Tensorflow ConvNet

MNIST Classification
CAT vs DOG Classification

최준명
Pusan National University
Vision & Intelligent system Lab
https://jimmylifestudy.blogspot.kr/
jmchoi@pusan.ac.kr

Tensorflow | 지난 시간 복습

- 먼저 **Graph**를 Build하고 **Session**을 통해 실행한다.
- 변수는 tf.Variable( ) 보단 **tf.get_variable( )**
- 입력은 **tf.placeholder( )** with feed_dict 나 **tf.data( )**를 쓰자.
- 모델 building은 **Layer API** 를 쓰면 간편하다. (단점: debugging이 어려움)
- 간단한 Linear Regression 모델
- 간단한 Logistic Regression 모델

Tensorflow | 오늘 목표

- Convolutional layer을 이용한 MNIST 분석
- 나만의 데이터를 이용하여 ConNet 구성

Convolutional layer in Tensorflow

Tensorflow | Convolutional layer

MNIST Dataset

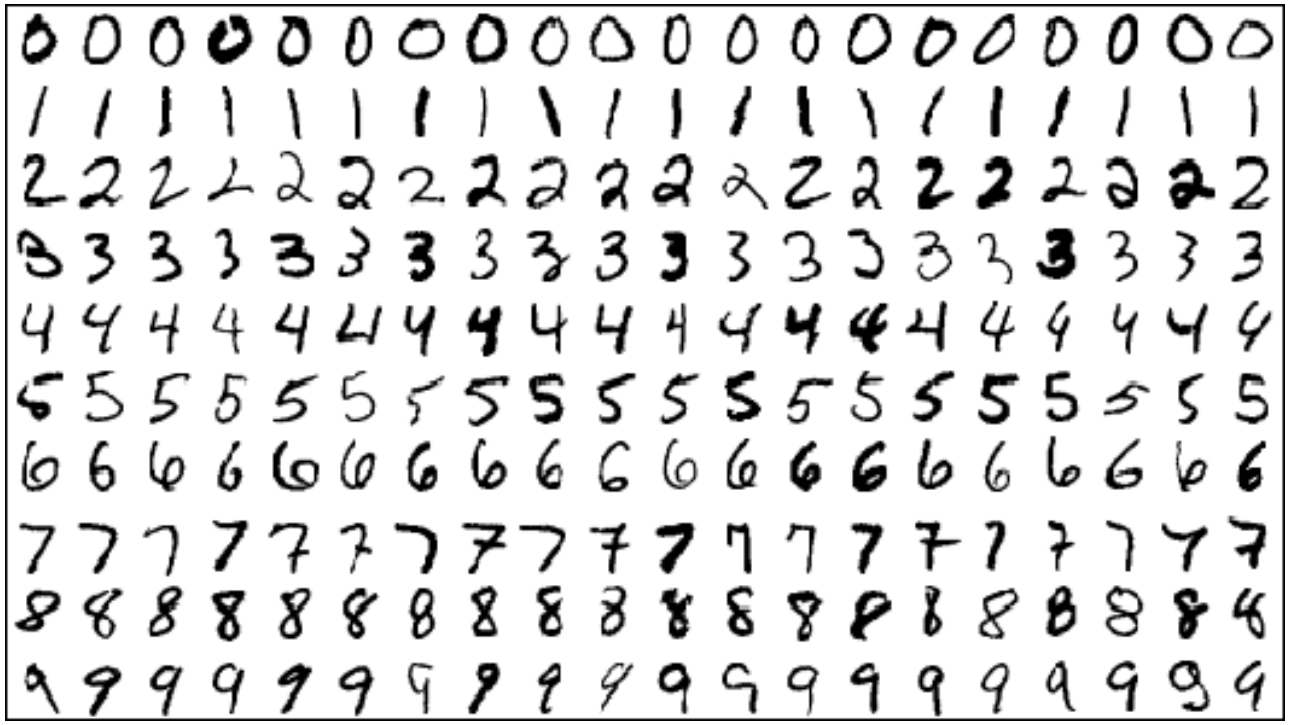
X = 28x28 array(1d size of 784)

