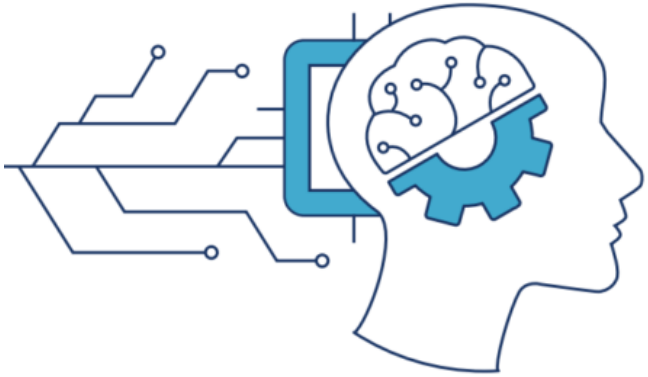
성남-KAIST AI
[Deep Learning 실습]

TAs: Kiwon Lee and Bae Gwangtak

Advisor: Junmo Kim

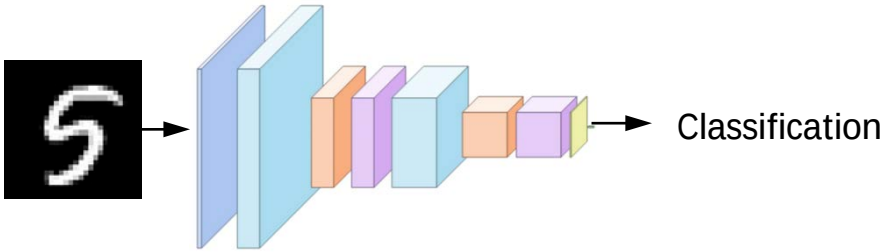
KAIST SIIT Lab



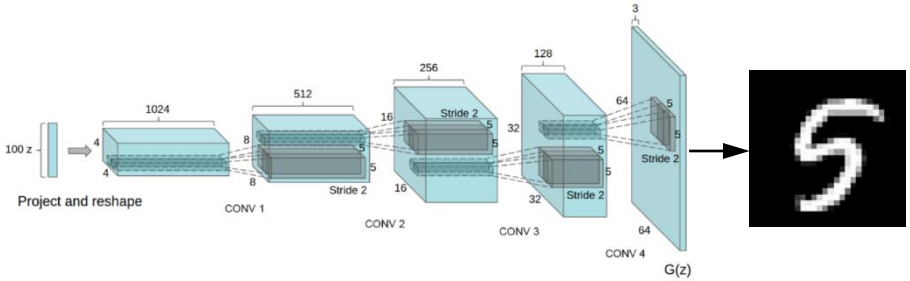
[Contact Info.]
Kiwon Lee: kaiser5072@kaist.ac.kr
Bae Gwangtak: gwangtak.bae@kaist.ac.kr
Junmo Kim: junmo.kim@kaist.ac.kr

Tutorial Guidelines

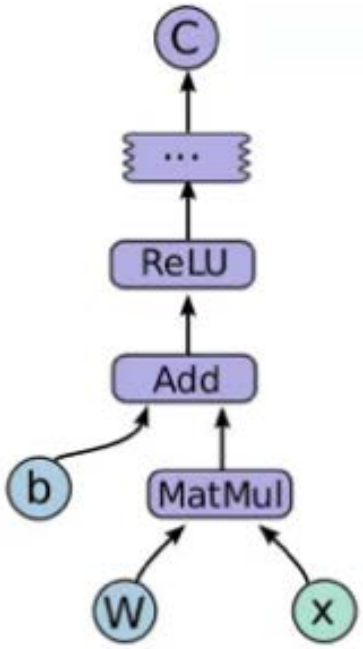
- **Session 1. Supervised Learning (Image Classification)**
  - Convolutional Neural Network (DIY network)



- **Session 2. Unsupervised Learning (Image Generation)**
  - Generative Adversarial Network (GAN)



What is Tensorflow?



- A python library
- Google (Brain)
- Open-source
- Library for numerical computation using data flow graphs
- CPU and GPU

https://www.tensorflow.org/

airbnb
AIRBUS DEFENCE & SPACE
CISCO
DeepMind
Google
IBM Power Systems
Movidius
SAP
ARM
CEVA
Dropbox
ebay
intel
JD.COM 京东
QUALCOMM
quantiphi
UBER

