

[Home](#) / [Dato Blog](#)

Deep Learning: Doubly Easy and Doubly Powerful with GraphLab Create



Posted by [Piotr Teterwak](#) on Dec 22, 2014
3:00:00 AM

[Tweet](#) [in](#) [Share](#) 39 

5 7

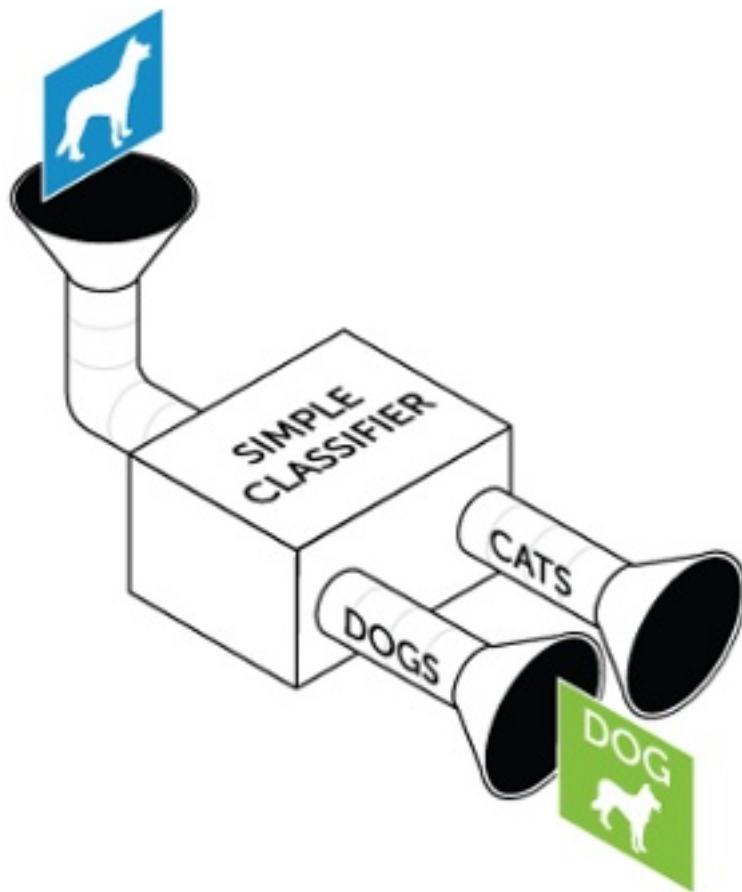
Note: Many of the code snippets in this blog post can take a very long time without GPU speedup. Please install the GPU version of GraphLab Create to follow along.

Subscribe to Dato Blog notifications

Recent Posts

- [Best TV Shows with Computer Science, Data Science, and Machine Learning](#)
- [How Fast Are Out-of-Core Algorithms?](#)
- [The Quest for Composable Data Vis](#)

One of machine learning's core goals is [classification](#) of input data. This is the task of taking novel data and assigning it to one of a pre-determined number of labels, based on what the classifier learns from a training set. For instance, a classifier could take an image and predict whether it is a cat or a dog.



The pieces of information fed to a classifier for each data point are called [features](#), and the category they belong to is

(React and ECharts)

- [Beginner's Guide to Click-Through Rate Prediction with Logistic Regression](#)
- ["No Better Magic than Machine Learning": Giving the Gift of ML Education](#)



SIGN UP HERE FOR OUR
UPCOMING FREE WEBINAR

"Image Similarity Using Deep Learning"

Posts by Topic

- [GraphLab Create \(20\)](#)
- [Data Science \(19\)](#)
- [SFrame \(9\)](#)
- [company news \(8\)](#)
- [Data Science Tools \(8\)](#)

[see all](#)

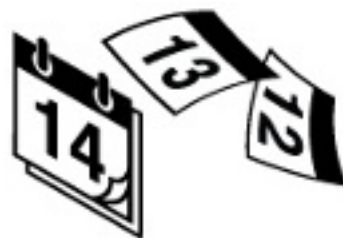
a **'target'** or **'label'**. Typically, the classifier is given data points with both features and labels, so that it can learn the correspondence between the two. Later, the classifier is queried with a data point and the classifier tries to predict what category it belongs to. A large group of these query data-points constitute a prediction-set, and the classifier is usually evaluated on its accuracy, or how many prediction queries it gets correct.

