ELEG 5040
Tutorial 1
Introduction to Python

Kai KANG (kkang@ee.cuhk.edu.hk)

Outline

- Tutorial Plan
- Piazza System
- Python Basics
- Popular Python Packages

Tutorial Plan

Week	Content	Note
1		
2	Python Introduction	
3	Theano	
4-5	CUDA/GPU Programming (Invited Talks)	Jan 24 & 25
6-7	Deep Learning Toolbox	
8-10	Caffe	
11-12	Research Experience on Deep Learning	
13-14	Review	

Piazza System

- Course website: www.piazza.com/cuhk.edu.hk/spring2015/eleg5040/home
- Resources: piazza.com/cuhk.edu.hk/spring2015/eleg5040/resources
  - Announcements (Email preferences)
  - Lecture and tutorial notes
  - Homework and solutions
  - Reading materials
- Q & A Section
  - Public and private questions, anonymous posts (if you are shy)
  - Collaborative editing: anyone can contribute
  - Image, attached files, LaTex math forms, code highlighting, etc

Why Python?

Why Python?

[If programming languages were vehicles]

Python is great for everyday tasks: easy to drive, versatile, comes with all the conveniences built in. It isn't fast or sexy, but neither are your errands.

Why Python?

- Convenience
  - Interpreted and interactive
  - Clear and concise syntax
  - Dynamic typing
  - Portable
  - Modules
- High-level
  - Object-oriented
  - Interfaces to many libraries
- Large community

Online Materials

- Official Python tutorial (https://docs.python.org/3/tutorial/index.html#tutorial-index)
- CodeAcademy interactive tutorial (http://www.codecademy.com/en/tracks/python)
- Dive Into Python free digital book (http://www.diveintopython.net)
- Python Challenge (http://www.pythonchallenge.com)

Python Basics

Numbers

Arithmetics

```
>>> 2 + 2
4
>>> 50 - 5*6
20
>>> (50 - 5*6) / 4
5.0
>>> 8 / 5  # division always returns a floating point number
1.6
```

Floor division and remainder

```
>>> 17 / 3  # classic division returns a float
5.666666666666667
>>>
>>> 17 // 3  # floor division discards the fractional part
5
>>> 17 % 3  # the % operator returns the remainder of the division
2
>>> 5 * 3 + 2  # result * divisor + remainder
17
```

Numbers

Power operations

```
>>> 5 ** 2  # 5 squared
25
>>> 2 ** 7  # 2 to the power of 7
128
```

Variables and assignments

```
>>> width = 20
>>> height = 5 * 9
>>> width * height
900
```

Numbers

Beyond int and float

```
>>> from fractions import Fraction
>>> Fraction(16, -10)
Fraction(-8, 5)
>>> Fraction(123)
Fraction(123, 1)
>>> Fraction()
Fraction(0, 1)
>>> Fraction('3/7')
Fraction(3, 7)

>>> 3+5j
(3+5j)
>>> 3+5J
(3+5j)
>>> 6+7j + 8-2J
(14+5j)
>>> complex('6-5j')
(6-5j)
```

Strings

Use either single quotes or double quotes

```
>>> 'spam eggs'  # single quotes
'spam eggs'
>>> 'doesn\'t'  # use \' to escape the single quote...
"doesn't"
>>> "doesn't"  # ...or use double quotes instead
"doesn't"
>>> '"Yes," he said.'
'"Yes," he said.'
```

Use \ to escape sequences

```
>>> "\"Yes,\" he said."
'"Yes," he said.'
>>> '"Isn\'t," she said.'
'"Isn\'t," she said.'
>>> 'Hello, World!\n'
'Hello, World!\n'
>>> print('Hello, World!\n')
Hello, World!

>>>
```

Strings

Concatenation and repetition

```
>>> 'He' + 'llo'
'Hello'
>>> ('He' + 'llo') * 3
'HelloHelloHello'
>>> 'He''llo'
'Hello'
>>> ('He' "llo") * 3
'HelloHelloHello'
```

Indexing (zero-based)

```
>>> word = 'Python'
>>> word[0]  # character in position 0
'P'
>>> word[5]  # character in position 5
'n'
>>> word[-1]  # last character
'n'
>>> word[-2]  # second-last character
'o'
>>> word[0:2]  # characters from position 0 (included) to 2 (excluded)
'Py'
>>> word[2:5]  # characters from position 2 (included) to 5 (excluded)
'tho'
```

Lists

Lists may not contain items with same types

```
list1 = ['physics', 'chemistry', 1997, 2000];
list2 = [1, 2, 3, 4, 5 ];
list3 = ["a", "b", "c", "d"];
```

Accessing, updating and deleting items

```
>>> list1 = ['physics', 'chemistry', 1997, 2000];
>>> list2 = [1, 2, 3, 4, 5, 6, 7 ];
>>> print "list1[0]: ", list1[0]
list1[0]:  physics
>>> print "list2[1:5]: ", list2[1:5]
list2[1:5]:  [2, 3, 4, 5]
>>> list1[2]=2001
>>> list1
['physics', 'chemistry', 2001, 2000]
>>> del list1[2]
>>> list1
['physics', 'chemistry', 2000]
```

Concatenating and repeating

```
>>> [1, 2, 3] + [4, 5]
[1, 2, 3, 4, 5]
>>> [6, 7] * 4
[6, 7, 6, 7, 6, 7, 6, 7]
```

Tuples

Tuples are very similar to lists, only difference is that tuples are immutable

```
>>> tup1 = ('physics', 'chemistry', 1997, 2000);
>>> tup2 = (1, 2, 3, 4, 5, 6, 7 );
>>> print "tup1[0]: ", tup1[0]
tup1[0]:  physics
>>> print "tup2[1:5]: ", tup2[1:5]
tup2[1:5]:  (2, 3, 4, 5)
>>> tup1 = (50,)
```

No enclosing delimiters needed

```
>>> a, b = 0, 1
>>> a, b = b, a + b
>>> a, b
(1, 1)
```

Dictionaries

Creating dictionaries with {}

```
>>> dict = {'Alice': '2341', 'Beth': '9102', 'Cecil': '3258'}
>>> dict
{'Beth': '9102', 'Alice': '2341', 'Cecil': '3258'}
>>> dict1 = { 'abc': 456 };
>>> dict2 = { 'abc': 123, 98.6: 37 };
>>> dict2
{98.6: 37, 'abc': 123}
```

Indexing with keys

```
>>> dict['Alice']
'2341'
>>> dict1['abc']
456
>>> dict2[98.6]
37
```

Looping through keys

```
>>> for key in dict2:
...     print key, dict2[key]
...
98.6 37
abc 123
```

`if` Statements

Indentation is important in Python, semicolons are not

```
>>> a = 3
>>> b = 4
>>> if a < b:
...     print(a)
... else:
...     print(b)
...
3
>>> print(a if a < b else b)
3
```

No switch and case statements, use if..elif..elif.. sequence

```
>>> if x < 0:
...     x = 0
...     print('Negative changed to zero')
... elif x == 0:
...     print('Zero')
... elif x == 1:
...     print('Single')
... else:
...     print('More')
```

`for` Statements

`for` statements iterate within sequences

```
>>> # Measure some strings:
... words = ['cat', 'window', 'defenestrate']
>>> for w in words:
...     print(w, len(w))
...
cat 3
window 6
defenestrate 12
```

Looping in a range

```
>>> for i in range(5):
...     print i
0
1
2
3
4
>>> range(1,5)
[1, 2, 3, 4]
>>> range(1,5,2)
[1, 3]
```

break and continue

break jumps out the smallest loop

```
>>> for n in range(2, 10):
...     for x in range(2, n):
...         if n % x == 0:
...             print n, 'equals', x, '*', n/x
...             break
...     else:
...         # loop fell through without finding a factor
...         print n, 'is a prime number'
```

continue jumps to next iteration

```
>>> for num in range(2, 10):
...     if num % 2 == 0:
...         print "Found an even number", num
...         continue
...     print "Found a number", num
```

`pass` Statements

pass statements do nothing but make syntax right

```
>>> while True:
...     pass  # Busy-wait for keyboard interrupt (Ctrl+C)
...
```

```
>>> class MyEmptyClass:
...     pass
...
```

```
>>> def initlog(*args):
...     pass   # Remember to implement this!
...
```

Functions

Define a function

```
>>> def fib(n):     # write Fibonacci series up to n
...     """Print a Fibonacci series up to n."""
...     a, b = 0, 1
...     while a < n:
...         print a,
...         a, b = b, a+b
...
>>> # Now call the function we just defined:
... fib(2000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597
```

Functions are objects

```
>>> fib
<function fib at 10042ed0>
>>> f = fib
>>> f(100)
0 1 1 2 3 5 8 13 21 34 55 89
```

Functions

return statements

```
>>> def fib2(n): # return Fibonacci series up to n
...     """Return a list containing the Fibonacci series up to
n."""
...     result = []
...     a, b = 0, 1
...     while a < n:
...         result.append(a)    # see below
...         a, b = b, a+b
...     return result
...
>>> f100 = fib2(100)    # call it
>>> f100                # write the result
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

Functions

Default argument values

```
>>> def inc(a, b = 1):
...     return a + b
...
>>> inc(10)
11
>>> inc(10,5)
15
```

Keyword arguments

```
>>> def inc(st, step=1):
...     return st + step
...
>>> inc(10)
11
>>> inc(10, 5)
15
>>> inc(10, step=6)
16
```

Arbitrary argument list

```
def write_multiple_items(file, separator, *args):
    file.write(separator.join(args))
```

Classes

Define a class: instance variables and methods

```
class MyClass:
    """A simple example class"""
    i = 12345
    def f(self):
        return 'hello world'
```

```
>>> x = MyClass()
>>> x.i
12345
>>> x.f()
'hello world'
```

Instantiations: __init__()

```
>>> class Complex:
...     def __init__(self, realpart, imagpart):
...         self.r = realpart
...         self.i = imagpart
...
>>> x = Complex(3.0, -4.5)
>>> x.r, x.i
(3.0, -4.5)
```

Classes

Instance variables

```
>>> class Dog:
...     def __init__(self, name, kind='canine'):
...         self.name = name
...         self.kind = kind
...
>>> d = Dog('Fido')
>>> e = Dog('Buddy', 'husky')
>>> d.kind
'canine'
>>> e.kind
'husky'
>>> d.name
'Fido'
>>> e.name
'Buddy'
```

Classes

Inheritance

```
>>> class Dog:
...     def __init__(self, name, kind='canine'):
...         self.name = name
...         self.kind = kind
...
```

```
>>> class Husky(Dog):
...     def __init__(self, name):
...         Dog.__init__(self, name, 'husky')
...
>>> h = Husky('Rocky')
>>> h.name
'Rocky'
>>> h.kind
'husky'
```

Run Python Scripts

```
#!/usr/bin/python3

class Duck:
    def quack(self):
        print('Quaaack!')

    def walk(self):
        print('Walks like a duck.')

def main():
    donald = Duck()
    donald.quack()
    donald.walk()

if __name__ == "__main__": main()
```

Popular Packages

- cPickle, Protocol Buffers
- NumPy, SciPy
- scikit-learn, scikit-image, PIL, matplotlib
- h5py, leveldb
- ipython
- …

NumPy for Matlab Users

- http://wiki.scipy.org/NumPy_for_Matlab_Users

MATLAB	numpy	Notes
a && b	a and b	short-circuiting logical AND operator (Python native operator); scalar arguments only
a || b	a or b	short-circuiting logical OR operator (Python native operator); scalar arguments only
`1*i,1*j,1i,1j`	`1j`	complex numbers
`eps`	`spacing(1)`	Distance between 1 and the nearest floating point number
`ode45`	`scipy.integrate.ode(f).set_integrator('dopri5')`	integrate an ODE with Runge-Kutta 4,5
`ode15s`	`scipy.integrate.ode(f).\ set_integrator('vode', method='bdf', order=15)`	integrate an ODE with BDF

NumPy for Matlab Users

- Linear Algebra Equivalents

MATLAB	numpy.array	numpy.matrix
`ndims(a)`	`ndim(a)` or `a.ndim`	
`numel(a)`	`size(a)` or `a.size`	
`size(a)`	`shape(a)` or `a.shape`	
`size(a,n)`	`a.shape[n-1]`	
`[ 1 2 3; 4 5 6 ]`	`array([[1.,2.,3.], [4.,5.,6.]])`	`mat([[1.,2.,3.], [4.,5.,6.]])` or
`[ a b; c d ]`	`vstack([hstack([a,b]), hstack([c,d])])`	`bmat('a b; c d')`
`a(end)`	`a[-1]`	`a[:,-1][0,0]`
`a(2,5)`	`a[1,4]`	
`a(2,:)`	`a[1]` or `a[1,:]`	
`a(1:5,:)`	`a[0:5]` or `a[:5]` or `a[0:5,:]`	
`a(end-4:end,:)`	`a[-5:]`	
`a(1:3,5:9)`	`a[0:3][:,4:9]`	
`a([2,4,5],[1,3])`	`a[ix_([1,3,4],[0,2])]`	

NumPy for Matlab Users

- Linear Algebra Equivalents

MATLAB	numpy.array	numpy.matrix
`a(3:2:21,:)`	`a[ 2:21:2,:]`	
`a(1:2:end,:)`	`a[ ::2,:]`	
`a(end:-1:1,:)` or `flipud(a)`	`a[ ::-1,:]`	
`a([1:end 1],:)`	`a[r_[:len(a),0]]`	
`a.'`	`a.transpose()` or `a.T`	
`a'`	`a.conj().transpose()` or `a.conj().T`	`a.H`
`a * b`	`dot(a,b)`	`a * b`
`a .* b`	`a * b`	`multiply(a,b)`
`a./b`	`a/b`	
`a.^3`	`a**3`	`power(a,3)`
`(a>0.5)`	`(a>0.5)`	
`find(a>0.5)`	`nonzero(a>0.5)`	
`a(:,find(v>0.5))`	`a[:,nonzero(v>0.5)[0]]`	`a[:,nonzero(v.A>0.5)[0]]`
`a(:,find(v>0.5))`	`a[:,v.T>0.5]`	`a[:,v.T>0.5)]`

NumPy for Matlab Users

- Linear Algebra Equivalents

MATLAB	numpy.array	numpy.matrix
a(a<0.5)=0	a[a<0.5]=0	
a .* (a>0.5)	a * (a>0.5)	mat(a.A * (a>0.5).A)
a(:) = 3	a[:] = 3	
y=x	y = x.copy()	
y=x(2,:)	y = x[1,:].copy()	
y=x(:)	y = x.flatten(1)	
1:10	arange(1.,11.) or r_[1.:11.] or r_[1:10:10j]	mat(arange(1.,11.)) or r_[1.:11.,'r']
0:9	arange(10.) or r_[:10.] or r_[:9:10j]	mat(arange(10.)) or r_[:10.,'r']
[1:10]'	arange(1.,11.)[:, newaxis]	r_[1.:11.,'c']
zeros(3,4)	zeros((3,4))	mat(...)
zeros(3,4,5)	zeros((3,4,5))	mat(...)
ones(3,4)	ones((3,4))	mat(...)
eye(3)	eye(3)	mat(...)
diag(a)	diag(a)	mat(...)
diag(a,0)	diag(a,0)	mat(...)
rand(3,4)	random.rand(3,4)	mat(...)

Reading and Writing Mat Files

```
>>> import scipy.io as sio
>>> import numpy as np
>>> x = np.arange(12).reshape((3,4))
>>> x
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
>>> sio.savemat('example.mat', {'x': x})
>>> y = sio.loadmat('example.mat')
>>> y
{'x': array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]]), '__version__': '1.0', '__header__':
'MATLAB 5.0 MAT-file Platform: posix, Created on: Fri Jan 16 13:04:04
2015', '__globals__': []}
>>> new_x = y['x']
>>> new_x
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

Pickle

```
>>> import cPickle
>>> import numpy as np
>>> x = np.arange(12).reshape((3,4))
>>> y = np.random.rand(3,4)
```

Serialization

```
>>> pkl_file = open('example.pkl','wb')
>>> cPickle.dump((x,y), pkl_file)
>>> pkl_file.close()
```

De-serialization

```
>>> new_x, new_y = cPickle.load(open('example.pkl','rb'))
>>> new_x
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
>>> new_y
array([[ 0.64956048,  0.79727908,  0.28350831,  0.07649368],
       [ 0.11551531,  0.11434357,  0.89143651,  0.23523877],
       [ 0.21506583,  0.16611042,  0.15059062,  0.30306736]])
```

References

- Official Python tutorial (https://docs.python.org/3/tutorial/index.html#tutorial-index)
- CodeAcademy interactive tutorial (http://www.codecademy.com/en/tracks/python)
- Dive Into Python free digital book (http://www.diveintopython.net)
- Python Challenge (http://www.pythonchallenge.com)
- NumPy for Matlab Users (http://wiki.scipy.org/NumPy_for_Matlab_Users)
- Scikit-learn (http://scikit-learn.org/stable/)
- SciPy, NumPy, matplotlib (http://www.scipy.org)
- h5py (http://docs.h5py.org/en/latest/)
- Style Guide for Python Code (https://www.python.org/dev/peps/pep-0008/)
- Google Python Style Guide (https://google-styleguide.googlecode.com/svn/trunk/pyguide.html)

Tutorial 2
Introduction to Theano

Kai KANG (kkang@ee.cuhk.edu.hk)

Outline

- CUDA/GPU Programming Invited Talks
- Theano Basics
- Beyond Basics
- Theano in Machine Learning and Deep Learning

CUDA/GPU Programming Tutorials

- Speaker: Bin Zhou,PhD
  - NVIDIA CUDA Fellow, USTC Adjunct Research Professor
  - Chief Scientist and Director of Marine Remote Sensing & Information Processing Lab, SDIOI
- First tutorial: 9:30 - 12:15 on Saturday, Jan 24, 2015, LSB LT6
- Second tutorial: 9:30 - 12:15 on Sunday, Jan 25, 2015, TY Wong Hall at SHB

Theano Basics

What is Theano?