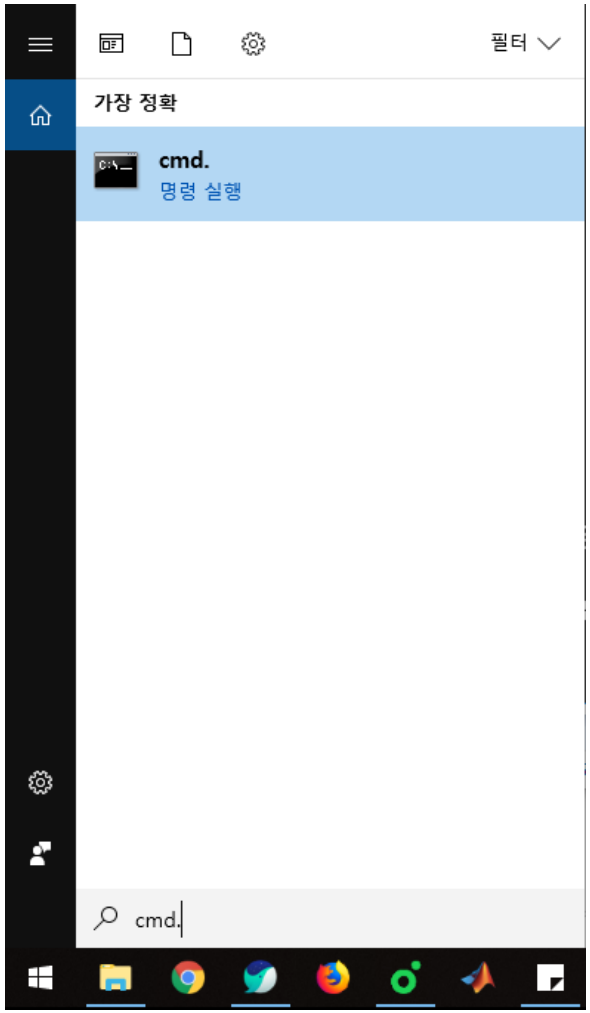
Install Requirements

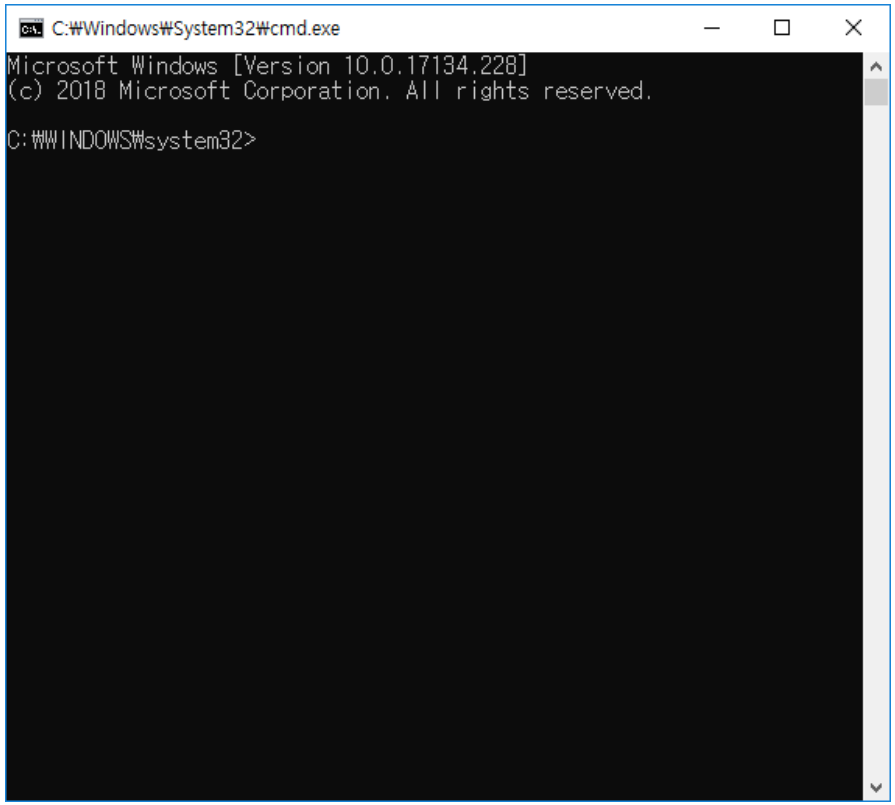
- **CPU-only**
  - Ubuntu 16.04 or later (64 bit)
  - masOS 10.12.6 (Sierra) or later (64 bit)
  - Windows 7 or later (64 bit) (Python 3 only)

- **GPU support**
  - NVIDIA GPU card withi CUDA Compute Capability 3.5 or higher https://developer.nvidia.com/cuda-gpus
  - GPU drivers, CUDA Toolkit, cuDNN...

