Lecture Notes for Machine Learning in Python

Professor Eric Larson

Week 12, 13, 14 Deep Learning

Class logistics

- Questions on Dask/Graphlab?
- Last time, age calculation on the fly
  - see my solution on GitHub

Lecture Agenda

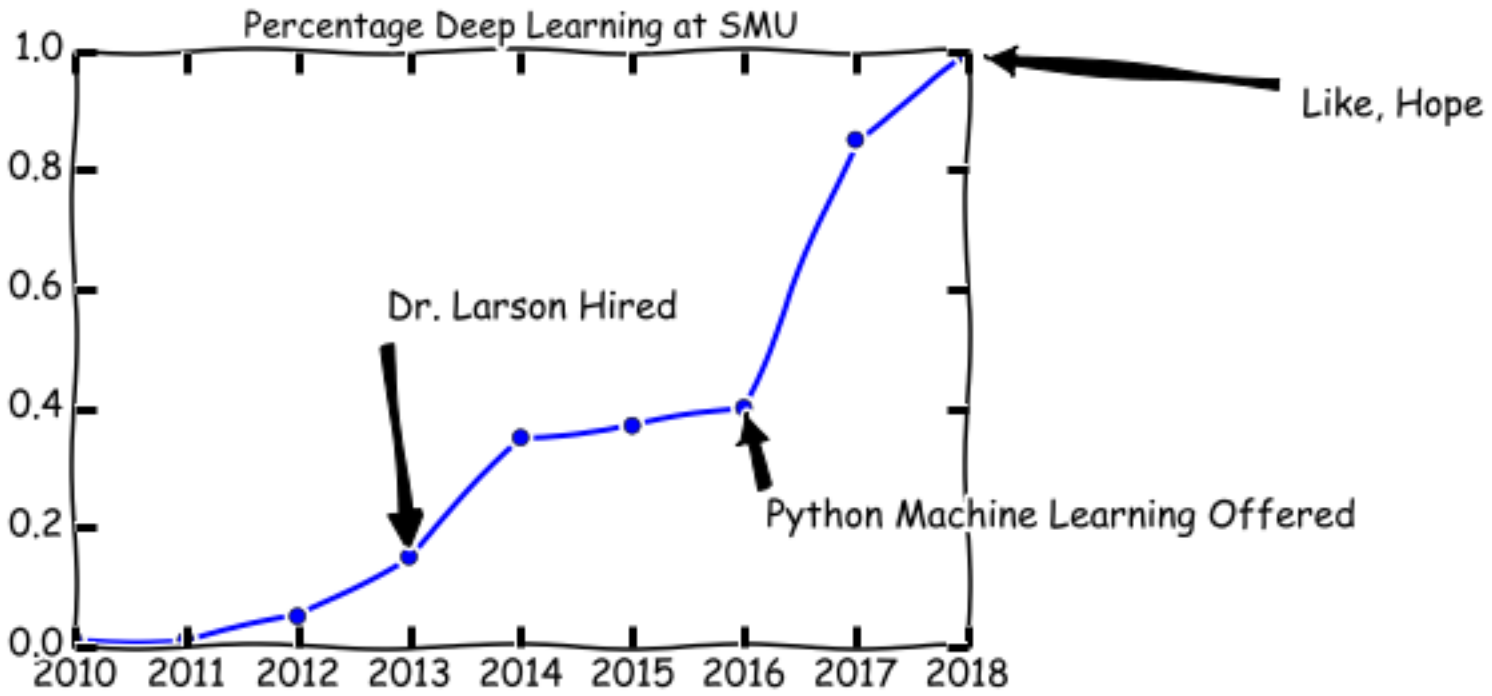
- Sampling and Modeling
- History
- Convolution Neural Networks
- Remaining Lectures
  - TensorFlow
  - Long- Short-term Memory Networks

Sampling and Modeling

Sampling and Modeling

- As of November 8th (morning) the highest rank for Trump winning the presidency was 33% (Nate Silver)
- Most other models were at <1% and were critical of *fivethirtyeight* (Nate Silver)
- **SELF TEST:** Why was this polling/modeling flawed?
  - (A) Insufficient sampling in key geographic areas
    - Under-sampling voters in rural areas (more difficult to contact)
    - Under-sampling minority voters (assume mirroring to Obama)
  - (B) Reliance on Bayesian estimation models
  - (C) Impossible cross validation due to changing environment
  - (D) Insufficient previous examples, even if environment was not changing
- Was there a way to make a better model of the election outcome?
- Some ensemble models performed well, but used features that only included:
  - time incumbent was in office, approval ratings for incumbent, and GDP growth, (other economic factors decreased performance of models)
  - even these models had wide confidence intervals, no clear winner

Some History of Deep Learning

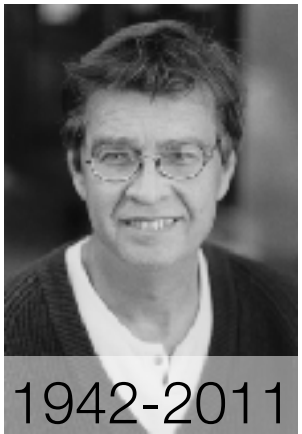



```
# Data to plot
dates = range(2010,2019)
percents = [0.01, 0.01, 0.05, 0.15, 0.35, 0.37, 0.40, 0.85, 0.99]

# Set the style to XKCD
plt.xkcd()

# Plot the percents
plt.plot(dates,percents, marker='o')
```

Neural Networks: Where we left it

- Before 1986: AI Winter
- 1986: *Rumelhart*, *Hinton*, and *Williams* popularize gradient calculation for multi-layer network
  - *technically* introduced by Werbos in 1982
- **difference**: Rumelhart *et al.* validated ideas with a computer
- until this point no one could train a multiple layer network consistently
- algorithm is popularly called **Back-Propagation**
- wins pattern recognition prize in 1993, becomes de-facto machine learning algorithm in the 90's

David Rumelhart



1942-2011

Geoffrey Hinton

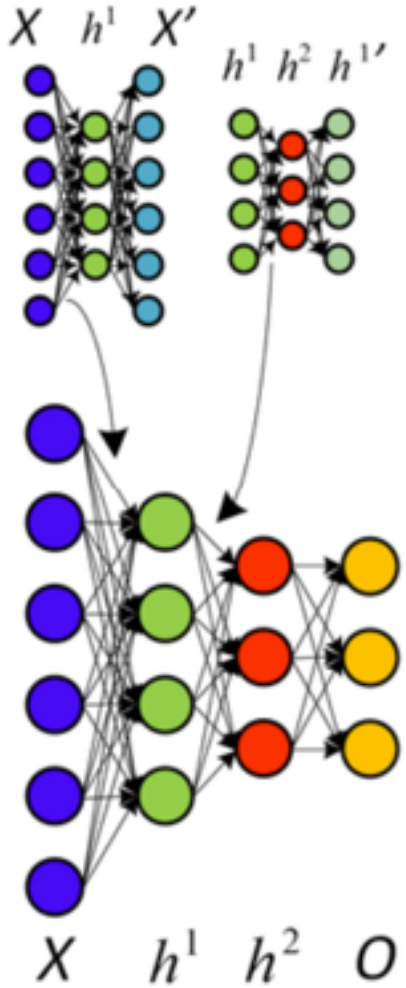
History of Deep Learning: Winter

- Up to this point: back propagation saved AI winter for NN (Hinton and others!)
- 80's, 90's, 2000's: neural networks for image processing start to get deeper
  - but back propagation no longer efficient for training
  - Back propagation gradient stagnates research—can't train deeper networks
- 2001: SVMs and Random Forests gain traction…
  - The second AI winter begins, research in NN plummets
  - Funding for and accepted papers that incorporate Neural Networks asymptotically approaches zero

Read this: http://www.andreykurenkov.com/writing/a-brief-history-of-neural-nets-and-deep-learning/

History of Deep Learning: Winter

- Winter is coming:



Easy to train, performs on par with other methods

Hard to train, performs worse than other methods

Researcher have difficulty reconciling expressiveness with performance

Read this: http://www.andreykurenkov.com/writing/a-brief-history-of-neural-nets-and-deep-learning/

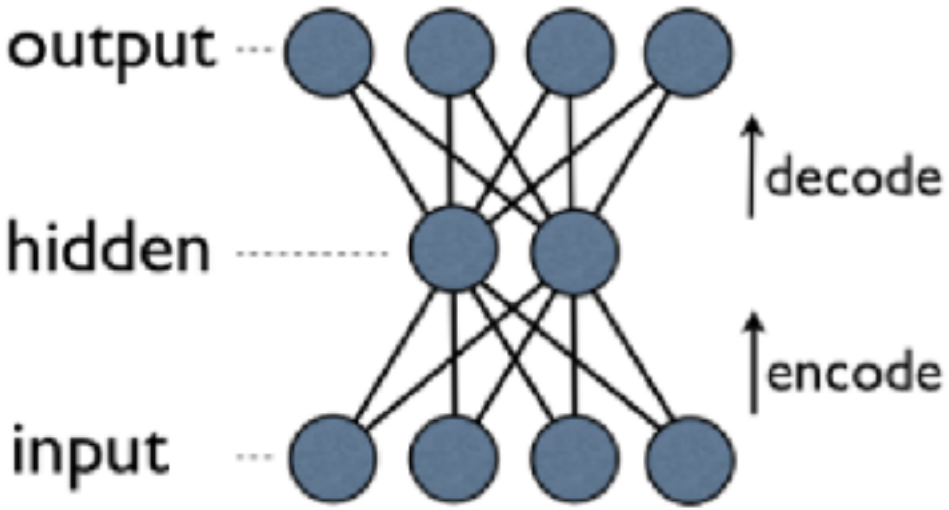
History of Deep Learning

- 2004: Hinton secures funding from CIFAR based on his reputation
  - *eventually*: Canada would be savior for NN
  - Hinton rebrands: Deep Learning
- 2006: Hinton publishes paper on using pre-training and Restricted Boltzmann Machines
- 2007: Another paper: Deep networks are more efficient when pre-trained
  - RBMs not really the important part

Read this: http://www.andreykurenkov.com/writing/a-brief-history-of-neural-nets-and-deep-learning/