There are many methods to perform classification, such as SVMs, logistic regression, deep learning, and more. To read about the different methods GraphLab Create supports, I invite you to read the [API Documentation](#). Today, however, we'll focus on deep learning methods, which have recently been shown to give incredible results on challenging problems. Yet this comes at cost of extreme sensitivity to model hyper-parameters and long training time. This means that one can spend months testing different model configurations, much too long to be worth the effort.

Posts by Author

- [Alice Zheng \(12\)](#)
- [Carlos Guestrin \(8\)](#)
- [Jennifer Bolton \(7\)](#)
- [Chris DuBois \(6\)](#)
- [Rajat Arya \(5\)](#)

[see all](#)





This blog post focuses on minimizing these pains, and exploring how GraphLab Create 1.1 makes deep learning Easy.

What is Deep Learning?

Before we start, let's explore the idea of deep learning. 'Deep learning' is a phrase being thrown around everywhere in the world of machine learning. In fact, it's even been in [The New York Times](#). It seems to be helping make tremendous breakthroughs, but what is it? It's a methodology for learning high-level concepts about data, frequently through models that have multiple layers of non-linear

transformations. Let's take a moment to analyze that last sentence. 'Learning high-level concepts about data' means that deep learning models take data, for instance raw pixel values of an image, and learns abstract ideas like 'is animal' or 'is cat' about that data. OK, easy enough, but what does having 'multiple layers of non-linear transformations' mean. Conceptually, all this means is that you have a composition of simple non-linear functions, forming a complex non-linear function, which can map things as complex as raw pixel values to image category. Let's illustrate a simple example of this:

$$f(x) = \cos(a * x)$$

$$g(x) = \exp(b * x)$$

$$f(g(x)) = \cos(a * \exp(b * x))$$

Notice how the composition of functions $f(g(x))$ is much more complex than either $f(x)$ or $g(x)$. Furthermore, by adjusting the values of a and b you can adjust the mapping

between input and output. These values, called parameters, are what is learned in a deep learning model. This same idea of composition is used many, many times within deep learning models, and can enable learning very complex relationships between input and output. This complexity is what allows deep learning models to attain such amazing results.

The most common class of methods, and what GraphLab uses, within the deep learning domain are Deep Neural Nets (DNNs). Deep Neural Networks are simply artificial neural networks with many hidden layers. To learn more about artificial neural networks and deep learning, click [here](#).

Typically, DNN's are used for classification of input, and frequently for images. As I mentioned before, they are very good at this. So we should simply take whatever algorithm I had for image classification before and replace it with a DNN?

Not so fast.

Before you can do this, you have to choose how many layers your network has. And how many hidden units each layer has. And how to initialize the model parameter values

(also known as weights). And how much L2-regularization to apply. There's a lot more, too. Basically, a deep learning model is a machine with many confusing knobs (called hyper-parameters, basically parameters that are not learned by the algorithm) and dials that will not work if set randomly.

Making Deep Learning Easy with GraphLab Create

GraphLab Create allows you to get started with neural networks without being an expert by eliminating the need to choose a good architecture and hyper-parameter starting values. Based on the input data, the `neuralnet_classifier.create()` function chooses an architecture to use and sets reasonable values for hyper-parameters. Let's check this out on MNIST, a dataset composed of handwritten digits where the task is to identify the digit:

```
>>> data = graphlab.SFrame('http://s3.amazonaws.com/GraphLab-Datasets/mnist/sframe/train')
>>> model = graphlab.neuralnet_classifier.create(data,
target='label', validation_set=None)
```

Evaluating this model on the prediction data will tell us how well the model functioned:

```
>>> testing_data = graphlab.SFrame('http://s3.amazonaws.com/GraphLab-Datasets/mnist/sframe/test')
>>> model.evaluate(testing_data)
```

```
{'accuracy': 0.9803000092506409, 'confusion_matrix': C
columns:
    target_label    int
    predicted_label int
    count           int
```

Rows: 65

Data:

target_label	predicted_label	count
0	0	974
2	0	3
5	0	1
6	0	7
8	0	6
9	0	5
0	1	1
1	1	1128

	2		1		1	
	6		1		3	
	
+-----+-----+-----+						

[65 rows x 3 columns]

Note: Only the head of the SFrame **is** printed.

