DEEP LEARNING AND GPU PARALLELIZATION IN JULIA

2015.10.28 18.337 Guest Lecture

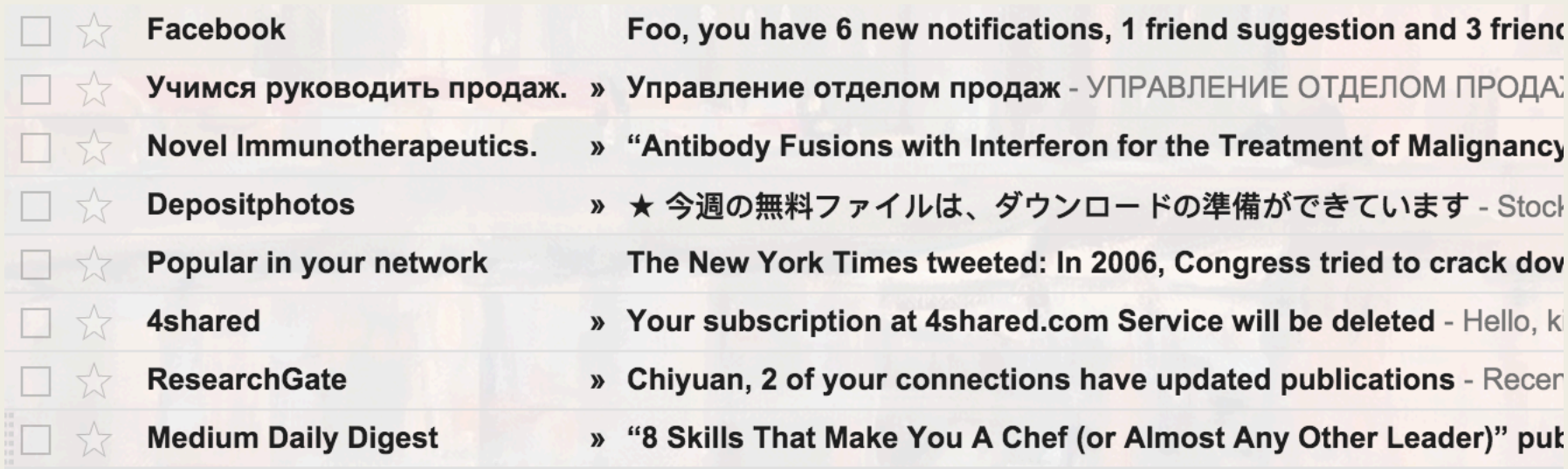
Chiyuan Zhang
CSAIL, MIT

MACHINE LEARNING AND DEEP LEARNING

A very brief introduction

What is Machine Learning?

- Typical machine learning example: email spam filtering



What is Machine Learning?

- Traditional Rule-based spam filtering:

```
for word in email
   if word ∈ [“buy”, “$$$”, “100% free”]
      return :spam
   end
end
return :good
```

- Issues
  - *Growing list of spam-triggering keywords*
  - *Longer word-sequences needed for higher accuracy, and rules could become very complicated and hard to maintain*
  - *…*

What is Machine Learning?

- Machine learning: training a model from examples
  - *Input 1: training data with labels, including spam email examples and good email examples, marked by human labeler as "spam" or "good"*
  - *Input 2: a parametric (usually probabilistic) model, describing a function*
    $$f_\theta: \mathcal{X} \rightarrow \{\pm 1\}$$
    where $\mathcal{X}$ is the space of all emails, +1 indicate good emails, and -1 indicate spam emails. θ is the parameters of the model, that is to be decided.
  - *Input 3: a cost function: $C(y, \hat{y})$, measuring the cost of predicting as $\hat{y}$ when the true label is y.*
  - *Training: essentially solving*
    $$\min_{\theta \in \Theta} \frac{1}{N} \sum_{i=1}^{N} C\left(y_i, f_\theta(x_i)\right)$$