- Symbolic computation library
- CPU and GPU infrastructure
- Optimized compiler

Online Materials

- Theano (http://deeplearning.net/software/theano/index.html)
- Theano introduction (http://deeplearning.net/software/theano/introduction.html#introduction)
- Installation guides (http://deeplearning.net/software/theano/install.html#install)
- Theano tutorials (http://deeplearning.net/software/theano/tutorial/index.html#tutorial)
- Documentation (http://deeplearning.net/software/theano/library/index.html#libdoc)
- GitHub page (https://github.com/Theano/Theano)

Algebra Basics

import theano

```
>>> import theano
Using gpu device 0: GeForce GTX 670
>>> import theano.tensor as T
>>> from theano import function
```

define variables

```
>>> x = T.fscalar('x')
>>> y = T.fscalar('y')
>>> x.type
TensorType(float32, scalar)
>>> y.type
TensorType(float32, scalar)
```

Algebra Basics

symbolic operations

```
>>> z = x + y
>>> z.type
TensorType(float32, scalar)
>>> w = x * y
>>> s = z + w
```

pretty print expressions

```
>>> from theano import pp
>>> pp(z)
'(x + y)'
>>> pp(w)
'(x * y)'
>>> s = z + w
>>> pp(s)
'((x + y) + (x * y))'
```

Algebra Basics

compile into functions

```
>>> from theano import function
>>> f1 = function([x, y], z)
>>> f2 = function([x, y], w)
>>> f3 = function([x, y], s)
```

call functions to get NumPy array

```
>>> f1(1.0, 2.0)
array(3.0, dtype=float32)
>>> f2(1.0, 2.0)
array(2.0, dtype=float32)
>>> f3(1.0, 2.0)
array(5.0, dtype=float32)
```

Algebra Basics

- 3 Steps in using Theano
  - Step 1: define symbolic variables

    ```
    >>> x = T.dscalar('x')
    >>> y = T.dscalar('y')
    ```

  - Step 2: define symbolic expressions using defined variables

    ```
    >>> z = x + y
    ```

  - Step 3: compile expressions into functions

    ```
    >>> f = function([x, y], z)
    ```

Example

Add two matrices

```
>>> m1 = T.dmatrix()
>>> m2 = T.dmatrix()
>>> sum = m1 + m2
>>> msum = function([m1, m2], sum)

>>> import numpy as np
>>> msum([[1, 3, 5], [7, 9, 11]], [[2, 4, 6], [8, 10, 12]])
array([[  3.,   7.,  11.],
       [ 15.,  19.,  23.]])
>>> msum(np.random.rand(2,3), np.random.rand(2,3))
array([[ 1.07868464,  1.01885801,  0.69778457],
       [ 0.9562832 ,  1.56608957,  1.19840735]])
>>> msum(np.random.rand(2,3), np.random.rand(2,3))
array([[ 1.04984599,  0.42415405,  1.51894434],
       [ 0.80635964,  0.17668743,  1.72312159]])
```

Data Types

- **byte:** bscalar, bvector, bmatrix, brow, bcol, btensor3, btensor4
- **16-bit integers:** wscalar, wvector, wmatrix, wrow, wcol, wtensor3, wtensor4
- **32-bit integers:** iscalar, ivector, imatrix, irow, icol, itensor3, itensor4
- **64-bit integers:** lscalar, lvector, lmatrix, lrow, lcol, ltensor3, ltensor4
- **float:** fscalar, fvector, fmatrix, frow, fcol, ftensor3, ftensor4
- **double:** dscalar, dvector, dmatrix, drow, dcol, dtensor3, dtensor4
- **complex:** cscalar, cvector, cmatrix, crow, ccol, ctensor3, ctensor4

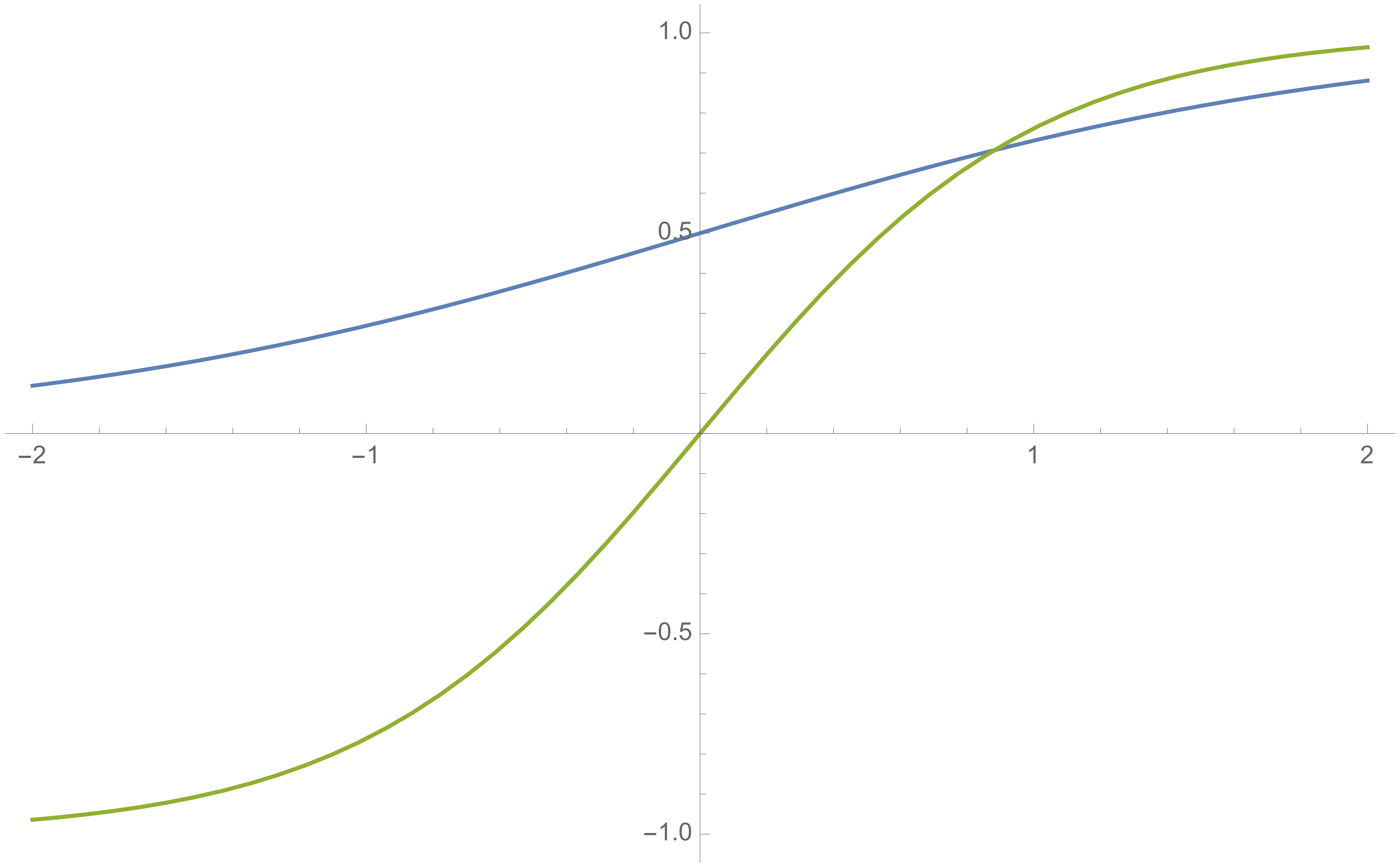
Beyond Basics

Elementwise Operations

```
>>> x = T.dmatrix('x')
>>> s = 1 / (1 + T.exp(-x))
>>> t = T.tanh(x)
>>> sigmoid = function([x], s)
>>> tanh = function([x], t)
>>> sigmoid([[0, 1], [-1, -2]])
array([[ 0.5       ,  0.73105858],
       [ 0.26894142,  0.11920292]])
>>> tanh([[0, 1], [-1, -2]])
array([[ 0.        ,  0.76159416],
       [-0.76159416, -0.96402758]])
>>> 2 * sigmoid([[0, 2], [-2, -4]]) - 1
array([[ 0.        ,  0.76159416],
       [-0.76159416, -0.96402758]])
```

Many mathematical operations, such as exp, log, abs, sin, cos, +, -, *, /, are elementwise operations

Elementwise Operations




```
Plot[{LogisticSigmoid[x],
      Tanh[x],
      2*LogisticSigmoid[2 x]-1},
     {x, -2, 2}]
```

WolframAlpha

Multiple Outputs

Theano functions support multiple outputs in a list

```
>>> a, b = T.dmatrices('a','b')
>>> diff = a - b
>>> abs_diff = abs(diff)
>>> squared_diff = diff ** 2
>>> f = function([a, b], [diff, abs_diff, squared_diff])
>>> f([[1, 1], [1, 1]], [[0, 1], [2, 3]])
[array([[ 1.,  0.],
       [-1., -2.]]),
 array([[ 1.,  0.],
       [ 1.,  2.]]),
 array([[ 1.,  0.],
       [ 1.,  4.]])]
>>> c, d, e = f([[1, 1], [1, 1]], [[0, 1], [2, 3]])
>>> c
array([[ 1.,  0.],
       [-1., -2.]])
>>> d
array([[ 1.,  0.],
       [ 1.,  2.]])
>>> e
array([[ 1.,  0.],
       [ 1.,  4.]])
```

Argument Default Values

Use Param class to specify function parameters

```
>>> from theano import Param
>>> input, step = T.dscalars('input', 'step')
>>> output = input + step
>>> incre = function([input, Param(step, default=1)], output)
>>> incre(33)
array(34.0)
>>> incre(33, 5)
array(38.0)
```

Set parameter values by name

```
>>> m = T.dscalar('m')
>>> output2 = output * m
>>> incre2 = function([input, Param(step, default=1), Param(m,
default=1, name='multiplier')], output2)
>>> incre2(33)
array(34.0)
>>> incre2(33, 5)
array(38.0)
>>> incre2(33, multiplier=2)
array(68.0)
```

Shared Variables

Theano functions can have internal states

```
>>> state = theano.shared(0.)
>>> inc_new = function([Param(step, default=1)], state+step,
updates=[(state, state+step)])
>>> inc_new()
array(1.0)
>>> inc_new()
array(2.0)
>>> inc_new(5)
array(7.0)
>>> inc_new(3)
array(10.0)
>>> inc_new()
array(11.0)
```

'updates' parameter is a list of shared variable and new expression pairs, which set the values of shared variables with those of new expressions

Shared Variables

getters and setters of shared variables

```
>>> state.get_value()
array(11.0)
>>> inc_new()
array(12.0)
>>> state.get_value()
array(12.0)
>>> inc_new(5)
array(17.0)
>>> state.get_value()
array(17.0)
>>> state.set_value(0)
>>> inc_new()
array(1.0)
>>> state.get_value()
array(1.0)
>>> inc_new(5)
array(6.0)
>>> state.get_value()
array(6.0)
```

Shared Variables

Shared variables are shared across functions

```
>>> dec = function([Param(step, default=1)], state-step,
updates=[(state, state-step)])
>>> dec()
array(5.0)
>>> dec(5)
array(0.0)
>>> state.get_value()
array(0.0)
>>> inc_new()
array(1.0)
>>> state.get_value()
array(1.0)
```

Expression Replacement

Use `givens` parameter to replace part of an expression with a new expression

```
>>> x = T.dmatrix('x')
>>> s = 1 / (1 + T.exp(-x))
>>> t = T.tanh(x)
>>> tanh1 = function([x], t)
>>> tanh2 = function([y], 2 * s - 1, givens={x: 2 * y})
>>> tanh1([[0, 1], [3, 5]])
array([[ 0.        ,  0.76159416],
       [ 0.99505475,  0.9999092 ]])
>>> tanh2([[0, 1], [3, 5]])
array([[ 0.        ,  0.76159416],
       [ 0.99505475,  0.9999092 ]])
```

Use theano.clone to do more advanced replacements

```
>>> tanh3 = function([x], theano.clone(2 * s - 1, replace={x: 2 * x}))
>>> tanh3([[0, 1], [3, 5]])
array([[ 0.        ,  0.76159416],
       [ 0.99505475,  0.9999092 ]])
```

Derivatives

Automatic derivatives with `tensor.grad`

```
>>> x = T.dscalar('x')
>>> y = x ** 2
>>> dy = T.grad(y, wrt=x)
>>> f = function([x], dy)
>>> f(3.0)
array(6.0)
>>> f(5.0)
array(10.0)
```

Theano optimizes expressions when compiling functions

```
>>> pp(dy)
'((fill((x ** TensorConstant{2}), TensorConstant{1.0}) *
TensorConstant{2}) * (x ** (TensorConstant{2} -
TensorConstant{1})))'
>>> pp(f.maker.fgraph.outputs[0])
'(TensorConstant{2.0} * x)'
```

Example - Logistic Regression

```
import numpy
import theano
import theano.tensor as T
rng = numpy.random

N = 400
feats = 784
D = (rng.randn(N, feats), rng.randint(size=N, low=0, high=2))
training_steps = 10000

# Declare Theano symbolic variables
x = T.matrix("x")
y = T.vector("y")
w = theano.shared(rng.randn(feats), name="w")
b = theano.shared(0., name=“b")
print "Initial model:"
print w.get_value(), b.get_value()


# Construct Theano expression graph
p_1 = 1 / (1 + T.exp(-T.dot(x, w) - b))   # Probability that target = 1
prediction = p_1 > 0.5                    # The prediction thresholded
xent = -y * T.log(p_1) - (1-y) * T.log(1-p_1) # Cross-entropy loss function
cost = xent.mean() + 0.01 * (w ** 2).sum()# The cost to minimize
gw, gb = T.grad(cost, [w, b])             # Compute the gradient of the cost
```

Example - Logistic Regression

```
# Compile
train = theano.function(
          inputs=[x,y],
          outputs=[prediction, xent],
          updates=((w, w - 0.1 * gw), (b, b - 0.1 * gb)))
predict = theano.function(inputs=[x], outputs=prediction)

# Train
for i in range(training_steps):
    pred, err = train(D[0], D[1])

print "Final model:"
print w.get_value(), b.get_value()
print "target values for D:", D[1]
print "prediction on D:", predict(D[0])
```

Theano in ML and DL

- Deep learning tutorial based on Theano (http://deeplearning.net/tutorial/contents.html)
    - Getting started tutorial (http://deeplearning.net/tutorial/gettingstarted.html)
- Pylearn2: machine learning library based on Theano (http://deeplearning.net/software/pylearn2/)
- GroundHog: RNN framework based on Theano (https://github.com/lisa-groundhog/GroundHog)
- Faster convolution methods (https://github.com/Theano/Theano/wiki/Convolution-methods)
- …

Tutorial 3. Deep Learning Toolbox

Xingyu ZENG(xyzeng@ee.cuhk.edu.hk)

DeepLearnToolbox

- A open-source Matlab toolbox for Deep Learning

- You can download in
  https://github.com/rasmusbergpalm/DeepLearnToolbox

- If you use this toolbox in your research please cite
  @MASTERSTHESIS\{IMM2012-06284, author = "R. B. Palm", title = "Prediction as a candidate for learning deep hierarchical models of data", year = "2012", }

DeepLearnToolbox

- Advantage
  - Matlab, easy to use
  - Open-source
- Disadvantage
  - Only CPU version, slow

Install Steps

1. Download the toolbox,
2. Addpath(genpath('DeepLearnToolbox'))

DeepLearnToolbox

- A Matlab toolbox for Deep Learning
  - NN/ - A library for Feedforward Backpropagation Neural Networks
  - CNN/ - A library for Convolutional Neural Networks
  - DBN/ - A library for Deep Belief Networks
  - SAE/ - A library for Stacked Auto-Encoders
  - CAE/ - A library for Convolutional Auto-Encoders
  - util/ - Utility functions used by the libraries
  - data/ - Data used by the examples
  - tests/ - unit tests to verify toolbox is working

DeepLearnToolbox

Feedforward Backpropagation Neural Networks

- Common Function
  - nnsetup.m
    - To setup one network
  - nntrain.m
    - To train one network
  - nnpredict.m
    - To test samples with one network

DeepLearnToolbox

- nnsetup.m

Usage example:

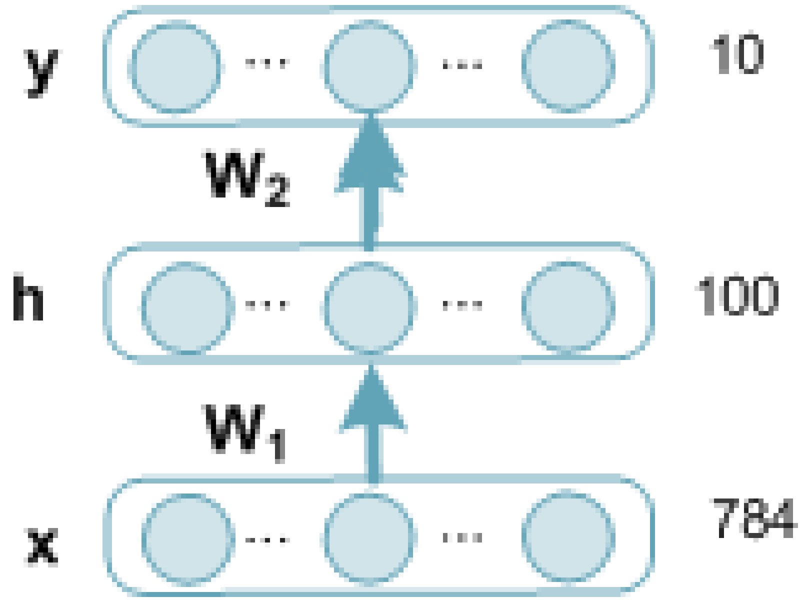
nn = nnsetup([784 100 10]); % to build up one three layers network

nn.activation_function = 'sigm';

nn.output= 'softmax';

h=sigmoid(W_1*x);

y=softmax(W_2*h);



DeepLearnToolbox

- nntrain.m

Usage example:

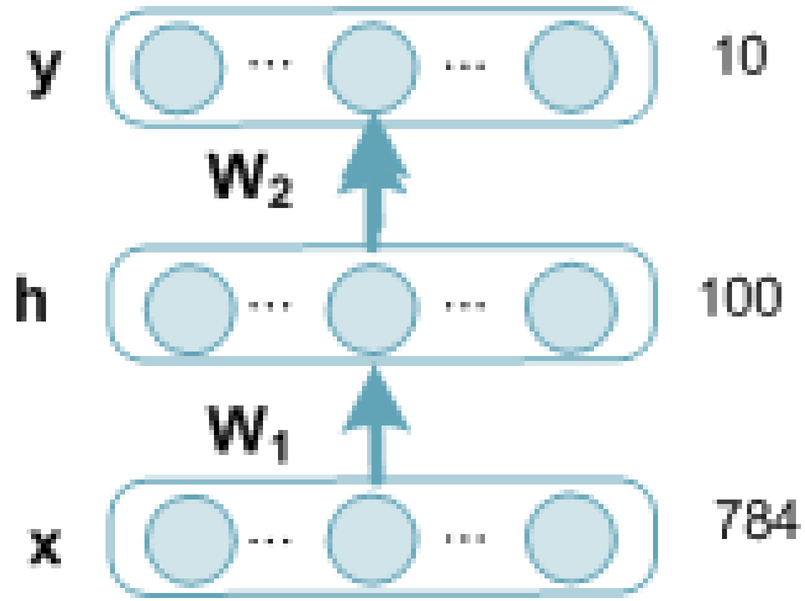
```
nn.learningRate = 0.1;
opts.numepochs =  1; %Number of full sweeps through data
opts.batchsize = 100;  %Take a mean gradient step over this many
                        %samples

nn = nntrain(nn, train_x, train_y, opts);
```

DeepLearnToolbox

- nnpredict.m

Usage example:

```
labels = nnpredict(nn, test_x);
```



Notes:

1. labels, the classes predicted by nn
2. nn.a{end}, the values of the output layer

```
[~,labels]=max(nn.a{end},[],2);
```

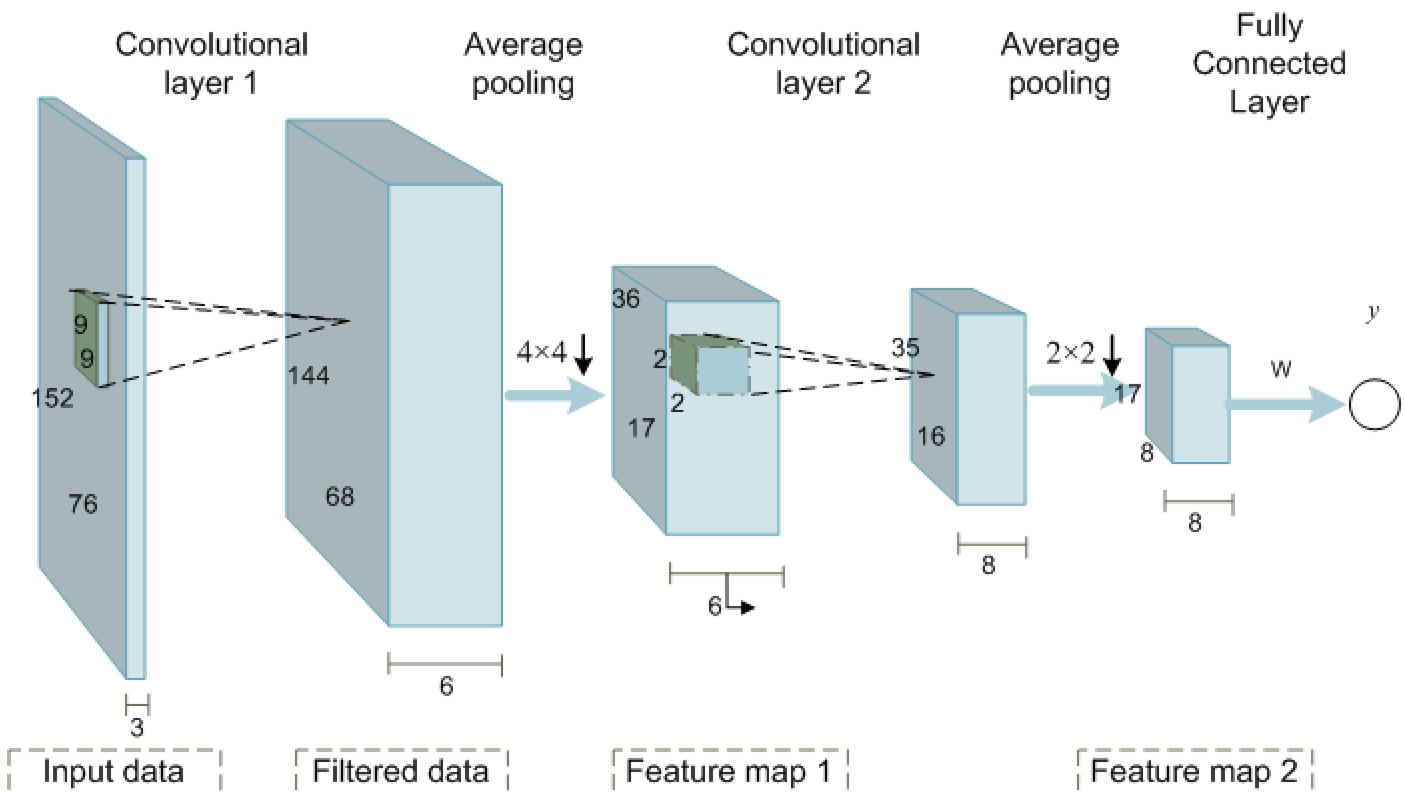
DeepLearnToolbox

Convolutional Neural Networks

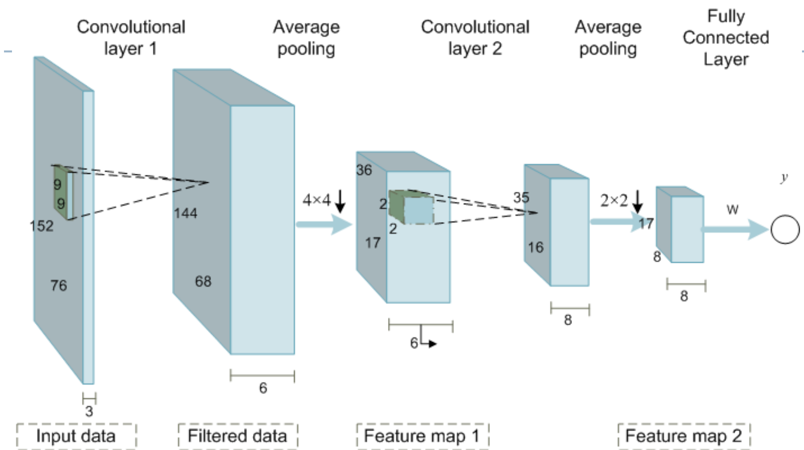
- Common Function
  - cnnsetup.m
    - To setup one convolutional network
  - cnntrain.m
    - To train one convolutional network
  - cnnff.m
    - Forward step with one convolutional network
  - cnntest.m
    - To test samples with one convlutional network

DeepLearnToolbox

- Suppose input size: 152*76*3



DeepLearnToolbox



- cnnsetup.m

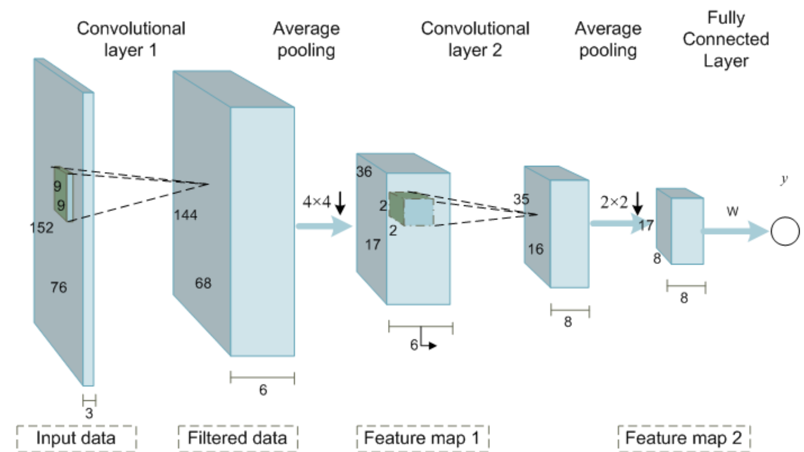
Usage example:

```
cnn.layers = {
    struct('type', 'i') %input layer
    struct('type', 'c', 'outputmaps', 6, 'kernelsize', 9) %convolution layer
    struct('type', 's', 'scale', 4) %sub sampling layer
    struct('type', 'c', 'outputmaps', 8, 'kernelsize', 2) %convolution layer
    struct('type', 's', 'scale', 2) %subsampling layer
};

% the size of {train_x, train_y} is useful for setup cnn
cnn = cnnsetup(cnn, train_x, train_y);
```

DeepLearnToolbox

- cnntrain.m

Usage example:

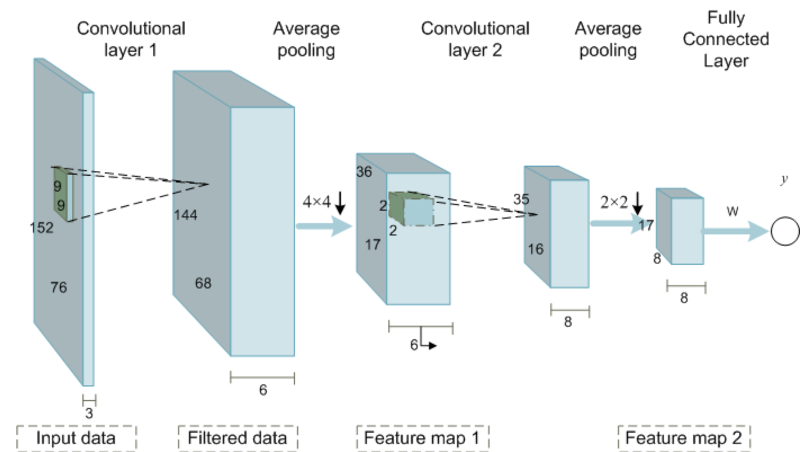



```
opts.alpha = 1; % learning rate
opts.batchsize = 50;
opts.numepochs = 1;

cnn = cnntrain(cnn, train_x, train_y, opts);
```

DeepLearnToolbox

- cnnff.m

Usage example:



cnn = cnnff(cnn, x);

Notes:

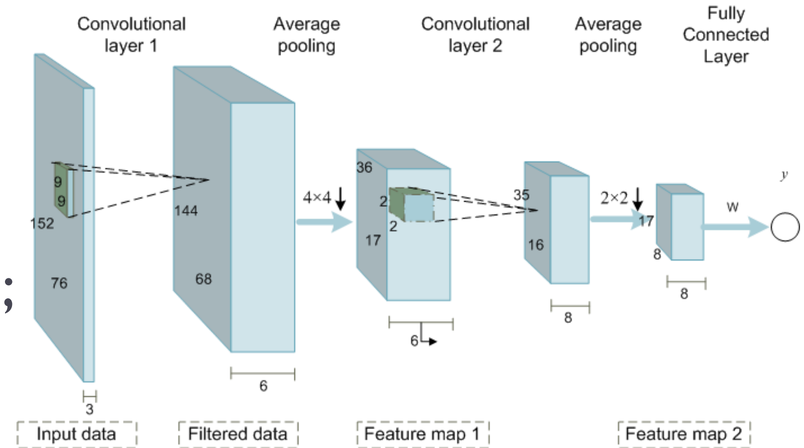
1. cnn.o, the output values of the output layer
2. cnn.fv, the feature of the input samples
3. cnn.o=sigm(cnn.ffW * cnn.fv + repmat(cnn.ffb, 1, size(cnn.fv, 2)));

DeepLearnToolbox

- cnntest.m

Usage example:

[er, bad] = cnntest(cnn, test_x, test_y);



Notes:

1. er, error fraction value
2. bad, index of misclassified testing samples

DeepLearnToolbox

- More examples can be found in

https://github.com/rasmusbergpalm/DeepLearnToolbox/blob/master/README.md

- More details about deep learning,

Book, 'Learning Deep Architectures for AI'

Tutorial 4. MatConvNet

Xingyu ZENG(xyzeng@ee.cuhk.edu.hk)

MatConvNet

- A open-source Matlab toolbox for Convolution Network

- You can download in

  https://github.com/vlfeat/matconvnet

- *If you use MatConvNet in your work, please cite:*

  "MatConvNet - Convolutional Neural Networks for MATLAB", A. Vedaldi and K. Lenc, arXiv:1412.4564, 2014.

MatConvNet

- Advantage
  - Matlab, easy to use
  - Pretrained models(VGG, AlexNet)
  - Support GPU
- Disadvantage
  - Complicated than DeepLearnToolbox
  - Support only Convolution Network

MatConvNet

- Installing and compiling the library
  - Matlab
    - run <path to MatConvNet>/matlab/vl_setupnn
    - vl_compilenn();
    - vl_compilenn('enableGpu', true); % for gpu
  - Shell
    - make ARCH=<your arch> MATLABROOT=<path to MATLAB>
    - make ENABLE_GPU=y ARCH=<your arch> MATLABROOT=<path to MATLAB> CUDAROOT=<path to CUDA>

Notes: the gpu version requre at least MATLAB2013

MatConvNet

- Overview
  1. Core functions
     - Different type of layers, including convolution, dropout
  2. Simple CNN functions
     - A simple wrapper for CNN, including training, testing, display
  3. Utility functions
     - Some helper function to initialize the toolbox

MatConvNet

- Pretrained Models (VGG, AlexNet)

Usage:

1. Download a pre-trained CNN
2. net=load('cnn.mat');
3. im=im-net.normalization.averageImage;
4. res=vl_simplenn(net,im);

Notes:

1. res(i).x, output for i-th layer
2. Each model has one averageImage value

MatConvNet

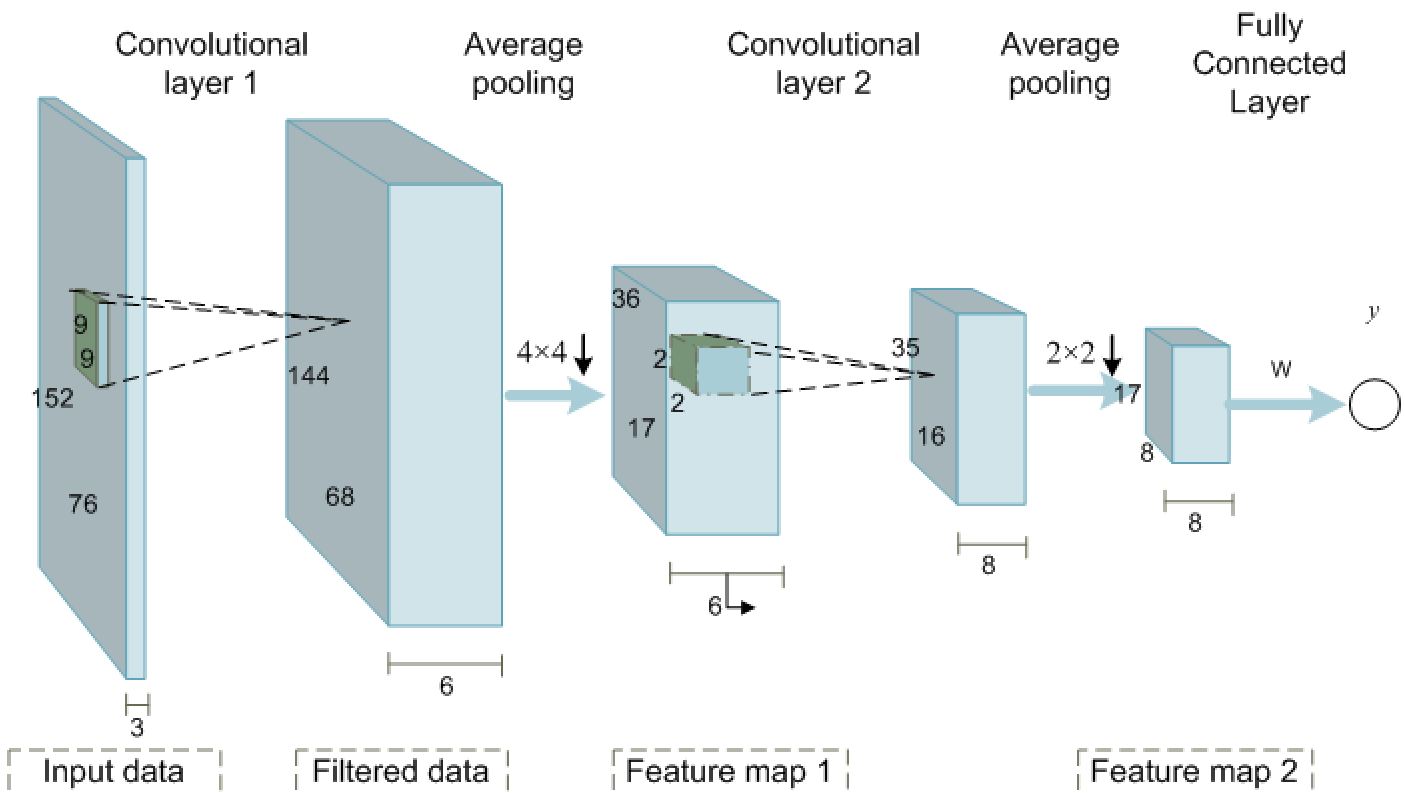
- Notes:
  1. Format: Height*Width*Channels*Num
  2. Pretrained Models:
     - VGG models

       `Very Deep Convolutional Networks for Large-Scale Image Recognition', Karen Simonyan and Andrew Zisserman, arXiv technical,report, 2014
     - AlexNet

       `ImageNet Classification with Deep Convolutional Neural Networks', Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, NIPS, 2012

MatConvNet

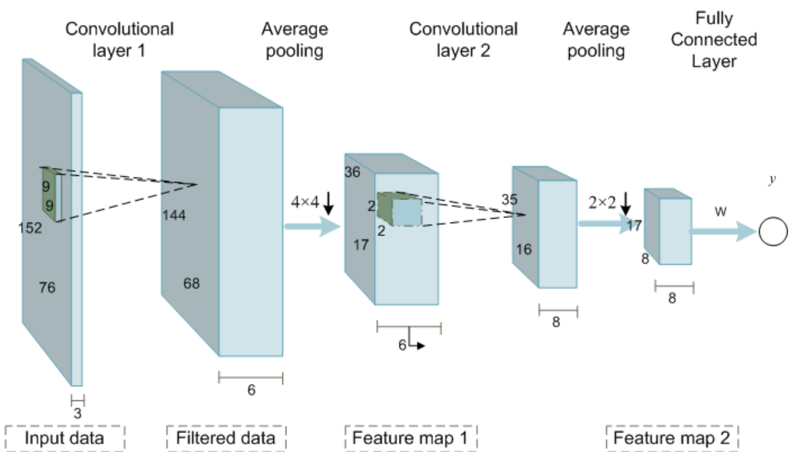
- Suppose input size: 152*76*3



MatConvNet

- Construct one CNN

```
net.layers = {} ;
net.layers{end+1} = struct('type', 'conv', ...
                           'filters', f*randn(9,9,3,6, 'single'), ...
                           'biases', zeros(1, 6, 'single'), ...
                           'stride', 1, ...
                           'pad', 0) ;
net.layers{end+1} = struct('type', 'pool', ...
                           'method', 'avg', ...
                           'pool', [4 4], ...
                           'stride', 4, ...
                           'pad', 0) ;
……..
net.layers{end+1} = struct('type', 'softmaxloss') ;
```



MatConvNet

- Layers
  1. Convolution Layer
  2. Pooling Layer
  3. RELU Layer
  4. Dropout Layer
  5. Softmax Layer
  6. Log-loss layer
  7. Softmax-log-loss-layer
  8. Custom Layer
     1. layer.type = 'custom'
     2. layer.forward: a function handle computing the block.
     3. layer.backward: a function handle computing the block derivative.

MatConvNet

- Training one CNN

Usage:

1. opts.train.batchSize=100;
2. opts.train.numEpochs=100;
3. opts.train.useGpu=false;
4. opts.train.learningRate=0.001;
5. [net,info]=cnn_train(net,imdb,@getBatch,opts);

Notes:

1. getBatch: one function, [im,labels]=getBatch(imdb,ind);
2. info: error, objective, topFiveError….