You can use `print_rows(num_rows=m, num_columns=n)` to **print** more rows **and** columns.}

We got **98.1% accuracy**. This is deep learning made Easy!

When Deep Learning made Easy is not easy enough, Making Deep Learning Doubly Easy

Although GraphLab Create tries to choose a good architecture and hyper-parameters, this automatic process often isn't enough. Optimal settings are often extremely problem specific, and it's impossible to determine them without good intuition, lots of experience, and many PhD students.

Yet, when good hyper-parameter settings come together, results are very strong. What's more, it's not uncommon for

the task you wanted to solve to be related something that has already been solved. Take, for example, the task of distinguishing cats from dogs. The famous ImageNet Challenge, for which DNN's are the state-of-the-art, asks the trained model to categorize input into one of 1000 classes (as Jay described in a previous [post](#)). Shouldn't features that distinguish between categories like lions and wolves should also be useful for discriminating between cats and dogs?

The answer is a definitive yes. It is accomplished by simply removing the output layer of the Deep Neural Network for 1000 categories, and taking the signals that would have been propagating to the output layer and feeding them as features to any classifier for our new cats vs dogs task. The training procedure breaks down something like this:

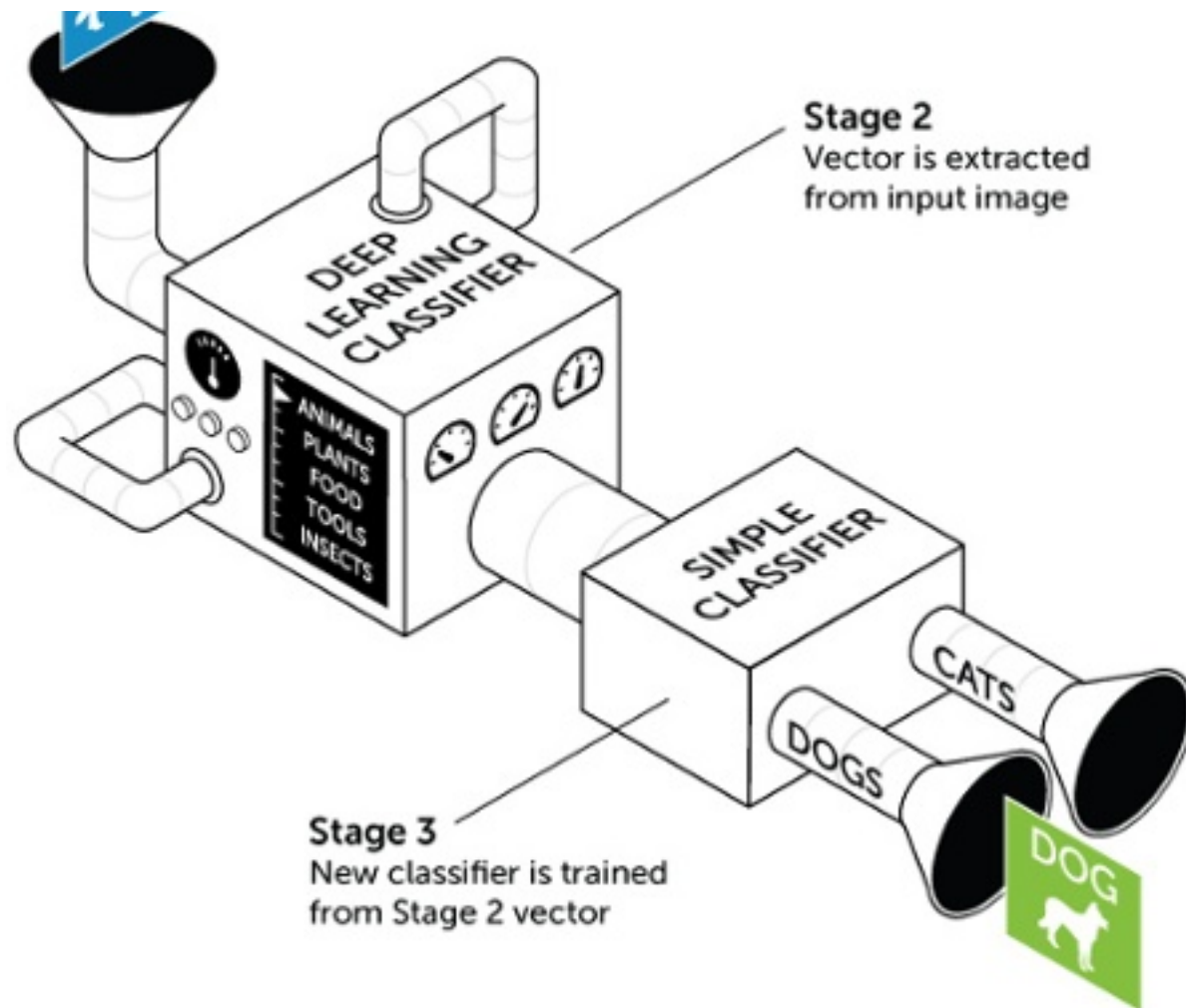
- **Stage 1:** Train a DNN classifier on a large, general dataset. A good example is ImageNet ,with 1000 categories and 1.2 million images. GraphLab hosts a model trained on ImageNet to allow you to skip this step in your own implementation. Simply load the model with

```
gl.load_model('http://s3.amazonaws.com/GraphLab-Datasets
```

