Practical Machine Learning

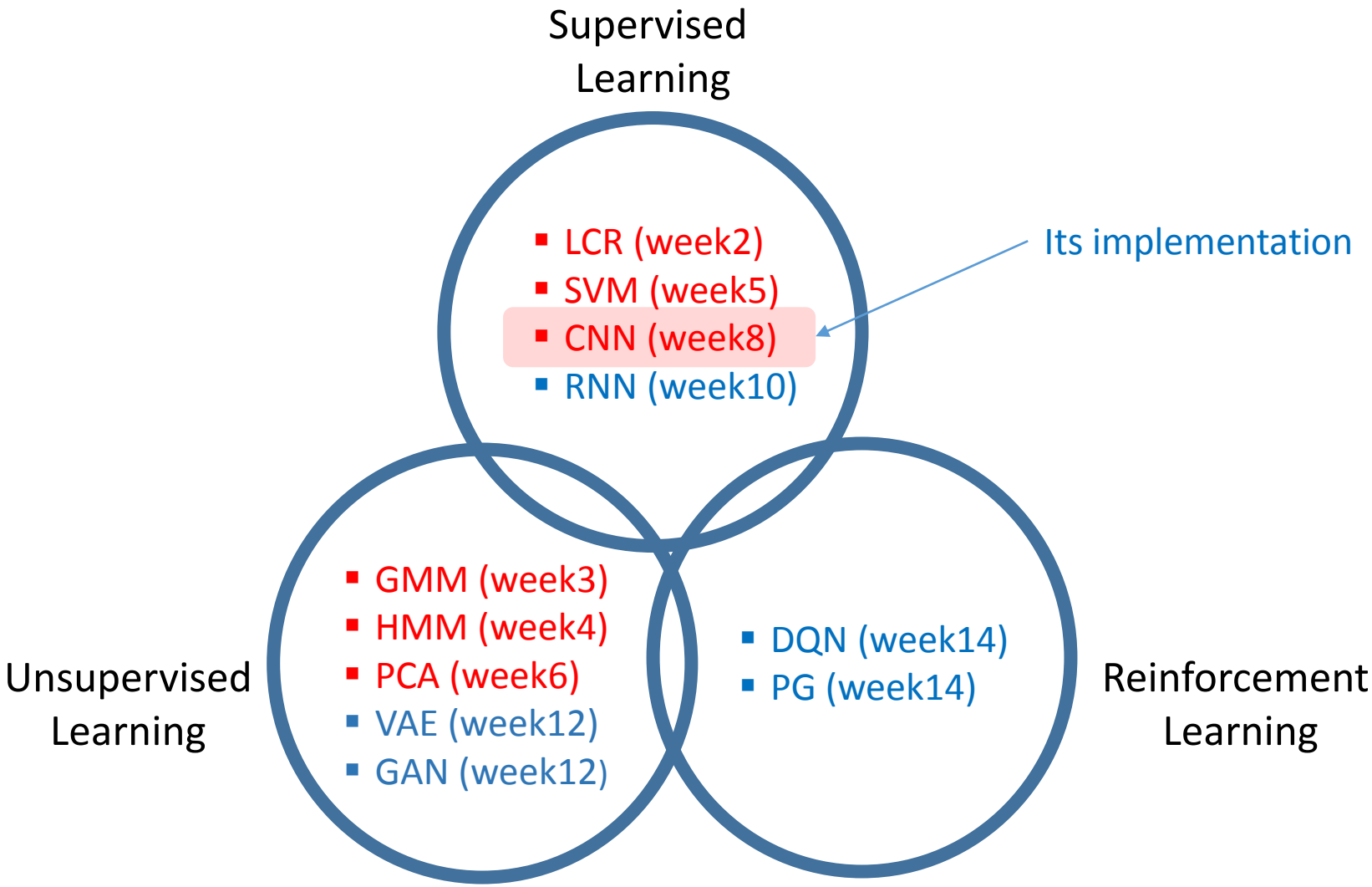
Lecture 9

TensorFlow – CNN implementation

Dr. Suyong Eum

OSAKA UNIVERSITY

Where we are

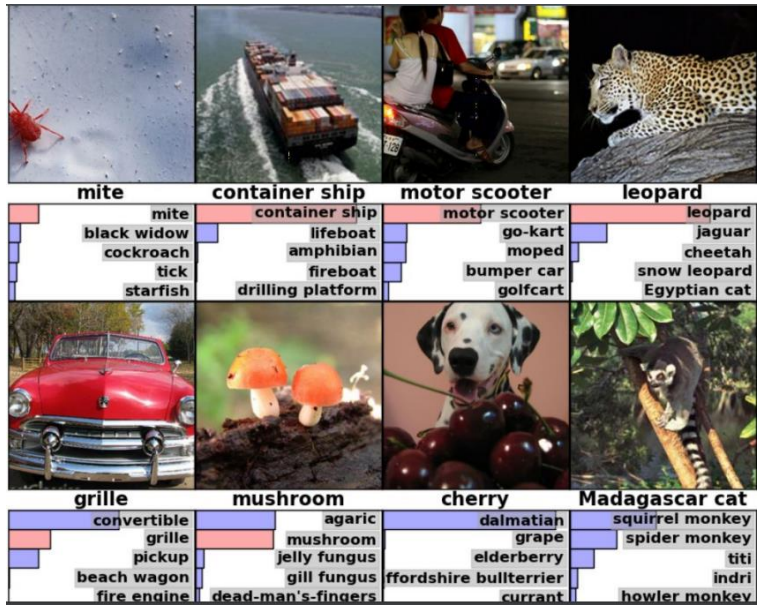

Supervised Learning
LCR (week2)
SVM (week5)
CNN (week8)
RNN (week10)
Its implementation
Unsupervised Learning
GMM (week3)
HMM (week4)
PCA (week6)
VAE (week12)
GAN (week12)
DQN (week14)
PG (week14)
Reinforcement Learning

You are going to learn

- ❑ About TensorFlow
- ❑ Its operational concept
- ❑ Convolutional Neural Network implementation
  - Data loading
  - Model
  - Loss and Accuracy
  - Training
  - Testing

Tensorflow History

- In 2011, Google developed a deep learning infrastructure called “DistBelief” for its internal use.
  - 1st Generation Machine Learning Framework
    - Concept “cat” learned from unlabeled YouTube images,
    - Improvement of speech recognition in the Google app by 25%,
    - Built image search in Google Photos,
    - Trained the inception model that won ImageNet competition in 2014,
    - Automated image captioning as well as deepdream.



Tensorflow History

- In 2011, Google developed a deep learning infrastructure called “DistBelief” for its internal use.
  - 1st Generation Machine Learning Framework
    - Concept “cat” learned from unlabeled YouTube images,
    - Improvement of speech recognition in the Google app by 25%,
    - Built image search in Google Photos,
    - Trained the inception model that won ImageNet competition in 2014,
    - Automated image captioning as well as deepdream.

- Problems with “DistBelief”
  - Uniquely targeted to Neural networks
  - Tightly coupled to Google’s internal infra: difficult to share and to open to public

- In Nov. 2015, google launched “TensorfFlow”
  - General, flexible, portable, easy-to-use, and open-source
  - Now “TensorFlow” replaced “DistBelief” in Google.

Why is Tensorflow?

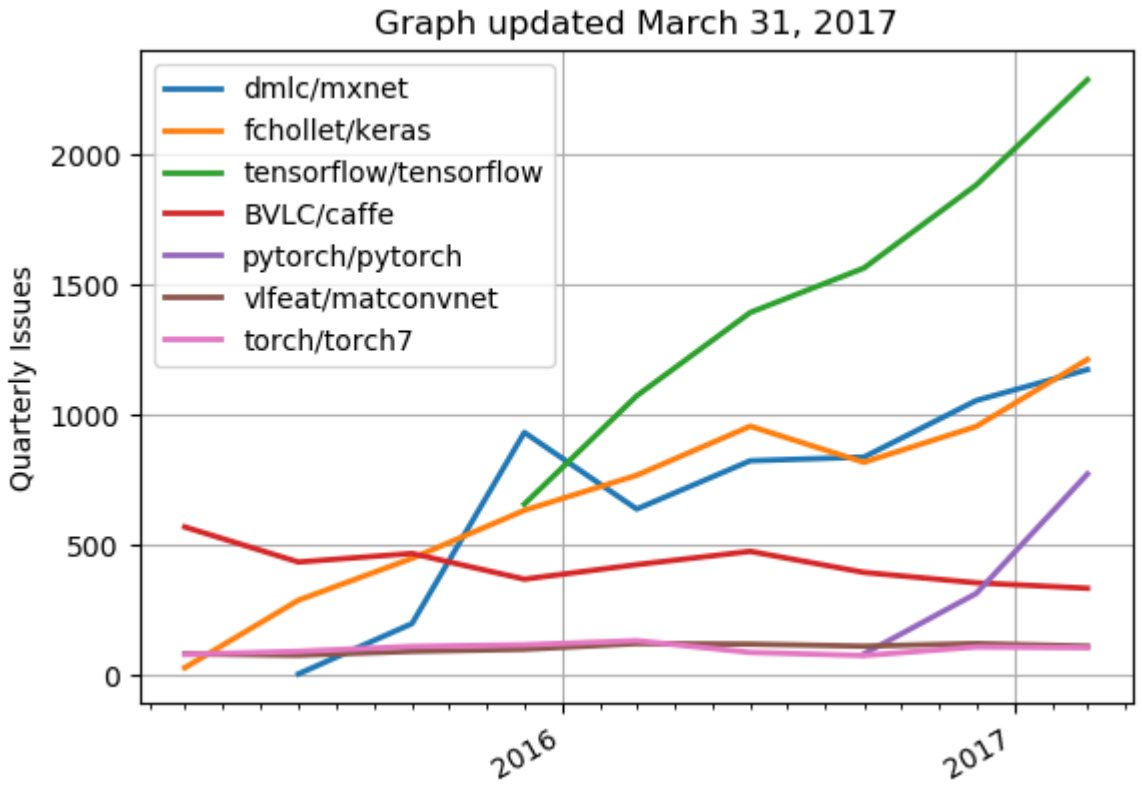
- Practically proven by google for their inner projects.
  - TF was built with production use in mind, whereas others were designed by researchers almost purely for research purposes.

- Easily build models that span multiple GPUs on a single machine, and to train large-scale networks in a distributed fashion.
  - Automatically discovers and uses GPUs and CPUs for computations.
  - By default, it occupies 100% of GPU resource. Of course, you can control.

- Although Tensorflow was inspired by Theano, major development of Theano would be ceased after its 1.0 release (after 2017) due to competing offerings by strong industrial players, e.g., google, facebook, MS, etc.

https://en.wikipedia.org/wiki/Theano_(software)

Popularity of other ML frameworks



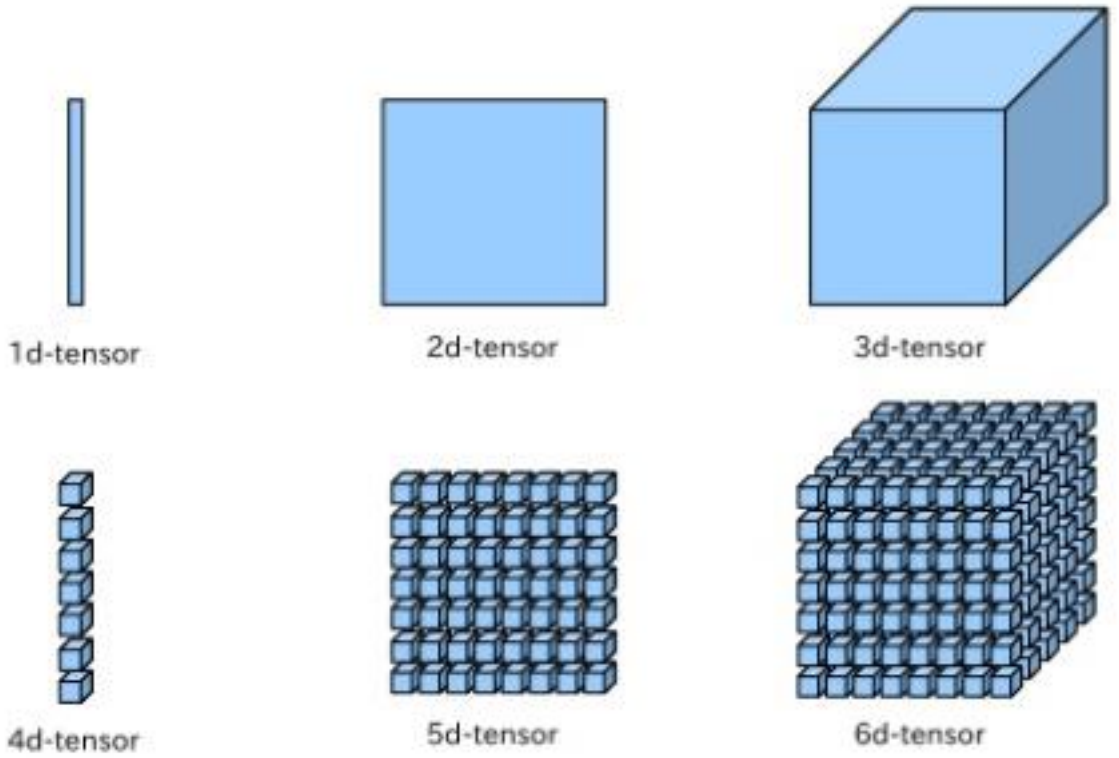
Deep learning libraries: accumulated GitHub metrics as of April 12, 2017