Example: the Naïve Bayes Model

$$
\begin{aligned}
f_\theta(x) &= \operatorname{argmax}_{y\in\{\pm 1\}} \mathbb{P}_\theta(y|x) \\
&= \operatorname{argmax}_{y\in\{\pm 1\}} \mathbb{P}_\theta(x|y)\mathbb{P}_\theta(y) \\
&= \operatorname{argmax}_{y\in\{\pm 1\}} \mathbb{P}_\theta(y) \prod_{j=1}^{M} \mathbb{P}_\theta(x_j|y)
\end{aligned}
$$

- Each x_i is the count of a specific word (e.g. “buy”) in our vocabulary
- The parameters θ encodes all the conditional probabilities, e.g. $\mathbb{P}_\theta(buy|\text{spam}) = 0.1$, $\mathbb{P}_\theta(buy|\text{good}) = 0.001$.
- The optimal θ is “learned” automatically from the examples in the training set.
- In practice, more complicated models can be built and used.
- Statistical and computational learning theory: learnability and performance gurantee.

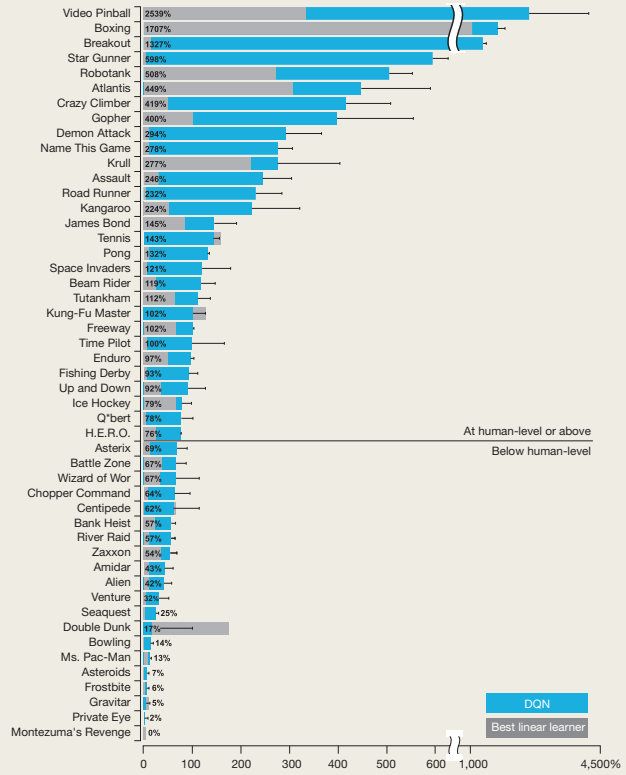
Machine Learning in the Wild

- Computer Vision
  - *Image classification: face recognition, object category identification*
  - *Image segmentation: find and locate objects, and carve out their boundaries*
  - *Scene understanding: high-level semantic information extraction*
  - *Image captioning: summarize an image with a sentence*

Andrej Karpathy and Li Fei-Fei. Deep Visual-Semantic Alignments for Generating Image Descriptions. CVPR 2015.

Machine Learning in the Wild

- Speech Recognition
  - *Input: audio signals; output: text transcription*
  - *Apple Siri, Google Now, Microsoft Cortana*
- Natural Language Processing
  - *Semantic parsing: output is syntax trees*
  - *Machine translation: output is a sentence in another language*
  - *Sentiment analysis: output "positive" or "negative"*
- Artificial Intelligence
  - *Google deepmind: reinforcement learning for playing video games*



Google Deep Mind. Human-level control through deep reinforcement learning. Nature, Feb. 2015.

What is Deep Learning then?

- Designing a good model is difficult
- Recall the *Naïve Bayes* model
  - *The prediction is parameterized by the probability of each word conditioned on the document being a spam or a good email.*
  - *The count of words in a (fixed) vocabulary is what we are looking at, those are called features or representations of the input data.*
  - *Two representations could contain the same information, but still be "good" or "bad", for a specific task.*
- Example: representations of a number

```
In [1]: a1 = 1234
        a2 = factor(a1)

        b1 = 2456
        b2 = factor(b1)

        println("$a1 + $b1 = $(a1+b1)")
        println("$a2 * $b2 = $(factor(a1*b1))")

        1234 + 2456 = 3690
        Dict(2=>1,617=>1) * Dict(2=>3,307=>1) = Dict(2=>4,307=>1,617=>1)
```

What is Deep Learning then?