- **Stage 2:** The outputs of each layer in the DNN can be viewed as a meaningful vector representation of each image. Extract these feature vectors from the layer prior to the output layer on each image of your task.
- **Stage 3:** Train a new classifier with those features as input for your own task.

At first glance, this seems even more complicated than just training the deep learning model. However, **Stage 1** is reusable for many different problems, and GraphLab is hosting the model so you don't have to train it yourself. **Stage 2** is easy to do with GraphLab's API (as shown below), and **Stage 3** is typically done with a simpler classifier than a deep learning model so it's easy to build yourself. In the end, this pipeline results in not needing to adjust hyper-parameters, faster training, and better performance even in cases where you don't have enough data to train a conventional deep learning model. What's more, this technique is effective even if your **Stage 3** classification task is relatively unrelated to the task **Stage 1** is trained on.





http://graphlab.com/learn/gallery/notebooks/build_imagenet_deeplearning.html

This idea was first explored by [Donahue et al. \(2014\)](#), and was used for the [Dogs vs Cats](#) competition as described for nolearn's [ConvNetFeatures](#). In our `NeuralNetworkClassifier` API, we put this functionality into the [.extract_features\(\)](#)

method. Let's explore the GraphLab Create API on the Cats vs. Dogs dataset. To get a feel for what we're trying to accomplish, a few sample images from the dataset are shown below:



First, let's load in the model trained on ImageNet. This corresponds to the end of Stage 1 in our pipeline:

```
>>> pretrained_model = graphlab.load_model('http://s3.amazonaws.com/GraphLab-Datasets/deeplearning/imagenet_model_iter45')
```

Now, let's load in the cats vs dogs images. We resize because the original ImageNet model was trained on 256 x 256 x 3 images:

```
>>> cats_dogs_sf = graphlab.SFrame('http://s3.amazonaws.com/GraphLab-Datasets/cats_vs_dogs/cats_dogs_sf')
>>> cats_dogs_sf['image'] = graphlab.image_analysis.resize(cats_dogs_sf['image'], 256, 256, 3)
```

And extract features, per Stage 2 of our pipeline:

```
>>> cats_dogs_sf['features'] = pretrained_model.extract_features(cats_dogs_sf)
```

```
>>> cats_dogs_train, cats_dogs_test = cats_dogs_sf.random_split(0.8)
```

And now, let's train a simple classifier as described by Stage 3

```
>>> simple_classifier = graphlab.classifier.create(cats_dogs_train, features = ['features'], target = 'label')
```

And now, to see how our trained model did, we evaluate it:

```
>>> simple_classifier.evaluate(cats_dogs_test)
{'accuracy': 0.9545091779728652, 'confusion_matrix': ConfusionMatrix(
  columns:
      target_label    str
      predicted_label str
      count          int

  Rows: 4

  Data:
  +-----+-----+-----+