MatConvNet

- More details can be found in

http://www.vlfeat.org/matconvnet/matconvnet-manual.pdf

- More examples can be found in

https://github.com/vlfeat/matconvnet/tree/master/examples

- The homepage of MatConvNet:

http://www.vlfeat.org/matconvnet/

Tutorial 7
Introduction to Caffe

Kai KANG (kkang@ee.cuhk.edu.hk)

Outline

- General Introduction
  - What is Caffe?
  - Why using Caffe?
  - How to get started?
- Caffe in details
  - Model and protocol buffers
  - Forward and backward computations
  - Loss and supervision
  - Solver and optimizations
  - Data and database
  - Interfaces and tools

What is Caffe?

- Open source deep learning framework maintained by Berkeley Vision and Learning Center (BVLC)
- Created by Yangqing Jia as an improved version of DeCaf
- Mainly written in C++ and CUDA C with Python and Matlab interfaces
- Tools, reference models and sample recipes included

Why Using Caffe? (Official)

- Expression: models and optimizations are defined as plaintext schemas instead of code.
- Speed: for research and industry alike speed is crucial for state-of-the-art models and massive data.
- Modularity: new tasks and settings require flexibility and extension.
- Openness: scientific and applied progress call for common code, reference models, and reproducibility.
- Community: academic research, startup prototypes, and industrial applications all share strength by joint discussion and development in a BSD-2 project.

Why We Use Caffe?

- Good starting point
- Reliability, especially for large scale problem
- Speed
- Popularity

How to Get Started?

- Official website (http://caffe.berkeleyvision.org)
- Download from the GitHub page (https://github.com/BVLC/caffe)
- Follow the installation instructions to compile and test (http://caffe.berkeleyvision.org/installation.html)
- Try the tutorials and reference models (http://caffe.berkeleyvision.org/tutorial/)
- Look through the detailed API documentations (http://caffe.berkeleyvision.org/doxygen/annotated.html)

Model
and
Protocol Buffers

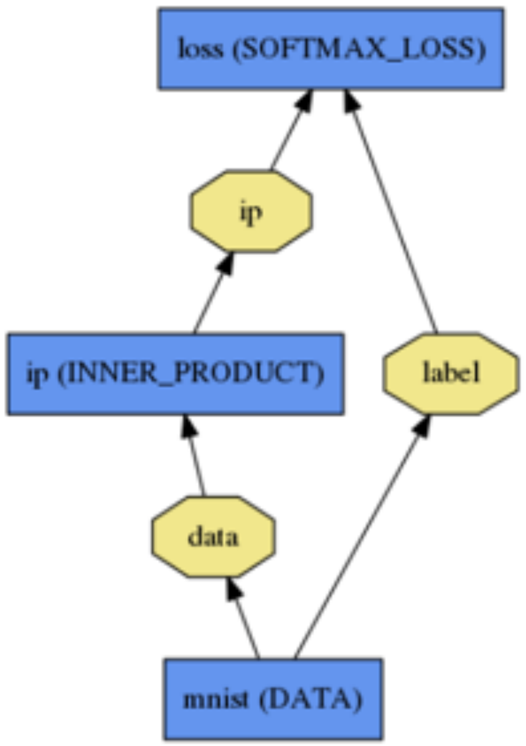
What is a Network?

- Text representations: human-readable network configurations
- Parameter representations: trained network parameters
- Class representations: network data and behavior implementations

What is a Caffe net?

```
name: "CaffeNet"
layers {
  name: "data"
  type: IMAGE_DATA
  ...
}
layers {
  name: "conv1"
  type: CONVOLUTION
  ...
}

...

layers {
  name: "loss"
  type: SOFTMAX_LOSS
  ...
}
```

```
message NetParameter {
  optional string name = 1;
  repeated LayerParameter layers =
2;
  repeated string input = 3;
  repeated int32 input_dim = 4;
  optional bool force_backward = 5
[default = false];
  optional NetState state = 6;
}
```



Text Representations ⇩ Net Definitions

Parameter Representations ⇩ Data Structure

Class Representations ⇩ Implementation

Protocol Buffers

- Google open-source project
- Represent structured data
- Support serialization and de-serialization
- Include C++, Python and Java interfaces
- Compilers compile protocol buffers into classes to access the data

Example - Blobs

- A Blob is a wrapper over the actual data in Caffe
- It's a 4D tensor with dimensions of (num, channels, height and width)
- It can represent data, results and network parameters
    - Data: 20 * 3 * 256 * 256
    - Results: 20 * 96 * 128 * 128
    - Convolution kernels: 96 * 3 * 7 * 7
    - Fully-connected vector: 4096 * 128 * 1 * 1
- Blobs store results of both forward and backward propagations

Example - Blobs

Protocol Buffers definitions

```
message BlobProto {
  optional int32 num = 1 [default = 0];
  optional int32 channels = 2 [default = 0];
  optional int32 height = 3 [default = 0];
  optional int32 width = 4 [default = 0];
  repeated float data = 5 [packed = true];
  repeated float diff = 6 [packed = true];
}
```

Represent multiple blobs

```
message BlobProtoVector {
  repeated BlobProto blobs = 1;
}
```

Example - Blobs

C++ class definitions

```
class Blob {
 public:
  Blob()
       : data_(), diff_(), num_(0), channels_(0), height_(0), width_(0),
       count_(0), capacity_(0) {}
  ...
  inline int num() const { return num_; }
  inline int channels() const { return channels_; }
  inline int height() const { return height_; }
  inline int width() const { return width_; }
  inline int count() const { return count_; }
  ...
  const Dtype* cpu_data() const;
  void set_cpu_data(Dtype* data);
  const Dtype* gpu_data() const;
  const Dtype* cpu_diff() const;
  const Dtype* gpu_diff() const;
  Dtype* mutable_cpu_data();
  Dtype* mutable_gpu_data();
  Dtype* mutable_cpu_diff();
  Dtype* mutable_gpu_diff();
  ...
};  // class Blob
```

Example - Layers

Protocol Buffers

```
message LayerParameter {
  repeated string bottom = 2; // the name of the bottom blobs
  repeated string top = 3; // the name of the top blobs
  optional string name = 4; // the layer name
  repeated NetStateRule include = 32;
  repeated NetStateRule exclude = 33;
  enum LayerType {
    ...
    CONVOLUTION = 4;
    ...
  }
  optional LayerType type = 5; // the layer type from the enum above

  repeated BlobProto blobs = 6;
  repeated float blobs_lr = 7;
  repeated float weight_decay = 8;
  repeated float loss_weight = 35;
  ...
  optional ContrastiveLossParameter contrastive_loss_param = 40;
  ...
  optional TransformationParameter transform_param = 36;

}
```

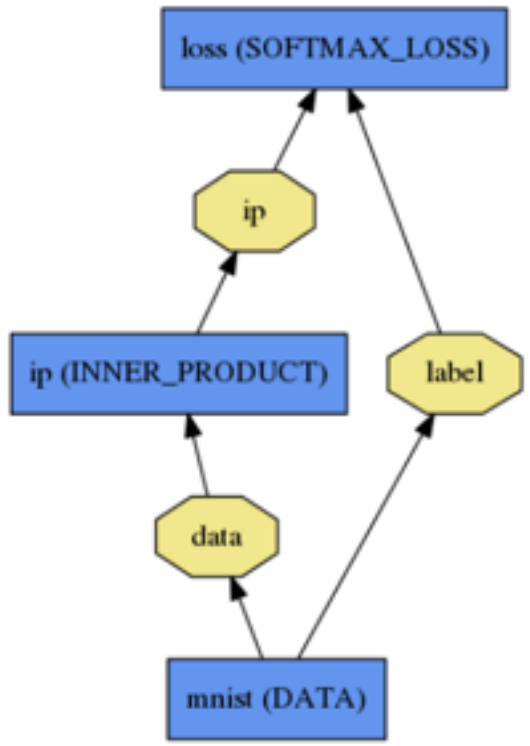
Example - Layers

- C++ class implementations
  - Setup: initialize the layer and connections
  - Forward: given input from bottoms compute outputs and pass to tops
  - Backward: given gradients wrt to top outputs, compute gradients wrt to input and sent to bottom. A layer with parameters computes gradients wrt to its parameters and store then internally
- A layer can have both CPU and GPU implementations for forward and backward propagations

Caffe Net

```
name: "CaffeNet"
layers {
  name: "data"
  type: IMAGE_DATA
  ...
}
layers {
  name: "conv1"
  type: CONVOLUTION
  ...
}

...

layers {
  name: "loss"
  type: SOFTMAX_LOSS
  ...
}
```

```
message NetParameter {
  optional string name = 1;
  repeated LayerParameter layers =
2;
  repeated string input = 3;
  repeated int32 input_dim = 4;
  optional bool force_backward = 5
[default = false];
  optional NetState state = 6;
}
```



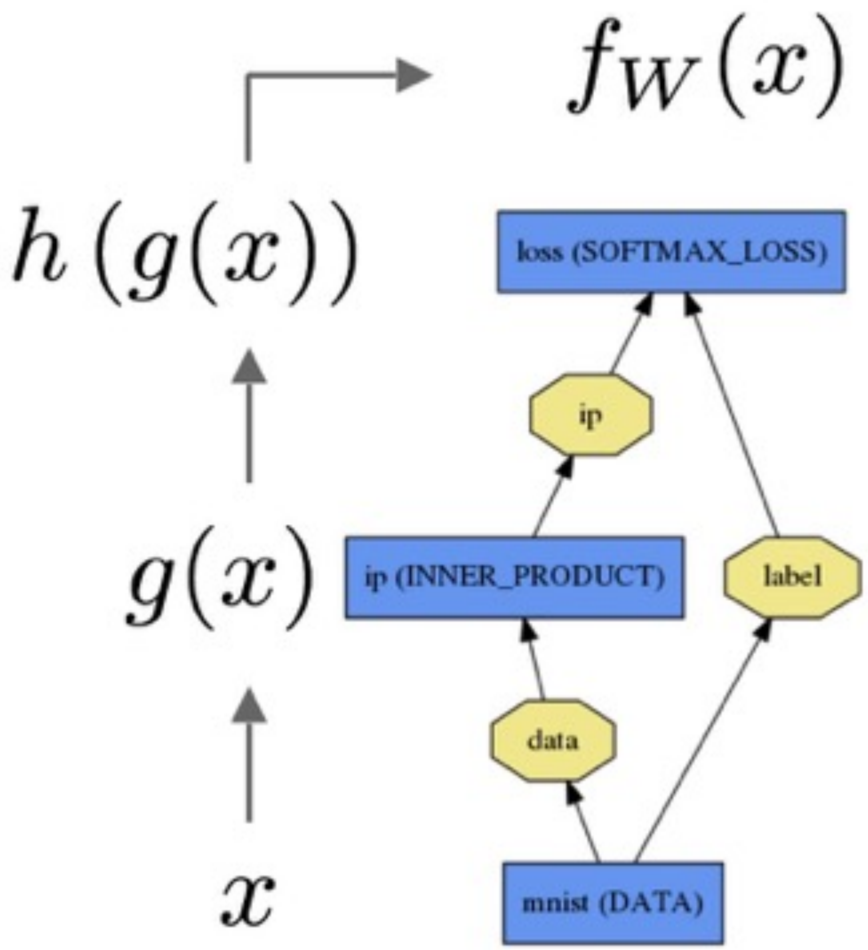
Text Representations ⇩ Net Definitions

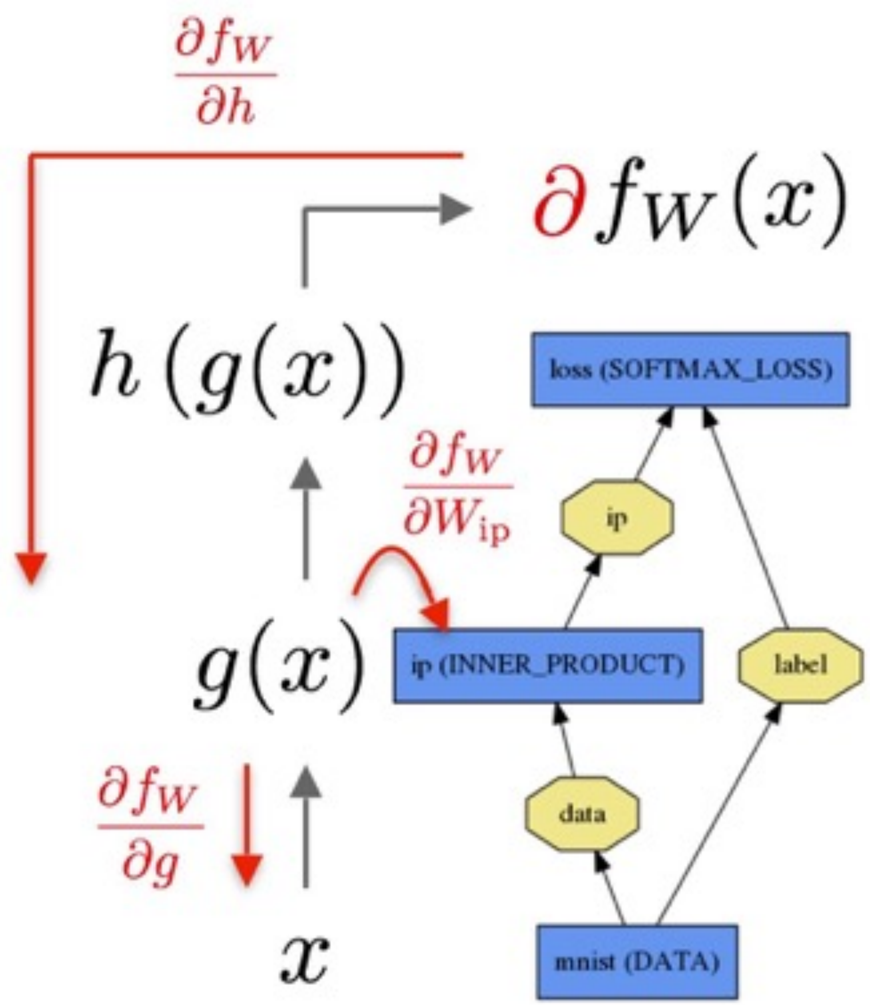
Parameter Representations ⇩ Data Structure

Class Representations ⇩ Implementation

Forward and Backward Propagation

Forward and Backward






```
Net::Forward();
Layer::Forward();
```

```
Net::Backward();
Layer::Backward();
```

Forward and Backward

- Each layer has CPU and GPU implementations for forward and backward:
  - Forward_cpu(), Forward_gpu()
  - Backward_cpu(), Backward_gpu()
- Solver calls forward first to get output and loss, then calls backward to calculate gradients of the model and update parameters

Loss and Supervision

Loss and Supervision

- Caffe is mainly used in supervised learning
- A loss function is the target function that we want to minimize
- Built-in loss functions:
    - Euclidean loss
    - Hinge loss
    - Multinomial logistic regression loss
    - Sigmoid cross entropy loss
    - Softmax loss
    - …
- Multiple loss functions for multitasking

Loss and Supervision

Single loss function

```
layers {
  name: "loss"
  type: SOFTMAX_LOSS
  bottom: "pred"
  bottom: "label"
  top: "loss"
}
```

Multiple loss functions with weights

```
layers {
  name: "loss_softmax"
  type: SOFTMAX_LOSS
  bottom: "pred"
  bottom: "label"
  top: "loss_softmax"
  loss_weight: 1
}
```

```
layers {
  name: "loss_eclidean"
  type: ECLIDEAN_LOSS
  bottom: "pred"
  bottom: "label"
  top: "loss_eclidean"
  loss_weight: 0.5
}
```

Caffe sums up all losses with non-zero loss_weight

Solver and optimizations

Solver and optimizations

- Solver
  - Creates training and validation networks
  - Iteratively calls forward/backward and updates parameters
  - Evaluates validation networks periodically
  - Snapshots model and solver states periodically

Solver and optimizations

Solver example

```
net: "examples/hdf5_classification/train_val.prototxt"
test_iter: 1000
test_interval: 1000
base_lr: 0.01
lr_policy: "step"
gamma: 0.1
stepsize: 5000
display: 1000
max_iter: 10000
momentum: 0.9
weight_decay: 0.0005
snapshot: 10000
snapshot_prefix: "examples/hdf5_classification/data/train"
solver_mode: GPU
```

Data and Database

Data and Database

- Data are represented in Caffe as Blobs
- Data layers provide data to the network
  - Data can also represented in Protocol Buffers
  - Structured data are serialized in binary format and stored in databases
- Databases
  - LevelDB
  - LMDB
  - HDF5
  - …

Data Layers

- Tops and bottoms: multiple tops (data, label, …), no bottoms
- Transformations: scaling, extracting mean, cropping, mirroring
- Prefetching: loading data in another thread
- Built-in data layers: DataLayer, ImageDataLayer, WindowDataLayer, HDF5DataLayer,…
- Multiple data layers or custom data layers for specific applications

Example - DataLayer

```
layers {
  name: "data"
  type: DATA
  top: "data"
  top: "label"
  data_param {
    source: "examples/imagenet/ilsvrc12_train_leveldb"
    batch_size: 256
  }
  transform_param {
    crop_size: 227
    mean_file: "data/ilsvrc12/imagenet_mean.binaryproto"
    mirror: true
  }
  include: { phase: TRAIN }
}
```

Example - ImageDataLayer

```
layers {
  name: "data"
  type: IMAGE_DATA
  top: "data"
  top: "label"
  image_data_param {
    source: "examples/_temp/file_list.txt"
    batch_size: 50
    new_height: 256
    new_width: 256
  }
  transform_param {
    crop_size: 227
    mean_file: "data/ilsvrc12/imagenet_mean.binaryproto"
    mirror: false
  }
}
```

Interfaces
and
Tools

Interfaces

- Command-line interfaces
- Python interfaces
- Matlab interfaces

Command-line Interfaces

Training

```
# train LeNet
caffe train -solver examples/mnist/lenet_solver.prototxt
# train on GPU 2
caffe train -solver examples/mnist/lenet_solver.prototxt -gpu 2
# resume training from the half-way point snapshot
caffe train -solver examples/mnist/lenet_solver.prototxt \
-snapshot examples/mnist/lenet_iter_5000.solverstate
```

Fine-tuning

```
# fine-tune CaffeNet model weights for style recognition
caffe train -solver examples/solver.prototxt \
-weights models/bvlc_reference_caffenet.caffemodel
```

Testing

```
# testing LeNet in lenet_train_test.prototxt
caffe test -model examples/mnist/lenet_train_test.prototxt \
-weights examples/mnist/lenet_iter_10000.caffemodel -gpu 0 \
-iterations 100
```

Tools

- convert_imageset.bin: convert image datasets
- compute_image_mean.bin: compute mean values
- extract_features.bin: extract features using trained models
- parse_log.sh: parse training log files
- …

Developing New Layers in Caffe

Kai KANG
kkang@ee.cuhk.edu.hk

Outline

- Look before You Leap
- Dive into Development
- Test Your Layers

Look Before You Leap

Look Before You Jump

- Do you really need a new layer?
- Have others implemented it yet?
- Is it worth the effort?

Do You Need a New Layer?