Aggregate popularity (30•contrib + 10•issues + 5•forks)•1e-3		
#1:	209.25	tensorflow/tensorflow
#2:	95.91	BVLC/caffe
#3:	82.36	fchollet/keras
#4:	61.69	dmlc/mxnet
#5:	41.20	Theano/Theano
#6:	35.00	deeplearning4j/deeplearning4j
#7:	32.17	Microsoft/CNTK
#8:	18.73	torch/torch7
#9:	17.29	baidu/paddle
#10:	15.14	pytorch/pytorch
#11:	14.22	pfnet/chainer
#12:	14.05	NVIDIA/DIGITS
#13:	12.62	tflearn/tflearn

https://github.com/mahehu/deeplearning_issues

Tensorflow = Tensor + Flow

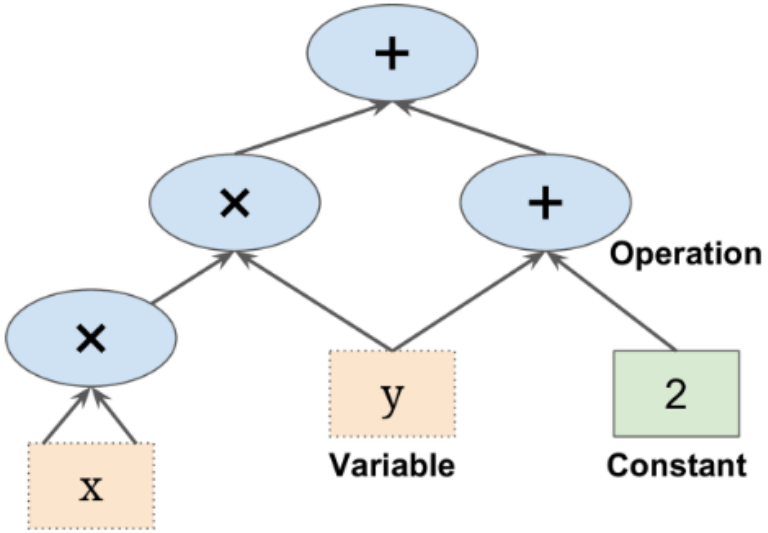
- Tensor: a multidimensional array of numbers, e.g., 4D [batch, height, width, channel]
  - 1d tensor : vector
  - 2d tensor : matrix
  - 3d tensor : cube
  - 4d tensor : a vector of cubes
  - 5d tensor : a matrix of cubes
  - 6d tensor : a cube of cubes

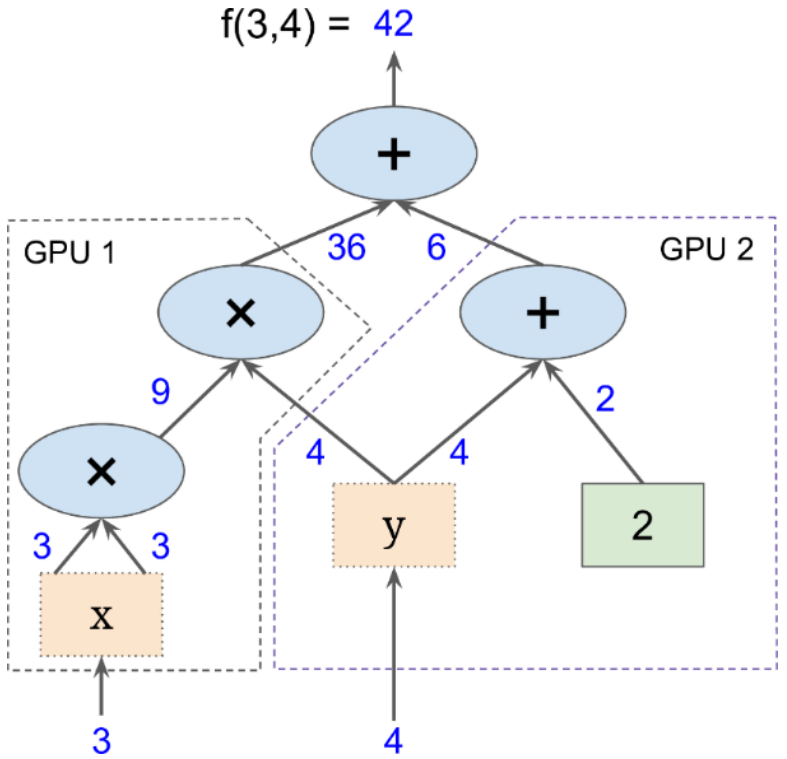


What is Tensorflow?

- Tensorflow is a programming system in which you represent computations as graphs.
  - Nodes in the graph are called *ops* (short for operations).
  - An *op* takes tensors to perform some computation and to produce tensors.
  - A tensorflow graph is a description of computations.

$$f(x, y) = x^2y + y + 2$$

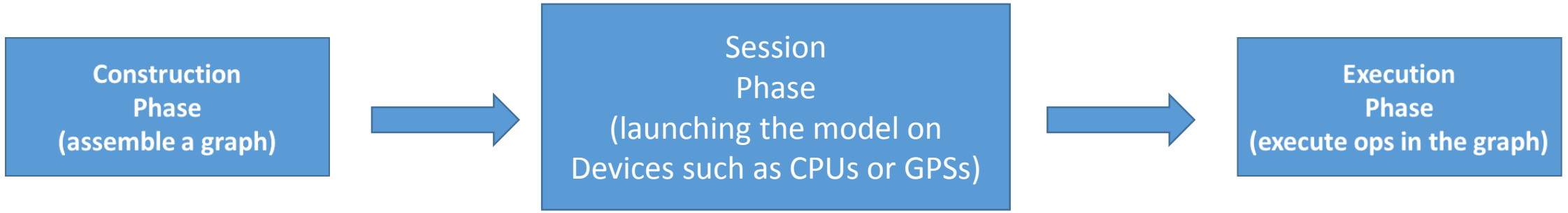




https://www.safaribooksonline.com/library/view/hands-on-machine-learning/9781491962282/ch09.html

Operational concept of Tensorflow

- Tensorflow programs are usually structured into a construction phase and an execution phase.
  - Construction phase: building a model using a graph, e.g., with python
  - Session phase: launching the model on devices such as CPU or GPU
  - Execution phase: executing the model on highly-optimized C++



Python: define the computational graph

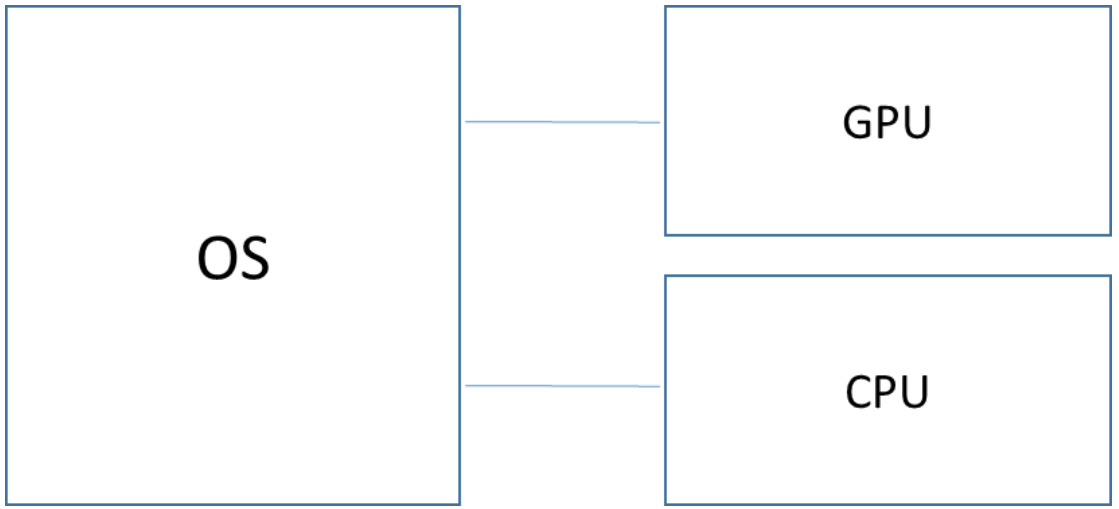
1) Allocate resource on devices, e.g., CPUs or GPUs,
2) Load the graphs on the devices,
3) Get ready to optimally execute the graph

C++: run the graph with the highly-optimized C++

Operational concept of Tensorflow: session

- Creating a session is similar to opening a file: communicating with a device, e.g., GPU
  - sess = tf.Session()
  - sess.close()
  - e.g., with tf.Session() as sess: # don't worry about closing session

- When a session begins, CPU or GPU resource is occupied.



Example of Tensorflow operation

```
import tensorflow as tf
```

```
a = tf.constant(5)
b = tf.constant(6)
c =  a + b
print (c)
```

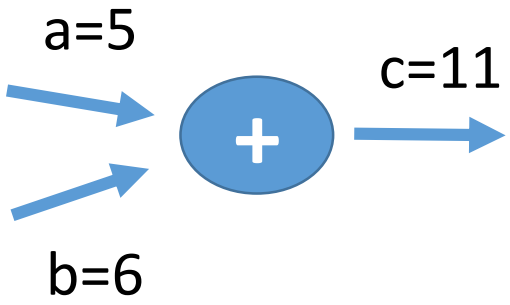
Construction phase

```
Tensor("add:0", shape=(), dtype=int32)
```

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

Execution phase

```
11
11
```



Three data types

Constant	Variable	Placeholder
• Constant value which does not change during runtime	• Weight, bias, etc., which keep being updated during runtime. • Need to be initialized.	• Data which gets fed into the model

```
a = tf.constant(1)
b = tf.constant(2)
c = a + b

sess = tf.Session()
sess.run(c)
```

```
3
```

```
a = tf.Variable(1)
```

```
sess = tf.Session()
sess.run(tf.global_variables_initializer())
```

```
sess.run(a)
```

```
1
```

```
sess.run(a.assign(3))
```

```
3
```

```
a = tf.placeholder(tf.float32)
b = tf.placeholder(tf.float32)
c = a + b

feed_dict = {a: 1., b: 2.}

sess = tf.Session()
sess.run(c, feed_dict)
```

```
3.0
```

Three data types - Variable

```
a = tf.Variable(tf.truncated_normal([3,2]), name = 'w1')
```

It is required when variables are saved and restored later.

```
sess = tf.Session()
sess.run(tf.global_variables_initializer())
```

This initialization step is mandatory .

```
sess.run(a)
```

```
array([[-1.0226047 ,  0.55575389],
       [-1.44942784,  0.52479738],
       [ 0.07580709,  0.00752988]], dtype=float32)
```

```
truncated_normal(
    shape,
    mean=0.0,
    stddev=1.0,
    dtype=tf.float32,
    seed=None,
    name=None
)
```

When the model does not learn from data, this is the one you need to check first.

Three data types - Placeholder

```
input = tf.placeholder(dtype=tf.float32)
model = 10*input
```

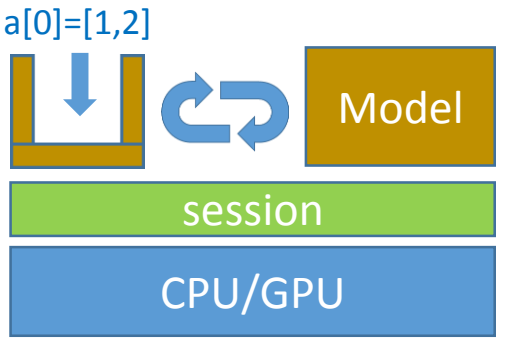
```
# Data define
a = [[1,2],[3,4]]

with tf.Session() as sess:

    feed_dict = {input: a[0]}
    print(sess.run(model, feed_dict))

    feed_dict = {input: a[1]}
    print(sess.run(model, feed_dict))
```

```
[10. 20.]
[30. 40.]
```

```
placeholder(
    dtype,
    shape=None,
    name=None
)
```

a=[[1,2],[3,4]]

a[0]=[1,2],
a[1]=[3,4]



- whole data set

  a=[[1,2],[3,4]]

- batch data set

  1) a[0]=[1,2]
  2) a[1]=[3,4]

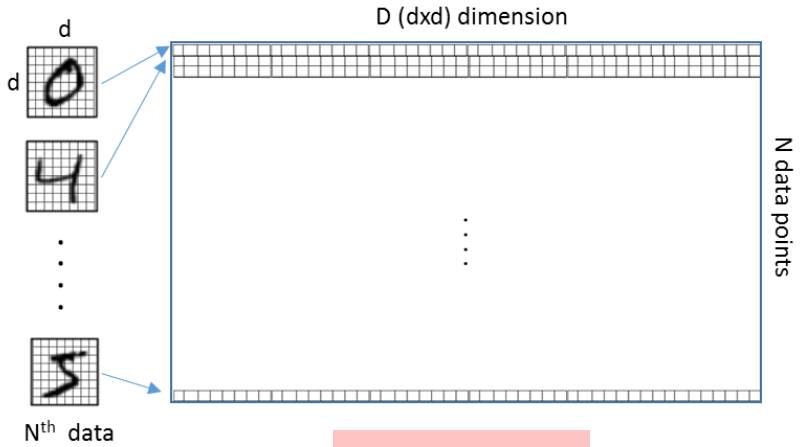
Convolutional Neural Network implementation

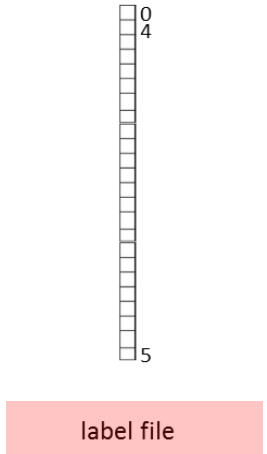
Implementation of CNN

1) Data loading
2) Model definition
3) Evaluation: Loss and Accuracy
4) Training
5) Testing: saving and reloading variables

1) Data loading: MNIST

- MNIST data set
  - http://yann.lecun.com/exdb/mnist/
  - Training data
    - One single file (45M) which includes 60,000 hand digit images for training,
    - One single file (59K) which includes corresponding labels.
  - Testing data
    - One single file (7.5M) which includes 10,000 hand digit images for testing,
    - One single file (9.8K) which includes corresponding labels.