```

target_label	predicted_label	count
0	0	2406
0	1	73
1	0	155
1	1	2378

[4 rows x 3 columns]}

We get ~**96%** accuracy! I don't know about you, but that feels like a pretty good number. For comparisons sake, let's try using just the `.create()` method.

```
>>> model = gl.neuralnet_classifier.create(cats_dogs_train, target='label', features = ['image'], validation_set=cats_dogs_test )
>>> model.evaluate(cats_dogs_test)
{'accuracy': 0.6049019694328308, 'confusion_matrix': C
olumns:
    target_label    int
    predicted_label int
    count          int

Rows: 4
```


Data:

target_label	predicted_label	count
0	0	922
1	0	415
0	1	1600
1	1	2163

[4 rows x 3 columns]}

Test accuracy is a disappointing **60%**. This means that there is either not enough data in the Cats vs. Dogs dataset to train the network or that the network architecture chosen by `.create()` is not complex enough. In any case, it's clear that extracting features with a network trained on ImageNet challenge helped tremendously. And you STILL didn't have to tune architecture or hyper-parameters. You don't even have to take the time to train a NeuralNet classifier, you can just repurpose one that already existed. Sounds like if using `.create()` was Easy, then using `.extract_features()` is Doubly Easy!

Making Doubly Easy also Doubly Powerful

It's always important to make sure any machine learning technique is consistent in its usefulness, and that its success is not a fluke. In order to do that, I tested it on the Caltech-101 dataset as described by [Donahue et al. \(2014\)](#). The Caltech-101 dataset has pictures of objects belonging to 101 categories, plus a junk category for a total of 102 classes. Each category contains between 40 and 800 images. A few examples are shown below:





Let's repeat the procedure we just went through for the Cats vs Dogs dataset, with a random 30 images per category used for training, and the rest for testing:

```
>>> caltech_train = graphlab.SFrame('http://s3.amazonaws.com/GraphLab-Datasets/caltech_101/caltech_101_train_sframe')
>>> caltech_test = graphlab.SFrame('http://s3.amazonaws.com/GraphLab-Datasets/caltech_101/caltech_101_test_sframe')
# preprocess
>>> caltech_train['image'] = graphlab.image_analysis.resize(caltech_train['image'], 256, 256, 3)
>>> caltech_test['image'] = graphlab.image_analysis.resize(caltech_test['image'], 256, 256, 3)
# Stage 2
```

```

>>> caltech_train['features'] = pretrained_model.extra
ct_features(caltech_train)
>>> caltech_test['features'] = pretrained_model.extrac
t_features(caltech_test)
# Stage 3
>>> classifier = graphlab.classifier.create(caltech_t
rain, features=['features'], target='label')
# Evaluate
>>> classifier.evaluate(caltech_test)

```

And evaluate:

```

{'accuracy': 0.8412228796844181, 'confusion_matrix': C
olumns:

```

```

    target_label    str
    predicted_label str
    count         int

```

Rows: 478

Data:

target_label	predicted_label	count
BACKGROUND_Google	Faces	15
BACKGROUND_Google	Faces_easy	2
BACKGROUND_Google	Leopards	3

BACKGROUND_Google	accordion	2	
BACKGROUND_Google	airplanes	1	
BACKGROUND_Google	anchor	5	
BACKGROUND_Google	ant	10	
BACKGROUND_Google	barrel	7	
BACKGROUND_Google	bass	6	
BACKGROUND_Google	beaver	1	
...	

+-----+-----+-----+

[478 rows x 3 columns]

Note: Only the head of the SFrame **is** printed.

You can use `print_rows(num_rows=m, num_columns=n)` to **print** more rows **and** columns.}

We get almost **85%** accuracy! This is on par with [highly complex, state-of-the-art methods](#). Clearly, feature extraction makes deep learning not only Doubly Easy, but also Doubly Powerful.

Deep learning Models are powerful, and are now easier to use than ever before. Download GraphLab Create , load in our ImageNet model, and tell us your deep learning success stories!

References

Donahue, J., Jia, Y., Vinyals, O., Homan, J., Zhang, N., Tzeng, E., and Darrell, T. DeCAF: A deep convolutional activation feature for generic visual recognition. In JMLR, 2014.

Krizhevsky, A., Sutskever, I. and Hinton, G. E.
ImageNet Classification with Deep Convolutional Neural Networks
NIPS 2012: Neural Information Processing Systems, Lake Tahoe, Nevada

Yang, J., L., Y., Tian, Y., Duan, L., and Gao, W. Group-sensitive multiple kernel learning for object categorization. In ICCV, 2009.