- Caffe may already have the layer
  - Read the documentations (http://caffe.berkeleyvision.org/doxygen/annotated.html)
- You may combine existing layers to have same effect
- Discover special usages of existing layers to simply the problem

Example - Sum Feature Maps

- Use 1x1 convolutions
  - Weights initialized to 1
  - Bias initialized to 0
  - Learning rates and weight decays set to 0

```
layer {
  name: "fusion"
  type: "Convolution"
  bottom: “feature_maps"
  top: "fusion"
  param { lr_mult: 0 decay_mult: 0 }
  param { lr_mult: 0 decay_mult: 0 }
  convolution_param {
    num_output: 1
    pad: 0
    kernel_size: 1
    stride: 1
    weight_filler { type: “constant" value: 1 }
    bias_filler { type: “constant" value: 0 }
  }
}
```

Example - Binary Error

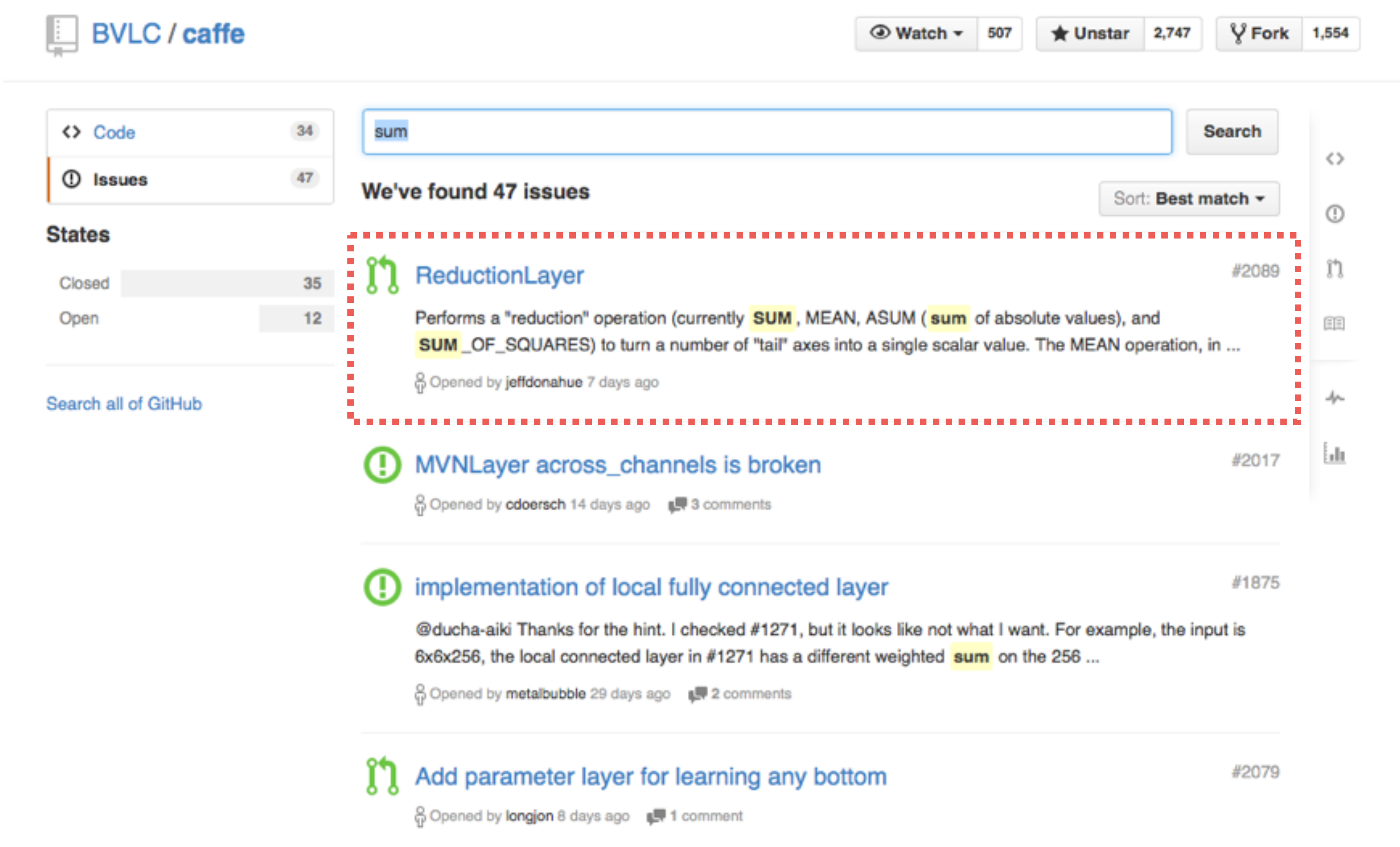
- Threshold layer to get decisions
- Elementwise layer to calculate difference
- Absolute layer to get error
- Write new layer to sum all elements

```
layer {
  name: "fusion"
  type: "Convolution"
  bottom: “feature_maps"
  top: "fusion"
  param { lr_mult: 0 decay_mult: 0 }
  param { lr_mult: 0 decay_mult: 0 }
  convolution_param {
    num_output: 1
    pad: 0
    kernel_size: 1
    stride: 1
    weight_filler { type: “constant" value: 1 }
    bias_filler { type: “constant" value: 0 }
  }
}
```

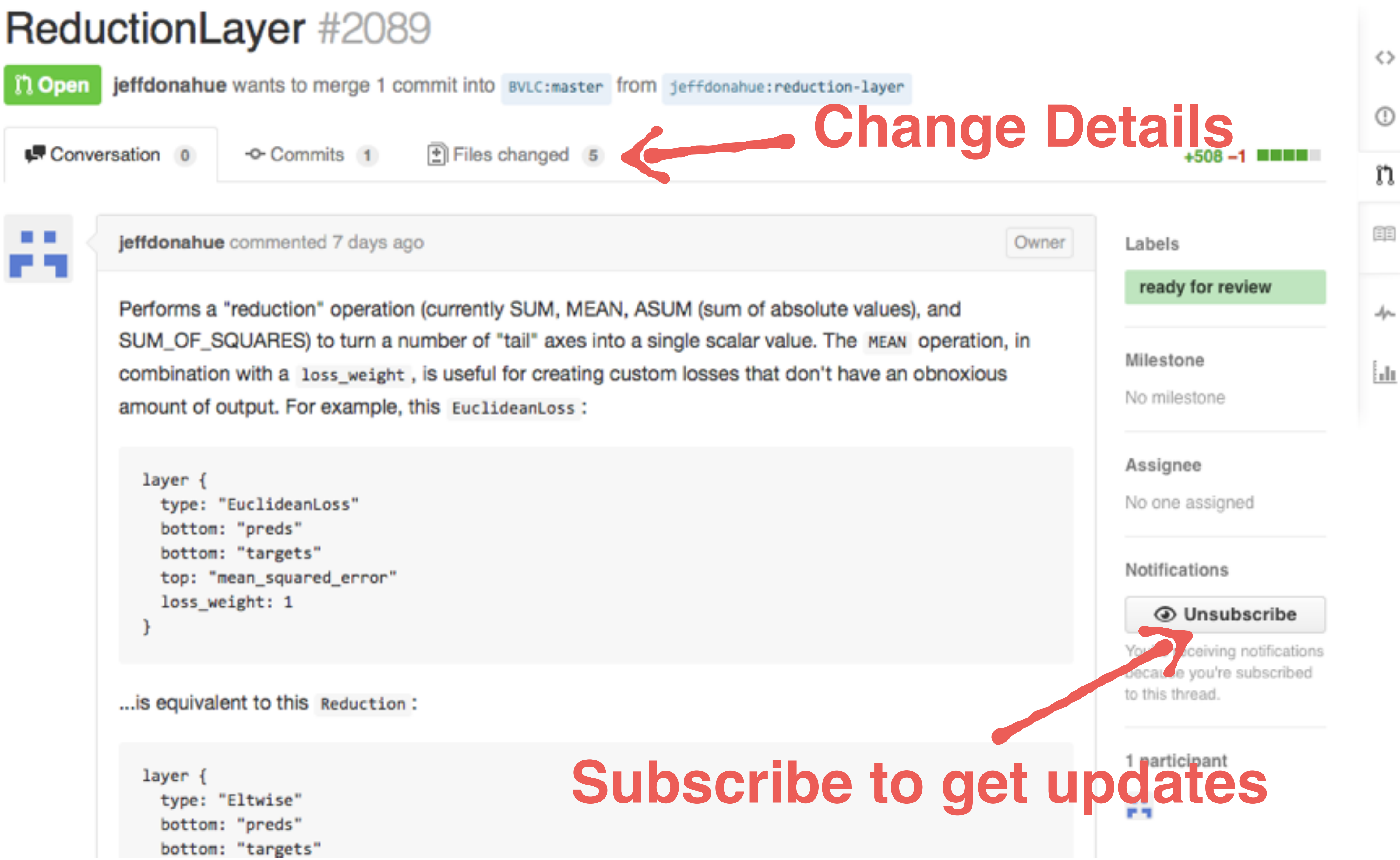
Have Others Done It?

- Discuss with other users
  - Colleagues or labmates
  - Caffe-user group
- Search the web
  - Search Issues or Pull-request on GitHub
  - Search related papers and ask for code
  - Search other users repositories

Example - Sum All Elements



Example - Sum All Elements



Is It Worth the Effort?

- Writing new layers takes time, especially for the first time
- Maybe other frameworks are better options
  - Unsupervised training
  - Generative models

Dive into Development

Dive into Development

- Choose a good starting point
- Find a reference layer
- Follow the development procedure

Starting Point

- Clone the latest Caffe repository from GitHub
  - Avoid known issues
  - Newer versions make development easier
  - Easier to maintain for the future
- Compile and run the test first
- Checkout a new branch and start writing

```
# Clone the latest repository from GitHub
# https://github.com/BVLC/caffe
git clone git@github.com:BVLC/caffe.git

# Compile and run the test
# http://caffe.berkeleyvision.org/installation.html
make all
make test
make runtest

# Checkout a new branch and start writing
git checkout -b mydev
```

Reference Layer

- Layer types
  - Common layers: ArgMax, ConCat, Eltwise, Flatten, InnerProduct, Silence, Softmax…
  - Data layers: Data, ImageData, WindowData…
  - Loss layers: Accuracy, EuclideanLoss, HingeLoss, SigmoidCrossEntropy…
  - Vision layers: Convolution, Pooling, Deconvolution
  - Neuron layers: ReLU, Sigmoid, Tanh…

Development Procedure

- Development wiki (https://github.com/BVLC/caffe/wiki/Development) for latest guides
- Add class declarations
- Implement you layers in C++ and CUDA
- Add Protocol Buffers parameters
- Add layer instantiation and registration
- Compile and debug

1. Add Class Declaration

- Add a class declaration in one of **`common_layer.hpp`**, **`data_layers.hpp`**, **`loss_layers.hpp`**, **`neuron_layers.hpp`** or **`vision_layers.hpp`**
  - Include an inline implementation of **`type()`** and **`*Blobs()`** methods
  - Omit **`*_gpu`** declarations if you'll only implement CPU code

Example - MapDataLayer

data_layer.hpp

```
template<typename Dtype>
class MapDataLayer : public BasePrefetchingDataLayer<Dtype> {
 public:
  explicit MapDataLayer(const LayerParameter& param)
      : BasePrefetchingDataLayer<Dtype>(param),
        label_transformer_(label_trans_param(param.transform_param()), this->phase_) {}
  virtual ~MapDataLayer();
  virtual void DataLayerSetUp(const vector<Blob<Dtype>*>& bottom,
      const vector<Blob<Dtype>*>& top);

  virtual inline const char* type() const { return "MapData"; }
  virtual inline int ExactNumBottomBlobs() const { return 0; }
  virtual inline int ExactNumTopBlobs() const { return 2; }

 protected:
  virtual void InternalThreadEntry();
  DataTransformer<Dtype> label_transformer_;
  Blob<Dtype> transformed_label_;

  shared_ptr<db::DB> db_;
  shared_ptr<db::Cursor> iter_;

 private:
  static TransformationParameter label_trans_param(
          const TransformationParameter& trans_param);
};
```

type()

*Blobs()

2. Implementation

- Implement your layer in layers/your_layer.cpp
  - (optional) LayerSetUp method for one-time initialization
  - Reshape method for computing sizes of top blobs, allocating buffers
  - Forward_cpu for forward propagation
  - Backward_cpu for gradient computations if needed
- Implement GPU versions Forward_gpu and Backward_gpu in layers/your_layer.cu

Example - MapDataLayer

map_data_layer.cpp

```
template <typename Dtype>
void MapDataLayer<Dtype>::DataLayerSetUp(const vector<Blob<Dtype>*>& bottom,
      const vector<Blob<Dtype>*>& top) {
  // Initialize DB
  // Read a data point and use it to initialize the top blob.
  // reshape data map
}

template<typename Dtype>
void MapDataLayer<Dtype>::InternalThreadEntry() {
  // Variable declarations
  // ...

  for (int item_id = 0; item_id < batch_size; ++item_id) {
    maps.ParseFromString(iter_->value());
    // Apply data and label transformations (mirror, scale, crop...)

    // go to the next iter
    iter_->Next();
    if (!iter_->valid()) {
      iter_->SeekToFirst();
    }
  }
}
```

Forward() for PrefetchDataLayer

3. Proto Declarations

- If needed, declare parameters in proto/caffe.proto

```
// NOTE
// Update the next available ID when you add a new LayerParameter field.
//
// LayerParameter next available layer-specific ID: 135 (last added:
reduction_param)
message LayerParameter {
  …
  // Layer type-specific parameters.
   //
   // Note: certain layers may have more than one computational engine
   // for their implementation. These layers include an Engine type and
   // engine parameter for selecting the implementation.
   // The default for the engine is set by the ENGINE switch at compile-
time.
   optional AccuracyParameter accuracy_param = 102;
   optional ArgMaxParameter argmax_param = 103;
   optional ConcatParameter concat_param = 104;
   optional MyLayerParameter mylayer_param = 135;
…
}
```

3. Proto Declarations

- If needed, declare parameters in proto/caffe.proto

```
// Message that stores parameters used by EltwiseLayer
message EltwiseParameter {
  enum EltwiseOp {
    PROD = 0;
    SUM = 1;
    MAX = 2;
  }
  optional EltwiseOp operation = 1 [default = SUM]; // element-wise
operation
  repeated float coeff = 2; // blob-wise coefficient for SUM operation

  // Whether to use an asymptotically slower (for >2 inputs) but stabler
method
  // of computing the gradient for the PROD operation. (No effect for SUM
op.)
  optional bool stable_prod_grad = 3 [default = true];
}
```

3. Layer Instantiation and Registration

- Add layer instantiation and registration in `my_layer.cpp`

```
INSTANTIATE_CLASS(MyLayer);
REGISTER_LAYER_CLASS(My);
```

- And in my_layer.cu

```
INSTANTIATE_LAYER_GPU_FUNCS(MyLayer);
```

Layer Testing and Maintenance

Layer Testing and Maintenance

- Write tests in `test/test_your_layer.cpp`.
- Use `test/test_gradient_check_util.hpp` to check Forward and Backward implementations are in numerical agreement
- Write tests in different scenarios
- Compile, run tests and fix bugs
- Frequently pull the latest changes and update your own layers if there are conflicts

ELEG 4040
Tutorial 9
Caffe Reference Models

Kai KANG
kkang@ee.cuhk.edu.hk

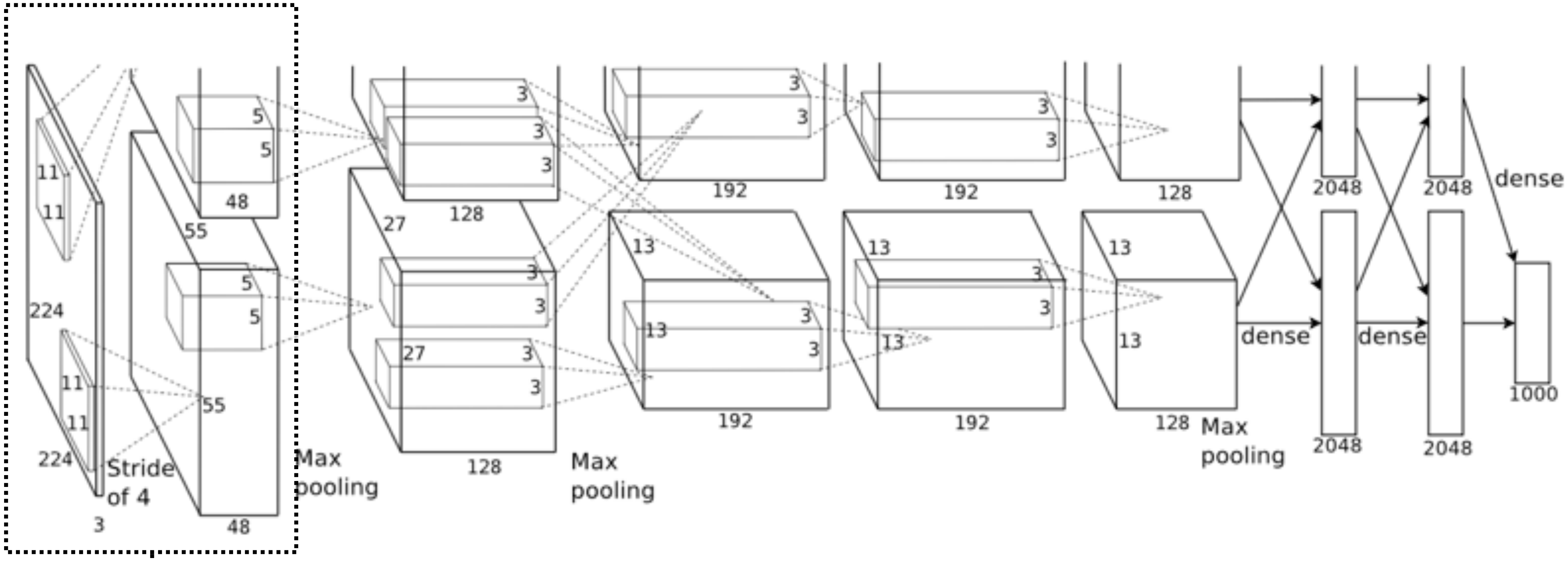
Reference Models

- AlexNet
- GoogLeNet
- Network in Network
- FCN-Xs
- …

AlexNet

- ImageNet 2012 Image Classification Challenge winner
- Originally trained on 2 GTX 580 GPUs because of insufficient memory
- Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.