1) Data loading: Tiny ImageNet

- Tiny ImageNet data set
  - https://tiny-imagenet.herokuapp.com/
  - 100,000 jpeg image files for training:
    - 200 classes (200 folders)
    - 500 images per class
    - Not only label but also coordination for object detection in each image file
      - words.txt
  - 10,000 jpeg image files for validating with label
  - 10,000 jpeg image files for testing without label

data folder

file1 - frog
file2 – fish
.
.
.
file - snake

label file

1) Data loading – Tiny ImageNet

- Tiny ImageNet data set

```
tiny-imagenet-200
├── label.txt
├── test
├── train
├── val
├── wnids.txt
└── words.txt
```

This is the labels for images under val directory (there are 10000 labels)

```
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
```

```
adagio@catt:tiny-imagenet-200$ ls train/
n01443537  n02113799  n02509815  n02988304  n03670208  n04146614  n04596742
n01629819  n02123045  n02666196  n02999410  n03706229  n04149813  n04597913
n01641577  n02123394  n02669723  n03014705  n03733131  n04179913  n06596364
n01644900  n02124075  n02699494  n03026506  n03763968  n04251144  n07579787
n01698640  n02125311  n02730930  n03042490  n03770439  n04254777  n07583066
n01742172  n02129165  n02769748  n03085013  n03796401  n04259630  n07614500
n01768244  n02132136  n02788148  n03089624  n03804744  n04265275  n07615774
n01770393  n02165456  n02791270  n03100240  n03814639  n04275548  n07695742
n01774384  n02190166  n02793495  n03126707  n03837869  n04285008  n07711569
n01774750  n02206856  n02795169  n03160309  n03838899  n04311004  n07715103
n01784675  n02226429  n02802426  n03179701  n03854065  n04328186  n07720875
n01855672  n02231487  n02808440  n03201208  n03891332  n04356056  n07734744
n01882714  n02233338  n02814533  n03250847  n03902125  n04366367  n07747607
n01910747  n02236044  n02814860  n03255030  n03930313  n04371430  n07749582
n01917289  n02268443  n02815834  n03355925  n03937543  n04376876  n07753592
n01944390  n02279972  n02823428  n03388043  n03970156  n04398044  n07768694
n01945685  n02281406  n02837789  n03393912  n03976657  n04399382  n07871810
n01950731  n02321529  n02841315  n03400231  n03977966  n04417672  n07873807
n01983481  n02364673  n02843684  n03404251  n03980874  n04456115  n07875152
n01984695  n02395406  n02883205  n03424325  n03983396  n04465501  n07920052
n02002724  n02403003  n02892201  n03444034  n03992509  n04486054  n09193705
n02056570  n02410509  n02906734  n03447447  n04008634  n04487081  n09246464
n02058221  n02415577  n02909870  n03544143  n04023962  n04501370  n09256479
n02074367  n02423022  n02917067  n03584254  n04067472  n04507155  n09332890
n02085620  n02437312  n02927161  n03599486  n04070727  n04532106  n09428293
n02094433  n02480495  n02948072  n03617480  n04074963  n04532670  n12267677
n02099601  n02481823  n02950826  n03637318  n04099969  n04540053
n02099712  n02486410  n02963159  n03649909  n04118538  n04560804
n02106662  n02504458  n02977058  n03662601  n04133789  n04562935
```

List of labels

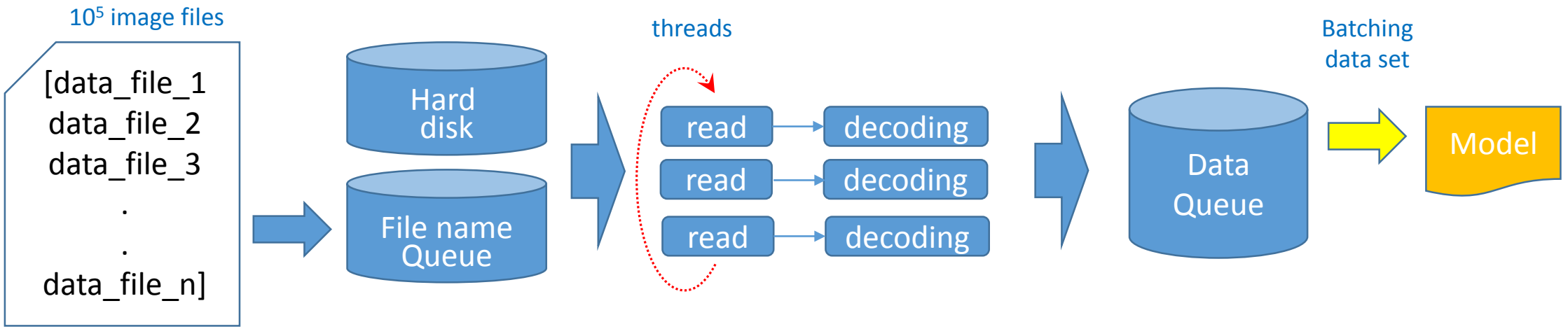
```
n02124075
n04067472
n04540053
n04099969
n07749582
```

Label description

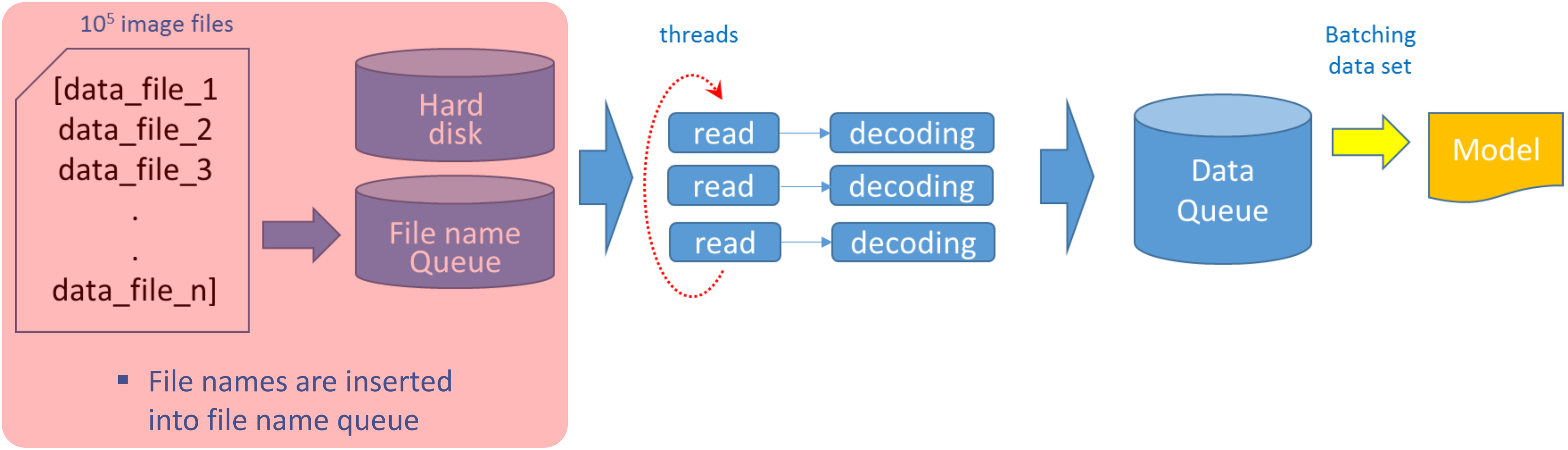
```
n00001740       entity
n00001930       physical entity
n00002137       abstraction, abstract entity
n00002452       thing
n00002684       object, physical object
n00003553       whole, unit
n00003993       congener
```

1) Data loading – queue runner

❑ When a large number of files are loaded, they are asynchronously loaded in parallel.



1) Data loading – queue runner

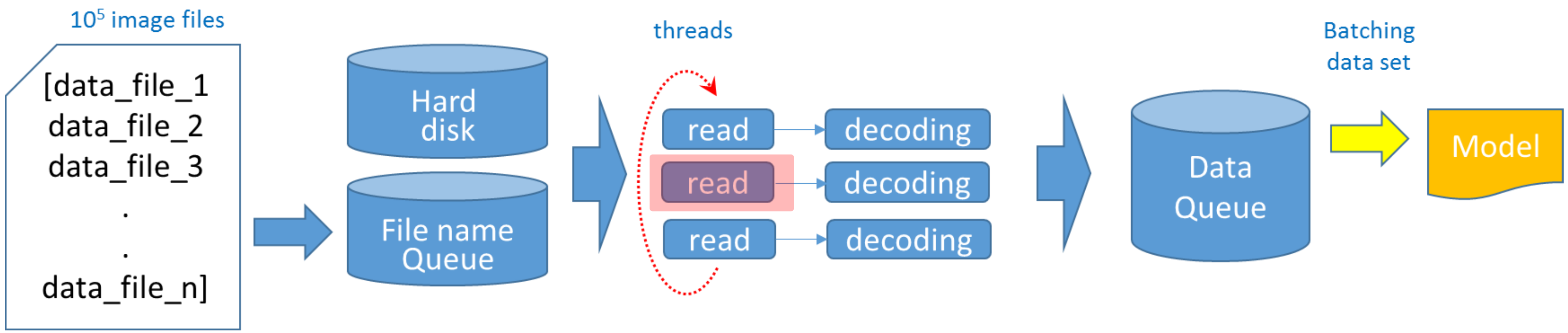



```
print(np.shape(train_images))
print(train_images[0:2])

print(np.shape(train_labels))
print(train_labels[0:2])
```

```
(100000,)
['../tiny-imagenet-200/train/n01443537/images/n01443537_0.JPEG',
'../tiny-imagenet-200/train/n01443537/images/n01443537_1.JPEG']
(100000,)
['1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0', '1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0']
```

Coding
❑ queue_name = tf.train.**slice_input_producer**([train_images, train_labels], shuffle=True) : creating queue which keeps the list of files

1) Data loading – queue runner



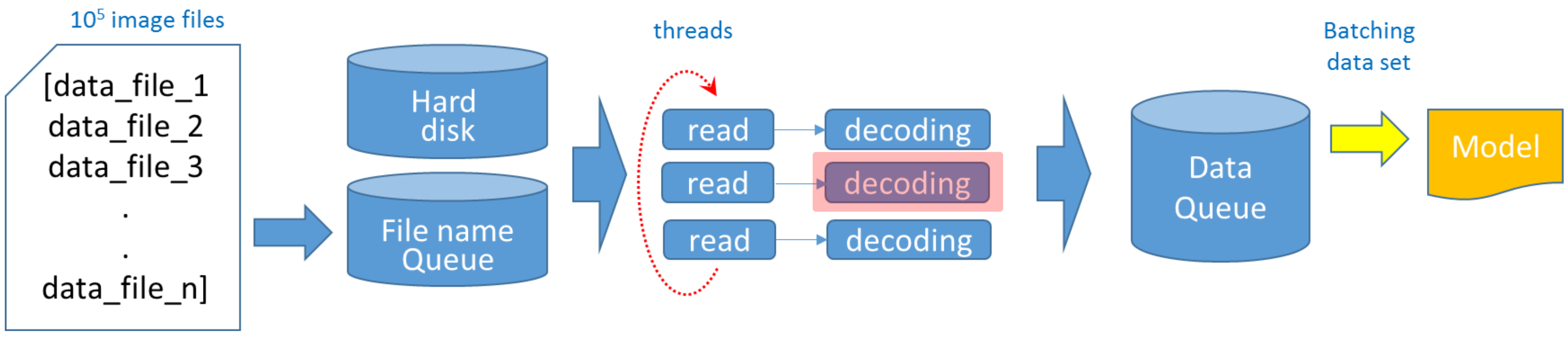
- File names are inserted into file name queue

```
print(np.shape(train_images))
print(train_images[0:2])

print(np.shape(train_labels))
print(train_labels[0:2])
```

```
(100000,)
['../tiny-imagenet-200/train/n01443537/images/n01443537_0.JPEG',
'../tiny-imagenet-200/train/n01443537/images/n01443537_1.JPEG']
(100000,)
['1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0', '1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0']
```

Coding
❑ queue_name = tf.train.**slice_input_producer**([train_images, train_labels], shuffle=True) : creating queue which keeps the list of files
❑ file_handler = tf.**read_file**(queue_name[0]) .... Why [0]? : reading one image file and creating its handler

1) Data loading – queue runner



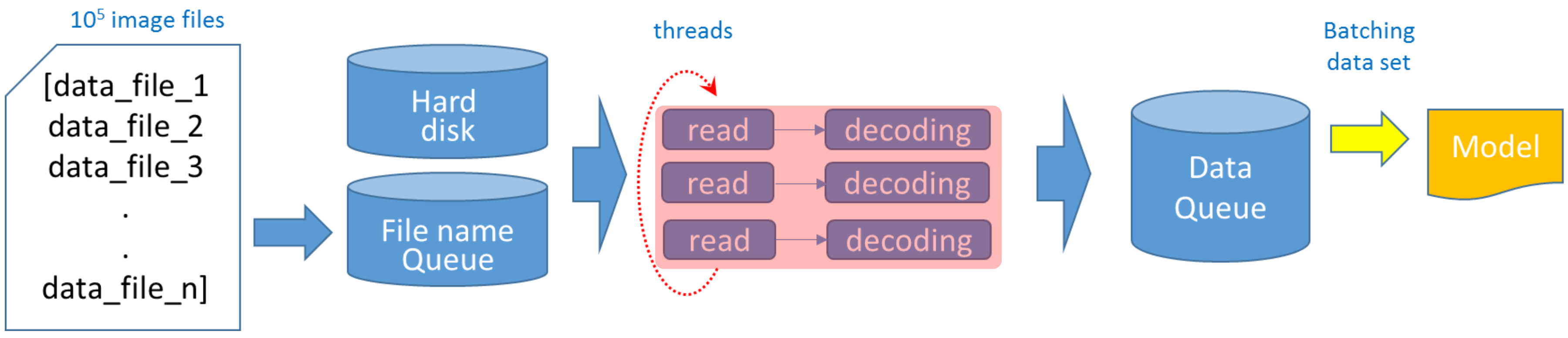
- File names are inserted into file name queue

```
print(np.shape(train_images))
print(train_images[0:2])

print(np.shape(train_labels))
print(train_labels[0:2])
```

```
(100000,)
['../tiny-imagenet-200/train/n01443537/images/n01443537_0.JPEG',
'../tiny-imagenet-200/train/n01443537/images/n01443537_1.JPEG']
(100000,)
['1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0', '1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0']
```