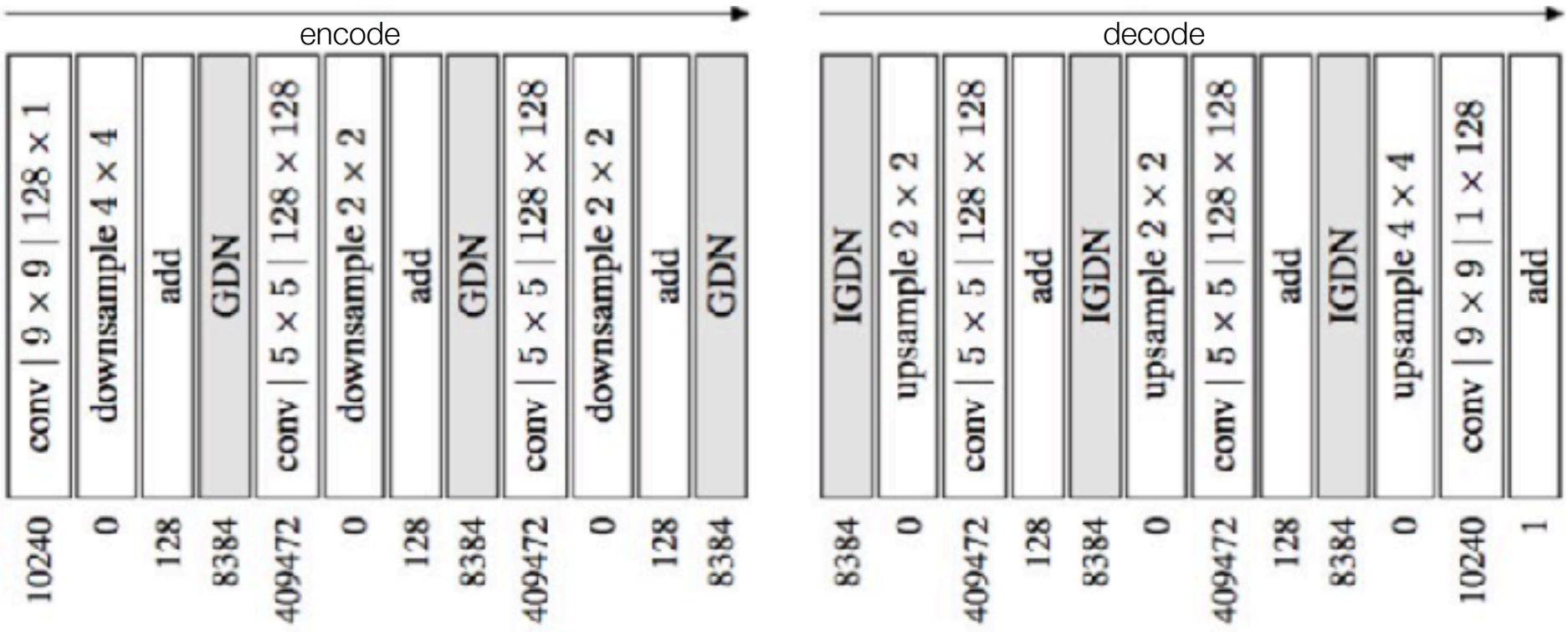
Pre-training: still in the long winter

- auto-encoding

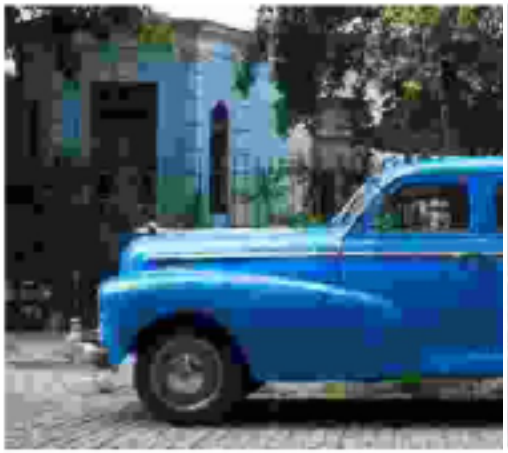


Pre-training: modern example

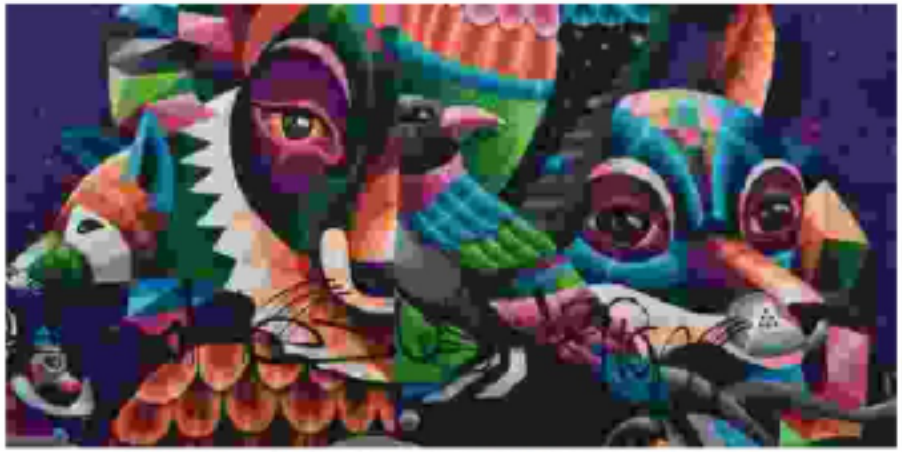
- auto-encoding: example from paper published Nov 5, 2016



https://arxiv.org/abs/1611.01704



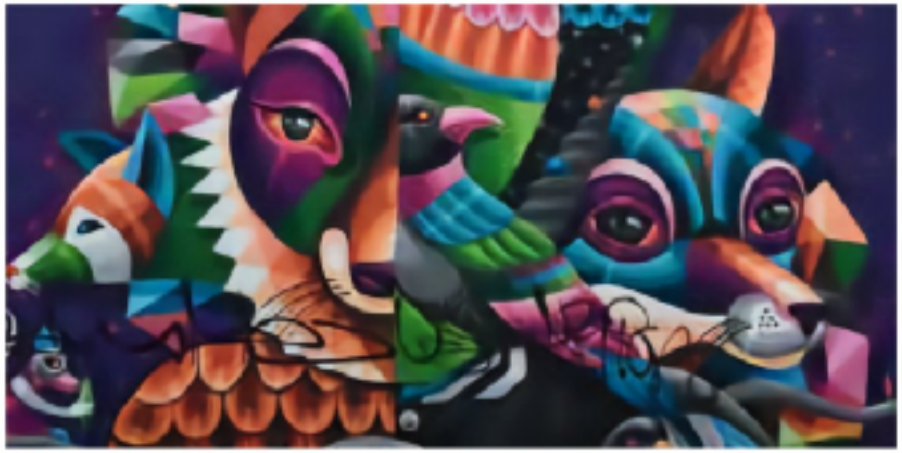
JPEG, 6006 bytes (0.170 bit/px), RMSE: 19.75



JPEG, 5928 bytes (0.168 bit/px), RMSE: 15.44/12.40, PSNR: 24.36 dB/26.26 dB

RMSE: 11.07/10.60, PSNR: 27.25 dB/27.63 dB

Proposed method, 5910 bytes (0.167 bit/px), RMSE



Proposed method, 5685 bytes (0.161 bit/px), RMSE: 10.41/5.98, PSNR: 27.78 dB/32.60 dB

it/px), RMSE: 6.10/5.09, PSNR: 32.43 dB/34.00 dB

JPEG 2000, 5918 bytes (0.167 bit/px), RMSE: 1

JPEG 2000, 5724 bytes (0.162 bit/px), RMSE: 13.75/7.02, PSNR: 25.36 dB/31.20 dB

x), RMSE: 8.56/5.71, PSNR: 29.49 dB/32.99 dB

History of Deep Learning

- 2009:
  - Andrew Ng gets involved
  - Hinton's lab start using GPUs
  - GPUs decrease training time by 70 fold…
- 2010: Hinton's and Ng's students go to internships with Microsoft, Google, IBM, and Facebook

Research at Google

Navdeep Jaitly
Google Brain Team

- Xbox Voice
- Android Speech Recognition
- IBM Watson
- DeepFace
- All of Baidu

George Dahl
Google Brain Team

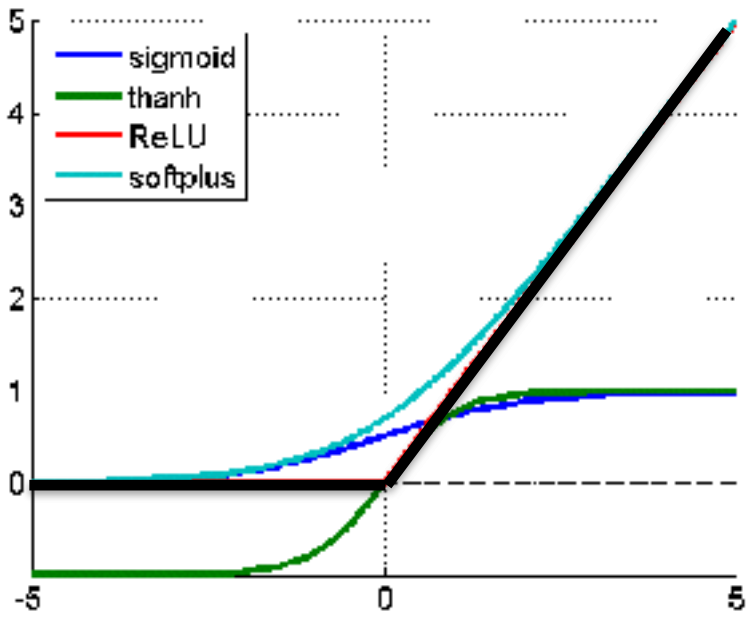
Abdel-rahman Mohamed
Microsoft Research
Redmond, Washington | Computer Software

Current	Microsoft
Previous	University of Toronto, IBM, Microsoft
Education	University of Toronto

Read this: http://www.andreykurenkov.com/writing/a-brief-history-of-neural-nets-and-deep-learning/

History of Deep Learning

- 2011: Glort and Y. Bengio investigate more systematic methods for why past deep architectures did not work
  - discover some interesting fixes: the type of neurons chosen and the selection of initial weights
  - do not require pre-training to get deep networks properly trained, just sparser representations and less complicated derivatives



$$\text{ReLU: } f(x) = \max(0,x)$$
$$f'(x) = 1 \text{ if } x>0 \text{ else } 0$$

that's a really easy gradient to compute!

…and it makes the weights more sparse

…and helps to solve the vanishing gradient problem

…and its inspired by biological vision community

Read this: http://www.andreykurenkov.com/writing/a-brief-history-of-neural-nets-and-deep-learning/

History of Deep Learning

- 2012: ImageNet competition occurs
  - **Second place**: 26.2% error rate
  - **First place**:
    - From Hinton's lab, uses convolutional neural networks with ReLU neurons and dropout
    - 15.2% error rate
  - Computer vision adopts deep learning with convolutional neural networks en mass

Fei Fei Li
Former
Director of Stanford's
AI Lab

I happened to witness this critical juncture in time first hand because the ImageNet challenge was over the last few years organized by Fei-Fei Li's lab (my lab), so I remember when my labmate gasped in disbelief as she noticed the (very strong) ConvNet submission come up in the submission logs. And I remember us pacing around the room trying to digest what had just happened. In the next few months ConvNets went from obscure models that were shrouded in skepticism to rockstars of Computer Vision, present as a core building block in almost every new Computer Vision paper.