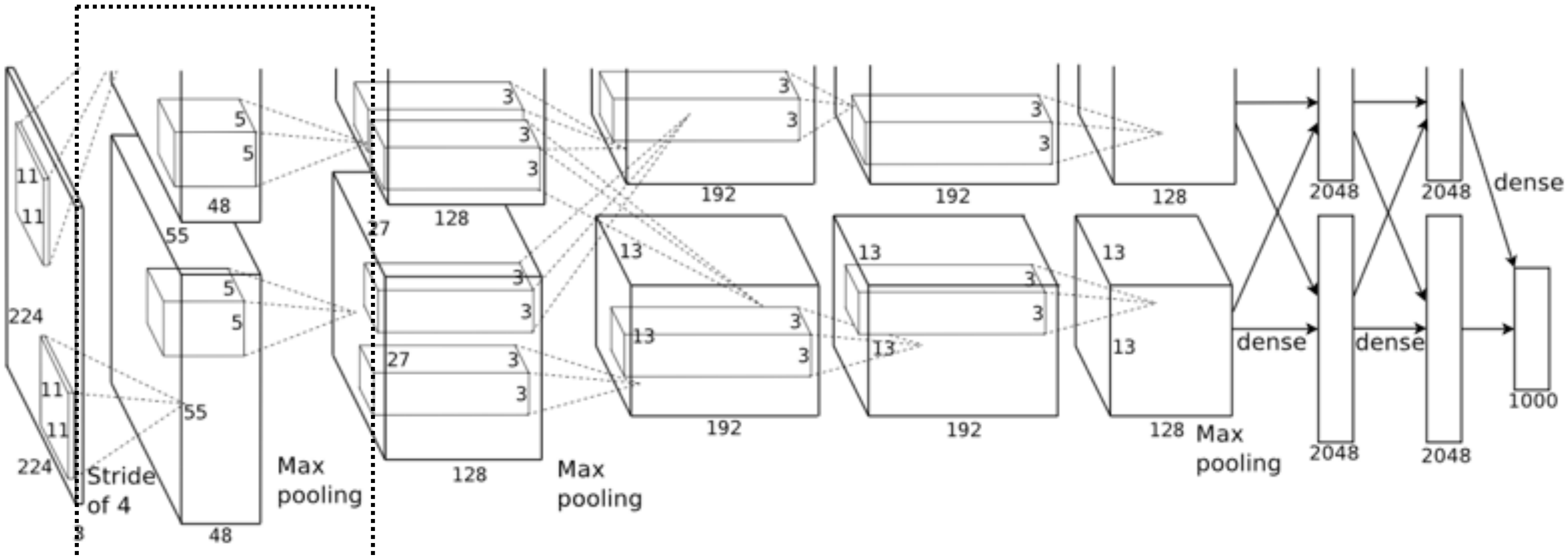
AlexNet




```
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  param { lr_mult: 1 decay_mult: 1 }
  param { lr_mult: 2 decay_mult: 0 }
  convolution_param {
    num_output: 96 kernel_size: 11 stride: 4
    weight_filler { type: "gaussian" std: 0.01 }
    bias_filler { type: "constant" value: 0 }
  }
}
```

put all 2x48 channels on a single GPU

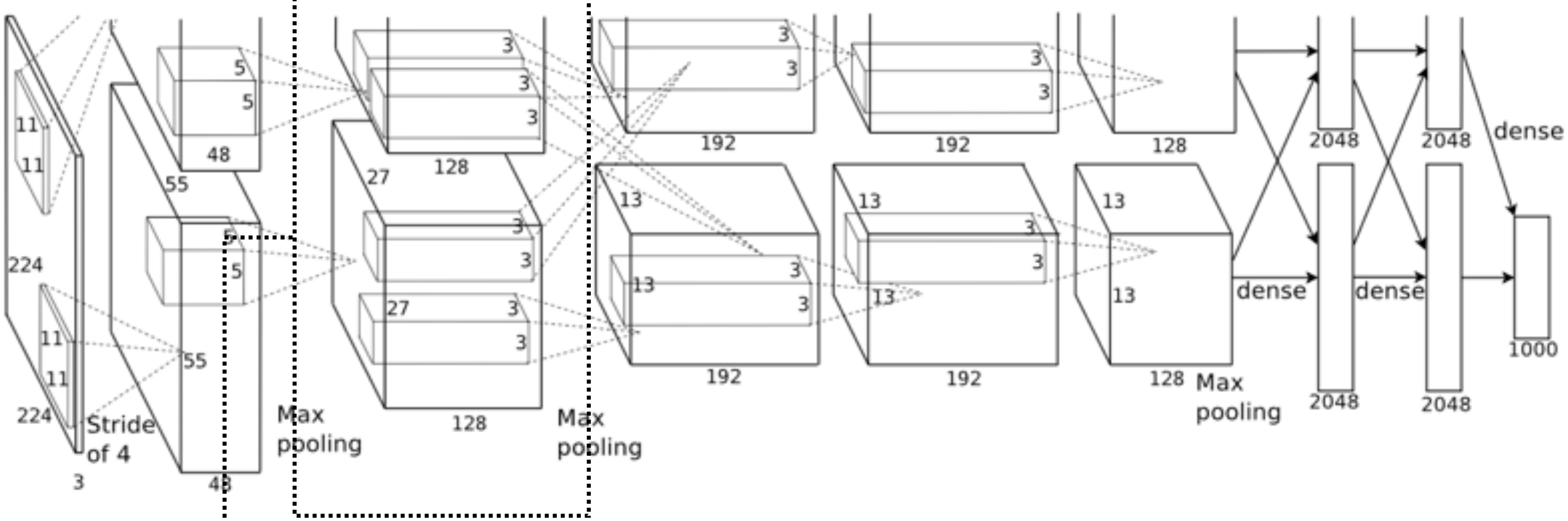
AlexNet




```
layer {
  name: "relu1" type: "ReLU" bottom: "conv1" top: "conv1"
}
layer {
  name: "norm1" type: "LRN" bottom: "conv1" top: "norm1"
  lrn_param { local_size: 5 alpha: 0.0001 beta: 0.75 }
}
layer {
  name: "pool1" type: "Pooling" bottom: "norm1" top: "pool1"
  pooling_param { pool: MAX kernel_size: 3 stride: 2 }
}
```

Local Response Normalization Layer

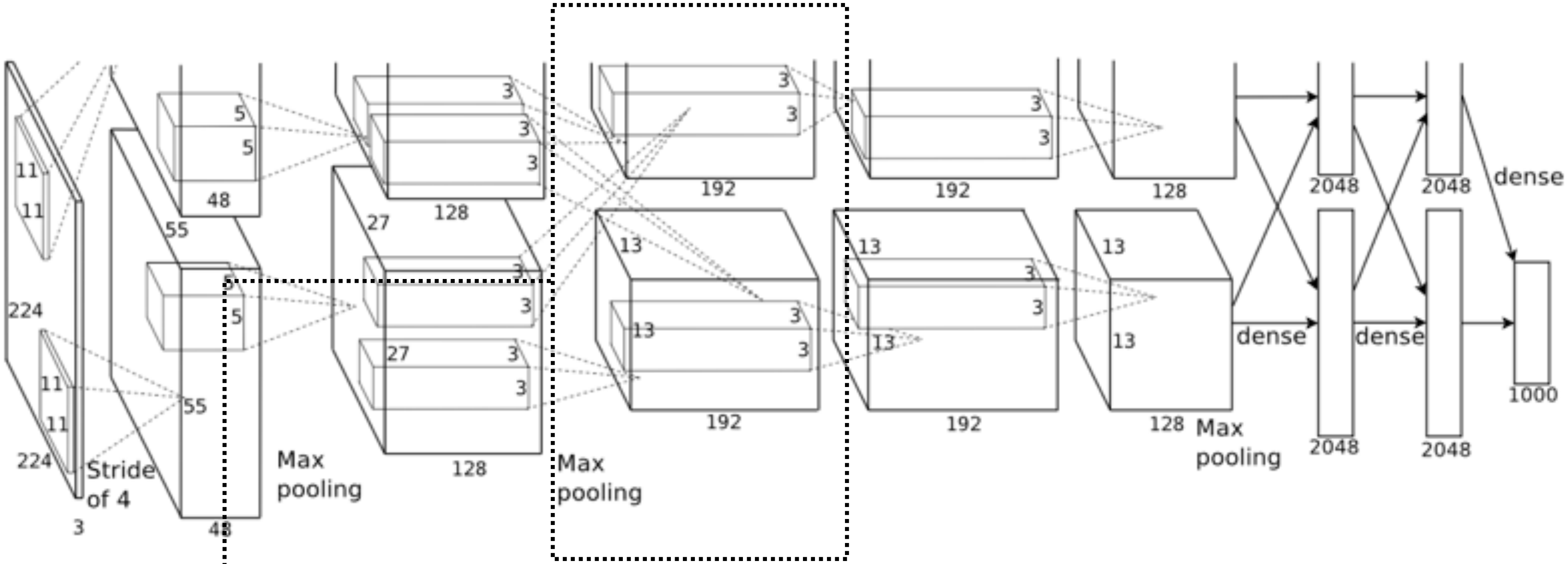
AlexNet




```
layer {
  name: "conv2"
  type: "Convolution"
  bottom: "pool1"
  top: "conv2"
  param { lr_mult: 1 decay_mult: 1 }
  param { lr_mult: 2 decay_mult: 0 }
  convolution_param {
    num_output: 256 pad: 2 kernel_size: 5 group: 2
    weight_filler { type: "gaussian" std: 0.01 }
    bias_filler { type: "constant" value: 0.1 }
  }
}
```

Divide input and output into two groups, equivalent to put on two GPUs

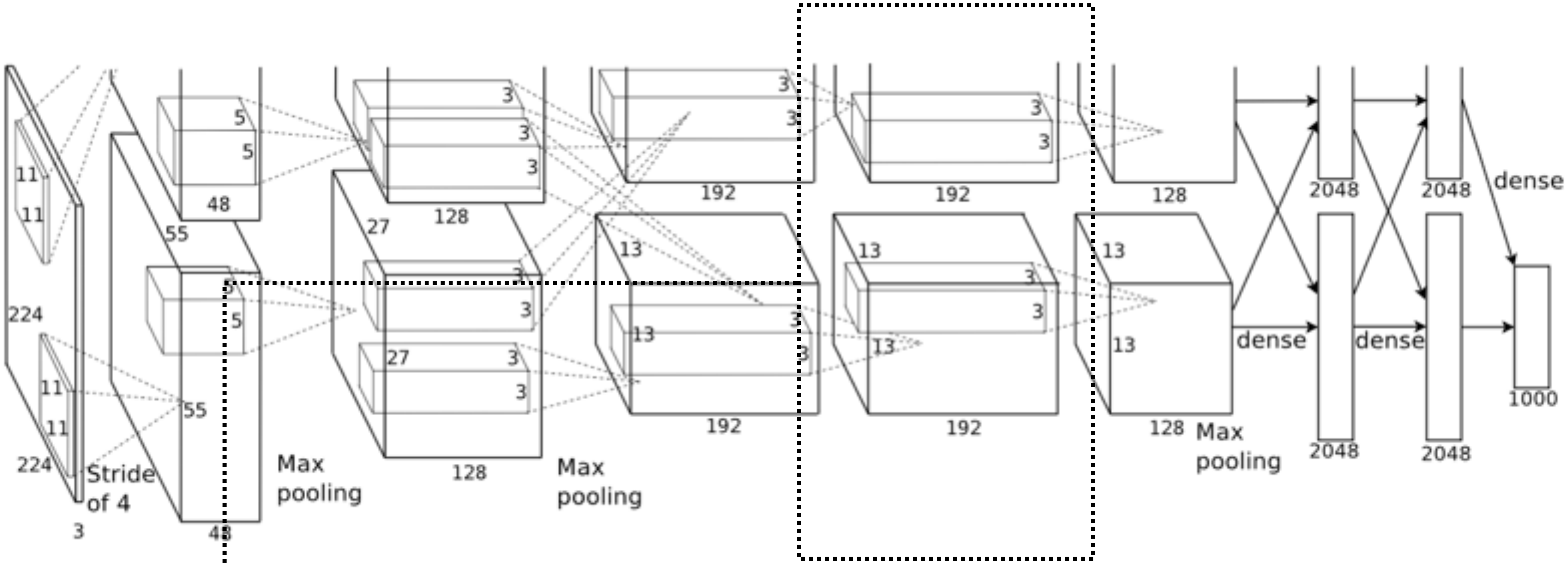
AlexNet




```
layer {
  name: "conv3"
  type: "Convolution"
  bottom: "pool2"
  top: "conv3"
  param { lr_mult: 1 decay_mult: 1 }
  param { lr_mult: 2 decay_mult: 0 }
  convolution_param {
    num_output: 384 pad: 1 kernel_size: 3
    weight_filler { type: "gaussian" std: 0.01 }
    bias_filler { type: "constant" value: 0 }
  }
}
```

No groups, use all input channels

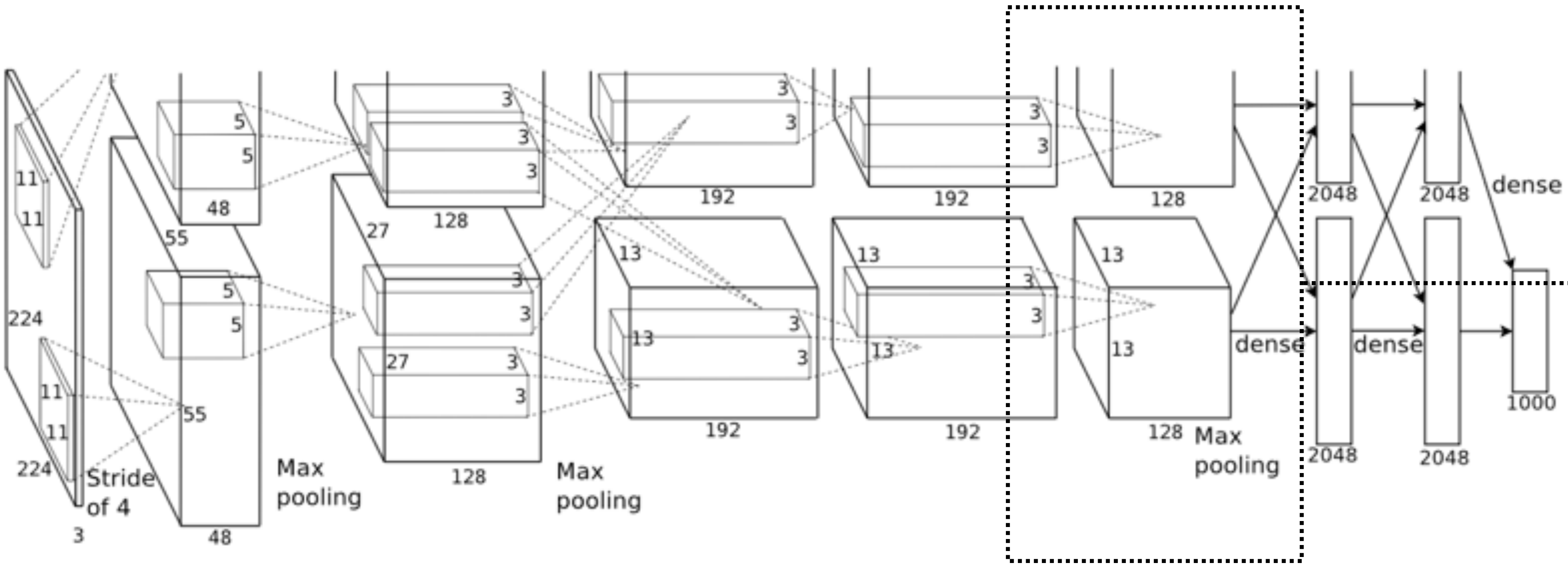
AlexNet




```
layer {
  name: "conv4"
  type: "Convolution"
  bottom: "conv3"
  top: "conv4"
  param { lr_mult: 1 decay_mult: 1 }
  param { lr_mult: 2 decay_mult: 0 }
  convolution_param {
    num_output: 384 pad: 1 kernel_size: 3 group: 2
    weight_filler { type: "gaussian" std: 0.01 }
    bias_filler { type: "constant" value: 0.1 }
  }
}
```

Divide input and output into two groups, equivalent to put on two GPUs

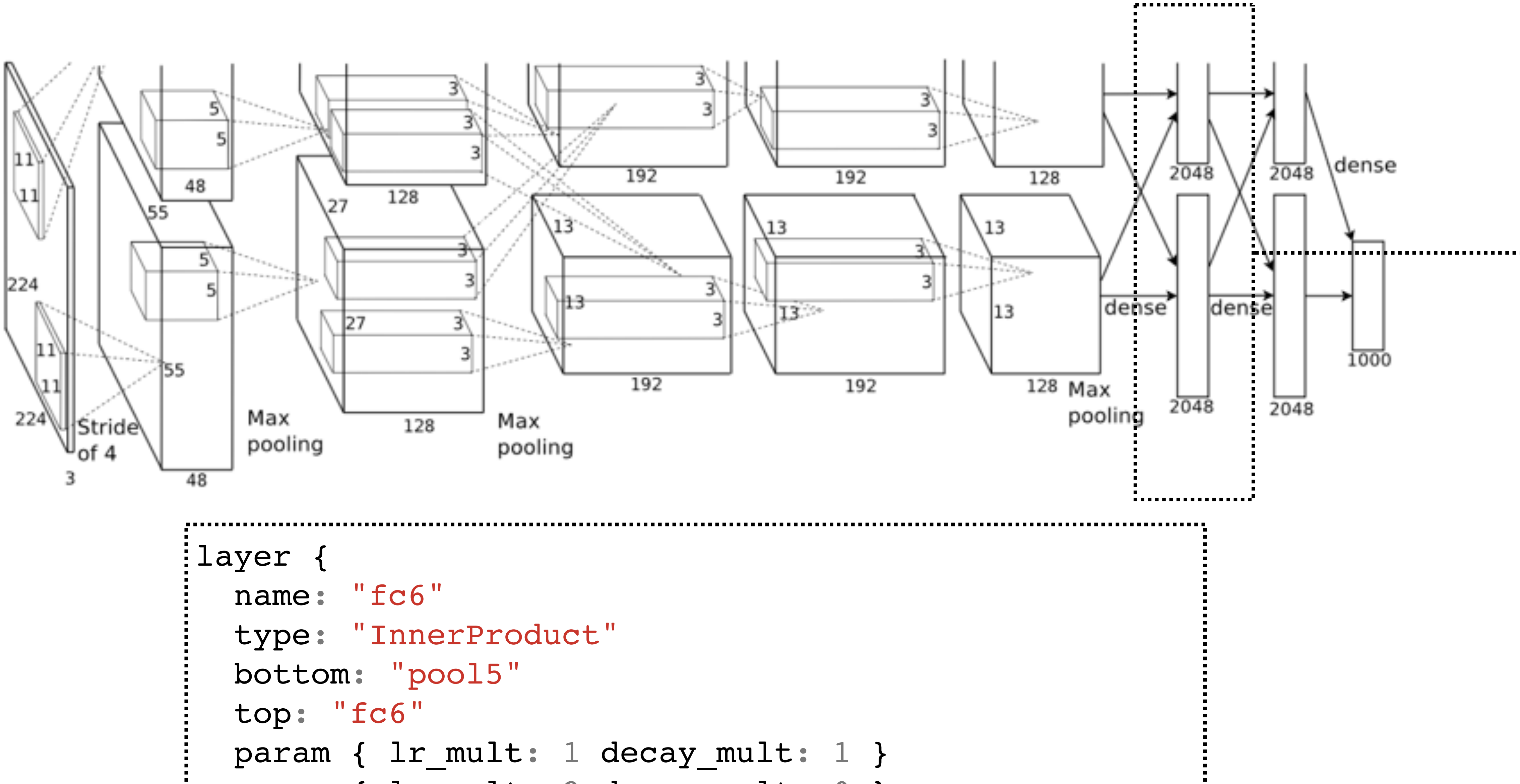
AlexNet




```
layer {
  name: "conv5"
  type: "Convolution"
  bottom: "conv4"
  top: "conv5"
  param { lr_mult: 1 decay_mult: 1 }
  param { lr_mult: 2 decay_mult: 0 }
  convolution_param {
    num_output: 256 pad: 1 kernel_size: 3 group: 2
    weight_filler { type: "gaussian" std: 0.01 }
    bias_filler { type: "constant" value: 0.1 }
  }
}
```