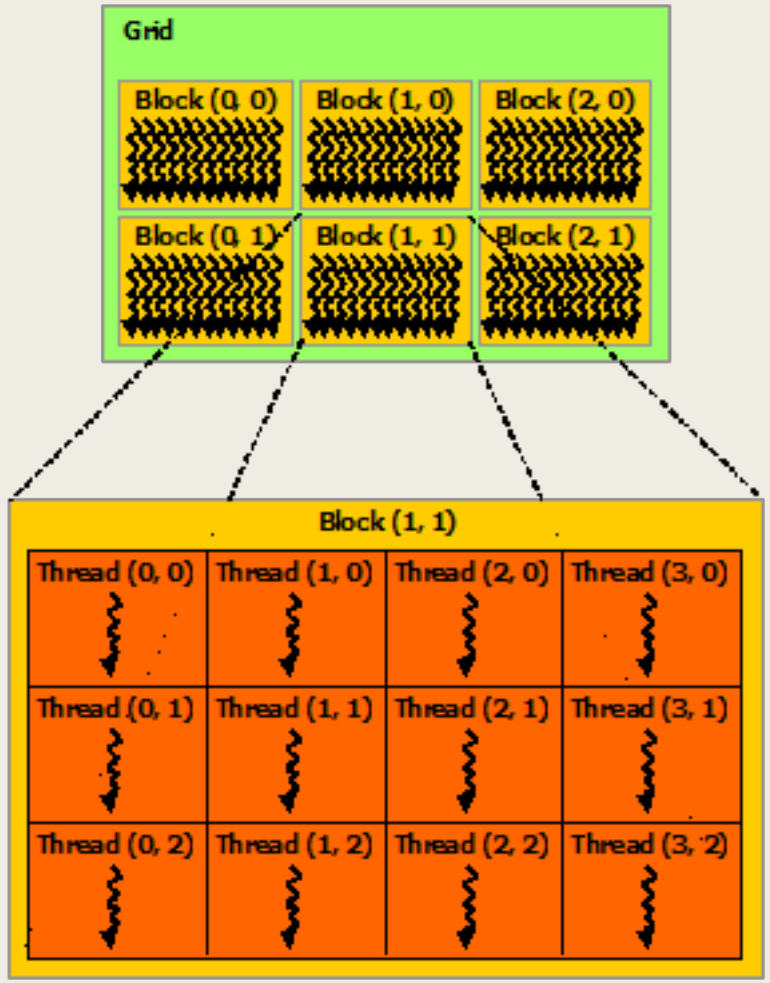
- Depending on the quality of the features, the learning problem might become easy or difficult.
- What features to look at when the input are complicated or unintuitive?
  - *E.g. for image input, looking at the raw pixels directly is usually not very helpful*
- Feature designing / engineering used to be a very important part of machine learning applications.
  - *SIFT in computer vision*
  - *MFCC in speech recognition*
- Deep Learning: learning both the representations and the model parameters automatically and jointly from the data.
  - *Recently become possible with huge amount of data (credit: internet, mobile devices, Mechanic Turk, …) and highly efficient computing devices (GPUs, …)*

DEEP LEARNING AND GPU PARALLELIZATION

In Julia... a tiny introduction

GPUs vs. CPUs

	CPUs	GPUs
Typical number of cores	Dozens of	Thousands of
Features	General purpose computing	“General” purpose computing
Parallelization	Arbitrarily complicated scheduling of different processes and threads performing heteogeneous tasks	All cores run the same “kernel” function, without or with very limited communication or sharing.
Example	One thread classifying emails and one thread displaying them in a GPU	Computing `max(X, 0)`, each core taking care of 1 element in the matrix X.