CPU-only Tensorflow (I)

- Click windows + s button and execute terminal



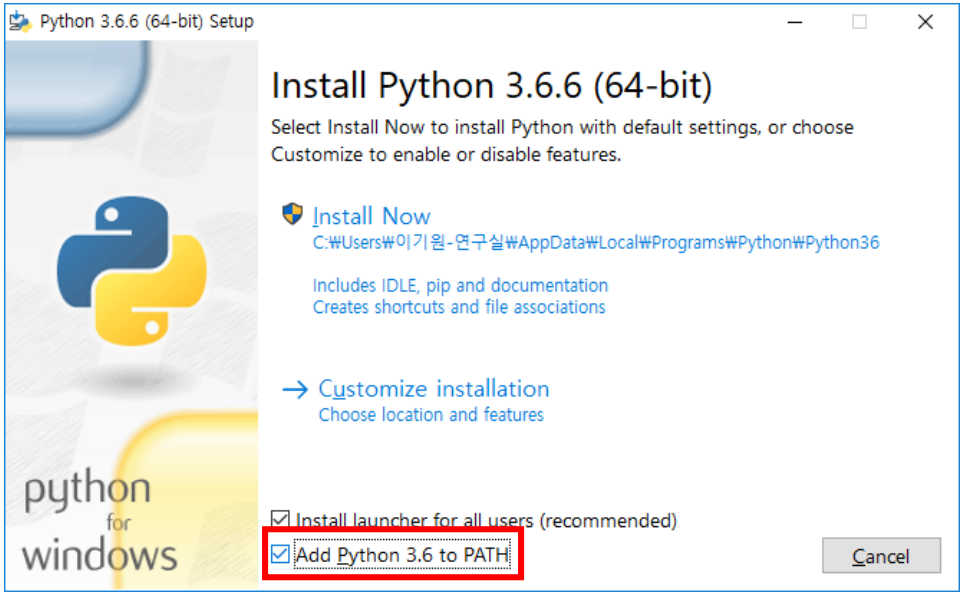



```
python3 –-version
pip3 –-version
virtualenv --version
```

CPU-only Tensorflow (II)

- Update build tools
  1. Go to the Visual Studio downloads
  2. Select *Redistributables and Build Tools*
  3. Download and install the Microsoft Visual C++ 2015 Redistributable Update 3.

- Install python 3



CPU-only Tensorflow (III)

- Install virtualenv

```
pip3 install –U pip virtualenv
```

- Install Tensorflow

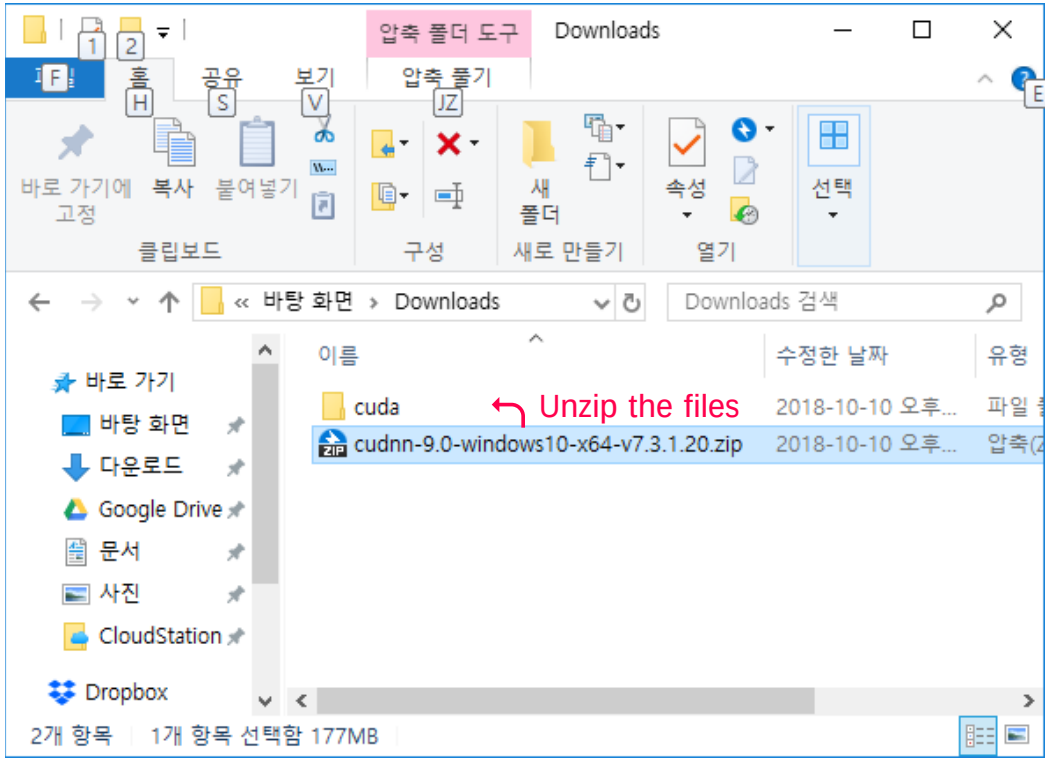
```
pip3 install –user –upgrade tensorflow
python3 –c “import tensorflow as tf; print(tf.__version__)”
```