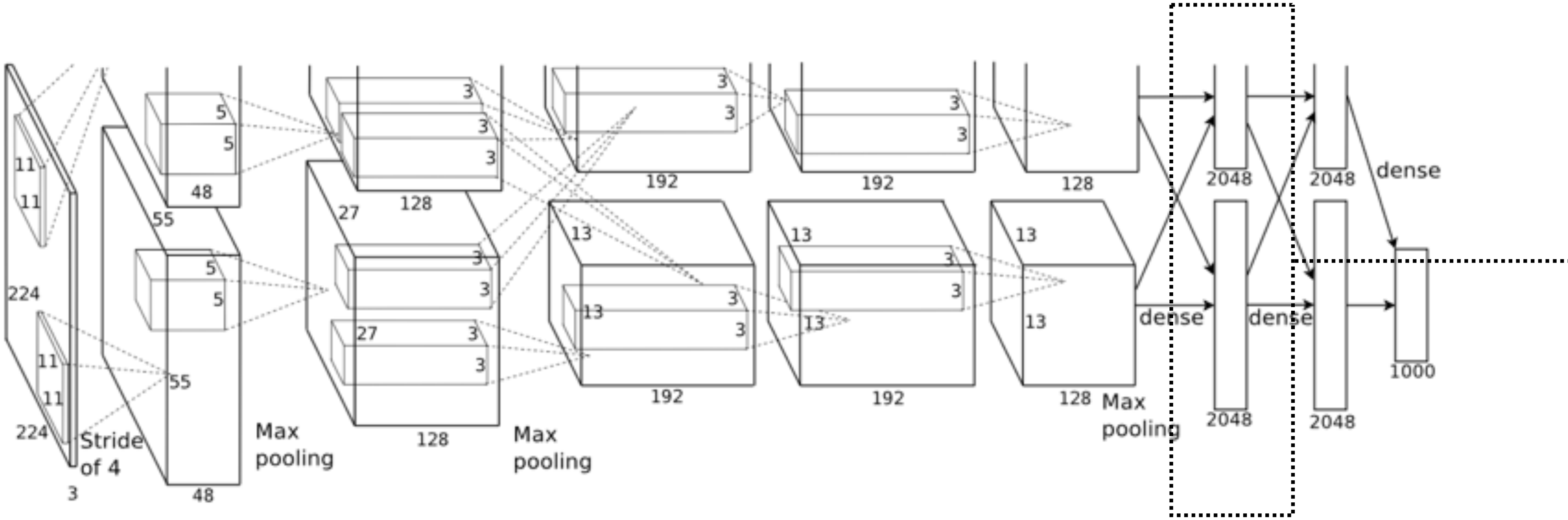
Divide input and output into two groups, equivalent to put on two GPUs

AlexNet



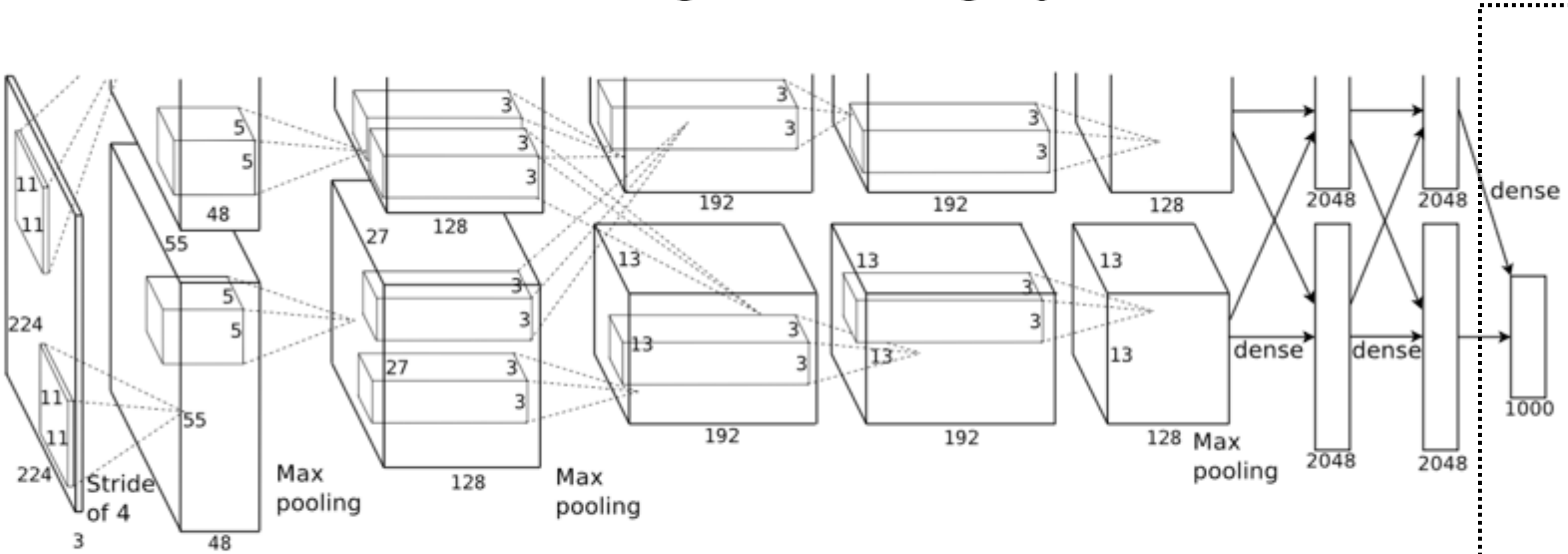

```
layer {
  name: "fc6"
  type: "InnerProduct"
  bottom: "pool5"
  top: "fc6"
  param { lr_mult: 1 decay_mult: 1 }
  param { lr_mult: 2 decay_mult: 0 }
  inner_product_param {
    num_output: 4096
    weight_filler { type: "gaussian" std: 0.005 }
    bias_filler { type: "constant" value: 0.1 }
  }
}
```

AlexNet




```
layer {
  name: "relu6"
  type: "ReLU"
  bottom: "fc6"
  top: "fc6"
}
layer {
  name: "drop6"
  type: "Dropout"
  bottom: "fc6"
  top: "fc6"
  dropout_param { dropout_ratio: 0.5 }
}
```

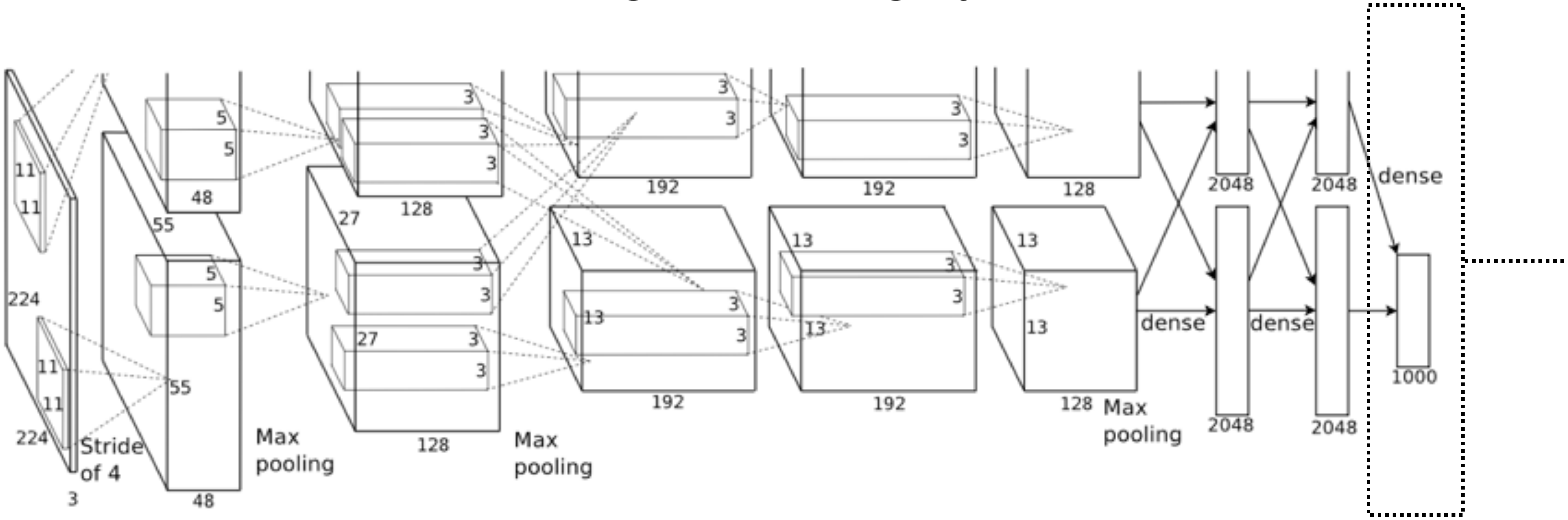
AlexNet




```
layer {
  name: "fc8"
  type: "InnerProduct"
  bottom: "fc7"
  top: "fc8"
  param { lr_mult: 1 decay_mult: 1 }
  param { lr_mult: 2 decay_mult: 0 }
  inner_product_param {
    num_output: 1000
    weight_filler { type: "gaussian" std: 0.01 }
    bias_filler { type: "constant" value: 0 }
  }
}
```

1000 neurons for
1000 classes

AlexNet




```
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "fc8"
  bottom: "label"
  top: "loss"
}
```

AlexNet

- Use `group` to convolve part of input channels to simulate two-GPU implementations
- Use local response normalization layer to normalize input responses
- Differences to original implementation
  - no relighting data augmentation
  - initializing non-zero biases to 0.1 instead of 1

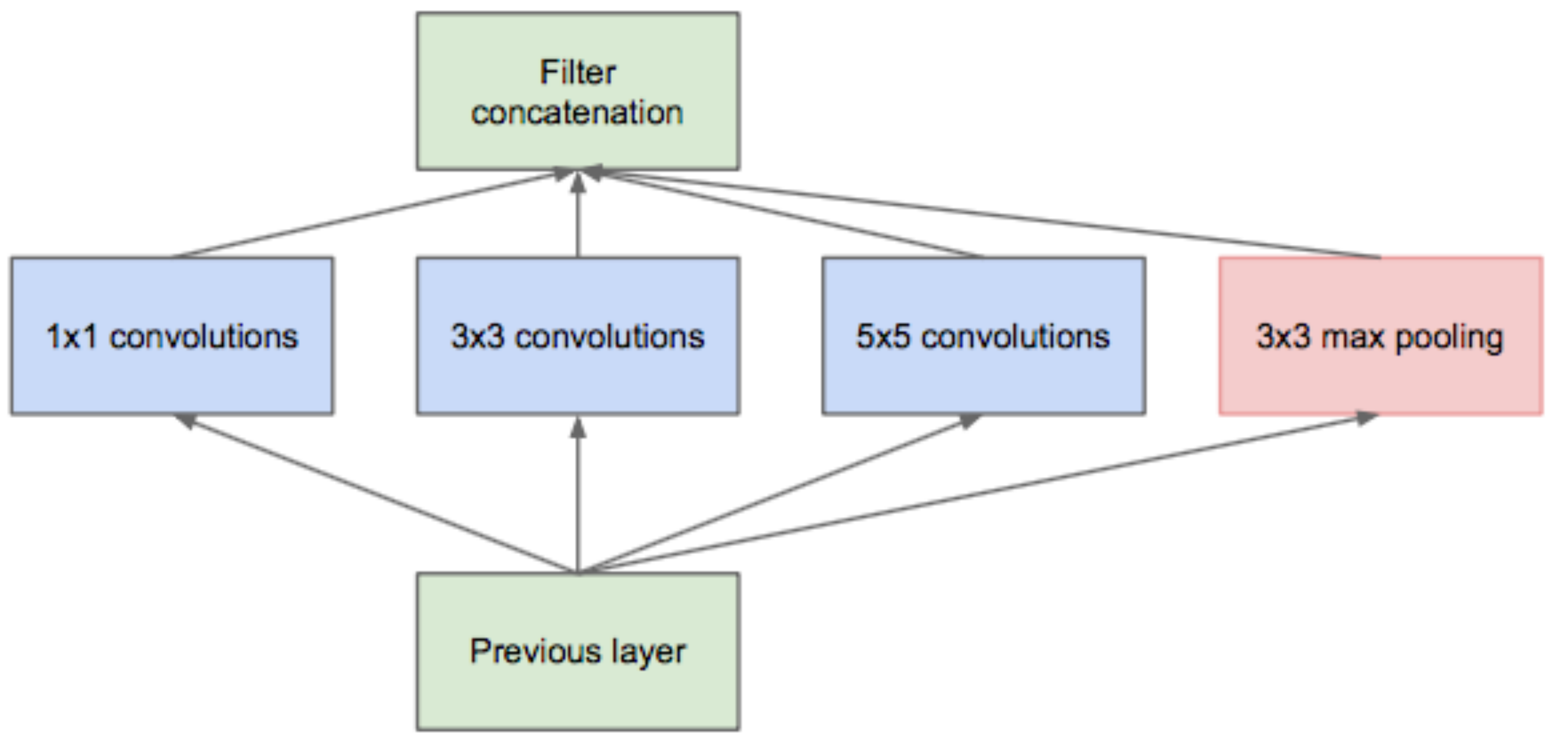
GoogLeNet

- ImageNet 2014 Image Classification and Object Detection winner

- Szegedy, Christian, et al. "Going deeper with convolutions." arXiv preprint arXiv:1409.4842 (2014).

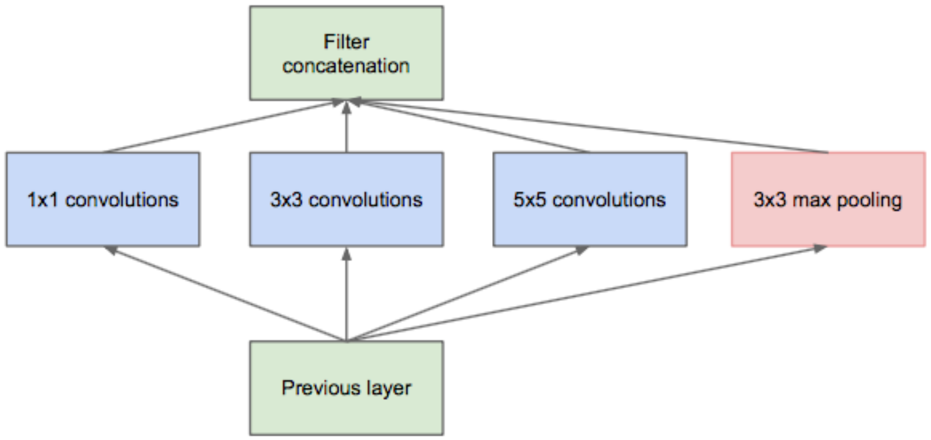
GoogLeNet

Inception Module - Naive Version



GoogLeNet

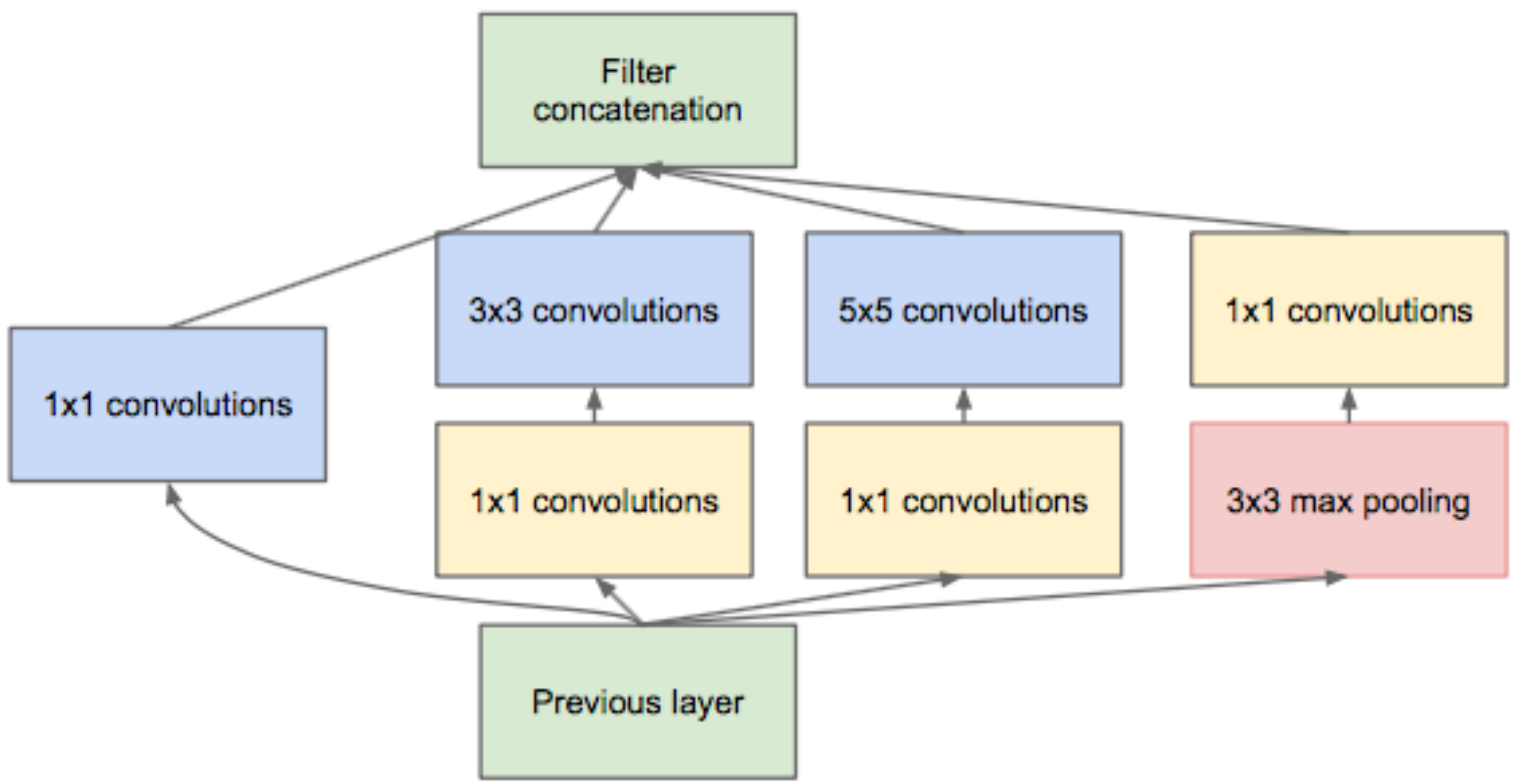
Inception Module - Naive Version




```
layer { name: "inception/1x1" type: "Convolution"
  bottom: "previous" top: "inception/1x1" ... }
layer { name: "inception/relu_1x1" type: "ReLU"
  bottom: "inception/1x1" top: "inception/1x1" }
layer { name: "inception/3x3" type: "Convolution"
  bottom: "previous" top: "inception/3x3" ... }
layer { name: "inception/relu_3x3" type: "ReLU"
  bottom: "inception/3x3" top: "inception/3x3" }
layer { name: "inception/5x5" type: "Convolution"
  bottom: "previous" top: "inception/5x5" ...}
layer { name: "inception/relu_5x5" type: "ReLU"
  bottom: "inception/5x5" top: "inception/5x5" ...}
layer { name: "inception/pool" type: "Pooling"
  bottom: "previous" top: "inception/pool"
  pooling_param { pool: MAX kernel_size: 3 stride: 1 pad: 1 } }
layer { name: "inception/relu_pool_proj" type: "ReLU"
  bottom: "inception/pool_proj" top: "inception/pool_proj" ...}
layer { name: "inception/output"
  type: "Concat"
  bottom: "inception/1x1"
  bottom: "inception/3x3"
  bottom: "inception/5x5"
  bottom: "inception/pool_proj"
  top: "inception/output"
}
```