Several Facts

- Many machine learning and deep learning algorithms fits nicely with GPU parallilization models: simple logic but massive parallel computation.
- Training time large deep neural networks:
  - *From ∞ (or probably finite, but takes years, nobody was able to do it in pre-GPU age)*
  - *To weeks or even days, with optimally designed models, computation kernels, IO, and multi-GPU parallizations*
- Julia is primarily designed for CPU parallelization and distributed computing, but GPU computing in Julia is gradually getting there
  - *https://github.com/JuliaGPU*

Deep Learning in Julia

- Now there are several packages available in Julia with GPU supports
  - *Mocha.jl*: https://github.com/pluskid/Mocha.jl
    Currently the most feature complete one. Design and architecture borrowed from the Caffe deep learning library.
  - *MXNet.jl*: https://github.com/dmlc/MXNet.jl
    A successor of Mocha.jl. Different design, with a language-agnostic C++ backend dmlc/libmxnet. Relatively new but very promising, with flexible symbolic API and efficient multi-GPU training support.
  - *Knet.jl*: https://github.com/denizyuret/Knet.jl
    Experimental symbolic neural network building script compilation.

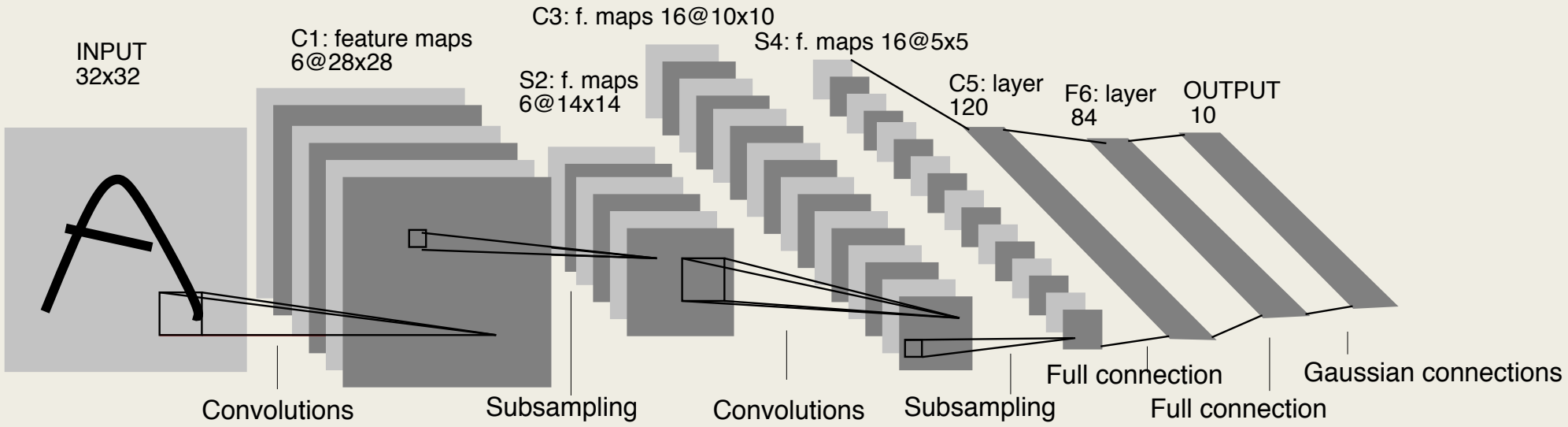
IMAGE CLASSIFICATION IN JULIA

A tutorial with MXNet.jl

Hello World: Handwritten Digits

- MNIST handwritten digit dataset
  - http://yann.lecun.com/exdb/mnist/
- Each digit is a 28-by-28 grayscale image
- 10 target classes: 0, 1, …, 9
- 60,000 training images, and 10,000 test images
- Considered as a fairly easy task nowdays, the “sanity-check” task for many machine learning algorithms

A Convolutional Neural Network: LeNet



LeCun, Yann, et al. "Gradient-based learning applied to document recognition." Proceedings of the IEEE 86.11 (1998): 2278-2324.

- A classical model invented by Yann LeCun, called the LeNet.
- Chain of convolution and pooling operations, followed by densely connected neural network layers.

What is Convolution and Pooling?