Y = Digit value

MNIST.train: 55,000 examples
MNIST.validation: 5,000 examples
MNIST.test: 10,000 examples

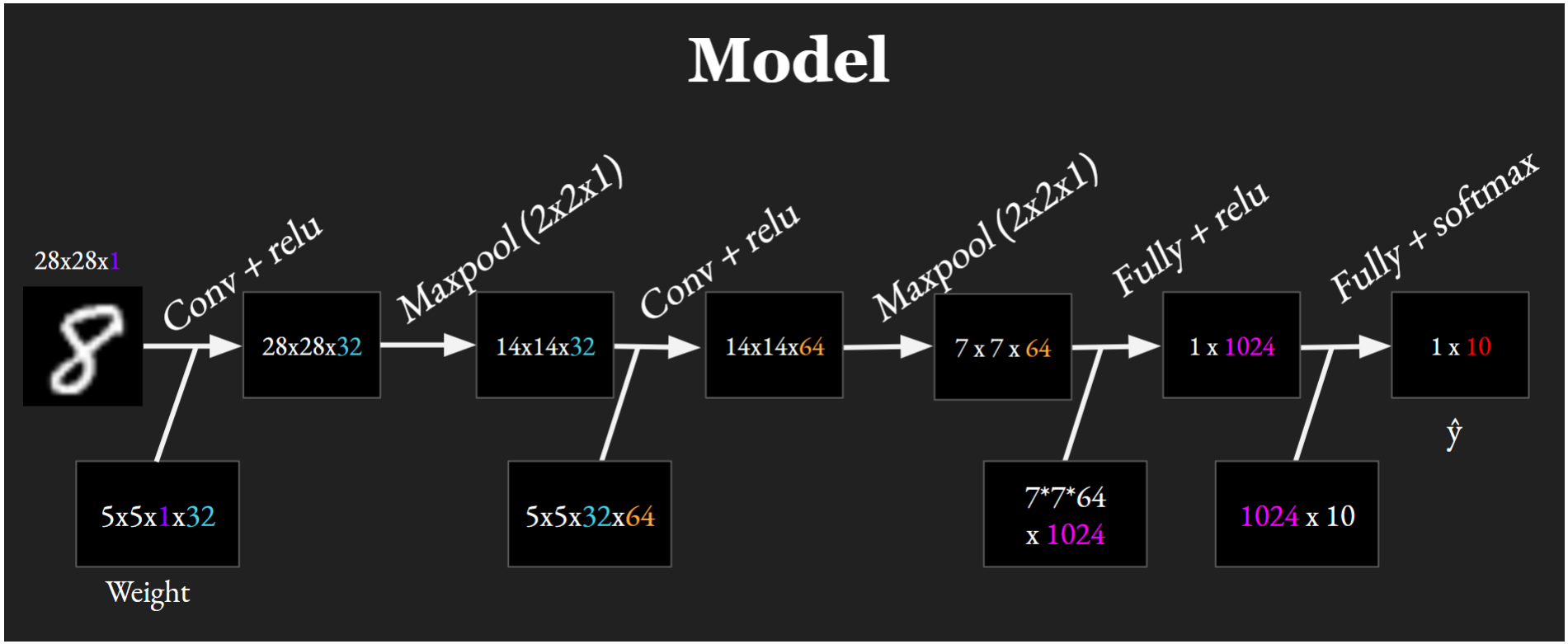
Inference: Y_predicted = softmax(X * w + b)

Cross entropy loss: −log(Y_predicted)



출처: https://docs.google.com/presentation/d/1lmcQVNAmJrL8x3Iq0VB1mVaka1r6pOIb-TMVTX5Rufc/edit#slide=id.g1c166da651_0_5