Editor's Note: This blog post was originally published in December, 2014. It was updated for accuracy in February,

2015 to correct an error in experimental results, and in March, 2015 to update links to cited papers.

Comments

Tangquan Qi Sun Jan 18 08:08:41 2015

this is a good tool.

↩ Reply to *Tangquan Qi*

Mark Dickson Mon Jan 26 12:55:26 2015

Hi Piotr,

Thank you for your very instructive post! I successfully replicated most of your results, but I did not achieve an accuracy close to 95% for the CIFAR dataset. I used the

code in your post, and here is the output for
classifier.evaluate(cifar_test):

{'accuracy': 0.711, 'confusion_matrix': Columns:
target_label int
predicted_label int
count int

Rows: 100

Data:

+-----+-----+-----+		
target_label	predicted_label	count
+-----+-----+-----+		
0	0	737
0	1	31
0	2	40
0	3	22
0	4	12
0	5	7
0	6	9
0	7	9

| 0 | 8 | 98 |

| 0 | 9 | 35 |

| ... | ... | ... |

+-----+-----+-----+

[100 rows x 3 columns]

Note: Only the head of the SFrame is printed.

You can use `print_rows(num_rows=m, num_columns=n)`
to print more rows and columns.}

Do you have any insights? I am also curious as to why your accuracy for label '0' somewhat less than for the rest of the dataset.

Mark Dickson

↩ Reply to *Mark Dickson*

Piotr Teterwak Mon Jan 26 20:20:00 2015

Hi Mark!

I'm glad you enjoyed the post!

I just took a look and confirmed that there is a

discrepancy between versions of GraphLab Create. I'll look into this further, thanks for bringing it up!

Cheers!

-Piotr

↩ Reply to *Piotr Teterwak*

Tarek Abdunabi Sun Feb 22 05:30:56 2015

Hi Piotr,

Thank you for your very interesting post.

I am considering using GraphLab tools in my PhD research, and have a question regarding extracting features using the model trained on ImageNet.

Does the step of extracting features from a new dataset requires a GPU? If so, is it possible to have the Dato Distributed service (On-Demand - \$3/hour/machine) with GPU support?

Thanks,

Tarek

← Reply to *Tarek Abdunabi*

Piotr Teterwak Wed Feb 25 16:24:58 2015

Hi Tarek,

I'm glad you enjoyed the post! Extracting features is much quicker and more feasible with a GPU.

Academic use of GraphLab Create is free, per

<https://dato.com/products/index.html>.

You could simply spin up an AWS instance and install the GPU egg!

Cheers!

-Piotr

← Reply to *Piotr Teterwak*

Tarek Wed Feb 25 18:53:58 2015

Thank you Piotr for your informative reply.

I've requested further info regarding Dato Distributed via the

"Contact us" form.

Cheers!

Tarek

↩ Reply to *Tarek*

Jensen Mon Mar 30 17:00:47 2015

Thank you Piotr. I also enjoyed your talk at GTC 2015, especially the last part of image search. But I cannot find this content here, is it possible to share it with us or tell us where to find? I would appreciate you very much. Thanks

↩ Reply to *Jensen*

Piotr Teterwak Mon Mar 30 19:06:05 2015

Hi Jensen,

I'm glad you enjoyed the talk at GTC! You can re-watch the talk at <http://on-demand.gputechconf.com/gtc/2015/video/S5630.html>.

There is also an example of image similarity search in

our Gallery:

https://dato.com/learn/gallery/notebooks/image_similarity.html

Here's an example on how to use image similarity to analyze architectural styles of buildings:

https://dato.com/learn/gallery/notebooks/bcn_buildings_graphlab.html

I hope this helps! Don't hesitate to ask if you have any more questions!

Cheers!

-Piotr

👉 Reply to *Piotr Teterwak*

Stefanie Mon Oct 19 14:45:34 2015

Hi Piotr!

Thank you for your blog post, very informative! I have one question, is it possible to use a pre-trained model (ImageNet) for predicting images which are not a typical category in the database? Like finding cells or galaxies?