- Convolution:
  - *Basically pattern matching across spatial locations, but…*
  - *The patterns (filters) are not designed a priori, but learned from the data and task.*
- Pooling:
  - *Accumulating local statisitcs of filter responses from the convolution layer.*
  - *Leads to local spatial invariance for the learned patterns.*

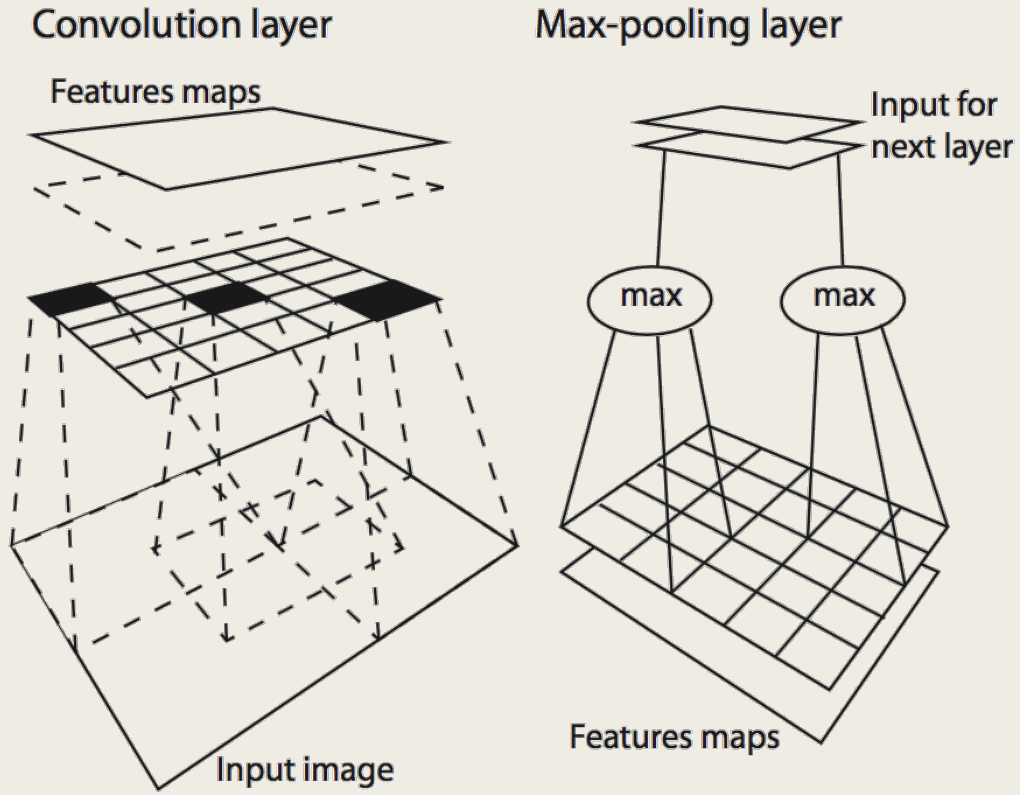


Image source: http://inspirehep.net/record/1252539

The LeNet in MXNet.jl

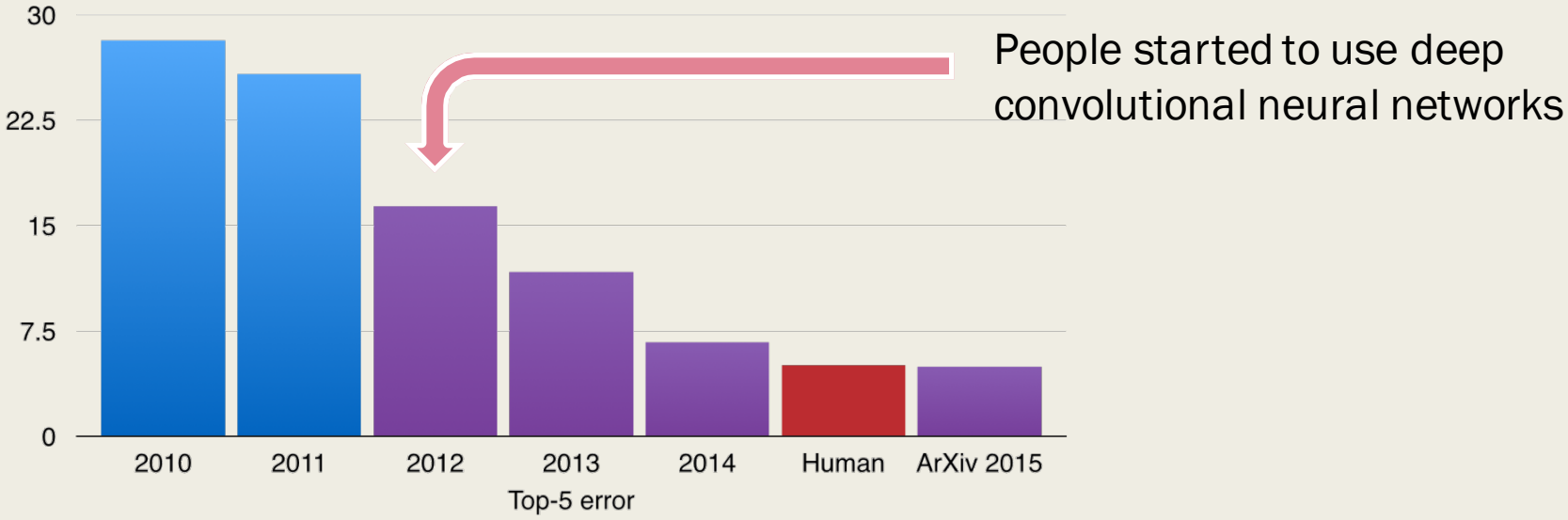
```
using MXNet

#--------------------------------------------------------------------------------
# define lenet

# input
data = mx.Variable(:data)

# first conv
conv1 = @mx.chain mx.Convolution(data=data, kernel=(5,5), num_filter=20)  =>
                  mx.Activation(act_type=:tanh) =>
                  mx.Pooling(pool_type=:max, kernel=(2,2), stride=(2,2))

# second conv
conv2 = @mx.chain mx.Convolution(data=conv1, kernel=(5,5), num_filter=50) =>
                  mx.Activation(act_type=:tanh) =>
                  mx.Pooling(pool_type=:max, kernel=(2,2), stride=(2,2))

# first fully-connected
fc1   = @mx.chain mx.Flatten(data=conv2) =>
                  mx.FullyConnected(num_hidden=500) =>
                  mx.Activation(act_type=:tanh)

# second fully-connected
fc2   = mx.FullyConnected(data=fc1, num_hidden=10)

# softmax loss
lenet = mx.Softmax(data=fc2, name=:softmax)
```

Loading the Data and Training the Model (Stochastic Gradient Descent)

```
31  #--------------------------------------------------------------------------------
32  # load data
33  batch_size = 100
34  include("mnist-data.jl")
35  train_provider, eval_provider = get_mnist_providers(batch_size; flat=false)
36
37  #--------------------------------------------------------------------------------
38  # fit model
39  model = mx.FeedForward(lenet, context=mx.gpu())
40
41  # optimizer
42  optimizer = mx.SGD(lr=0.05, momentum=0.9, weight_decay=0.00001)
43
44  # fit parameters
45  mx.fit(model, optimizer, train_provider, n_epoch=20, eval_data=eval_provider)
```