Coding
❑ queue_name = tf.train.**slice_input_producer**([train_images, train_labels], shuffle=True) : creating queue which keeps the list of files
❑ file_handler = tf.**read_file**(queue_name[0]) …. Why [0]? : reading one image file and creating its handler
❑ image = tf.image.**decode_jpeg**(file_handler) : decoding the data value (e.g., png, jpeg, gif, csv)

1) Data loading – queue runner



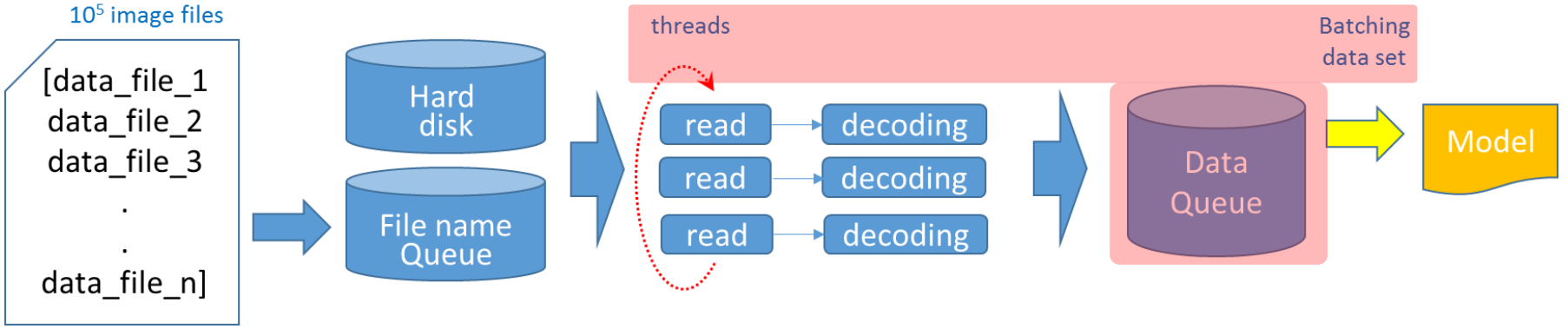
- File names are inserted into file name queue

```
print(np.shape(train_images))
print(train_images[0:2])

print(np.shape(train_labels))
print(train_labels[0:2])
```

```
(100000,)
['../tiny-imagenet-200/train/n01443537/images/n01443537_0.JPEG',
'../tiny-imagenet-200/train/n01443537/images/n01443537_1.JPEG']
(100000,)
['1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0', '1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0']
```

Coding
❑ queue_name = tf.train.**slice_input_producer**([train_images, train_labels], shuffle=True) : creating queue which keeps the list of files
❑ file_handler = tf.**read_file**(queue_name[0]) .... Why [0]? : reading one image file and creating its handler
❑ image = tf.image.**decode_jpeg**(file_handler) : decoding the data value (e.g., png, jpeg, gif, csv)
❑ x, y = tf.train.**suffle_batch**([image, label], batch_size = 10, ...) : batching image files

1) Data loading – queue runner



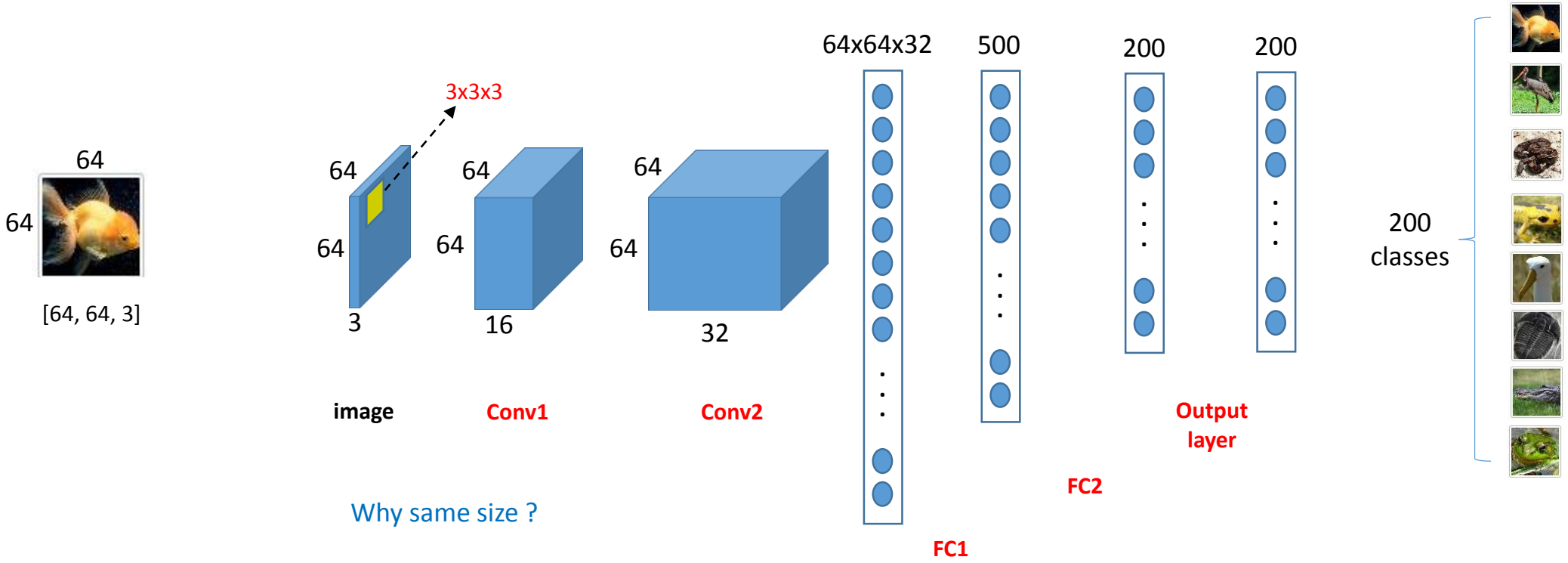

```
sess = tf.Session()
coord = tf.train.Coordinator()
thread = tf.train.start_queue_runners(sess, coord)

for i range (5)
        x_batch, y_batch = sess.run([x,y])
        feed_dict = {x_: x_batch, y_: y_batch}

coord.request_stop()
coord.join(thread)
```

```
sess = tf.Session()
coord = tf.train.Coordinator()
thread = tf.train.start_queue_runners(sess, coord)

for i in range(5):
    x_batch, y_batch = sess.run([x,y_])
    feed_dict = {x:x_batch, y_:y_batch}

    print (x)

coord.request_stop()
coord.join(thread)
```

```
Tensor("shuffle_batch/shuffle_batch:0", shape=(32, 64, 64, 3), dtype=float32)
Tensor("shuffle_batch/shuffle_batch:0", shape=(32, 64, 64, 3), dtype=float32)
Tensor("shuffle_batch/shuffle_batch:0", shape=(32, 64, 64, 3), dtype=float32)
Tensor("shuffle_batch/shuffle_batch:0", shape=(32, 64, 64, 3), dtype=float32)
Tensor("shuffle_batch/shuffle_batch:0", shape=(32, 64, 64, 3), dtype=float32)
```

32 x Image (64x64x3)

2) Model: Convolutional Neural Networks



2) Model: Convolutional Neural Networks - Conv

Conv1

```
# Convolutional Layer 1

W_h1 = tf.Variable(tf.truncated_normal([3,3,3,16], stddev=1./math.sqrt(3*3*3)))
b_h1 = tf.Variable(tf.zeros([16]))

x_image = tf.reshape(x, [-1, image_height, image_width, 3])

conv1   = tf.nn.conv2d(x_image, W_h1, strides=[1,1,1,1], padding='SAME')
hidden1 = tf.nn.relu(conv1 + b_h1)

pool1 = tf.nn.max_pool(hidden1, ksize=[1,2,2,1], strides=[1,1,1,1], padding='SAME')
hidden1 = pool1
```

Conv2

```
# Concolutional Layer 2

W_h2 = tf.Variable(tf.truncated_normal([3,3,16,32], stddev=1./math.sqrt(3*3*16)))
b_h2 = tf.Variable(tf.zeros([32]))

conv2   = tf.nn.conv2d(hidden1, W_h2, strides=[1,1,1,1], padding='SAME')
hidden2 = tf.nn.relu(conv2 + b_h2)

pool2 = tf.nn.max_pool(hidden2, ksize=[1,2,2,1], strides=[1,1,1,1], padding='SAME')
hidden2 = pool2
```

FC1

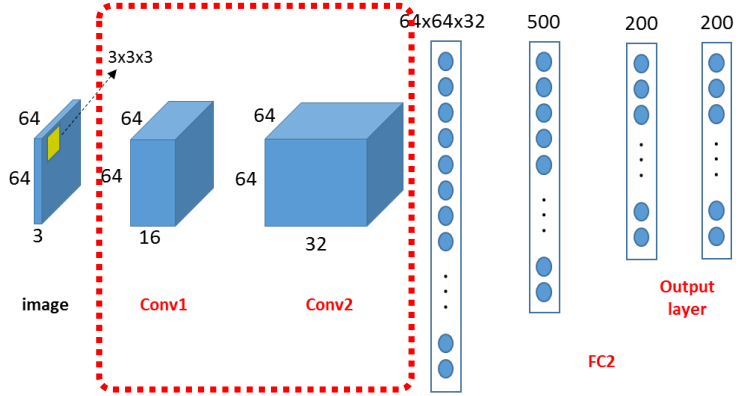
```
# Fully connected Layer 1

h_flat1 = tf.reshape(hidden2, [-1, 64*64*32])
fc_w1   = tf.Variable(tf.truncated_normal([64*64*32, 500], stddev=1./math.sqrt(64*64*32)))
fc_b1   = tf.Variable(tf.zeros([500]))

h_fc1 = tf.nn.relu(tf.matmul(h_flat1, fc_w1) + fc_b1)
```

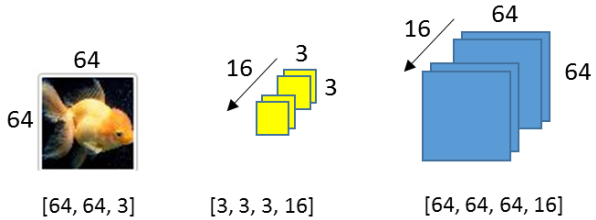
FC2

```
# Fully connected Layer 2

fc_w2   = tf.Variable(tf.truncated_normal([500,200], stddev=1./math.sqrt(500*200)))
fc_b2   = tf.Variable(tf.zeros([200]))

h_fc2 = tf.nn.relu(tf.matmul(h_fc1, fc_w2) + fc_b2)
```

Output layer

```
# Output layer

W_o = tf.Variable(tf.truncated_normal([200, 200], stddev=1./math.sqrt(200*200)))
b_o = tf.Variable(tf.truncated_normal([200]))

pred = tf.matmul( h_fc2 , W_o) + b_o
```





- ❑ truncated_normal ()
  - shape =[height, width, channel, batch_size])
- ❑ conv2d()
  - strides = [batch_size, height, width, channel]
- ❑ max_pool()
  - ksize = [batch_size, height, width, channel]
  - strides = [batch_size, height, width, channel]
  - padding = ‘SAME’ or ‘VALID’

2) Model: Convolutional Neural Networks - FC

Conv1

```
# Convolutional Layer 1

W_h1 = tf.Variable(tf.truncated_normal([3,3,3,16], stddev=1./math.sqrt(3*3*3)))
b_h1 = tf.Variable(tf.zeros([16]))

x_image = tf.reshape(x, [-1, image_height, image_width, 3])

conv1   = tf.nn.conv2d(x_image, W_h1, strides=[1,1,1,1], padding='SAME')
hidden1 = tf.nn.relu(conv1 + b_h1)

pool1 = tf.nn.max_pool(hidden1, ksize=[1,2,2,1], strides=[1,1,1,1], padding='SAME')
hidden1 = pool1
```

Conv2

```
# Concolutional Layer 2

W_h2 = tf.Variable(tf.truncated_normal([3,3,16,32], stddev=1./math.sqrt(3*3*16)))
b_h2 = tf.Variable(tf.zeros([32]))

conv2   = tf.nn.conv2d(hidden1, W_h2, strides=[1,1,1,1], padding='SAME')
hidden2 = tf.nn.relu(conv2 + b_h2)

pool2 = tf.nn.max_pool(hidden2, ksize=[1,2,2,1], strides=[1,1,1,1], padding='SAME')
hidden2 = pool2
```

FC1

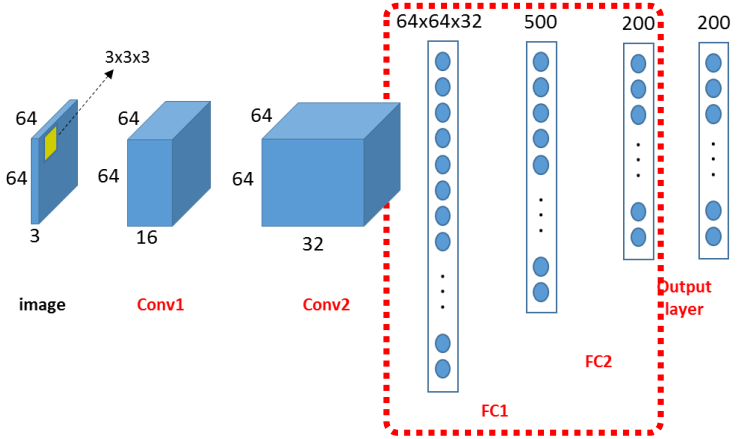
```
# Fully connected Layer 1

h_flat1 = tf.reshape(hidden2, [-1, 64*64*32])
fc_w1   = tf.Variable(tf.truncated_normal([64*64*32, 500], stddev=1./math.sqrt(64*64*32)))
fc_b1   = tf.Variable(tf.zeros([500]))

h_fc1 = tf.nn.relu(tf.matmul(h_flat1, fc_w1) + fc_b1)
```