GPU support Tensorflow (I)

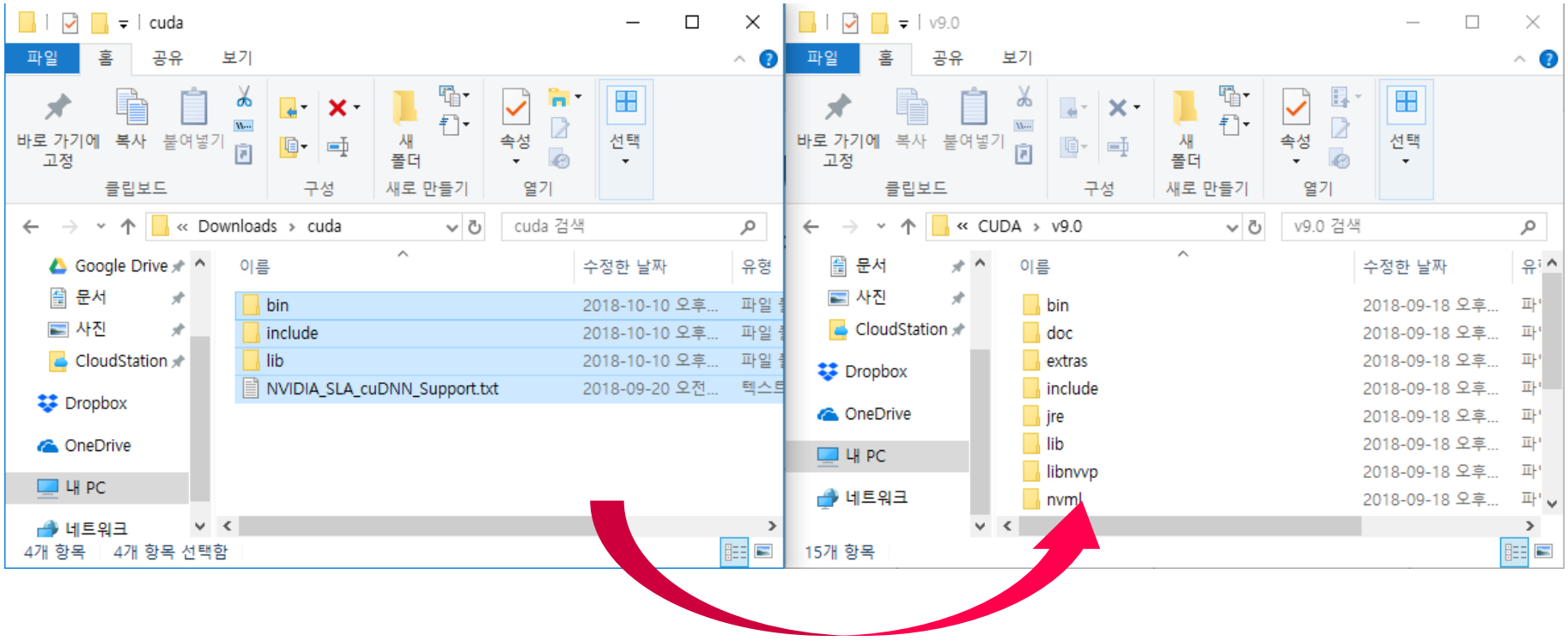
- Requirements

https://www.tensorflow.org/install/gpu

  - CUDA Toolkit – Tensorflow supports CUDA 9.0
  - cuDNN SDK – (≥7.2)



GPU support Tensorflow (II)



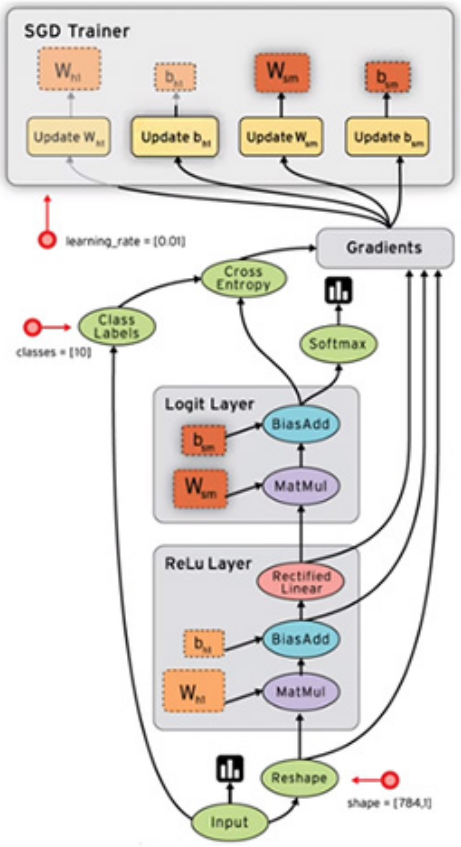
Copy the files to the folder where cuda is installed