History of Deep Learning

- 2012: Hinton Lab, Google, IBM, and Microsoft jointly publish paper, popularity for deep learning methods increases

Deep Neural Networks for Acoustic Modeling in Speech Recognition

[The shared views of four research groups]

[Geoffrey Hinton, Li Deng, Dong Yu, George E. Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N. Sainath, and Brian Kingsbury]

https://www.cs.toronto.edu/~gdahl/papers/deepSpeechReviewSPM2012.pdf

Read this: http://www.andreykurenkov.com/writing/a-brief-history-of-neural-nets-and-deep-learning/

History of Deep Learning

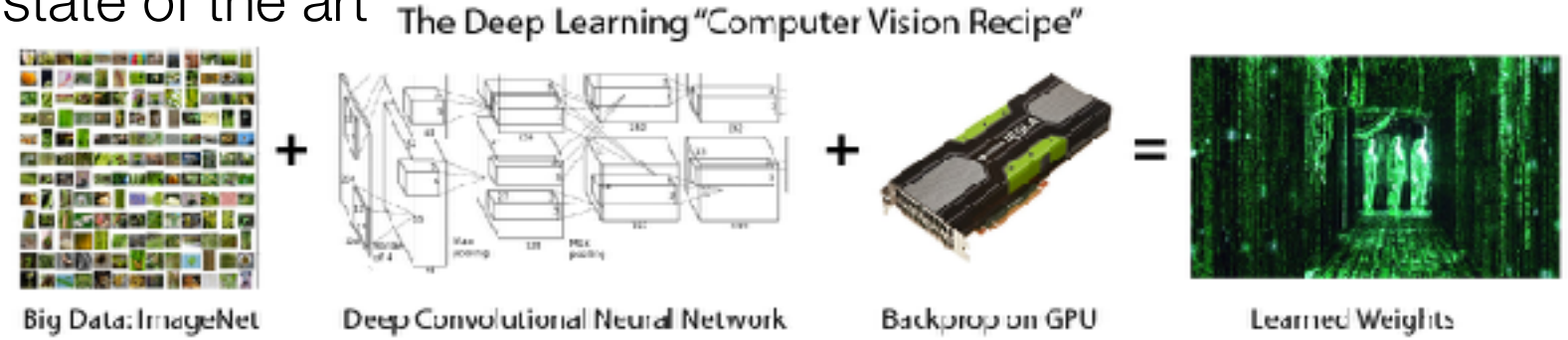
- 2013: Ng and Google (founded BrainTeam)
  - run unsupervised feature creation on YouTube videos (becomes computer vision benchmark)

The work resulted in unsupervised neural net learning of an unprecedented scale - 16,000 CPU cores powering the learning of a whopping 1 billion weights. The neural net was trained on Youtube videos, entirely without labels, and learned to recognize the most common objects in those videos.

Read this: http://www.andreykurenkov.com/writing/a-brief-history-of-neural-nets-and-deep-learning/

History of Deep Learning

- Hinton summarized what we learned in deep learning from the 2006 to present. Where we went wrong before present day:
  - labeled dataset were 1000s of times too small
  - computers were millions of times too slow
  - weights were initialized in stupid ways
  - we used the wrong non-linearities
- Or in my terms:
  - use a GPU, dropout, and ReLU neurons where it makes sense (like in early layers)
- Modern day deep learning uses simple, tried methods to achieve state of the art



Read this: http://www.andreykurenkov.com/writing/a-brief-history-of-neural-nets-and-deep-learning/

A summary of the Deep Learning people:

Stayed at University

Advises IBM

Heads Facebook AI Team

Google

Coursera
Baidu
Google

Read this paper from 2015, as it sums up advancements nicely

And predicts the future of deep learning

REVIEW

doi:10.1038/nature14539

Deep learning

Yann LeCun[1,2], Yoshua Bengio[3] & Geoffrey Hinton[4,5]

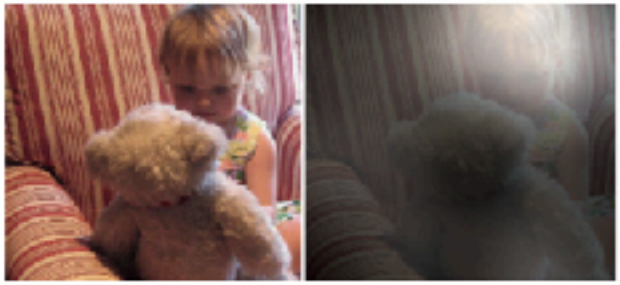
Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction. These methods have dramatically improved the state-of-the-art in speech recognition, visual object recognition, object detection and many other domains such as drug discovery and genomics. Deep learning discovers intricate structure in large data sets by using the backpropagation algorithm to indicate how a machine should change its internal parameters that are used to compute the representation in each layer from the representation in the previous layer. Deep convolutional nets have brought about breakthroughs in processing images, video, speech and audio, whereas recurrent nets have shone light on sequential data such as text and speech.

Famous examples:

A woman is throwing a **frisbee** in a park.

A **dog** is standing on a hardwood floor.

A **stop** sign is on a road with a mountain in the background

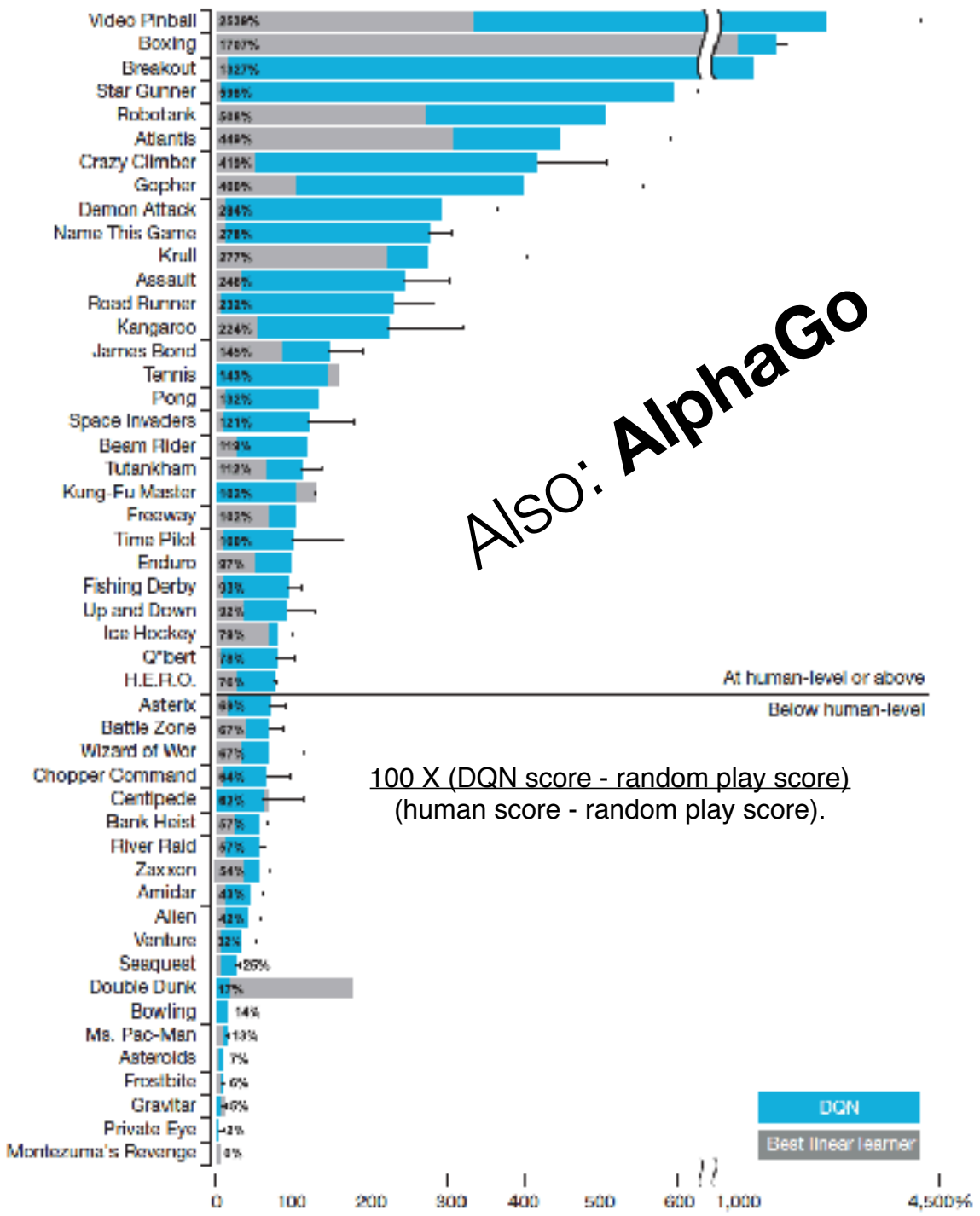


A little **girl** sitting on a bed with a teddy bear.

A group of **people** sitting on a boat in the water.

A giraffe standing in a forest with **trees** in the background.

Famous examp



Also: **AlphaGo**

100 X (DQN score - random play score)
(human score - random play score).

Convolutional Neural Networks

CNN Overview

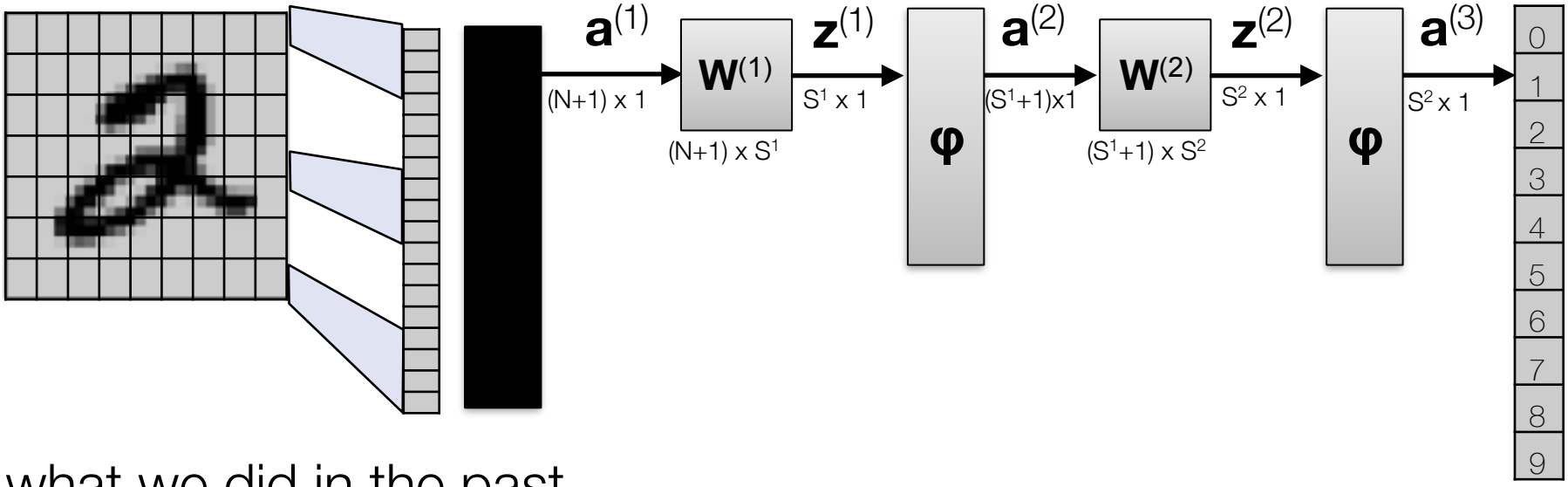
- First layer(s):
  - convolution with different filters
  - nonlinearity
  - pooling
  - Each pooling layer *can* make the input image "smaller"
    - allowing more summative explanations
- Final layers are densely connected
  - typically multi-layer perceptrons

CNN Overview: Self Test

- First layer(s):
  - convolution with different filters
  - nonlinearity
  - pooling
- Final layers are densely connected
  - typically multi-layer perceptrons

- Where are unstable gradients **most** problematic?
  - (A) During Convolution Layer(s) updates
  - (B) During Fully Connected Layer(s) updates
  - (C) Both A and B
  - (D) They are not a problem

From Fully Connected to CNN



what we did in the past

If image is 9x9, and each fully connected layer is 20 hidden neurons wide, how many parameters are in this NN (ignore bias)?