Tensorflow | Convolutional layer



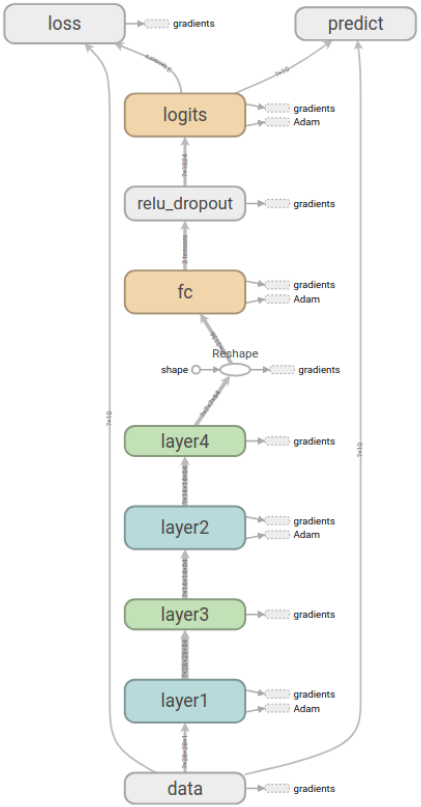
출처: https://docs.google.com/presentation/d/17VTArfQVtapBqfYecyvp3Kp9HKy8Pw2WI12acYME2nI/edit#slide=id.g1c60f09bdb_0_336

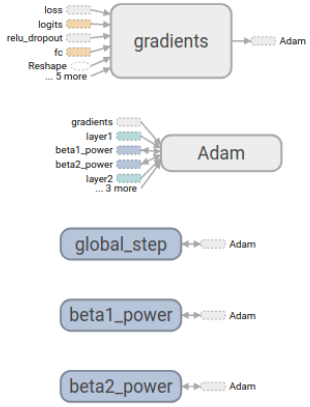
Tensorflow | Convolutional layer

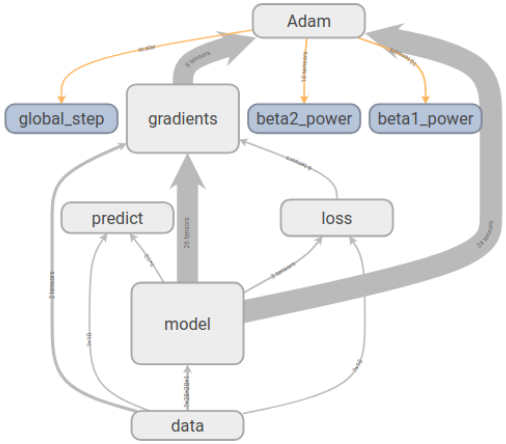
```python
def conv2d(inputs, nb_filter, filter_size, strides=1, padding='same',
           activation=tf.nn.relu, bias=True, weights_init=variance_scaling_initializer,
           bias_init=zeros_initializer, trainable=True, scope='Conv2d'):
  with tf.variable_scope(scope,  values=[inputs], reuse=tf.AUTO_REUSE) as scope:
    W_init = weights_init
    bias_init = bias_init
    x = tf.layers.conv2d(inputs,filters=nb_filter,kernel_size=filter_size,strides=[strides,strides],padding=padding,
                         kernel_initializer=W_init,bias_initializer=bias_init,activation=activation,trainable=trainable)
  return x

def maxpool(inputs, filter_size=2, strides=1, padding='same', scope='pool'):
  with tf.variable_scope(scope, reuse=tf.AUTO_REUSE) as scope:
    x = tf.layers.max_pooling2d(inputs,pool_size=[filter_size,filter_size],strides=[strides,strides],padding=padding)
  return x

def fully_connected(inputs, nb_filter, activation=tf.nn.relu, bias=True, weights_init=variance_scaling_initializer,
           bias_init=zeros_initializer, trainable=True,  scope='FC'):

  if len(inputs.get_shape().as_list())>2:
    n_inputs = int(np.prod(inputs.get_shape().as_list()[1:]))
    inputs = tf.reshape(inputs, [-1, n_inputs])
  with tf.variable_scope(scope, reuse=tf.AUTO_REUSE) as scope:
    x = tf.layers.dense(inputs,units=nb_filter,activation=activation,use_bias=bias,kernel_initializer=weights_init,
                        bias_initializer=bias_init,trainable=trainable)
  return x
```