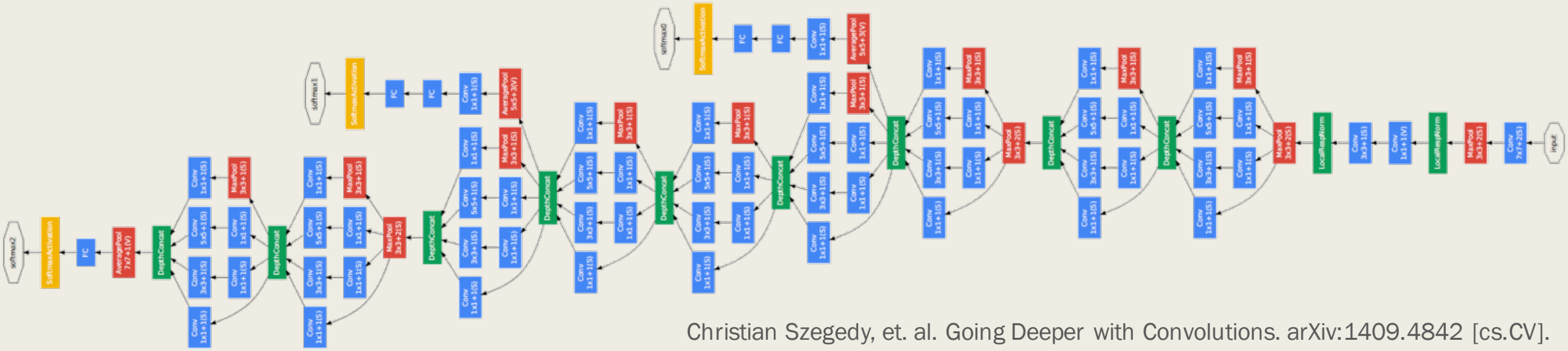
A More Interesting Example: Imagenet

- The Imagenet dataset: http://www.image-net.org/
  - 14,197,122 full-resolution images, 21,841 target classes
  - Challenges every year (Imagenet Large Scale Visual Recognition Challenge, ILSVRC)
  - A smaller subset with ~1,000,000 images and 1,000 categories is typically used



The Google “Inception” Model

- Winner of ILSVRC 2014, 27 layers, ~7 million parameters
- With a highly optimized library, on 4 GPU cards, training a similar model takes 8.5 days (see http://mxnet.readthedocs.org/en/latest/tutorial/imagenet_full.html)



Christian Szegedy, et. al. Going Deeper with Convolutions. arXiv:1409.4842 [cs.CV].

Image Classification with a Pre-trained Model

- Because we cannot have a 8.5-day long class…
- We will show a demo on using pre-trained model to do image classification
- The IJulia Notebook is at: http://nbviewer.ipython.org/github/dmlc/MXNet.jl/blob/master/examples/imagenet/ijulia-pretrained-predict/Prediction%20with%20Pre-trained%20Model.ipynb

In [13]:
```
pred = mx.predict(model, mx.ArrayDataProvider(img))
classes = open(joinpath(model_dir, "synset.txt")) do s
    map(x -> replace(strip(x), r"^n[0-9]+ ", ""), readlines(s))
end
println(classes[indmax(pred)])
```

```
Egyptian cat
```

GPU Programming in Julia: Status

- High-level programming APIs
  - *CUFFT.jl, CUBLAS.jl, CLBLAS.jl, CUDNN.jl, CUSPARSE.jl, etc…*
- Intermediate-level programming APIs
  - *CUDArt.jl, OpenCL.jl*
  - *Write kernel functions in C++, but high-level program logic in Julia*
- Low-level programming APIs
  - *Using Julia FFI, to call into libcudart.so etc.*

```
ccall((:cuLaunchKernel, “libcuda”),
(Ptr{Void}, …), kernel_hdr, gx, gy, ...)
```

```
import OpenCL
const cl = OpenCL

const sum_kernel = "
   __kernel void sum(__global const float *a,
                     __global const float *b,
                     __global float *c)
    {
      int gid = get_global_id(0);
      c[gid] = a[gid] + b[gid];
    }
"
a = rand(Float32, 50_000)
b = rand(Float32, 50_000)

device, ctx, queue = cl.create_compute_context()

a_buff = cl.Buffer(Float32, ctx, (:r, :copy), hostbuf=a)
b_buff = cl.Buffer(Float32, ctx, (:r, :copy), hostbuf=b)
c_buff = cl.Buffer(Float32, ctx, :w, length(a))

p = cl.Program(ctx, source=sum_kernel) |> cl.build!
k = cl.Kernel(p, "sum")

cl.call(queue, k, size(a), nothing, a_buff, b_buff, c_buff)

r = cl.read(queue, c_buff)

if isapprox(norm(r - (a+b)), zero(Float32))
    info("Success!")
else
    error("Norm should be 0.0f")
end
```