$(K^2 \times 20) + (20 \times 20) + (20 \times 10) = 600 + 20\ K^2$

for $9 \times 9 = 600 + 20 \times 9^2 =$ 2,220 parameters

Lecture Notes for Machine Learning in Python

Professor Eric Larson

Week 12, 13, 14 Deep Learning

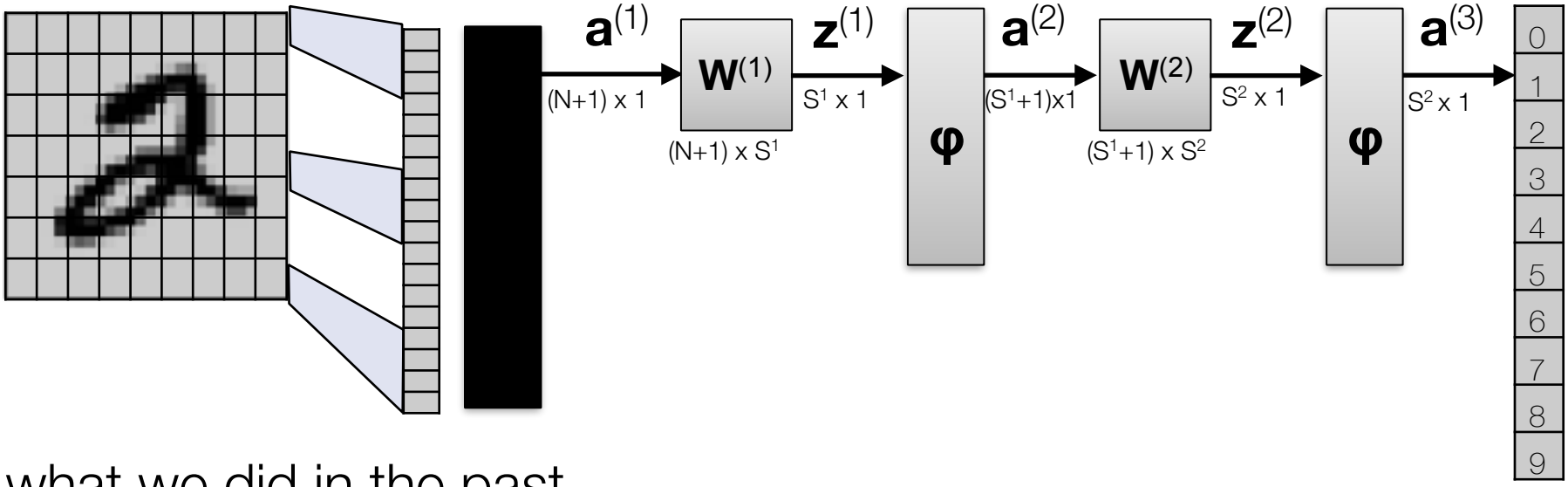
Class logistics

- Lab 3 has been up!
  - Due Dec 13 by 5PM
  - Sorry, No Resubmits for A3
  - No Project Presentations
- Grades will be coming in more steadily

Lecture Agenda

- Finish Convolutional Neural Networks
- Introduction to TensorFlow
  - Tensors
  - Namespaces
  - Numerical methods
- Demo of TensorFlow with CNNs
- Remaining lecture topics:
  - Long- Short-term Memory Networks

From Fully Connected to CNN



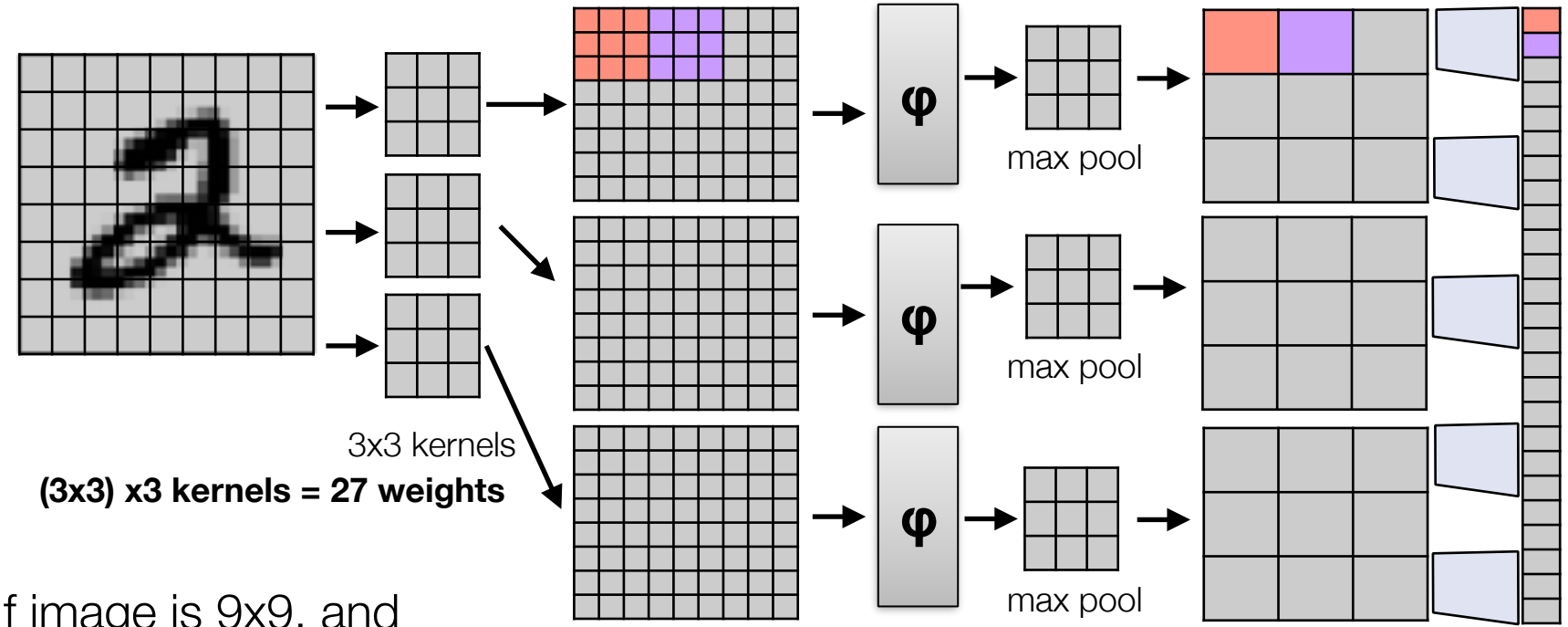
what we did in the past

If image is 9x9, and each fully connected layer is 20 hidden neurons wide, how many parameters are in this NN (ignore bias)?

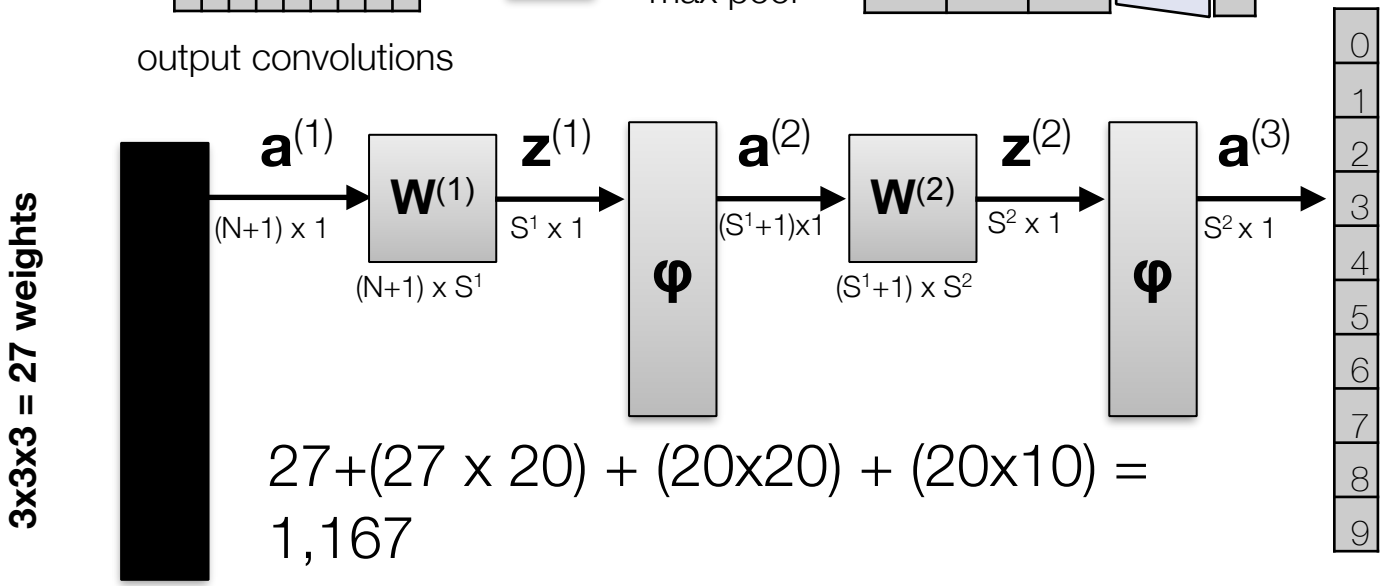
$(K^2 \times 20) + (20 \times 20) + (20 \times 10) = 600 + 20\ K^2$