Tensorflow | Convolutional layer

```python
def inference(self):
  with tf.variable_scope('model') as scope:
    conv1 = conv2d(inputs=self.img, nb_filter=32, filter_size=5, trainable=self.training, scope='layer1')
    pool1 = maxpool(conv1, 2, 2, 'same', scope='layer2')
    conv2 = conv2d(inputs=pool1, nb_filter=64, filter_size=5, trainable=self.training, scope='layer3')
    pool2 = maxpool(conv2, 2, 2, 'same', scope='layer4')
    fc = fully_connected(pool2, 1024, trainable=self.training, scope='layer5')
    self.logits = fully_connected(fc, self.n_classes, trainable=self.training, scope='logits')
```

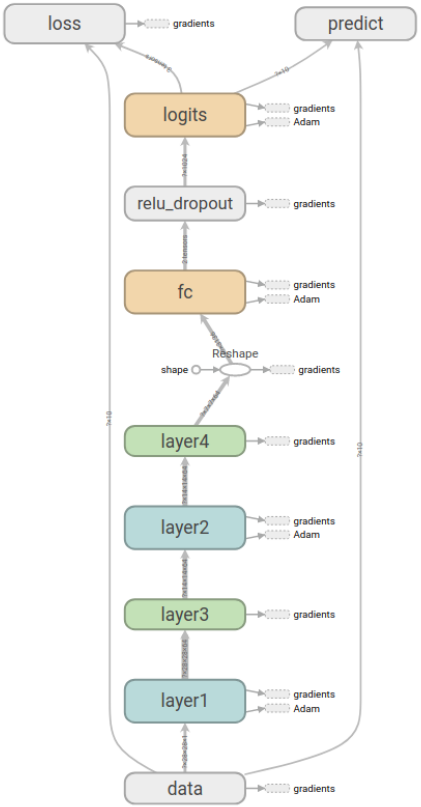


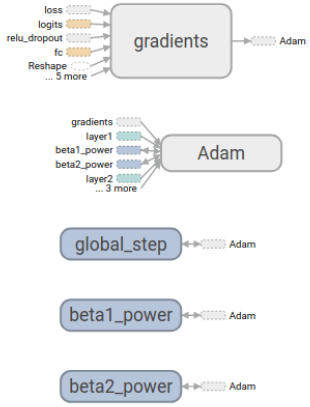


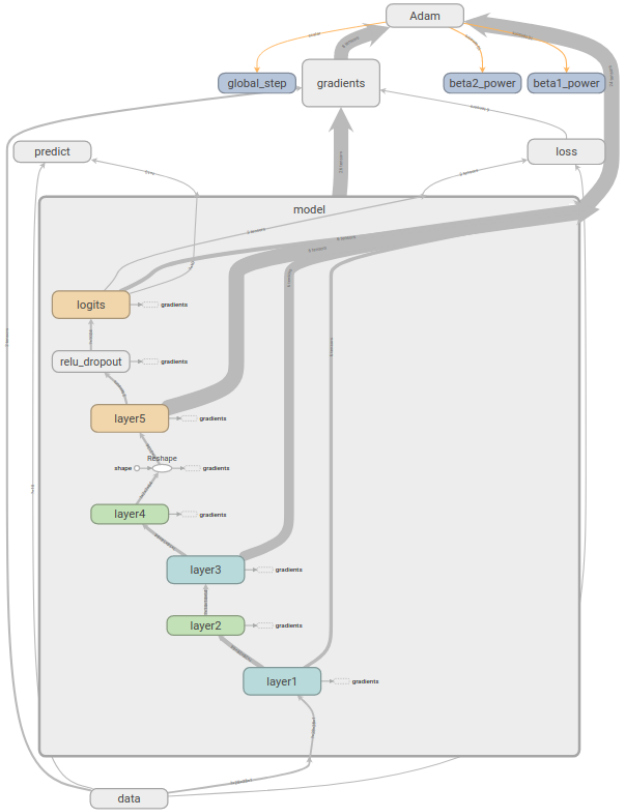


Tensorflow | Convolutional layer

```python
def inference(self):
  with tf.variable_scope('model') as scope:
    conv1 = conv2d(inputs=self.img, nb_filter=32, filter_size=5, trainable=self.training, scope='layer1')
    pool1 = maxpool(conv1, 2, 2, 'same', scope='layer2')
    conv2 = conv2d(inputs=pool1, nb_filter=64, filter_size=5, trainable=self.training, scope='layer3')
    pool2 = maxpool(conv2, 2, 2, 'same', scope='layer4')
    fc = fully_connected(pool2, 1024, trainable=self.training, scope='layer5')
    self.logits = fully_connected(fc, self.n_classes, trainable=self.training, scope='logits')
```