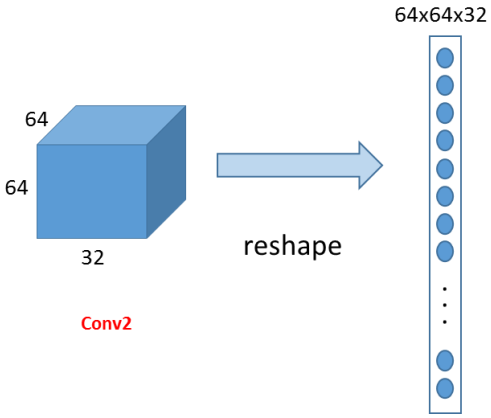
FC2

```
# Fully connected Layer 2

fc_w2   = tf.Variable(tf.truncated_normal([500,200], stddev=1./math.sqrt(500*200)))
fc_b2   = tf.Variable(tf.zeros([200]))

h_fc2 = tf.nn.relu(tf.matmul(h_fc1, fc_w2) + fc_b2)
```

Output layer

```
# Output layer

W_o = tf.Variable(tf.truncated_normal([200, 200], stddev=1./math.sqrt(200*200)))
b_o = tf.Variable(tf.truncated_normal([200]))

pred = tf.matmul( h_fc2 , W_o) + b_o
```





- tf.reshape(hidden2, [-1, 64*64*32])

2) Model: Convolutional Neural Networks – Output layer

Conv1

```
# Convolutional Layer 1

W_h1 = tf.Variable(tf.truncated_normal([3,3,3,16], stddev=1./math.sqrt(3*3*3)))
b_h1 = tf.Variable(tf.zeros([16]))

x_image = tf.reshape(x, [-1, image_height, image_width, 3])

conv1   = tf.nn.conv2d(x_image, W_h1, strides=[1,1,1,1], padding='SAME')
hidden1 = tf.nn.relu(conv1 + b_h1)

pool1 = tf.nn.max_pool(hidden1, ksize=[1,2,2,1], strides=[1,1,1,1], padding='SAME')
hidden1 = pool1
```

Conv2

```
# Concolutional Layer 2

W_h2 = tf.Variable(tf.truncated_normal([3,3,16,32], stddev=1./math.sqrt(3*3*16)))
b_h2 = tf.Variable(tf.zeros([32]))

conv2   = tf.nn.conv2d(hidden1, W_h2, strides=[1,1,1,1], padding='SAME')
hidden2 = tf.nn.relu(conv2 + b_h2)

pool2 = tf.nn.max_pool(hidden2, ksize=[1,2,2,1], strides=[1,1,1,1], padding='SAME')
hidden2 = pool2
```

FC1

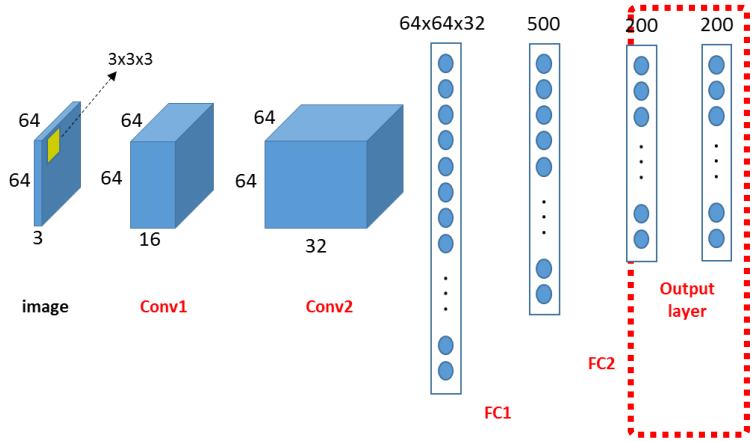
```
# Fully connected Layer 1

h_flat1 = tf.reshape(hidden2, [-1, 64*64*32])
fc_w1   = tf.Variable(tf.truncated_normal([64*64*32, 500], stddev=1./math.sqrt(64*64*32)))
fc_b1   = tf.Variable(tf.zeros([500]))

h_fc1 = tf.nn.relu(tf.matmul(h_flat1, fc_w1) + fc_b1)
```

FC2

```
# Fully connected Layer 2

fc_w2   = tf.Variable(tf.truncated_normal([500,200], stddev=1./math.sqrt(500*200)))
fc_b2   = tf.Variable(tf.zeros([200]))

h_fc2 = tf.nn.relu(tf.matmul(h_fc1, fc_w2) + fc_b2)
```

Output layer

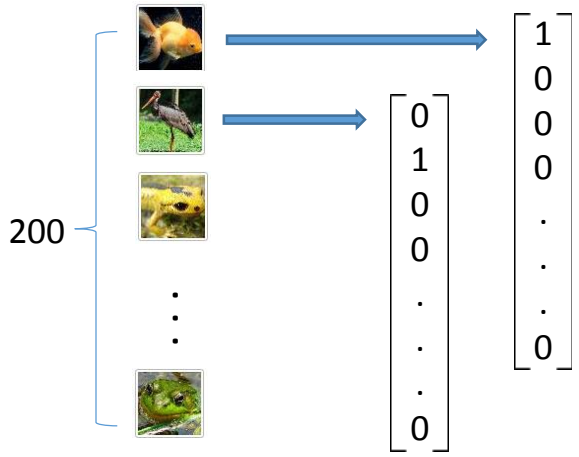
```
# Output layer

W_o = tf.Variable(tf.truncated_normal([200, 200], stddev=1./math.sqrt(200*200)))
b_o = tf.Variable(tf.truncated_normal([200]))

pred = tf.matmul( h_fc2 , W_o) + b_o
```



- We are going to use “softmax_cross_entropy_with_logits()”, which includes “softmax” activation function inside. Thus, the outcome is used without going through any activation function.

3) Evaluation: loss calculation

```
# Loss and Accuracy

loss =  tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=pred, labels=y_))
train = tf.train.AdamOptimizer().minimize(loss)

correct_prediction = tf.equal(tf.argmax(pred, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```



- tf.nn.softmax_cross_entropy_with_logits(prediction, label)
  - prediction: predicted label which is the output from the previous layer
    - e.g., [0.1, 0.4, 0.5]
  - label: true label, one-hot encoded
    - e.g., [0, 0, 1]

Output from softmax function

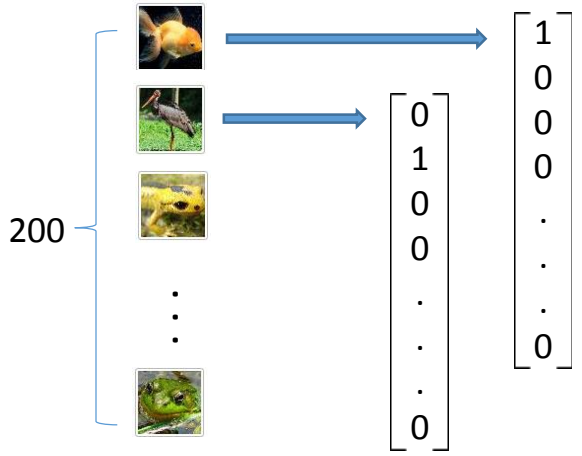
Output from softmax function

batch1	Prediction	Label	Cross Entropy (error)	batch2	Prediction	Label	Cross Entropy(error)
Data1-1	0.1, 0.2, 0.7	0, 0, 1	-ln(0.1)*0-ln(0.2)*0-ln(0.7)*1 = 0.357	Data2-1	0.3, 0.3, 0.4	0, 0, 1	-ln(0.3)*0-ln(0.3)*0-ln(0.4)*1 = 0.916
Data1-2	0.1, 0.6, 0.3	0, 1, 0	-ln(0.1)*0-ln(0.6)*1-ln(0.3)*0 = 0.511	Data2-2	0.3, 0.4, 0.3	0, 1, 0	-ln(0.3)*0-ln(0.4)*1-ln(0.3)*0 = 0.916
Data1-3	0.3, 0.3, 0.4	1, 0, 0	-ln(0.3)*1-ln(0.3)*0-ln(0.4)*0 = 1.204	Data2-3	0.1, 0.1, 0.8	1, 0, 0	-ln(0.1)*1-ln(0.1)*0-ln(0.8)*0 = 2.303
		Mean	0.691			Mean	1.287

3) Evaluation: loss calculation

```
# Loss and Accuracy
loss =  tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=pred, labels=y_))
train = tf.train.AdamOptimizer().minimize(loss)

correct_prediction = tf.equal(tf.argmax(pred, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

Mean of results from batch images

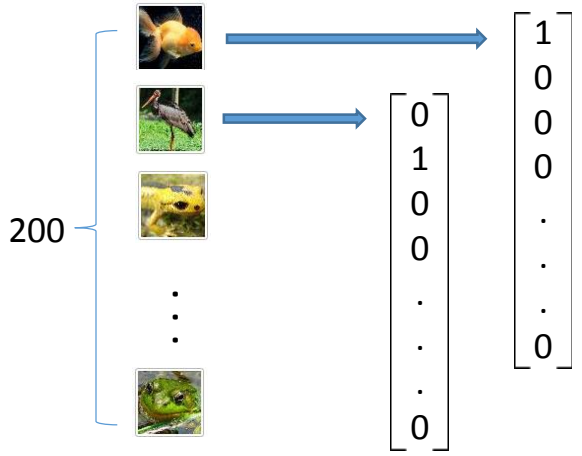


- tf.nn.softmax_cross_entropy_with_logits(prediction, label)
  - prediction: predicted label which is the output from the previous layer
    - e.g., [0.1, 0.4, 0.5]
  - label: true label, one-hot encoded
    - e.g., [0, 0, 1]

batch1	Prediction	Label	Cross Entropy (error)	batch2	Prediction	Label	Cross Entropy(error)
Data1-1	0.1, 0.2, 0.7	0, 0, 1	-ln(0.1)*0-ln(0.2)*0-ln(0.7)*1 = 0.357	Data2-1	0.3, 0.3, 0.4	0, 0, 1	-ln(0.3)*0-ln(0.3)*0-ln(0.4)*1 = 0.916
Data1-2	0.1, 0.6, 0.3	0, 1, 0	-ln(0.1)*0-ln(0.6)*1-ln(0.3)*0 = 0.511	Data2-2	0.3, 0.4, 0.3	0, 1, 0	-ln(0.3)*0-ln(0.4)*1-ln(0.3)*0 = 0.916
Data1-3	0.3, 0.3, 0.4	1, 0, 0	-ln(0.3)*1-ln(0.3)*0-ln(0.4)*0 = 1.204	Data2-3	0.1, 0.1, 0.8	1, 0, 0	-ln(0.1)*1-ln(0.1)*0-ln(0.8)*0 = 2.303
		Mean	0.691			Mean	1.287

3) Evaluation: loss calculation

```
# Loss and Accuracy

loss =  tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=pred, labels=y_))
train = tf.train.AdamOptimizer().minimize(loss)

correct_prediction = tf.equal(tf.argmax(pred, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```



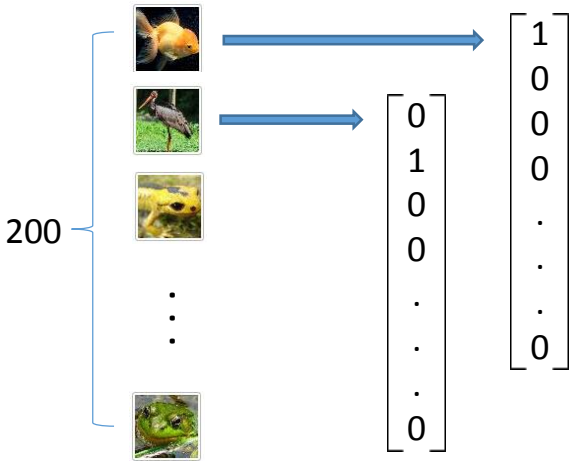
- tf.nn.softmax_cross_entropy_with_logits(prediction, label)
  - prediction: predicted label which is the output from the previous layer
    - e.g., [0.1, 0.4, 0.5]
  - label: true label, one-hot encoded
    - e.g., [0, 0, 1]

- Do not implement the functions separately: (numerical unstability)
  - tf.nn.softmax(prediction)
  - tf.reduce_mean(-tf.reduce_sum(label*tf.log(prediction), reduction_indices=[1]))

batch1	Prediction	Label	Cross Entropy (error)	batch2	Prediction	Label	Cross Entropy(error)
Data1-1	0.1, 0.2, 0.7	0, 0, 1	-ln(0.1)*0-ln(0.2)*0-ln(0.7)*1 = 0.357	Data2-1	0.3, 0.3, 0.4	0, 0, 1	-ln(0.3)*0-ln(0.3)*0-ln(0.4)*1 = 0.916
Data1-2	0.1, 0.6, 0.3	0, 1, 0	-ln(0.1)*0-ln(0.6)*1-ln(0.3)*0 = 0.511	Data2-2	0.3, 0.4, 0.3	0, 1, 0	-ln(0.3)*0-ln(0.4)*1-ln(0.3)*0 = 0.916
Data1-3	0.3, 0.3, 0.4	1, 0, 0	-ln(0.3)*1-ln(0.3)*0-ln(0.4)*0 = 1.204	Data2-3	0.1, 0.1, 0.8	1, 0, 0	-ln(0.1)*1-ln(0.1)*0-ln(0.8)*0 = 2.303
		Mean	0.691			Mean	1.287

3) Evaluation: accuracy calculation