GPU support Tensorflow (III)

- Install Tensorflow

```
pip3 install –user –upgrade tensorflow-gpu
python3 –c “import tensorflow as tf; print(tf.__version__)”
```

Principle



[Tensorflow workflow]

1. Draw your graph
2. Feed data
3. ... and optimize

[Graph]

- Placeholders: gates where we introduce examples.
- Model: makes predictions. Set of variables and operations.
- Cost function: function that computes the model error.
- Optimizer: algorithm that optimizes the variables.

Example Code

- import tensorflow as tf

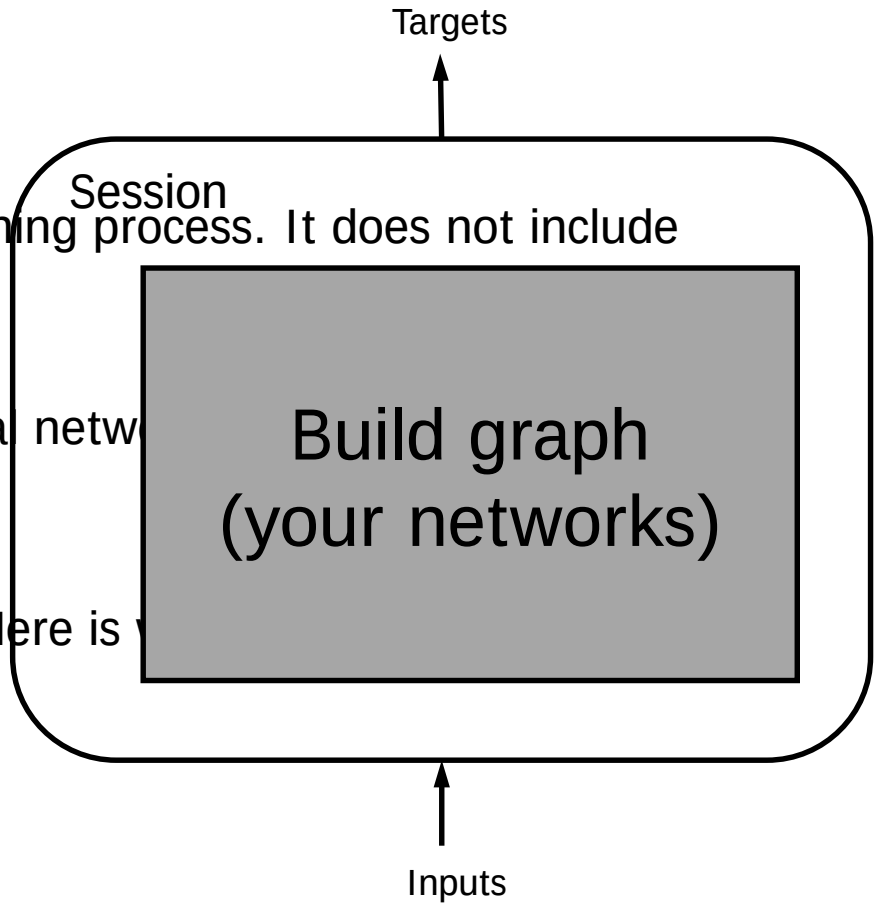
- mnist = tf.contrib.learn.datasets.load_dataset()
- model.build_graph()

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

- for t in range(max_iters):
- 	sess.run(model.cost, model.train_op)

Graph, Data and Session

- Graph
  - Layout of the prediction and learning process. It does not include data.
- Data
  - Examples that will train the neural netw
    inputs and targets.
- Session
  - Where everything takes places. Here is w
    with data.