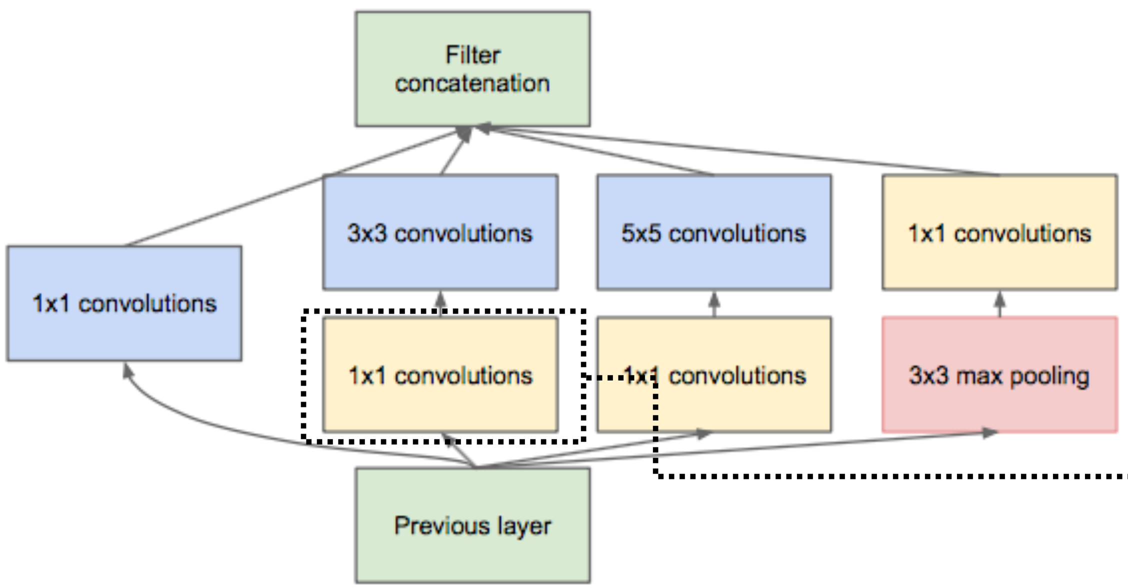
GoogLeNet

Inception Module - Dimension Reduction



GoogLeNet

Inception Module - Dimension Reduction

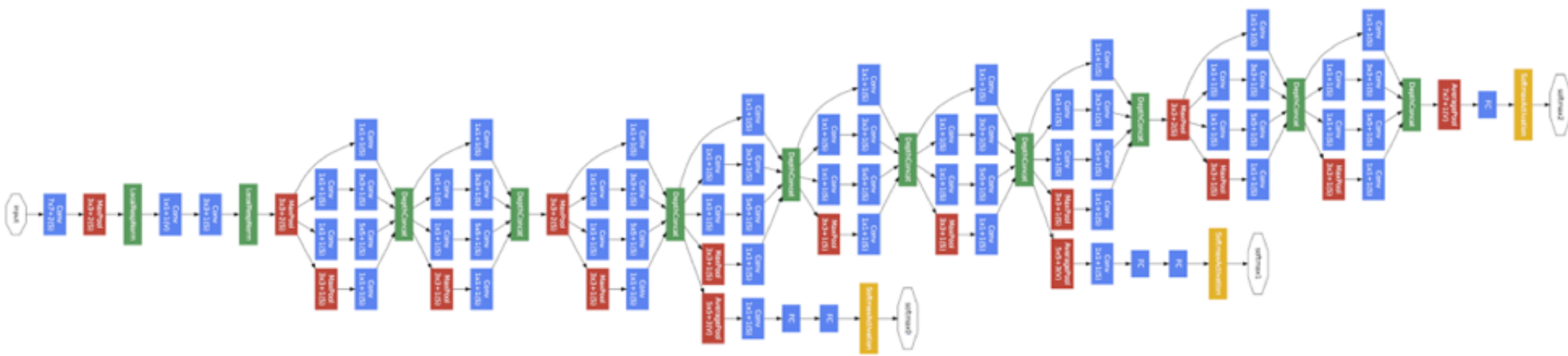



```
layer {
  name: "inception/3x3_reduce"
  type: "Convolution"
  bottom: "previous"
  top: "inception/3x3_reduce"
  param { lr_mult: 1 decay_mult: 1 }
  param { lr_mult: 2 decay_mult: 0 }
  convolution_param {
    num_output: 96 kernel_size: 1
    weight_filler { type: "xavier" std: 0.09 }
    bias_filler { type: "constant" value: 0.2 } }
}
layer {
  name: "inception/relu_3x3_reduce"
  type: "ReLU"
  bottom: "inception/3x3_reduce"
  top: "inception/3x3_reduce"
}
```

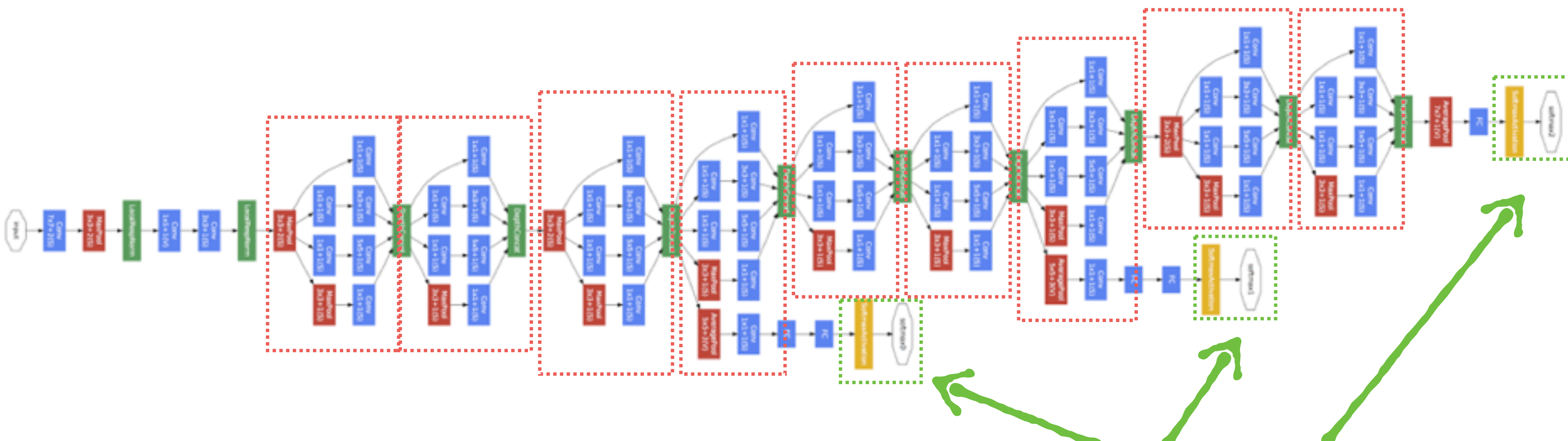
reduce channels with 1x1 kernels

GoogLeNet

Net Structure



GoogLeNet



Inception Modules

Supervision

```
layer {
  name: "loss1/classifier"
  type: "InnerProduct"
  bottom: "loss1/fc"
  top: "loss1/classifier"
  param { lr_mult: 1 decay_mult: 1 }
  param { lr_mult: 2 decay_mult: 0 }
  inner_product_param {
    num_output: 1000
    weight_filler { type: "xavier" std: 0.0009765625 }
    bias_filler { type: "constant" value: 0 }
  }
}
```

```
layer {
  name: "loss1/loss"
  type: "SoftmaxWithLoss"
  bottom: "loss1/classifier"
  bottom: "label"
  top: "loss1/loss1"
  loss_weight: 0.3
}
```

GoogLeNet

- Differences:
  - not training with the relighting data-augmentation;
  - not training with the scale or aspect-ratio data-augmentation;
  - uses "xavier" to initialize the weights instead of "gaussian";

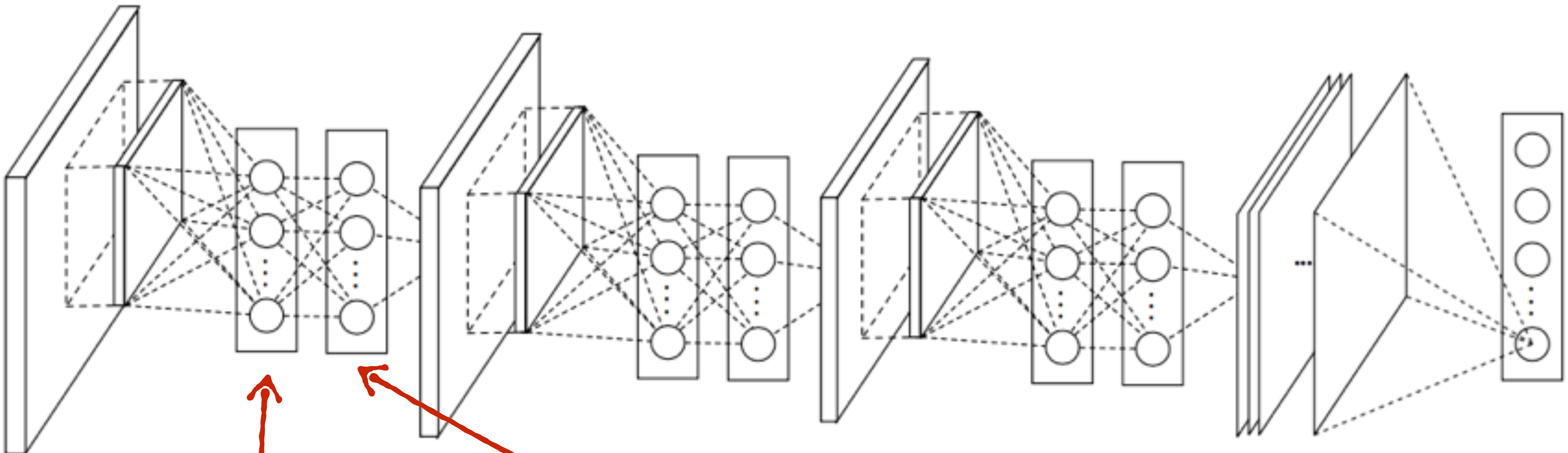
Network in Network

- ImageNet 2014 Object Detection (with provided data only) winner

- Use global pooling rather than fully connected layers for classification

- Lin, Min, Qiang Chen, and Shuicheng Yan. "Network in network." arXiv preprint arXiv:1312.4400 (2013).

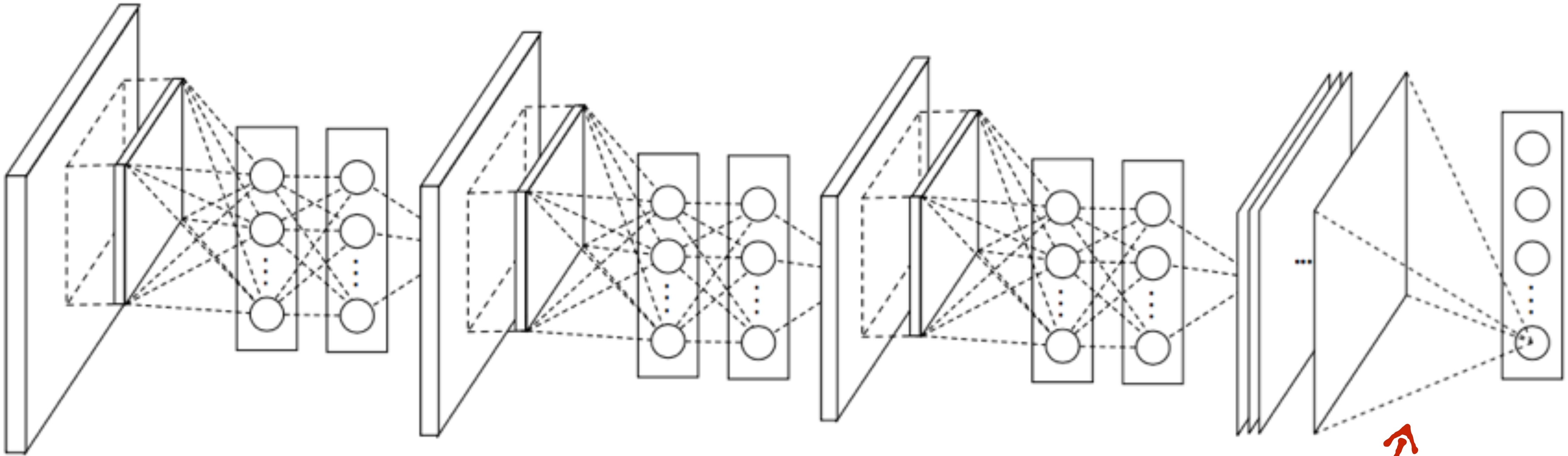
Network in Network



```
layers {
  bottom: "conv1"
  top: "cccp1"
  name: "cccp1"
  type: CONVOLUTION
  convolution_param {
    num_output: 96
    kernel_size: 1
    stride: 1
    weight_filler { type: "gaussian" mean: 0 std: 0.05 }
    bias_filler { type: "constant" value: 0 }
  }
}
layers {
  bottom: "cccp1"
  top: "cccp1"
  name: "relu1"
  type: RELU
}
```

```
layers {
  bottom: "cccp1"
  top: "cccp2"
  name: "cccp2"
  type: CONVOLUTION
  convolution_param {
    num_output: 96
    kernel_size: 1
    stride: 1
    weight_filler { type: "gaussian" mean: 0 std: 0.05 }
    bias_filler { type: "constant" value: 0 }
  }
}
layers {
  bottom: "cccp2"
  top: "cccp2"
  name: "relu2"
  type: RELU
}
```

Network in Network

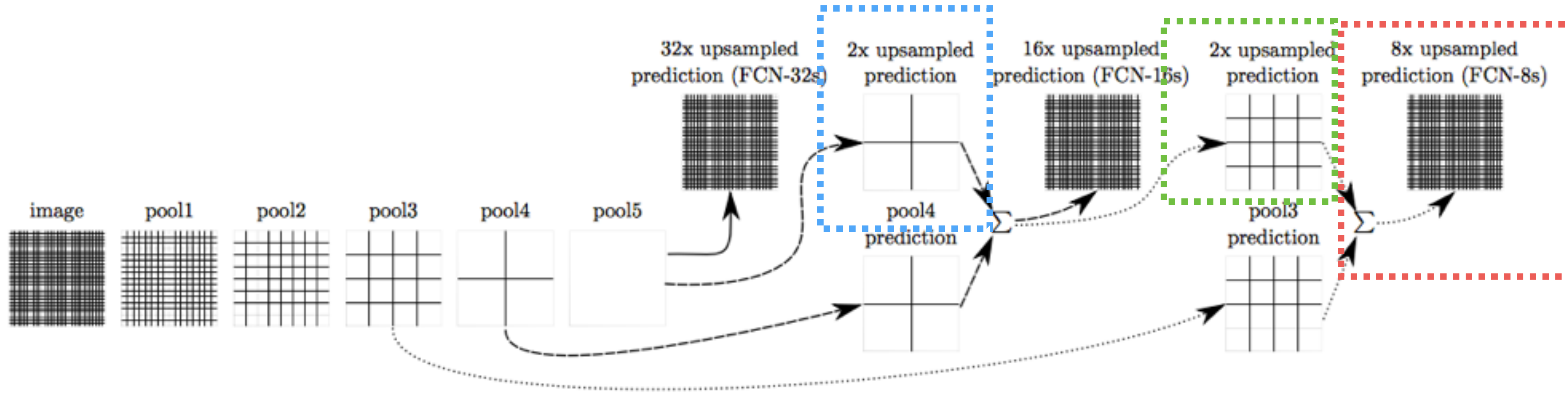


```
layers {
  bottom: "cccp8"
  top: "pool4"
  name: "pool4"
  type: POOLING
  pooling_param {
    pool: AVE
    global_pooling: true
  }
}
```

FCN-Xs

- State-of-the-art results on PASCAL VOC segmentation challenges
- Use trainable deconvolution layer to get dense prediction
- https://github.com/longjon/caffe/tree/future
- Long, Jonathan, Evan Shelhamer, and Trevor Darrell. "Fully convolutional networks for semantic segmentation." arXiv preprint arXiv:1411.4038 (2014).

FCN-Xs




```
layers { type: DECONVOLUTION name: 'score2' bottom: 'score' top: 'score2'
  blobs_lr: 1 blobs_lr: 2 weight_decay: 1 weight_decay: 0
  convolution_param { kernel_size: 4 stride: 2 num_output: 21 } }
```

```
layers { type: DECONVOLUTION name: 'score4' bottom: 'score-fused'
  top: 'score4'
  blobs_lr: 1 weight_decay: 1
  convolution_param { bias_term: false kernel_size: 4 stride: 2 num_output: 21 } }
```

```
layers { type: DECONVOLUTION name: 'upsample'
  bottom: 'score-final' top: 'bigscore'
  blobs_lr: 0
  convolution_param { bias_term: false num_output: 21 kernel_size: 16 stride: 8 } }
```

deconvolution
for upsampling