```
# Loss and Accuracy

loss =  tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=pred, labels=y_))
train = tf.train.AdamOptimizer().minimize(loss)

correct_prediction = tf.equal(tf.argmax(pred, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

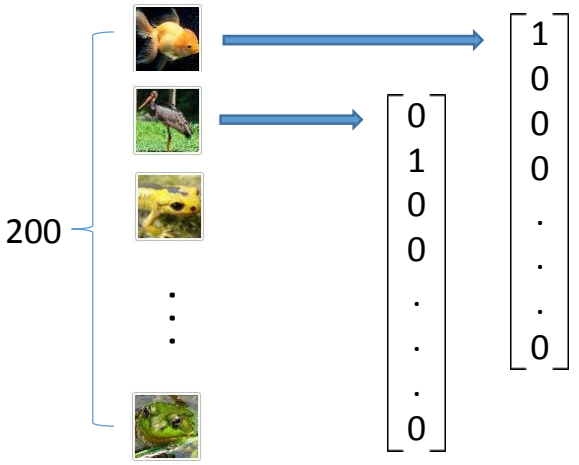


- tf.argmax(tensor, 1): return index of the item which has the max value
- tf.equal(x, y): return true if x==y otherwise false

	prediction	label	x = tf.argmax (prediction)	y = tf.argmax (label)	z = tf.equal(x,y)
Data1-1	0.1, 0.2, 0.7	0, 0, 1	2	2	True
Data1-2	0.1, 0.6, 0.3	0, 1, 0	1	1	True
Data1-3	0.3, 0.3, 0.4	1, 0, 0	2	0	False

3) Evaluation: accuracy calculation

```
# Loss and Accuracy

loss =  tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=pred, labels=y_))
train = tf.train.AdamOptimizer().minimize(loss)

correct_prediction = tf.equal(tf.argmax(pred, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```



- tf.argmax(tensor, 1): return index of the item which has the max value
- tf.equal(x, y): return true if x==y otherwise false

	prediction	label	x = tf.argmax (prediction)	y = tf.argmax (label)	z = tf.equal(x,y)	tf.cast(z)
Data1-1	0.1, 0.2, 0.7	0, 0, 1	2	2	True	1
Data1-2	0.1, 0.6, 0.3	0, 1, 0	1	1	True	1
Data1-3	0.3, 0.3, 0.4	1, 0, 0	2	0	False	0
					accuracy	2/3

4) Training

```
# Loss and Accuracy

loss =  tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=pred, labels=y_))
train = tf.train.AdamOptimizer().minimize(loss)

correct_prediction = tf.equal(tf.argmax(pred, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

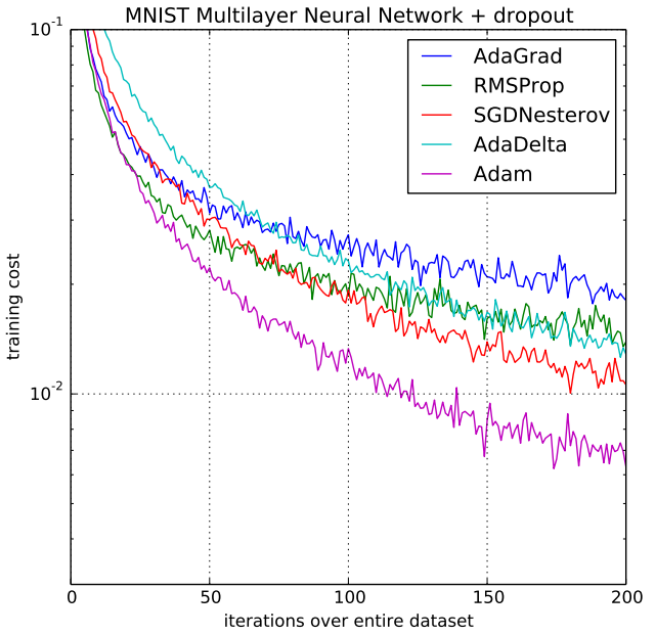
- tf.train.Optimizer
- tf.train.GradientDescentOptimizer
- tf.train.AdadeltaOptimizer
- tf.train.AdagradOptimizer
- tf.train.AdagradDAOptimizer
- tf.train.MomentumOptimizer
- tf.train.AdamOptimizer
- tf.train.FtrlOptimizer
- tf.train.ProximalGradientDescentOptimizer
- tf.train.ProximalAdagradOptimizer
- tf.train.RMSPropOptimizer

❑ tf.train.AdamOptimizer(learning_rate).minimize(loss)

- https://arxiv.org/pdf/1412.6980.pdf
- Now Adam is now recommended as the default algorithm to use.

```
__init__(
    learning_rate=0.001,
    beta1=0.9,
    beta2=0.999,
    epsilon=1e-08,
    use_locking=False,
    name='Adam'
)
```

```
minimize(
    loss,
    global_step=None,
    var_list=None,
    gate_gradients=GATE_OP,
    aggregation_method=None,
    colocate_gradients_with_ops=False,
    name=None,
    grad_loss=None
)
```



5) Testing: checkpoint saving

```
# Convolutional Layer 1

W_h1 = tf.Variable(tf.truncated_normal([3,3,3,16], stddev=1./math.sqrt(3*3*3)))
b_h1 = tf.Variable(tf.zeros([16]))

x_image = tf.reshape(x, [-1, image_height, image_width, 3])

conv1   = tf.nn.conv2d(x_image, W_h1, strides=[1,1,1,1], padding='SAME')
hidden1 = tf.nn.relu(conv1 + b_h1)

pool1 = tf.nn.max_pool(hidden1, ksize=[1,2,2,1], strides=[1,1,1,1], padding='SAME')
hidden1 = pool1
```

```
# Concolutional Layer 2

W_h2 = tf.Variable(tf.truncated_normal([3,3,16,32], stddev=1./math.sqrt(3*3*16)))
b_h2 = tf.Variable(tf.zeros([32]))

conv2   = tf.nn.conv2d(hidden1, W_h2, strides=[1,1,1,1], padding='SAME')
hidden2 = tf.nn.relu(conv2 + b_h2)

pool2 = tf.nn.max_pool(hidden2, ksize=[1,2,2,1], strides=[1,1,1,1], padding='SAME')
hidden2 = pool2
```

```
# Fully connected Layer 1

h_flat1 = tf.reshape(hidden2, [-1, 64*64*32])
fc_w1   = tf.Variable(tf.truncated_normal([64*64*32, 500], stddev=1./math.sqrt(64*64*32)))
fc_b1   = tf.Variable(tf.zeros([500]))

h_fc1 = tf.nn.relu(tf.matmul(h_flat1, fc_w1) + fc_b1)
```

```
# Fully connected Layer 2

fc_w2   = tf.Variable(tf.truncated_normal([500,200], stddev=1./math.sqrt(500*200)))
fc_b2   = tf.Variable(tf.zeros([200]))

h_fc2 = tf.nn.relu(tf.matmul(h_fc1, fc_w2) + fc_b2)
```

```
# Output layer

W_o = tf.Variable(tf.truncated_normal([200, 200], stddev=1./math.sqrt(200*200)))
b_o = tf.Variable(tf.truncated_normal([200]))

pred = tf.matmul(drop_fc, W_o) + b_o
```

```
# Check point define

var_list = [W_h1, b_h1, W_h2, b_h2]
saver = tf.train.Saver(var_list)
```

saver = tf.train.Saver()

Save all values

- Creating an instance of Saver()
  - In default, all values in the model are saved,
  - Values can be saved selectively.

```
__init__(
    var_list=None,
    reshape=False,
    sharded=False,
    max_to_keep=5,
    keep_checkpoint_every_n_hours=10000.0,
    name=None,
    restore_sequentially=False,
    saver_def=None,
    builder=None,
    defer_build=False,
    allow_empty=False,
    write_version=tf.train.SaverDef.V2,
    pad_step_number=False,
    save_relative_paths=False,
    filename=None
)
```

keep last 5 checkpoints

5) Testing: checkpoint saving

```
# Check point define

var_list = [W_h1, b_h1, W_h2, b_h2]
saver = tf.train.Saver(var_list)
```

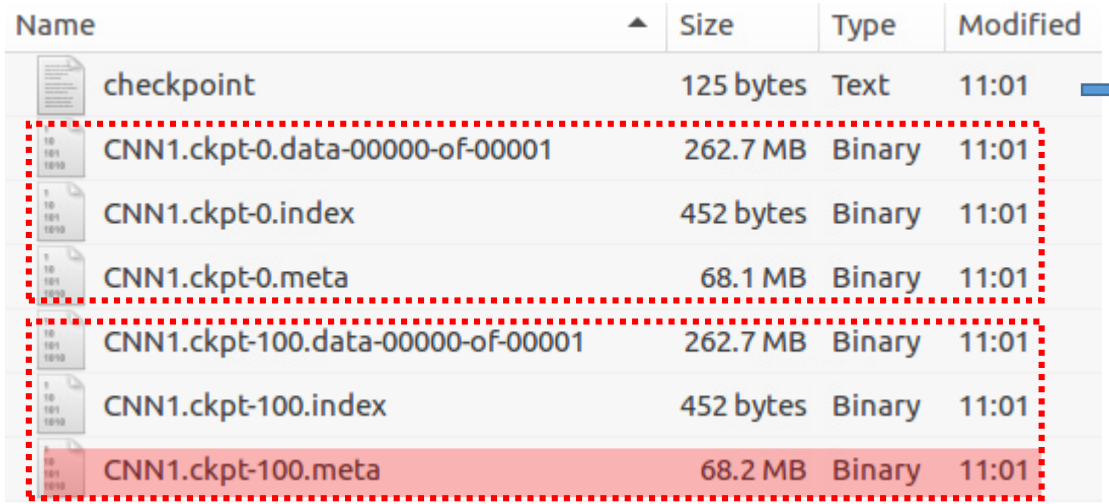
```
# Running

with tf.Session() as sess:
    coord = tf.train.Coordinator()
    thread = tf.train.start_queue_runners(sess, coord)

    sess.run(tf.global_variables_initializer())

    for i in range(1000):

        sess.run(train)
        _loss, _accuracy, _pred, _y, _x = sess.run([loss, accuracy, pred, y_, x])

        if i%100 == 0:
            saver.save(sess, './summary/CNN1.ckpt', i)
            print ("++++++++++++++++++++++++++++++++++++++++", i)
            print ("Completion: ", i/10000, "%")
            print ("loss: ", _loss)
            print ("accuracy: ", _accuracy)

            prediction = sess.run(tf.argmax(_pred, 1))
            print (prediction)

            label_ = sess.run(tf.argmax(_y, 1))
            print (label_)

    coord.request_stop()
    coord.join(thread)
```

- Running a save() method on a session.

```
save(
    sess,
    save_path,
    global_step=None,
    latest_filename=None,
    meta_graph_suffix='meta',
    write_meta_graph=True,
    write_state=True
)
```

5) Testing: checkpoint saving

- After saving, there will be three files under the specified directory
  - checkpoint: list of saving points including the latest one.
  - .meta: the complete Tensorflow graph structure which is used for restoring the graph.
  - .index and .data: actual values of variables: weights, biases, etc.



Name	Size	Type	Modified
checkpoint	125 bytes	Text	11:01
CNN1.ckpt-0.data-00000-of-00001	262.7 MB	Binary	11:01
CNN1.ckpt-0.index	452 bytes	Binary	11:01
CNN1.ckpt-0.meta	68.1 MB	Binary	11:01
CNN1.ckpt-100.data-00000-of-00001	262.7 MB	Binary	11:01
CNN1.ckpt-100.index	452 bytes	Binary	11:01
CNN1.ckpt-100.meta	68.2 MB	Binary	11:01

Most recent one

```
model_checkpoint_path: "CNN1.ckpt-100"
all_model_checkpoint_paths: "CNN1.ckpt-0"
all_model_checkpoint_paths: "CNN1.ckpt-100"
```

Directory name

```
check_point_state = tf.train.get_checkpoint_state("summary")

print (check_point_state.model_checkpoint_path)
print (check_point_state.all_model_checkpoint_paths)
```

```
summary/CNN1.ckpt-100
['summary/CNN1.ckpt-0', 'summary/CNN1.ckpt-100']
```

5) Testing: checkpoint restoring

- Reusing the graph definition used in the training.
  - For validating your trained CNN when they still have labels.

```
# Check point define

saver = tf.train.Saver()
```

```
with tf.Session() as sess:

    coord = tf.train.Coordinator()
    thread = tf.train.start_queue_runners(sess, coord)

    saver.restore(sess, './summary/CNN1.ckpt-1000')

    for i in range(10):

        _loss, _accuracy, _pred, _y, _x = sess.run([loss, accuracy, pred, y_, x])

        print ("++++++++++++++++++++++++++++++++++++", i)
        print ("loss: ", _loss)
        print ("accuracy: ", _accuracy)

        prediction = sess.run(tf.argmax(_pred, 1))
        print (prediction)

        label_ = sess.run(tf.argmax(_y, 1))
        print (label_)

    coord.request_stop()
    coord.join(thread)
```

- Redefining a clean test graph
  - For testing your trained CNN with individual image files without labels.