Constant, variable, and placeholder

- Difference between constant/variable and placeholder.
- Placeholder: Feeding data to your graph during session.
- Variable: learnable/not learnable
- Constant: a fixed variable

[**Placeholder**]
```
images = tf.placeholder(tf.float32, shape=(hps.batch_size, 28, 28, 1), name='images')
```

[**Variable**]
```
w = tf.get_variable('weight/DW', [x.get_shape()[1], out_dim],
                            initializer=tf.uniform_unit_scaling_initializer(factor=1.0),
                    trainable=True)
```

[Constant]
```
Init = tf.ones(...) / tf.random_normal(...)
```

Operations

- Tensorflow math ops are pretty standard, quite similar to Numpy.
- Visit https://www.tensorflow.org/api_docs/cc/group/math-ops

Category	Examples
Element-wise mathematical operations	Add, Sub, Mul, Div, Exp, Log, Greater, Less, Equal, ...
Array operations	Concat, Slice, Split, Constant, Rank, Shape, Shuffle, ...
Matrix operations	MatMul, MatrixInverse, MatrixDeterminant, ...
Stateful operations	Variable, Assign, AssignAdd, ...
Neural network building blocks	SoftMax, Sigmoid, ReLU, Convolution2D, MaxPool, ...
Checkpointing operations	Save, Restore
Queue and synchronization operations	Enqueue, Dequeue, MutexAcquire, MutexRelease, ...
Control flow operations	Merge, Switch, Enter, Leave, NextIteration

```
tf.add(a, b) # >> [5 8]
tf.add_n([a, b, b]) # >> [7 10]. Equivalent to a + b + b
tf.mul(a, b) # >> [6 12] because mul is element wise
tf.matmul(a, b) # >> ValueError
tf.matmul(tf.reshape(a, shape=[1, 2]), tf.reshape(b, shape=[2, 1])) # >> [[18]]
tf.div(a, b) # >> [1 3]
tf.mod(a, b) # >> [1 0]
```

Feeds and Fetches

- Feeds
  - Designating specific operations to be ‘feed’ by using tf.placeholder().
- Fetches
  - Execute the graph with a run() call on the Session object and pass in the tensors to retrieve.
  - All the ops needed to produce the values of the requested tensors are run once.

```
input1 = tf.placeholder(tf.float32)
input2 = tf.placeholder(tf.float32)
input3 = tf.placeholder(tf.float32)
intermed = tf.add(input2, input3)
mul = tf.mul(input1, intermed)

With tf.Session() as sess:
  result = sess.run([output],
                feed_dict={input1: 3.0,
                           input2: 2.0,
                               input3: 5.0})
  print(result)
```

```
input1 = tf.constant([3.0])
input2 = tf.constant([2.0])
```

```
input3 = tf.constant([5.0])
intermed = tf.add(input2, input3)
mul = tf.mul(input1, intermed)

With tf.Session() as sess:
  result = sess.run([mul, intermed])
  print(result)
```

Coding by Examples

Image Classification

- Task: image classification
- Dataset: MNIST (28x28)
- Network: Simple CNN (VGG-like)

1. Prepare your data
2. Build graph
3. Make session & Initialize
4. Run session

1. Prepare your data

- Load MNIST data

```
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets('./data/', one_hot=True)
```

- Split train/val/test (image, label) pairs

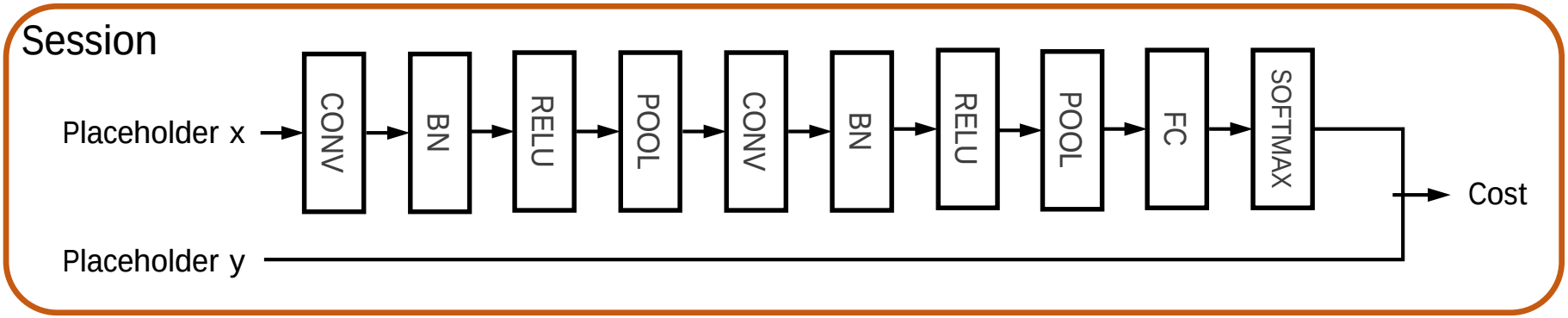
```
train_images = mnist.train.images
train_labels = mnist.train.labels
train_images = train_images.reshape([-1, 28, 28, 1]) / 255.
train_images = train_images[0:1000]

val_images = mnist.validation.images
val_labels = mnist.validation.labels
val_images = val_images.reshape([-1, 28, 28, 1]) / 255.
```