for $9 \times 9 = 600 + 20 \times 9^2 =$ 2,220 parameters

From Fully Connected to CNN

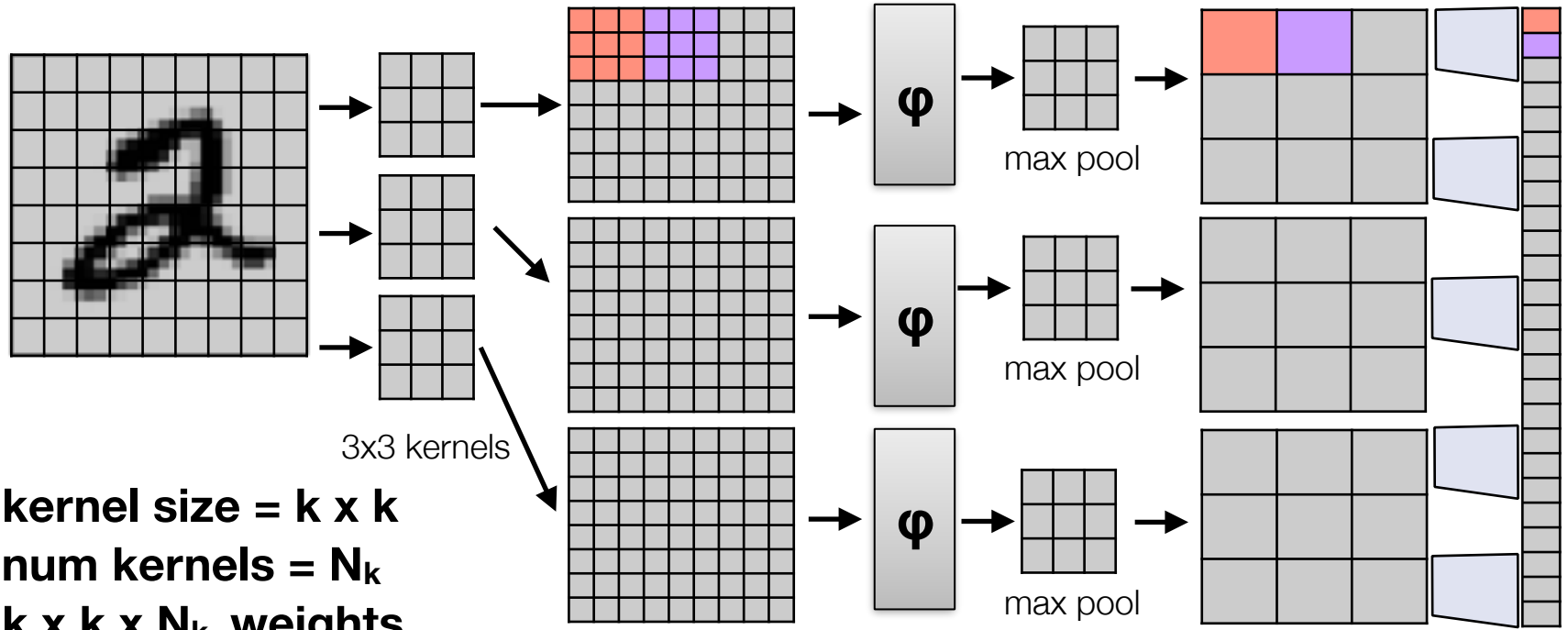


If image is 9x9, and each fully connected layer is 20 hidden neurons wide, how many parameters are in this NN (ignore bias)?



27+(27 x 20) + (20x20) + (20x10) = 1,167

From Fully Connected to CNN

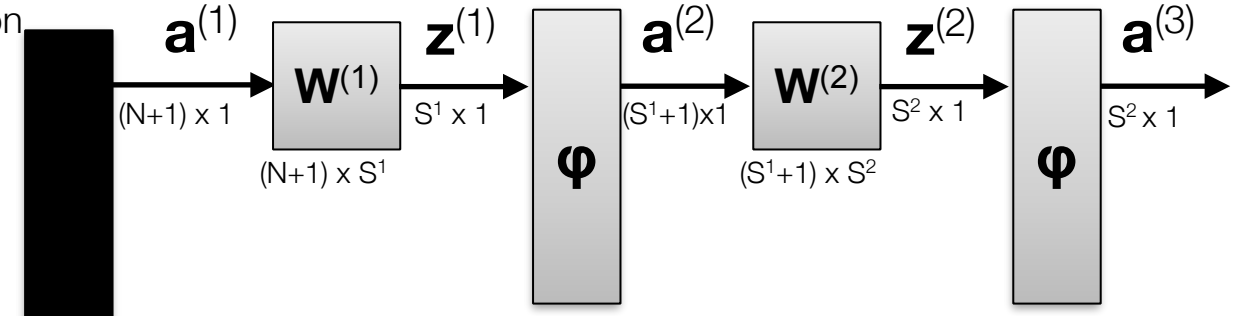


kernel size = k x k
num kernels = N_k
k x k x N_k weights

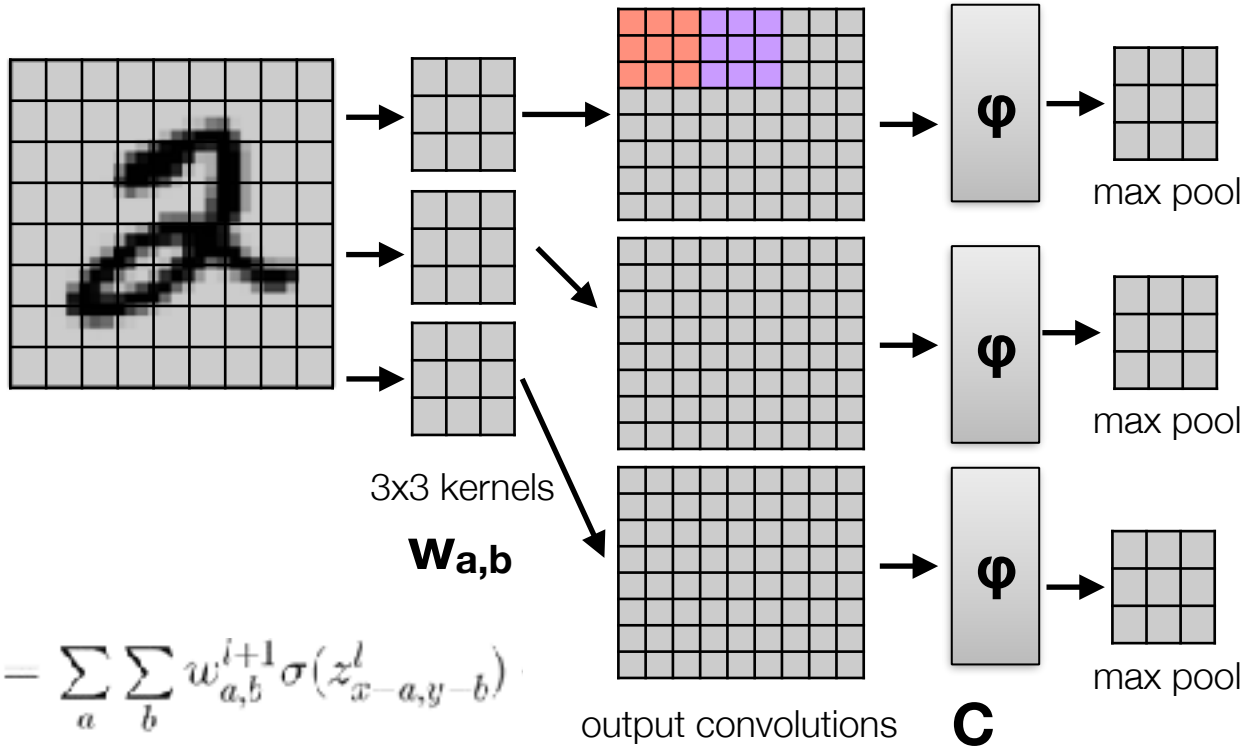
pooling outputs
if stride = k
(K/k) x (K/k) x N_k

total inputs
N_k x (k^2 +K^2/k^2)

num filters
filter dimension
image dimension



CNN gradient



Derivative of max pool is easy:

for each input x_i

$f'(x_i) = 1$ if x_i is max
0 else

$$-\sum_a \sum_b w_{a,b}^{l+1} \sigma(z_{x-a,y-b}^l)$$

Derivative of convolution is more involved:

$$\frac{\partial C}{\partial w_{a,b}^l} = \sum_x \sum_y \frac{\partial C}{\partial z_{x,y}^l} \frac{\partial z_{x,y}^l}{\partial w_{a,b}^l} = \sum_x \sum_y \delta_{x,y}^l \frac{\partial (\sum_{a'} \sum_{b'} w_{a',b'}^l \sigma(z_{x-a',y-b'}^l) + b_{x,y}^l)}{\partial w_{a,b}^l} =$$

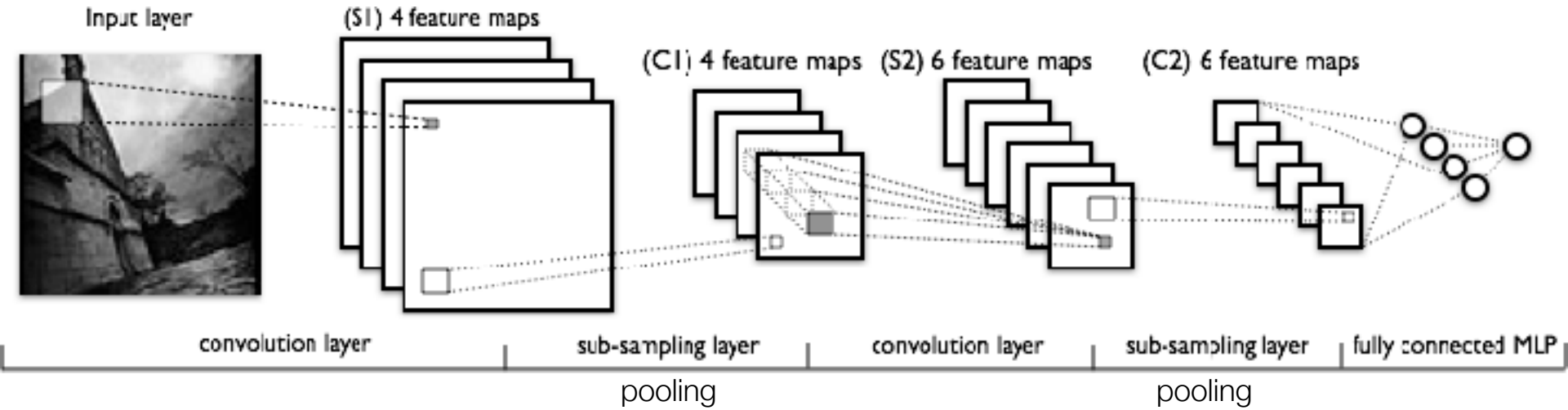
$$\sum_x \sum_y \delta_{x,y}^l \sigma(z_{x-a,y-b}^{l-1}) = \delta_{a,b}^l * \sigma(z_{-a,-b}^{l-1}) = \delta_{a,b}^l * \sigma(ROT180(z_{a,b}^{l-1}))$$