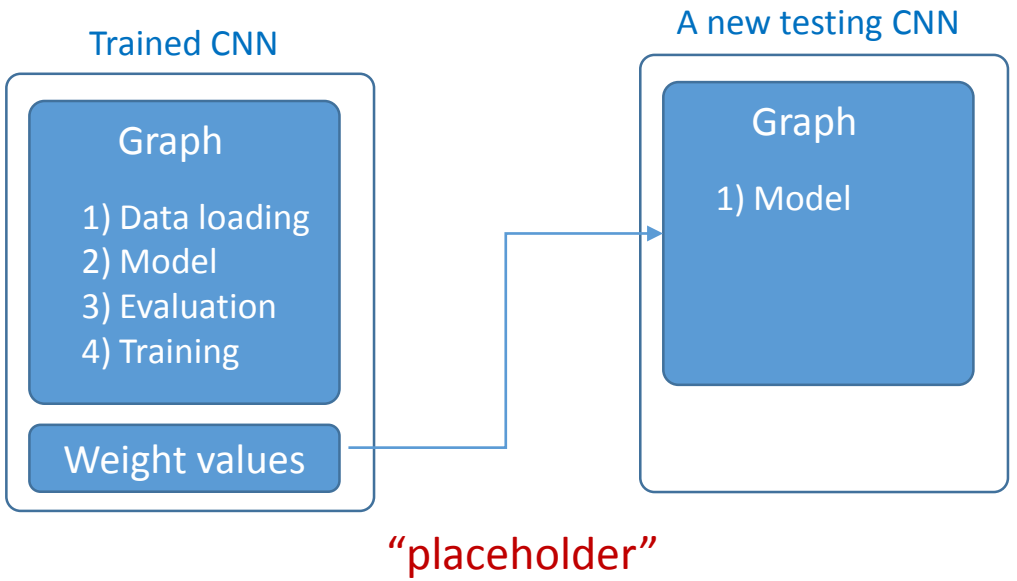
5) Testing: checkpoint restoring

- Reusing the graph definition used in the training.
  - For validating your trained CNN when they still have labels.

```
# Check point define

saver = tf.train.Saver()
```

```
with tf.Session() as sess:

    coord = tf.train.Coordinator()
    thread = tf.train.start_queue_runners(sess, coord)

    saver.restore(sess, './summary/CNN1.ckpt-1000')

    for i in range(10):

        _loss, _accuracy, _pred, _y, _x = sess.run([loss, accuracy, pred, y_, x])

        print ("++++++++++++++++++++++++++++++++++++", i)
        print ("loss: ", _loss)
        print ("accuracy: ", _accuracy)

        prediction = sess.run(tf.argmax(_pred, 1))
        print (prediction)

        label_ = sess.run(tf.argmax(_y, 1))
        print (label_)

    coord.request_stop()
    coord.join(thread)
```

- Redefining a clean test graph
  - For testing your trained CNN with individual image files without labels.
  - It is difficult to feed the data to the training model since it is optimized for reading data from files.
  - For evaluation, we only need “CNN” part.



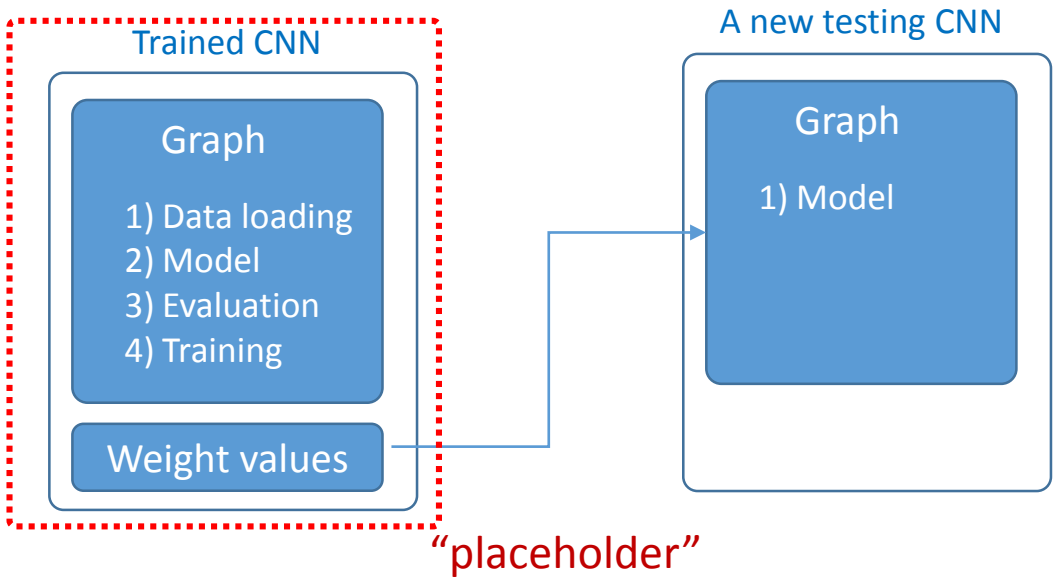
5) Testing: checkpoint restoring

```
# Call the training model with data

sess = tf.Session()

#First let's load meta graph and restore weights
training_saver = tf.train.import_meta_graph('./summary/CNN1.ckpt-900.meta')
training_saver.restore(sess,tf.train.latest_checkpoint('./summary'))
training_graph = tf.get_default_graph()

# Let's get the parameter of the test model from training graph

Wh1 = training_graph.get_tensor_by_name("cnn_layer1/W_h1:0")
bh1 = training_graph.get_tensor_by_name("cnn_layer1/b_h1:0")

Wh2 = training_graph.get_tensor_by_name("cnn_layer2/W_h2:0")
bh2 = training_graph.get_tensor_by_name("cnn_layer2/b_h2:0")

fcw1 = training_graph.get_tensor_by_name("fc_layer1/fc_w1:0")
fcb1 = training_graph.get_tensor_by_name("fc_layer1/fc_b1:0")

fcw2 = training_graph.get_tensor_by_name("fc_layer2/fc_w2:0")
fcb2 = training_graph.get_tensor_by_name("fc_layer2/fc_b2:0")

Wo = training_graph.get_tensor_by_name("out_layer/W_o:0")
bo = training_graph.get_tensor_by_name("out_layer/b_o:0")
```

Importing the trained CNN from a checkpoint

Extracting only weight and bias to build our new testing CNN

```
# Let's define a cnn test graph

#input layer
image_path = tf.placeholder(tf.string)
file_content = tf.read_file(image_path)
x = tf.cast(tf.image.decode_jpeg(file_content, channels=3), tf.float32)

# CNN layer 1
W_h1    = tf.placeholder(tf.float32)
b_h1    = tf.placeholder(tf.float32)
x_image = tf.reshape(x, [-1, 64, 64, 3], name='x_image')
conv1   = tf.nn.conv2d(x_image, W_h1, strides=[1,1,1,1], padding='SAME', name='conv1')
hidden1 = tf.nn.relu(conv1 + b_h1, name='hidden1')
pool1   = tf.nn.max_pool(hidden1, ksize=[1,2,2,1], strides=[1,1,1,1], padding='SAME', name='pool1')
hidden1 = pool1

# CNN layer 2
W_h2    = tf.placeholder(tf.float32)
b_h2    = tf.placeholder(tf.float32)
conv2   = tf.nn.conv2d(hidden1, W_h2, strides=[1,1,1,1], padding='SAME', name='conv2')
hidden2 = tf.nn.relu(conv2 + b_h2, name='hidden2')
pool2   = tf.nn.max_pool(hidden2, ksize=[1,2,2,1], strides=[1,1,1,1], padding='SAME', name='pool2')
hidden2 = pool2

# Fully connected layer 1
h_flat1 = tf.reshape(hidden2, [-1, 64*64*32], name='h_flat1')
fc_w1   = tf.placeholder(tf.float32)
fc_b1   = tf.placeholder(tf.float32)
h_fc1   = tf.nn.relu(tf.matmul(h_flat1, fc_w1) + fc_b1, name='h_fc1')

# Fully connected layer 2
fc_w2   = tf.placeholder(tf.float32)
fc_b2   = tf.placeholder(tf.float32)
h_fc2   = tf.nn.relu(tf.matmul(h_fc1, fc_w2) + fc_b2, name='h_fc2')

# Output layer
W_o     = tf.placeholder(tf.float32)
b_o     = tf.placeholder(tf.float32)
pred    = tf.matmul(h_fc2, W_o, name='pred') + b_o
```



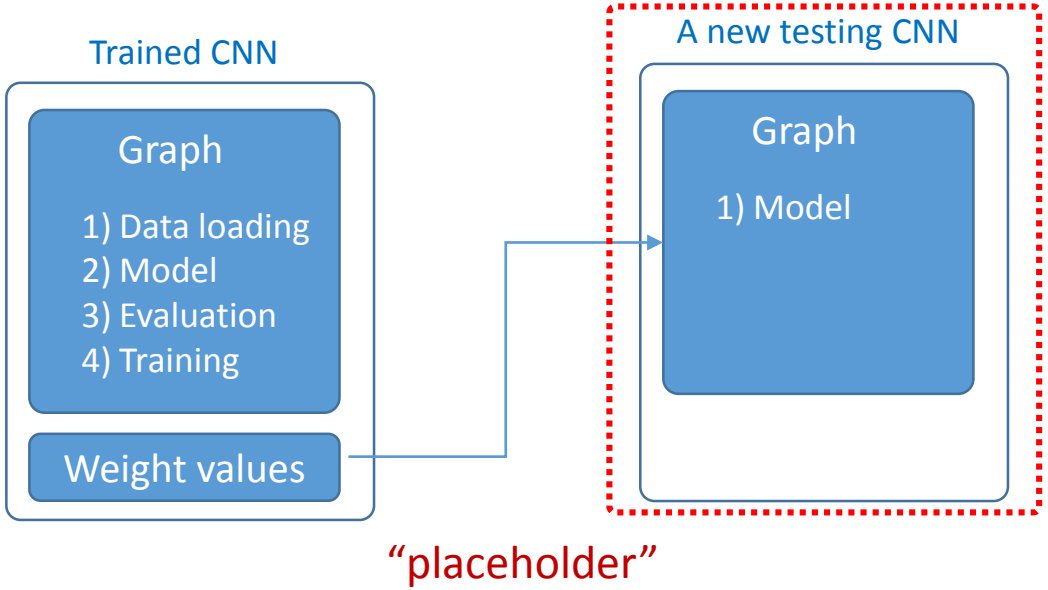
5) Testing: checkpoint restoring

```python
# Call the training model with data

sess = tf.Session()

#First let's load meta graph and restore weights
training_saver = tf.train.import_meta_graph('./summary/CNN1.ckpt-900.meta')
training_saver.restore(sess,tf.train.latest_checkpoint('./summary'))
training_graph = tf.get_default_graph()
```

```python
# Let's get the parameter of the test model from training graph

Wh1 = training_graph.get_tensor_by_name("cnn_layer1/W_h1:0")
bh1 = training_graph.get_tensor_by_name("cnn_layer1/b_h1:0")

Wh2 = training_graph.get_tensor_by_name("cnn_layer2/W_h2:0")
bh2 = training_graph.get_tensor_by_name("cnn_layer2/b_h2:0")

fcw1 = training_graph.get_tensor_by_name("fc_layer1/fc_w1:0")
fcb1 = training_graph.get_tensor_by_name("fc_layer1/fc_b1:0")

fcw2 = training_graph.get_tensor_by_name("fc_layer2/fc_w2:0")
fcb2 = training_graph.get_tensor_by_name("fc_layer2/fc_b2:0")

Wo = training_graph.get_tensor_by_name("out_layer/W_o:0")
bo = training_graph.get_tensor_by_name("out_layer/b_o:0")
```

```python
# Let's define a cnn test graph

#input layer
image_path = tf.placeholder(tf.string)
file_content = tf.read_file(image_path)
x = tf.cast(tf.image.decode_jpeg(file_content, channels=3), tf.float32)

# CNN layer 1
W_h1    = tf.placeholder(tf.float32)
b_h1    = tf.placeholder(tf.float32)
x_image = tf.reshape(x, [-1, 64, 64, 3], name='x_image')
conv1   = tf.nn.conv2d(x_image, W_h1, strides=[1,1,1,1], padding='SAME', name='conv1')
hidden1 = tf.nn.relu(conv1 + b_h1, name='hidden1')
pool1   = tf.nn.max_pool(hidden1, ksize=[1,2,2,1], strides=[1,1,1,1], padding='SAME', name='pool1')
hidden1 = pool1

# CNN layer 2
W_h2    = tf.placeholder(tf.float32)
b_h2    = tf.placeholder(tf.float32)
conv2   = tf.nn.conv2d(hidden1, W_h2, strides=[1,1,1,1], padding='SAME', name='conv2')
hidden2 = tf.nn.relu(conv2 + b_h2, name='hidden2')
pool2   = tf.nn.max_pool(hidden2, ksize=[1,2,2,1], strides=[1,1,1,1], padding='SAME', name='pool2')
hidden2 = pool2

# Fully connected layer 1
h_flat1 = tf.reshape(hidden2, [-1, 64*64*32], name='h_flat1')
fc_w1   = tf.placeholder(tf.float32)
fc_b1   = tf.placeholder(tf.float32)
h_fc1   = tf.nn.relu(tf.matmul(h_flat1, fc_w1) + fc_b1, name='h_fc1')

# Fully connected layer 2
fc_w2   = tf.placeholder(tf.float32)
fc_b2   = tf.placeholder(tf.float32)
h_fc2   = tf.nn.relu(tf.matmul(h_fc1, fc_w2) + fc_b2, name='h_fc2')

# Output layer
W_o     = tf.placeholder(tf.float32)
b_o     = tf.placeholder(tf.float32)
pred    = tf.matmul(h_fc2, W_o, name='pred') + b_o
```

Defining a new testing CNN with “placeholder”



