function that normalizes z column-wise so that all the n gates add to a vector of ones, and $\odot$ is elementwise multiplication. In our experiments, we use f = elementwise sigmoid and p = the softmax function. This gate mechanism is similar to the Gated Recurrent Unit[1]. The gates adaptively learn which of the n representations are best for classifying the given image, and weight the better representations more heavily in the classification task.

3.2.1 Dataset

We test the GTN on the MIT-Scenes dataset[3], a challenging dataset because it is significantly different from ImageNet. Note the difference in distribution between MIT-Scenes and ImageNet; the ImageNet images tend to have objects centrally located in the image and most of the semantic meaning is in that central object. The MIT-Scenes dataset, in contrast, has semantic meaning dispersed throughout the whole image. In addition, the objects one might look for are not centrally located, but rather, are distributed throughout the scene.

3.2.2 Data Processing and Model Choice

We use the BVLC Reference CaffeNet, an architecture similar to AlexNet. It was trained on ImageNet2012. To preprocess the data, we read in the images from MIT-Scenes and resize them to (3,227,227). We then subtract the mean image of ImageNet2012.

3.3. Results

	Test Accuracy
Softmax FC2	0.54263
Softmax FC1	0.55315
Ensemble Softmax FC1, Softmax FC2	0.5812
Simple Average FC1,FC2	0.56593
CNN-SVM	58.4
Gated Transfer	**0.59912**

The above table shows our results. Softmax FC1 and FC2 are training a new Softmax on the features extracted from FC1 or FC2. Average FC1,FC2 is training a new Softmax on the average of the features extracted from the two fully connected layers.

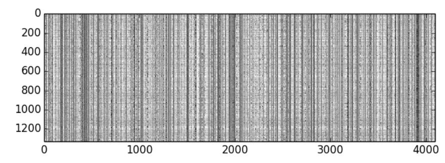


Figure 9. Gates on FC6 layer

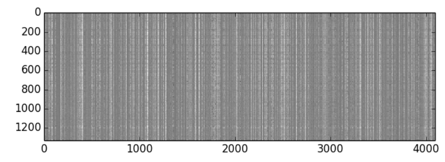


Figure 10. Gates on FC7 layer

CNN-SVM is reported from [4]. Their methodology is extracting features from FC1 and training an SVM on top. CNN-SVM already outperforms most non-CNN based classification schemes, as reported in the paper.

3.3.1 Analysis of Results

The fact that the ensemble of Softmax FC6 and Softmax FC7 performs significantly better than either classifier gives good evidence that the two feature representations are better at predicting different images.

See Figures 9 and 10 for visualizations of the gates. Each column is a dimension, and each row is an image in the test set. It is clear that the gates are learning that some dimensions from the second fully connected layer are better than the corresponding dimensions in the first fully connected layer and vice versa. The feature representation that ends up going to the classifier at the end can be thought of as the sum of the two arrays being visualized here. In this way, the GTN can selectively pick certain features from FC6 and certain ones from FC7.

In addition, close inspection of the visualizations show that, especially for the gates on FC6, different images have a different pattern. This lends credence to the intuition that the gates are adaptively learning which features are better for which image, which in turn gives credence to the intuition that some features are better for some images, while others are better for others. In general, it seems that FC7 is better on more dimensions than FC6; it is promising that the gates learn to selectively pick the dimensions that each are better in.

3.4. Future Work

There is much more work to be done on the problem of how to best Transfer Learn from pre-trained models. We would like to experiment more with fine-tuning hyperparameters and data augmentations, as time and technical constraints prevented us from rigorously cross-validating various settings. We were unable to get any good run of a full fine-tuning of the model. In addition, increasing the number n of layers we send to the gates is also something to try, though the parameter space gets much bigger as we would need to include a fully-connected layer so that all the vectors have the same dimensionality d. With enough correct data augmentation, this is probably doable, and is promising for future research.

In addition, we would like to explore methods that explicitly recognize the fact that the distribution of images in the new dataset is different than the distribution of images in the original dataset. For example, consider how a human would interact if given a new dataset of images and tasked with classifying them. They would try to represent the new images in a schema that makes sense for their experiences; perhaps by scanning through the images and looking for areas that they can easily understand. In this way, they are converting the new dataset of images into a distribution more like their ”old dataset of images”: they throw away parts of the image that are undecipherable to them, and only keep parts of the image that fit the distribution they are used to.

In a similar way, we might be able to devise an architecture that first learns to converts a new image into one that better fits the distribution of the original dataset (perhaps a recurrent attention mechanism, perhaps a convolutional/deconvolutional encoder), and then propagates that new image through the original network. This would better solve the problem that Transfer Learning faces, which is the difference in source distributions of the datasets.

References

[1] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014.

[2] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[3] A. Quattoni and A.Torralba. Recognizing indoor scenes. *IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

[4] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. CNN features off-the-shelf: an astounding baseline for recognition. *CoRR*, abs/1403.6382, 2014.

[5] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? *CoRR*, abs/1411.1792, 2014.