2. Build graph

- From inputs to targets.
- Data:

```
x = tf.placeholder(tf.float32, shape=[hps.batch_size, 28, 28, 1])
y = tf.placeholder(tf.float32, shape=[hps.batch_size, 10])
```

- Network:
  - (CONV-BN-RELU) – POOL – (CONV-BN-RELU) – POOL – FC - SOFTMAX



2. Build graph - Convolution

- Network:
  - (CONV-BN-RELU) – POOL – (CONV-BN-RELU) – POOL – FC - SOFTMAX

```
def conv(x, filter_size, in_filters, out_filters, strides=[1, 1, 1, 1]):
 w = tf.get_variable(
            'weight/DW', [filter_size, filter_size, in_filters, out_filters],
                  tf.float32, initializer=tf.uniform_unit_scaling_initializer(factor=2.0))
 return tf.nn.conv2d(x, w, strides=[1, 1, 1, 1], padding='SAME')
```

Usage: y = conv(x, 3, in_filter, out_filter, stride)

2. Build graph – Batch normalization

- Network:
  - (CONV-BN-RELU) – POOL – (CONV-BN-RELU) – POOL – FC - SOFTMAX

```
def batch_norm(x):
  params_shape = [x.get_shape()[-1]]

  beta = tf.get_variable('beta', params_shape, tf.float32,
       initializer=tf.constant_initializer(0.0, tf.float32))
  gamma = tf.get_variable('gamma', params_shape, tf.float32,
       initializer=tf.constant_initializer(1.0, tf.float32))
  mean, variance = tf.nn.moments(x, [0, 1, 2])

  return  tf.nn.batch_normalization(x, mean, variance, beta, gamma)
```

Usage: y = batch_norm(x)

2. Build graph – ReLU

- Network:
  - (CONV-BN-RELU) – POOL – (CONV-BN-RELU) – POOL – FC - SOFTMAX

```
def relu(x, leakiness=0.0):
  return tf.maximum(x, leakiness*x)
```

Usage: y = relu(x, leakiness)

2. Build graph – Pooling

- Network:
  - (CONV-BN-RELU) – POOL – (CONV-BN-RELU) – POOL – FC - SOFTMAX

```
def max_pool(x):
 return tf.nn.max_pool(x, ksize=[1, 2, 2, 1],
                       strides=[1, 2, 2, 1], padding='SAME')

def avg_pool(x):
 return tf.nn.avg_pool(x, ksize=[1, 2, 2, 1],
                       strides=[1, 2, 2, 1], padding='SAME')
```

Usage: y = max_pool(x)

2. Build graph – Fully connected

- Network:
  - (CONV-BN-RELU) – POOL – (CONV-BN-RELU) – POOL – FC - SOFTMAX

```
def fully_connected(self, name, x, out_dim):
    x = tf.reshape(x, [self.hps.batch_size, -1])
    w = tf.get_variable(
        'weight/DW', [x.get_shape()[1], out_dim],
        initializer=tf.uniform_unit_scaling_initializer(factor=1.0))
    b = tf.get_variable('bias', [out_dim],
                        initializer=tf.constant_initializer())
  return tf.nn.xw_plus_b(x, w, b)
```

Usage: y = fully_connected(x, num_classes)

2. Build graph – Cost function & Optimizer

- Network:
  - (CONV-BN-RELU) – POOL – (CONV-BN-RELU) – POOL – FC - SOFTMAX

Cost function
(Cross entropy)

```
cent = tf.nn.softmax_cross_entropy_with_logits(logits=logits, labels=self.labels)
self.cost = tf.reduce_mean(cent, name='cent')
```

Optimizer
(SGD+momentum)

```
def build_train_op():
    optimizer = tf.train.MomentumOptimizer(self.lrn_rate, 0.9)
    train_vars = tf.trainable_variables()
    grads = tf.gradients(self.cost, train_vars)
    apply_op = optimizer.apply_gradients(zip(grads, train_vars),
                                         global_step=self.global_step,
                                         name='train_step')
    train_ops = [apply_op] + self._extra_train_ops
    self.train_op = tf.group(*train_ops)
```

3. Make session & Initialize

- A Session object encapsulates the environment in which Operation objects are executed, and Tensor objects are evaluated.