5) Testing: checkpoint restoring

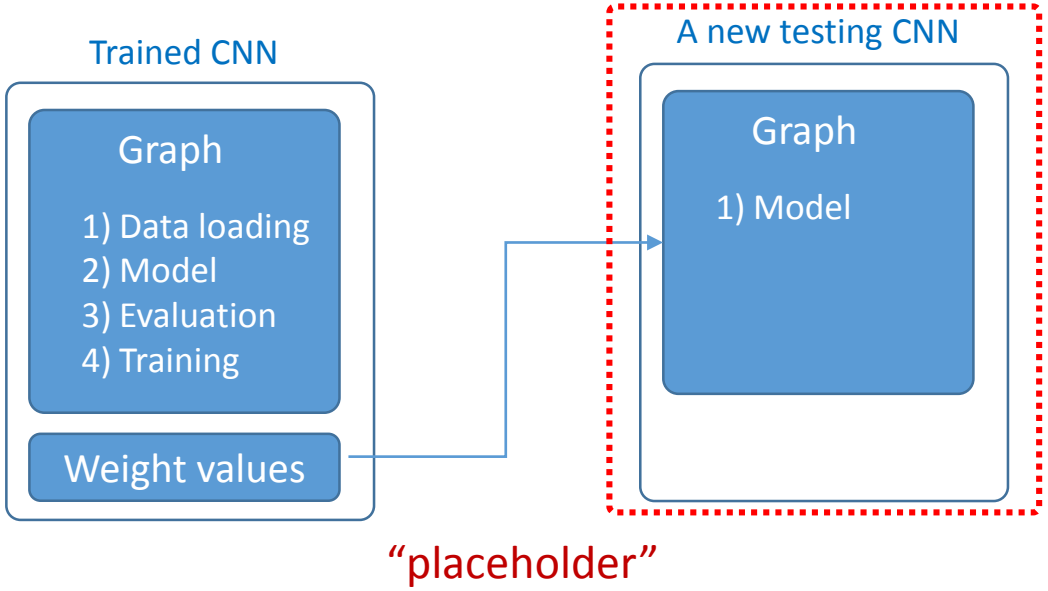
```
# Call the training model with data

sess = tf.Session()

#First let's load meta graph and restore weights
training_saver = tf.train.import_meta_graph('./summary/CNN1.ckpt-900.meta')
training_saver.restore(sess,tf.train.latest_checkpoint('./summary'))
training_graph = tf.get_default_graph()
```

```
# Let's get the parameter of the test model from training graph

Wh1 = training_graph.get_tensor_by_name("cnn_layer1/W_h1:0")
bh1 = training_graph.get_tensor_by_name("cnn_layer1/b_h1:0")

Wh2 = training_graph.get_tensor_by_name("cnn_layer2/W_h2:0")
bh2 = training_graph.get_tensor_by_name("cnn_layer2/b_h2:0")

fcw1 = training_graph.get_tensor_by_name("fc_layer1/fc_w1:0")
fcb1 = training_graph.get_tensor_by_name("fc_layer1/fc_b1:0")

fcw2 = training_graph.get_tensor_by_name("fc_layer2/fc_w2:0")
fcb2 = training_graph.get_tensor_by_name("fc_layer2/fc_b2:0")

Wo = training_graph.get_tensor_by_name("out_layer/W_o:0")
bo = training_graph.get_tensor_by_name("out_layer/b_o:0")
```

```
# Let's define a cnn test graph

#input layer
image_path = tf.placeholder(tf.string)
file_content = tf.read_file(image_path)
x = tf.cast(tf.image.decode_jpeg(file_content, channels=3), tf.float32)

# CNN layer 1
W_h1    = tf.placeholder(tf.float32)
b_h1    = tf.placeholder(tf.float32)
x_image = tf.reshape(x, [-1, 64, 64, 3], name='x_image')
conv1   = tf.nn.conv2d(x_image, W_h1, strides=[1,1,1,1], padding='SAME', name='conv1')
hidden1 = tf.nn.relu(conv1 + b_h1, name='hidden1')
pool1   = tf.nn.max_pool(hidden1, ksize=[1,2,2,1], strides=[1,1,1,1], padding='SAME', name='pool1')
hidden1 = pool1

# CNN layer 2
W_h2    = tf.placeholder(tf.float32)
b_h2    = tf.placeholder(tf.float32)
conv2   = tf.nn.conv2d(hidden1, W_h2, strides=[1,1,1,1], padding='SAME', name='conv2')
hidden2 = tf.nn.relu(conv2 + b_h2, name='hidden2')
pool2   = tf.nn.max_pool(hidden2, ksize=[1,2,2,1], strides=[1,1,1,1], padding='SAME', name='pool2')
hidden2 = pool2

# Fully connected layer 1
h_flat1 = tf.reshape(hidden2, [-1, 64*64*32], name='h_flat1')
fc_w1   = tf.placeholder(tf.float32)
fc_b1   = tf.placeholder(tf.float32)
h_fc1   = tf.nn.relu(tf.matmul(h_flat1, fc_w1) + fc_b1, name='h_fc1')

# Fully connected layer 2
fc_w2   = tf.placeholder(tf.float32)
fc_b2   = tf.placeholder(tf.float32)
h_fc2   = tf.nn.relu(tf.matmul(h_fc1, fc_w2) + fc_b2, name='h_fc2')

# Output layer
W_o     = tf.placeholder(tf.float32)
b_o     = tf.placeholder(tf.float32)
pred    = tf.matmul(h_fc2, W_o, name='pred') + b_o
```

```
feed_dict = {
    W_h1: Wh1.eval(session=sess),
    b_h1: bh1.eval(session=sess),
    W_h2: Wh2.eval(session=sess),
    b_h2: bh2.eval(session=sess),
    fc_w1: fcw1.eval(session=sess),
    fc_b1: fcb1.eval(session=sess),
    fc_w2: fcw2.eval(session=sess),
    fc_b2: fcb2.eval(session=sess),
    W_o: Wo.eval(session=sess),
    b_o:bo.eval(session=sess),
    image_path: image_list_with_full_dir[i]
}
```

Defining a new testing CNN with "placeholder"



Backup Slides

Tensorboard: plotting defined graph

- tf.summary.merge_all()
- train_writer = tf.summary.FileWriter('./tensorboard/', sess.graph)
- train_writer.close() # closing when jobs are done

creating an event file in the given directory and add summaries and events to it.

```
with tf.Session() as sess:
    coord = tf.train.Coordinator()
    thread = tf.train.start_queue_runners(sess, coord)

    sess.run(tf.global_variables_initializer())

    # Tensorboard
    merged = tf.summary.merge_all()

    for i in range(1000):

        sess.run(train)
        summary, _loss, _accuracy, _pred, _y, _x = sess.run([merged, loss, accuracy, pred, y_, x])

        if (i)%100 == 0:

            # Tensorboard
            train_writer = tf.summary.FileWriter('./tensorboard/', sess.graph)

            saver.save(sess, './summary/CNN1.ckpt', i)
            print ("++++++++++++++++++++++++++++++++++++", i)
            print ("Completion: ", i/100000, "%")
            print ("loss: ", _loss)
            print ("accuracy: ", _accuracy)

            prediction = sess.run(tf.argmax(_pred, 1))
            print (prediction)

            label_ = sess.run(tf.argmax(_y, 1))
            print (label_)

            train_writer.close()

    coord.request_stop()
    coord.join(thread)
```

- Under the directory of "./tensorboard/", the files are created as follows:

Name	Size	Type
events.out.tfevents.1508717197.catt	136.2 MB	Binary
events.out.tfevents.1508717201.catt	136.3 MB	Binary
events.out.tfevents.1508717206.catt	136.4 MB	Binary
events.out.tfevents.1508717210.catt	136.5 MB	Binary
events.out.tfevents.1508717215.catt	136.6 MB	Binary
events.out.tfevents.1508717219.catt	136.7 MB	Binary
events.out.tfevents.1508717224.catt	136.8 MB	Binary
events.out.tfevents.1508717229.catt	136.9 MB	Binary
events.out.tfevents.1508717234.catt	137.0 MB	Binary
events.out.tfevents.1508717238.catt	137.1 MB	Binary

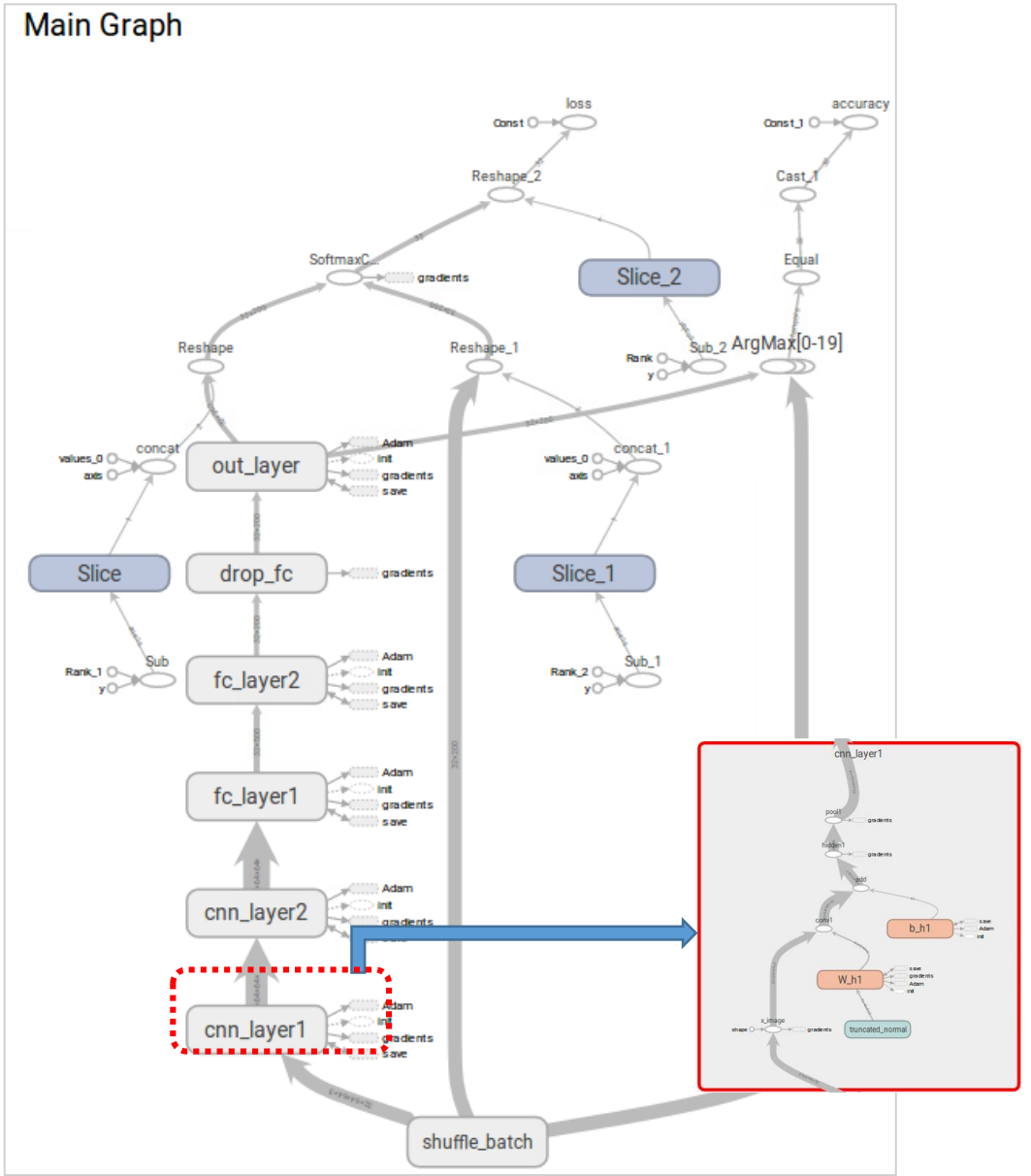
Tensorboard: plotting defined graph

- Once the log files are created, execute the command below:
  - tensorboard --logdir=path/to/log-directory

- Then, in a browser, typing the below (similar to Jupyter notebook)
  - localhost:6006

- When a graph looks complicated, a scope can be defined.
- All information under the scope is displayed under the name of the scope.
  - with tf.name_scope('cnn_layer1'):

```
# Convolutional Layer 1
with tf.name_scope('cnn_layer1'):
    W_h1 = tf.Variable(tf.truncated_normal([3,3,3,16], stddev=1./math.sqrt(3*3*3)), name='W_h1')
    b_h1 = tf.Variable(tf.zeros([16]), name='b_h1')

    x_image = tf.reshape(x, [-1, image_height, image_width, 3], name='x_image')

    conv1   = tf.nn.conv2d(x_image, W_h1, strides=[1,1,1,1], padding='SAME', name='conv1')
    hidden1 = tf.nn.relu(conv1 + b_h1, name='hidden1')

    pool1 = tf.nn.max_pool(hidden1, ksize=[1,2,2,1], strides=[1,1,1,1], padding='SAME', name='pool1')
    hidden1 = pool1
```



Tensorboard: plotting loss and accuracy

```
# Loss and Accuracy

loss =  tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=pred, labels=y_), name='loss')
correct_prediction = tf.equal(tf.argmax(pred, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32), name='accuracy')

# for plotting loss in tensorboard
tf.summary.scalar('loss', loss)
tf.summary.scalar('accuracy', accuracy)
```

```
with tf.Session() as sess:
    coord = tf.train.Coordinator()
    thread = tf.train.start_queue_runners(sess, coord)

    sess.run(tf.global_variables_initializer())

    # Tensorboard - accuracy
    merged = tf.summary.merge_all()

    for i in range(1000):

        sess.run(train)
        summary, _loss, _accuracy, _pred, _y, _x = sess.run([merged, loss, accuracy, pred, y_, x])
        if (i)%100 == 0:

            # Tensorboard - graph
            train_writer = tf.summary.FileWriter('./tensorboard/', sess.graph)

            # Tensorboard - loss, accuracy
            train_writer.add_summary(summary, i)

            saver.save(sess, './summary/CNN1.ckpt', i)
            print ("++++++++++++++++++++++++++++++++++++++", i)
            print ("Completion: ", i/100000, "%")
            print ("loss: ", _loss)
            print ("accuracy: ", _accuracy)

            prediction = sess.run(tf.argmax(_pred, 1))
            print (prediction)

            label_ = sess.run(tf.argmax(_y, 1))
            print (label_)

            train_writer.close()

    coord.request_stop()
    coord.join(thread)
```

- Summary operations
  - Summaries provides a way to export condensed information about a model, which is then accessible by Tensorboard.

- add_summary: a method of FileWriter class
  - Adds the scalar values to the event file.