https://grzegorzgwardys.wordpress.com/2016/04/22/8/

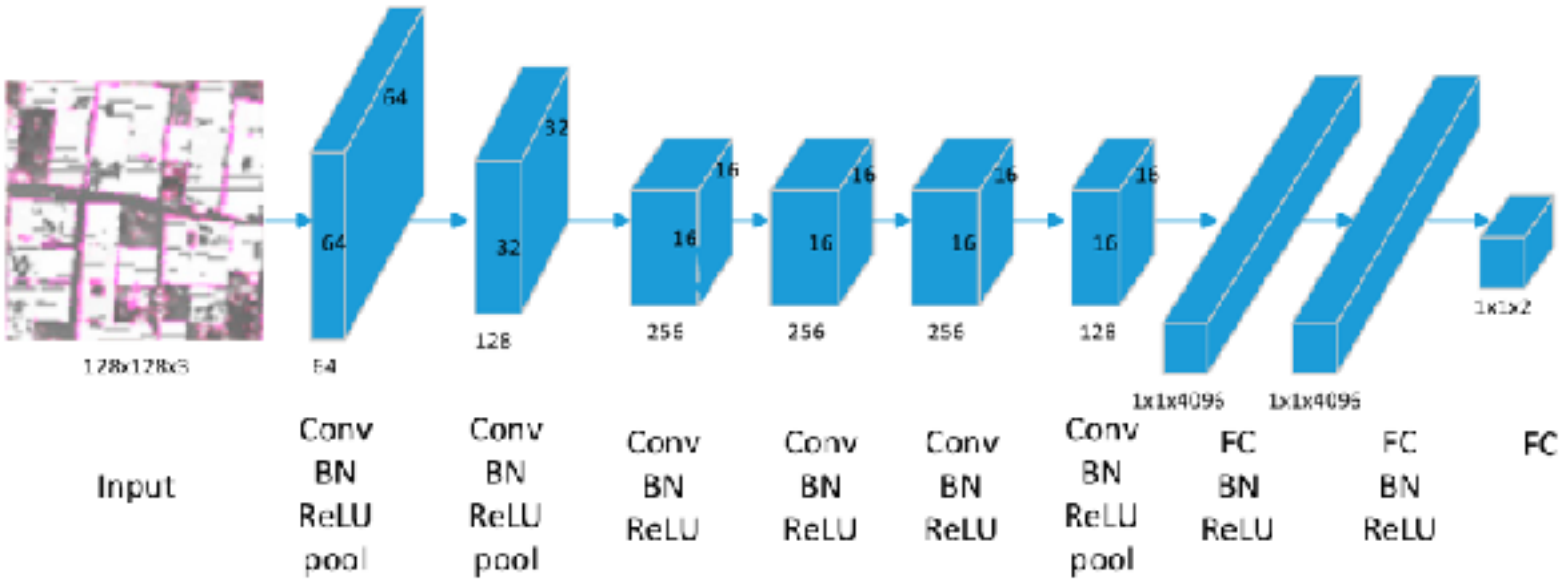
CNN gradient

- But we really want to understand the process!
- These are great guides:
  - https://grzegorzgwardys.wordpress.com/2016/04/22/8/
  - http://andrew.gibiansky.com/blog/machine-learning/convolutional-neural-networks/

Common CNN Architecture



pooling pooling



CNN: What does it all mean?

Deep Visualization Toolbox

yosinski.com/deepvis

#deepvis


Jason Yosinski


Jeff Clune


Anh Nguyen

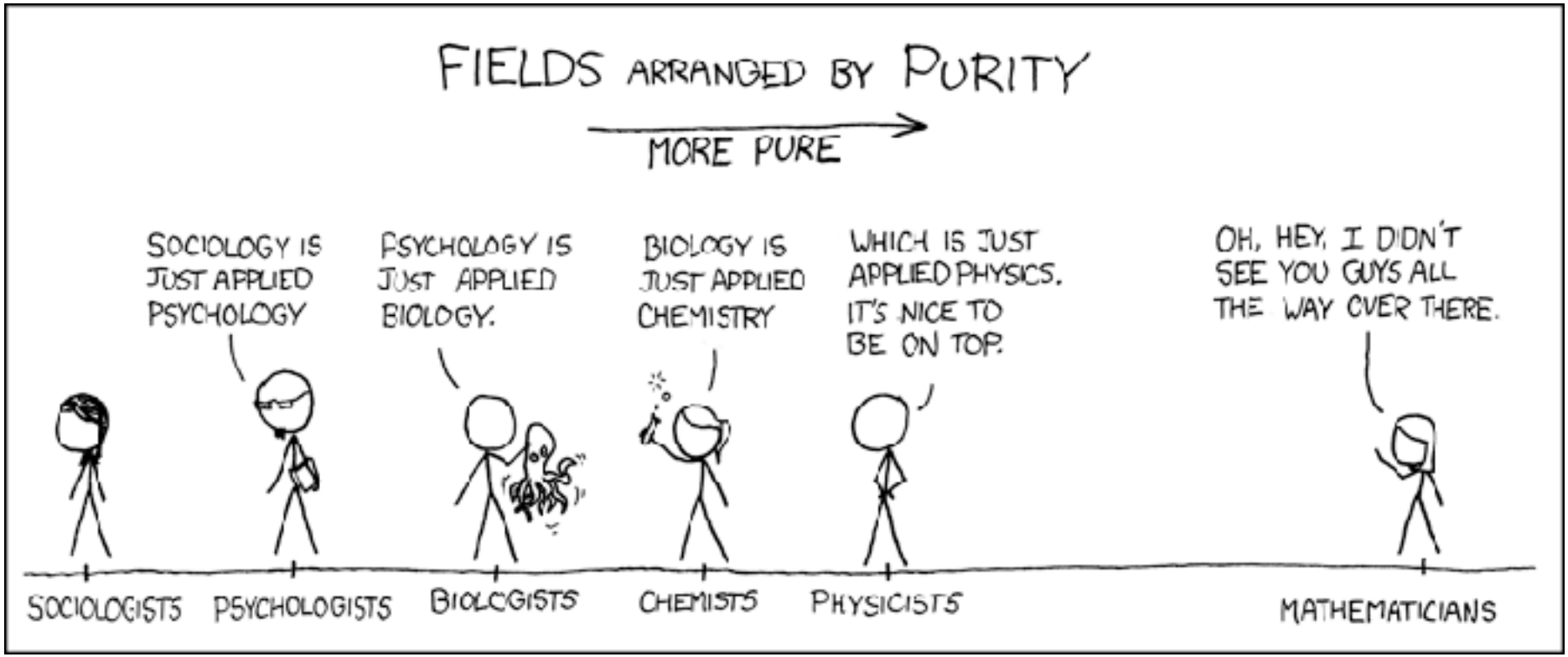

Thomas Fuchs


Hod Lipson

Programming CNNs

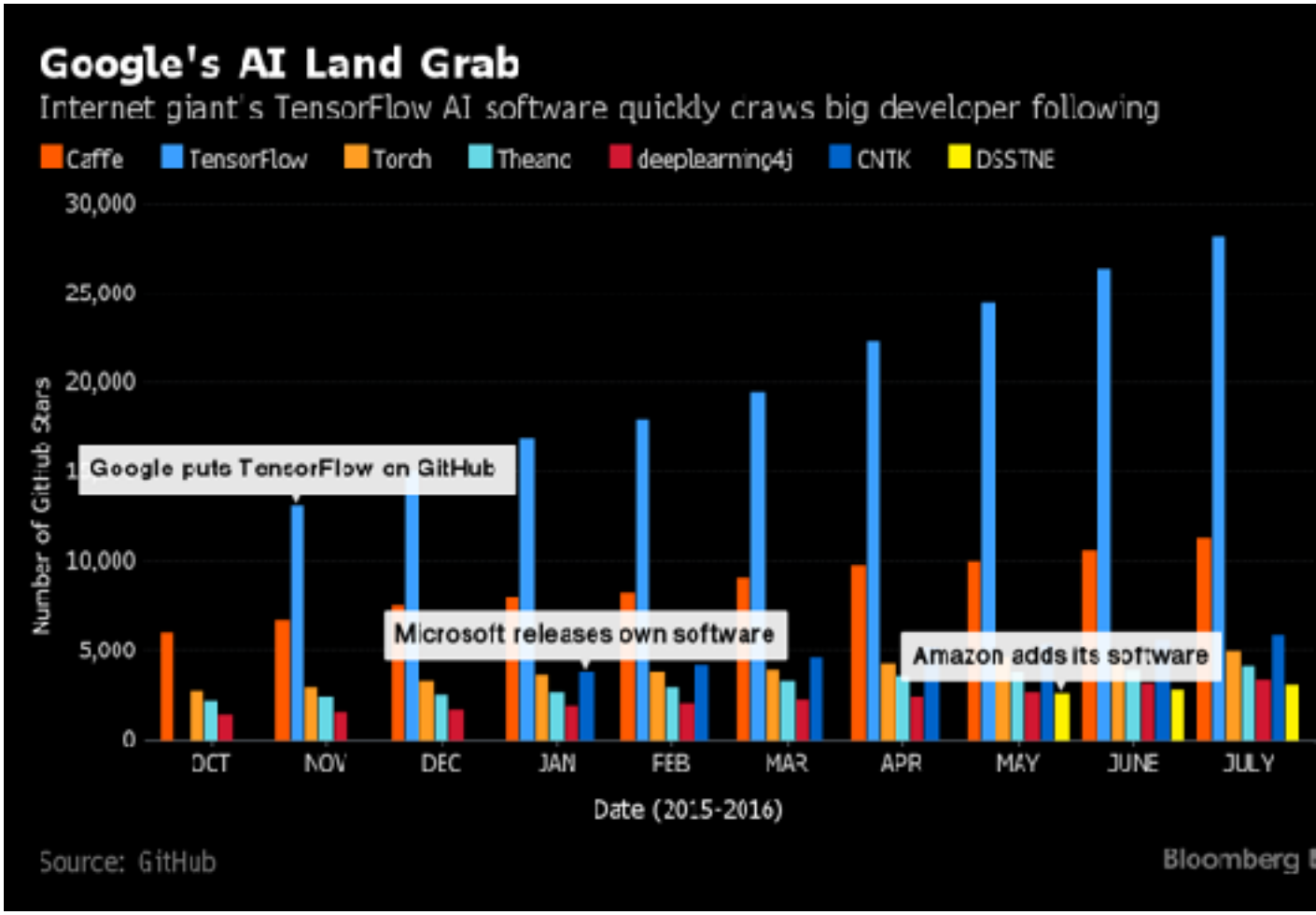
- We need a deep learning toolkit!