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

4. Run session

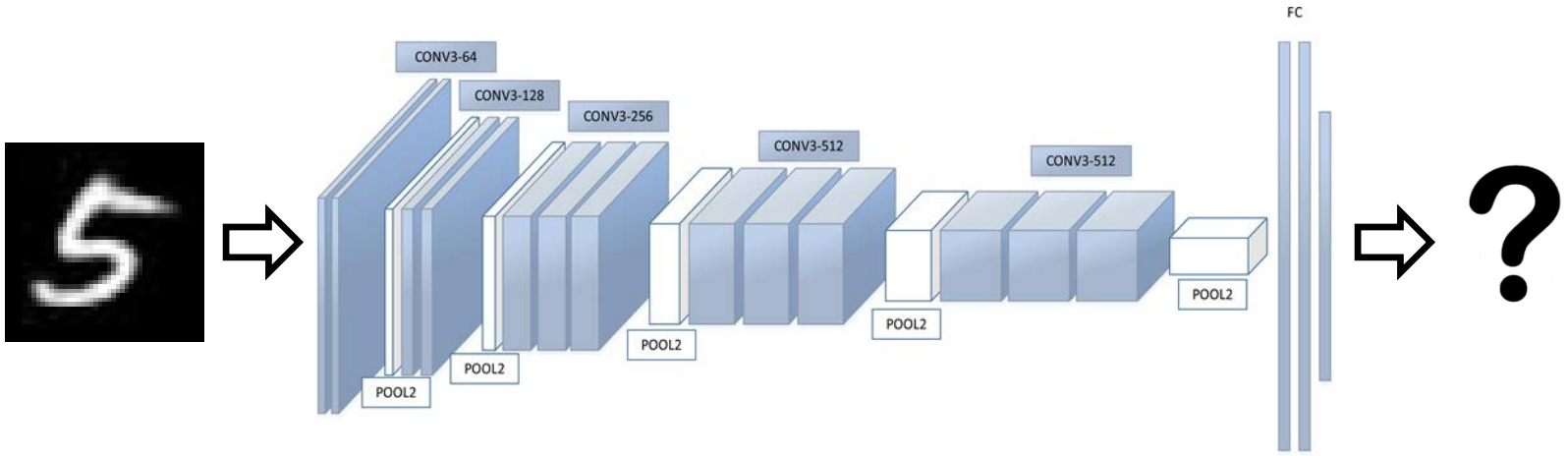
```
(_, loss) = sess.run([model.train_op, model.cost],
                     feed_dict={model.lrn_rate: lrn_rate,
                  model.images: batch_images,
                               model.labels: batch_labels})
```

Experiment

Image Classification

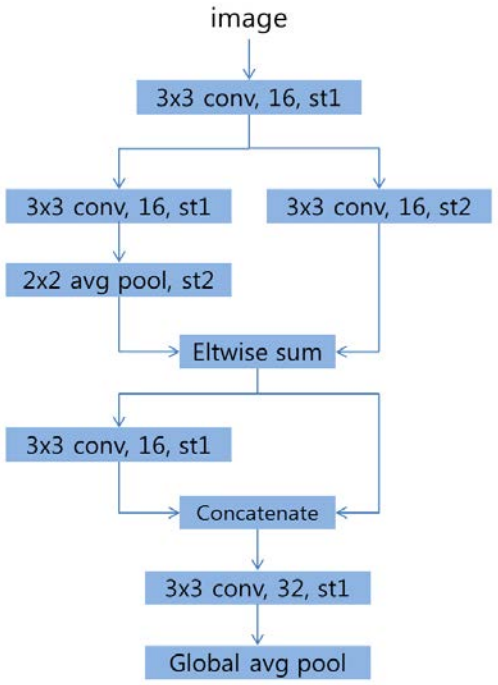
Task: Image Classification

- MNIST classification
- Provided code: VGG / ResNet architecture.



Task: Image Classification

- Project goal 1: (Simple Network)
  - Implement the following architecture.
- Project goal 2: (DIY Network)
  - Make your own architecture.




```
## BUILD GRAPH
## Option 1: resnet
## Option 2: vggnet
## Option 3: simple network (fill in the blank!)
## Option 4: DIY network (fill in the blank!)

def build_graph(self):
  ## GLOBAL ITERATION
  self.global_step = tf.contrib.framework.get_or_create_global_step()

  if self.mode == 'train':
    ## FEATURE EXTRACTION
    with tf.variable_scope('embed') as scope:
      #feats = self.resnet(self.images)
      #feats = self.vggnet(self.images)
      feats = self.simple_network(self.images)
      #feats = self.DIY_network(self.images)
```

```
##############################
####### FILL IN THE BLANK ######
##############################
##############################
####### BE CREATIVE! ##########
##############################
def DIY_network(self, images):
  """ Your Code """


  img_feat = self._global_avg_pool(x)
  return img_feat
```