Thanks

↩ Reply to *Stefanie*

Piotr Teterwak Mon Oct 19 21:13:53 2015

Hi Stefanie,

In general, the further you drift from the existing categories, the worse results you get. However, that said, ImageNet-trained models work surprisingly well on a very wide variety of image domains, and I wouldn't be too surprised if features extracted by an ImageNet model worked reasonably well for both of those domains.

Cheers!

-Piotr

↩ Reply to *Piotr Teterwak*

Stefanie Tue Oct 20 04:54:14 2015

Thanks for your answer, Piotr!

Just for curiosity, how many images per category would be required for a good neural network model? Millions or might thousands of images be enough?
Stefanie

↩ Reply to *Stefanie*

Piotr Teterwak Mon Nov 2 12:26:26 2015

Hi Stefanie,

Thousands may be enough, depending on the task. Not many people have million images per category (though you might have over a million total).

Hope this helps !

-Piotr

↩ Reply to *Piotr Teterwak*

Daniel Lee Thu Oct 29 23:15:19 2015

Hello Piotr:

Do you think ImageNet model worked well for medical

related images, i.e. retina images and breast image etc,,

Cheers,

Daniel

↩ Reply to *Daniel Lee*

Daniel Lee Thu Oct 29 23:55:01 2015

Hello Piotr:

Do you think ImageNet model worked well for medical related images, i.e. retina images and breast image etc,,

Cheers,

Daniel

↩ Reply to *Daniel Lee*

Piotr Teterwak Mon Nov 2 12:28:38 2015

Hi Daniel,

I'm guessing you might be better of learning a network from scratch with medical images, since the domain is so different from imagenet, but it never hurts to give it a shot. In any Computer Vision problem, I always start with extracted ImageNet features to see how far we can get.

Hope this helps!

Cheers!

-Piotr

✉ Reply to *Piotr Teterwak*

Daniel Lee Tue Nov 3 21:35:28 2015

Hello Piotr:

Are there other pre-trained models available@Dato besides the pre-trained imagenet_model_iter45?

Cheers,

Daniel

↩ Reply to *Daniel Lee*

Kai Schaffer Thu Nov 12 07:35:58 2015

Hello Piotr:

I followed the procedures and there are always some errors in certain steps.

Firstly, evaluating the model on the prediction data did give me 97% of accuracy. However, when I tried it with the EC2 on Amazon, the result became around 10%. The codes are same so I did not understand why there is a huge difference between two test results.

Secondly, when I tried the following codes, it always returned an errors:

```
cats_dogs_sf['features'] =
```

```
pretrained_model.extract_features(cats_dogs_sf)
```

```
cats_dogs_train, cats_dogs_test =  
cats_dogs_sf.random_split(0.8)
```

The error message is "RuntimeError: Communication Failure: 113."

Since the error cannot be corrected, I was not able to continue later procedures. It would be great if you can help out with this.

Thank you.

✉ Reply to *Kai Schaffer*

Piotr Teterwak Thu Nov 12 14:36:01 2015

Hi Kai,

Could you send a support ticket to support@dato.com with the details of the problem? We will help you out there.

Cheers!

-Piotr

✉ Reply to *Piotr Teterwak*

First Name

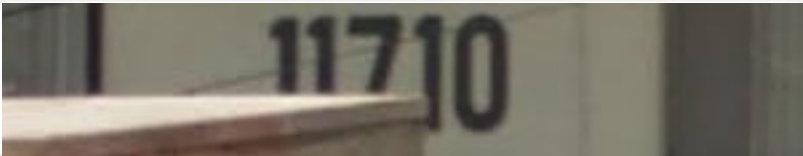
Last Name

Email



Website

Comment

☐ Subscribe to follow-up comments for this post



[Privacy & Terms](#)



Submit Comment

ABOUT

Contact

Team

Blog

Careers

Press

Investors

CONNECT WITH US



SUBSCRIBE TO OUR MACHINE LEARNING BLOG

Get data science insights and best practices from the Dato team sent right to your inbox.

Enter email to subscribe

SUBSCRIBE

© 2015 Dato · All Rights Reserved. No part of this website may be reproduced without Dato's expressed consent. Dato, GraphLab, GraphLab Create and logos are property of Dato. [Privacy Policy](#) | [Terms of Use](#)