TensorFlow


FIELDS ARRANGED BY PURITY
MORE PURE
SOCIOLOGY IS JUST APPLIED PSYCHOLOGY
PSYCHOLOGY IS JUST APPLIED BIOLOGY.
BIOLOGY IS JUST APPLIED CHEMISTRY
WHICH IS JUST APPLIED PHYSICS. IT'S NICE TO BE ON TOP.
OH, HEY, I DIDN'T SEE YOU GUYS ALL THE WAY OVER THERE.
SOCIOLOGISTS
PSYCHOLOGISTS
BIOLOGISTS
CHEMISTS
PHYSICISTS
MATHEMATICIANS

Options for Deep Learning Toolkits

- Caffe
- TensorFlow
- Torch
- CuDNN
- MxNet
- Theano
- CNTK
- DSSTNE



Programmatic creation

- Most toolkits use python to build a computation graph of operations
  - In many ways similar to Dask and Graphlab
  - Build up computations
  - Execute computations
- Theano/CNTK are completely valid alternatives
  - its really up to you what you want to use
  - Theano originated at Berkeley and can be wrapped with Keras or Lasagne
    - http://deeplearning.net/software/theano/
  - CNTK is a Microsoft product with python wrappers and has some impressive speed graphics compared to all other languages (as of one year ago)
    - https://github.com/Microsoft/CNTK

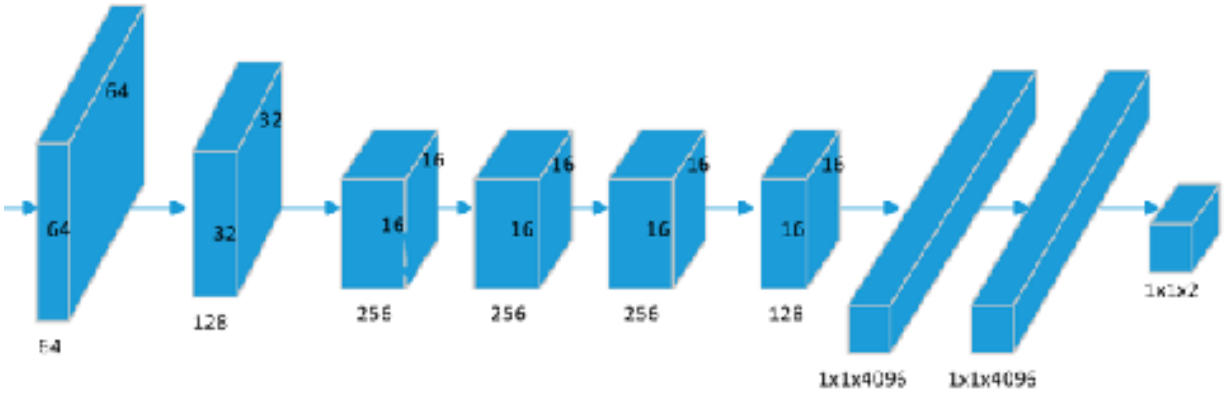
Tensorflow

- Open sourced library form Google
- Second generation release from Google Brain
  - supported for Linux and Unix (some windows support if you build from source)
  - Also works on Android/iOS
- Released November 9th, 2015
- **Supports:**
  - tensor creation
  - functions on tensors
  - automatic derivative computation

Tensors

- Tensors are just multidimensional arrays
  - like in Numpy
  - typically, for NN,
    - scalars (biases and constants)
    - vectors (inputs)
    - 2D matrices (images)
    - 3D matrices (color images)
    - 4D matrices (batches of color images)

```
a = tf.constant(5.0)

b = tf.constant(6.0)
```



Tensor basic functions

```
a = tf.constant(5.0)

b = tf.constant(6.0)

c = a * b
```

- Easy to define operations on tensors

Numpy	TensorFlow
`a = np.zeros((2,2)); b = np.ones((2,2))`	`a = tf.zeros((2,2)), b = tf.ones((2,2))`
`np.sum(b, axis=1)`	`tf.reduce_sum(a,reduction_indices=[1])`
`a.shape`	`a.get_shape()`
`np.reshape(a, (1,4))`	`tf.reshape(a, (1,4))`
`b * 5 + 1`	`b * 5 + 1`
`np.dot(a,b)`	`tf.matmul(a, b)`
`a[0,0], a[:,0], a[0,:]`	`a[0,0], a[:,0], a[0,:]`

- Also supports convolution: `tf.nn.conv2d, tf.nn.conv3D`

https://cs224d.stanford.edu/lectures/CS224d-Lecture7.pdf

Tensor neural network functions

- Easy to define operations on layers of networks

```
• tf.nn.relu(features, name=None)
• tf.nn.bias_add(value, bias, data_format=None, name=None)
• tf.sigmoid(x, name=None)
• tf.tanh(x, name=None)
• tf.nn.conv2d(input, filter, strides, padding)
• tf.nn.conv1d(value, filters, stride, padding)
• tf.nn.conv3d(input, filter, strides, padding)
• tf.nn.conv3d_transpose(value, filter, output_shape, strides)
• tf.nn.sigmoid_cross_entropy_with_logits(logits, targets)
• tf.nn.softmax(logits, dim=-1)
• tf.nn.log_softmax(logits, dim=-1)
• tf.nn.softmax_cross_entropy_with_logits(logits, labels, dim=-1)
```

- Each function creates layers easily, *knows its gradient*
- But… lets start simple…

Tensor function evaluation

- Easy to define operations on tensors
- Nothing evaluated until you define a session and tell it to evaluate it
- Session defines configuration of execution
  - like GPU versus CPU

```
a = tf.constant(5.0)

b = tf.constant(6.0)

c = a * b

with tf.Session() as sess:
    print(sess.run(c))
    print(c.eval())
```

output = 30

https://cs224d.stanford.edu/lectures/CS224d-Lecture7.pdf

Tensor Variables

- Variables that must be initialized before use

```
W1 = tf.ones((2,2))
```
immutable

```
W2 = tf.Variable(tf.zeros((2,2)), name="weights")
```
mutable

```
with tf.Session() as sess:
  print(sess.run(W1))
  sess.run(tf.initialize_all_variables())
  print(sess.run(W2))
```

```
W = tf.Variable(tf.zeros((2,2)), name="weights")
R = tf.Variable(tf.random_normal((2,2)), name="random_weights")
```


https://cs224d.stanford.edu/lectures/CS224d-Lecture7.pdf

Tensor Variables

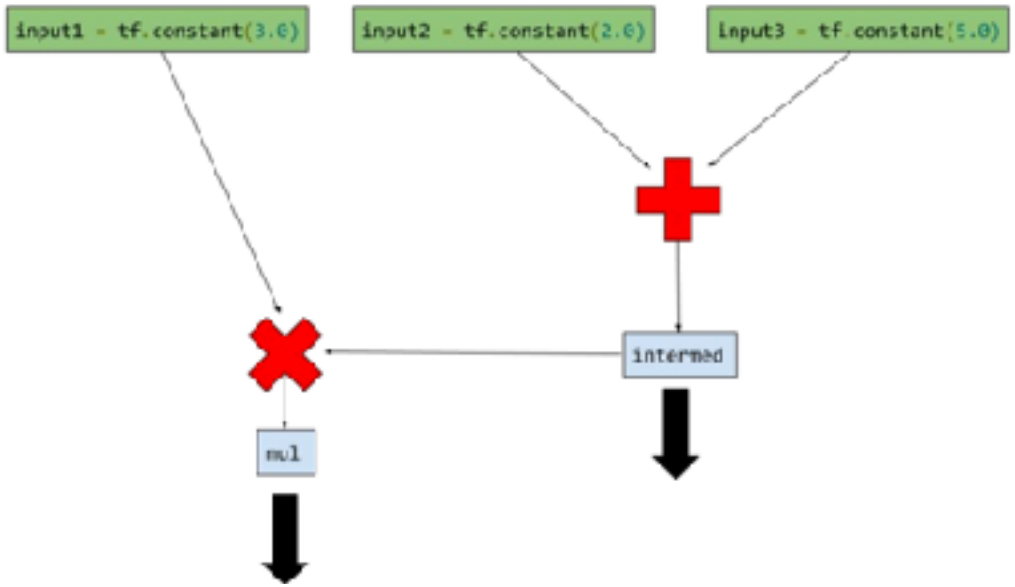
- Variables that must be initialized before use

```
input1 = tf.constant(3.0)
input2 = tf.constant(2.0)
input3 = tf.constant(5.0)
intermed = tf.add(input2, input3)
mul = tf.mul(input1, intermed)
with tf.Session() as sess:
      result = sess.run([mul, intermed])
      print(result)
```

mutable variable

mutable variable

fetch variable values

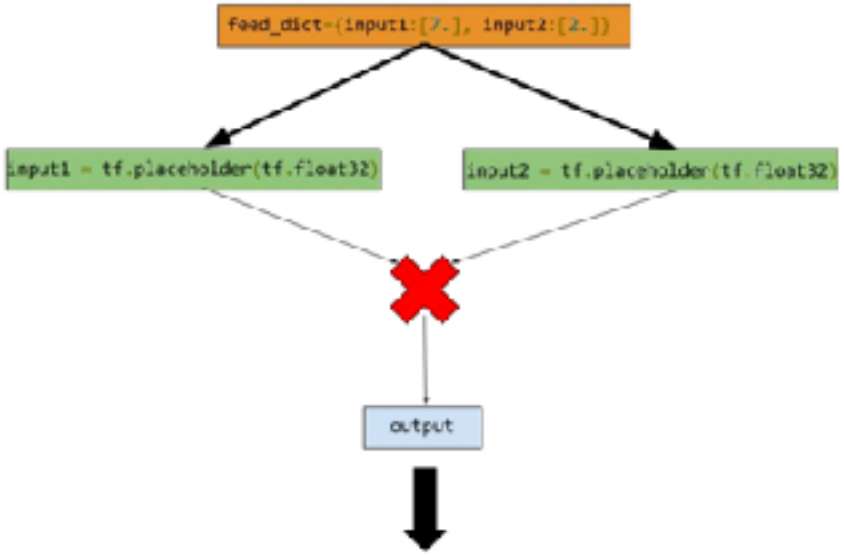


https://cs224d.stanford.edu/lectures/CS224d-Lecture7.pdf

Tensor Placeholders

- Placeholders await values and can be streamed in from the computation graph

```
input1 = tf.placeholder(tf.float32)

input2 = tf.placeholder(tf.float32)

output = tf.mul(input1, input2)

with tf.Session() as sess:
      print(sess.run([output], feed_dict={input1:[7.], input2:[2.]}))
```



https://cs224d.stanford.edu/lectures/CS224d-Lecture7.pdf

Tensor get_variable and scope

- Just a form of scoping
  - define and reuse variables
  - if you have not defined variable as “reuse,” then you can only access it once via get_variable

Variable Scope Example

Variable Scope mechanism in TensorFlow consists of 2 main functions:

- `tf.get_variable(<name>, <shape>, <initializer>)`: Creates or returns a variable with a given name.
- `tf.variable_scope(<scope_name>)`: Manages namespaces for names passed to `tf.get_variable()`.

```
with tf.variable_scope("foo"):
    v = tf.get_variable("v", [1])
with tf.variable_scope("foo", reuse=True):
    v1 = tf.get_variable("v", [1])
```

```
v.name == "foo/v:0"
```

without “reuse” this is an error

https://cs224d.stanford.edu/lectures/CS224d-Lecture7.pdf

https://www.tensorflow.org/versions/r0.11/how_tos/variable_scope/index.html

Tensor get_variable and scope

- Why do we need this?
  - Reusing variable creation code

```
def my_image_filter(input_images):
    conv1_weights = tf.Variable(tf.random_normal([5, 5, 32, 32]),
        name="conv1_weights")
    conv1_biases = tf.Variable(tf.zeros([32]), name="conv1_biases")
    conv1 = tf.nn.conv2d(input_images, conv1_weights,
        strides=[1, 1, 1, 1], padding='SAME')
    relu1 = tf.nn.relu(conv1 + conv1_biases)

    conv2_weights = tf.Variable(tf.random_normal([5, 5, 32, 32]),
        name="conv2_weights")
    conv2_biases = tf.Variable(tf.zeros([32]), name="conv2_biases")
    conv2 = tf.nn.conv2d(relu1, conv2_weights,
        strides=[1, 1, 1, 1], padding='SAME')
    return tf.nn.relu(conv2 + conv2_biases)
```

explicit definition of repeated operations

https://www.tensorflow.org/versions/r0.11/how_tos/variable_scope/index.html

Tensor get_variable and scope

- Why do we need this?
  - Reusing variable creation code

more readable code

```
def conv_relu(input, kernel_shape, bias_shape):
    # Create variable named "weights".
    weights = tf.get_variable("weights", kernel_shape,
        initializer=tf.random_normal_initializer())
    # Create variable named "biases".
    biases = tf.get_variable("biases", bias_shape,
        initializer=tf.constant_initializer(0.0))
    conv = tf.nn.conv2d(input, weights,
        strides=[1, 1, 1, 1], padding='SAME')
    return tf.nn.relu(conv + biases)
```

```
def my_image_filter(input_images):
    with tf.variable_scope("conv1"):
        # Variables created here will be named "conv1/weights", "conv1/biases".
        relu1 = conv_relu(input_images, [5, 5, 32, 32], [32])
    with tf.variable_scope("conv2"):
        # Variables created here will be named "conv2/weights", "conv2/biases".
        return conv_relu(relu1, [5, 5, 32, 32], [32])
```


https://www.tensorflow.org/versions/r0.11/how_tos/variable_scope/index.html

Tensor Differentiation

- This seems like a lot of new syntax
- Why can't we just use numpy?
  - automatic differentiation, i.e., "*your life, easier*"

```
# Define variables to be learned
with tf.variable_scope("linear-regression"):
  W = tf.get_variable("weights", (1, 1),
                      initializer=tf.random_normal_initializer())
  b = tf.get_variable("bias", (1,),
                      initializer=tf.constant_initializer(0.0))
  y_pred = tf.matmul(X, W) + b
  loss = tf.reduce_sum((y - y_pred)**2/n_samples)

  opt = tf.train.AdamOptimizer()
  opt_operation = opt.minimize(loss)
  with tf.Session() as sess:
      sess.run(tf.initialize_all_variables())
      sess.run([opt_operation], feed_dict={X: X_data, y: y_data})
```

$$J(W,b) = \frac{1}{N}\sum_{i=1}^{N}(y_i - (Wx_i + b))^2$$

https://cs224d.stanford.edu/lectures/CS224d-Lecture7.pdf

Tensor Mini-batching?

```
opt = tf.train.AdamOptimizer()
opt_operation = opt.minimize(loss)

with tf.Session() as sess:
  # Initialize Variables in graph
  sess.run(tf.initialize_all_variables())
  # Gradient descent loop for 500 steps
  for _ in range(500):
    # Select random minibatch
    indices = np.random.choice(n_samples, batch_size)
    X_batch, y_batch = X_data[indices], y_data[indices]
    # Do gradient descent step
    _, loss_val = sess.run([opt_operation, loss], feed_dict={X: X_batch, y: y_batch})
```

- Each node in a TensorFlow graph knows its gradient
- Which means back-propagation is easy to carry out computationally

https://cs224d.stanford.edu/lectures/CS224d-Lecture7.pdf

Tensor-flow Simplification

- It seems like many common NN optimizations can be preprogrammed
- **Self Test**: Can the syntax be simplified?
  - (A) **Yes**, we could write a generic mini-batch optimization computation graph, then use it for arbitrary inputs
  - (B) **Yes**, but we lose control over the optimization procedures
  - (C) **Yes**, but we lose control over the NN models that we can create via Tensorflow
  - (D) **Yes**, and Dr. Larson is going to make us write it ourselves

Lecture Notes for Machine Learning in Python

Professor Eric Larson

Week 12, 13, 14 Deep Learning

Class logistics

- Lab 3 has been up!
  - Due Dec 13 by 5PM
  - Sorry, No Resubmits for A3
  - No Project Presentations
- Grades will be coming in more steadily

Lecture Agenda

- Finish Convolutional Neural Networks
- Introduction to TensorFlow
  - Tensors
  - Namespaces
  - Numerical methods
- Demo of TensorFlow with CNNs
- Remaining lecture topics:
  - Long- Short-term Memory Networks

Tensor-flow Simplification

- It seems like many common NN optimizations can be preprogrammed
- **Self Test**: Can the syntax be simplified?
  - (A) **Yes**, we could write a generic mini-batch optimization computation graph, then use it for arbitrary inputs
  - (B) **Yes**, but we lose control over the optimization procedures
  - (C) **Yes**, but we lose control over the NN models that we can create via Tensorflow
  - (D) **Yes**, and Dr. Larson is going to make us write it ourselves

Tensor-flow Simplification: Scikit-Flow (SKFLow)

- Mirrors syntax of scikit-learn
  - but is TensorFlow under the hood
- Defines many standard models out of the box
- Allows creation of custom models
  - with simplifications for commonly created layers
- Allows many controls over the optimization used
- Was third party (while in alpha)
  - but now integrated and merged with TensorFlow core
    - `tf.contrib.learn`
  - **TF Learn** created and maintained by Google developers!
  - Google recommends this:
    - for learning Tensorflow
    - for getting a model off the ground quickly

TensorFlow and SKFlow

Building and simplifying
Linear Regression

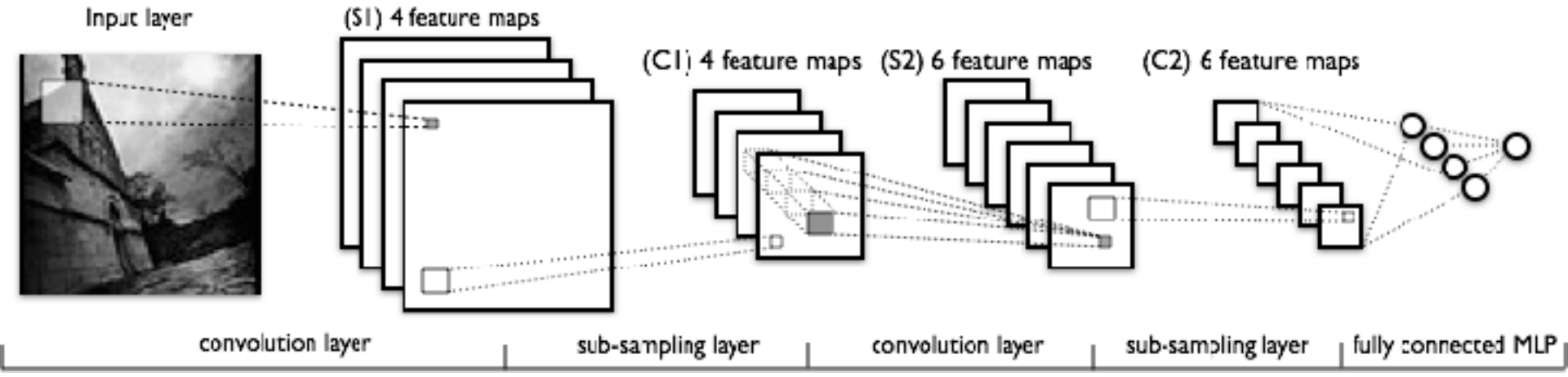
Other tutorials:

https://github.com/jtoy/awesome-tensorflow

https://github.com/tensorflow/tensorflow/tree/r0.11/tensorflow/examples/skflow

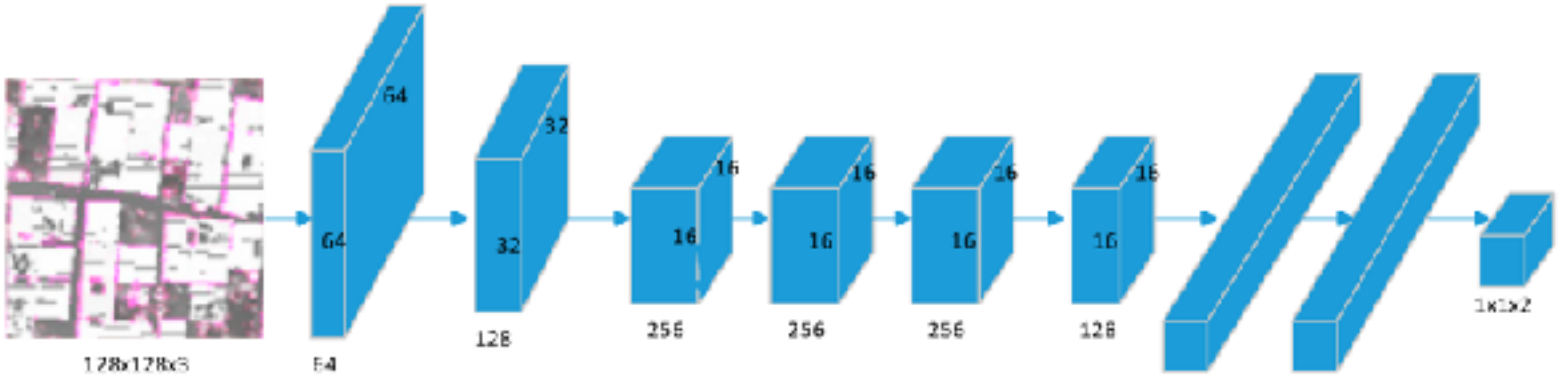
Or do a Google search!!! They are everywhere!!!

A brief reminder about CNNs:



pooling pooling

layer1 layer2 layer3 layer4 layer5 layer6 layer7 layer8 layer9



TensorFlow and SKFlow

Convolutional Neural Networks
in TensorFlow
with SKFlow

For remainder of semester

- Recurrent Neural Network Architectures
- Mainly, LSTMs