Tensorflow | Convolutional layer

```python
def loss(self):
    '''
    define loss function
    use softmax cross entropy with logits as the loss function
    compute mean cross entropy, softmax is applied internally
    '''
    #
    with tf.name_scope('loss'):
        entropy = tf.nn.softmax_cross_entropy_with_logits(labels=self.label, logits=self.logits)
        self.loss = tf.reduce_mean(entropy, name='loss')

def optimize(self):
    '''
    Define training op
    using Adam Gradient Descent to minimize cost
    '''
    self.opt = tf.train.AdamOptimizer(self.lr).minimize(self.loss,
                                                        global_step=self.gstep)

def summary(self):
    '''
    Create summaries to write on TensorBoard
    '''
    with tf.name_scope('summaries'):
        tf.summary.scalar('loss', self.loss)
        tf.summary.scalar('accuracy', self.accuracy_holder)
        tf.summary.histogram('histogram loss', self.loss)
        self.summary_op = tf.summary.merge_all()

def build(self):
    '''
    Build the computation graph
    '''
    self.get_data()
    self.inference()
    self.loss()
    self.optimize()
    self.eval()
    self.summary()
```

Tensorflow | Convolutional layer

```python
def train_one_epoch(self, sess, saver, writer, epoch, step):
  start_time = time.time()
  self.training = True
  n_batches = int(self.mnist.train.num_examples/self.batch_size)
  total_loss = 0
  for j in range(n_batches):
    self.X_batch,self.Y_batch = self.mnist.train.next_batch(self.batch_size)
    _, l = sess.run([self.opt, self.loss], feed_dict={self.X: self.X_batch, self.label: self.Y_batch})
    step += 1
    total_loss += l
  saver.save(sess, 'checkpoints/convnet_mnist/mnist-convnet', step)
  print('Average loss at epoch {0}: {1} Took: {2} seconds'.format(epoch, total_loss/n_batches,time.time()-start_time))
  return step

def eval_once(self, sess, writer, epoch, step):
  start_time = time.time()
  self.training = False
  total_correct_preds = 0
  n_batches = int(self.mnist.test.num_examples/100)
  for j in range(n_batches):
    self.X_batch, self.Y_batch = self.mnist.test.next_batch(100)
    accuracy_batch = sess.run(self.accuracy, feed_dict={self.X: self.X_batch, self.label: self.Y_batch})
    total_correct_preds += accuracy_batch
  summaries = sess.run(self.summary_op, feed_dict={self.X:self.X_batch,self.label:self.Y_batch,
                                                   self.accuracy_holder:total_correct_preds/n_batches})
  writer.add_summary(summaries, global_step=step)
  print('Accuracy at epoch {0}: {1} Took: {2} seconds'.format(epoch, total_correct_preds/n_batches,
                                                              time.time()-start_time))
```

Tensorflow | Convolutional layer

```python
def train(self, n_epochs):
    '''
    The train function alternates between training one epoch and evaluating
    '''
    utils.safe_mkdir('checkpoints')
    utils.safe_mkdir('checkpoints/convnet_mnist')
    writer = tf.summary.FileWriter('./graphs/convnet', tf.get_default_graph())

    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())
        saver = tf.train.Saver()
        ckpt = tf.train.get_checkpoint_state(os.path.dirname('checkpoints/convnet_mnist/checkpoint'))
        if ckpt and ckpt.model_checkpoint_path:
            saver.restore(sess, ckpt.model_checkpoint_path)

        step = self.gstep.eval()

        for epoch in range(n_epochs):
            step = self.train_one_epoch(sess, saver, writer, epoch, step)
            self.eval_once(sess, writer, epoch, step)
    writer.close()
```

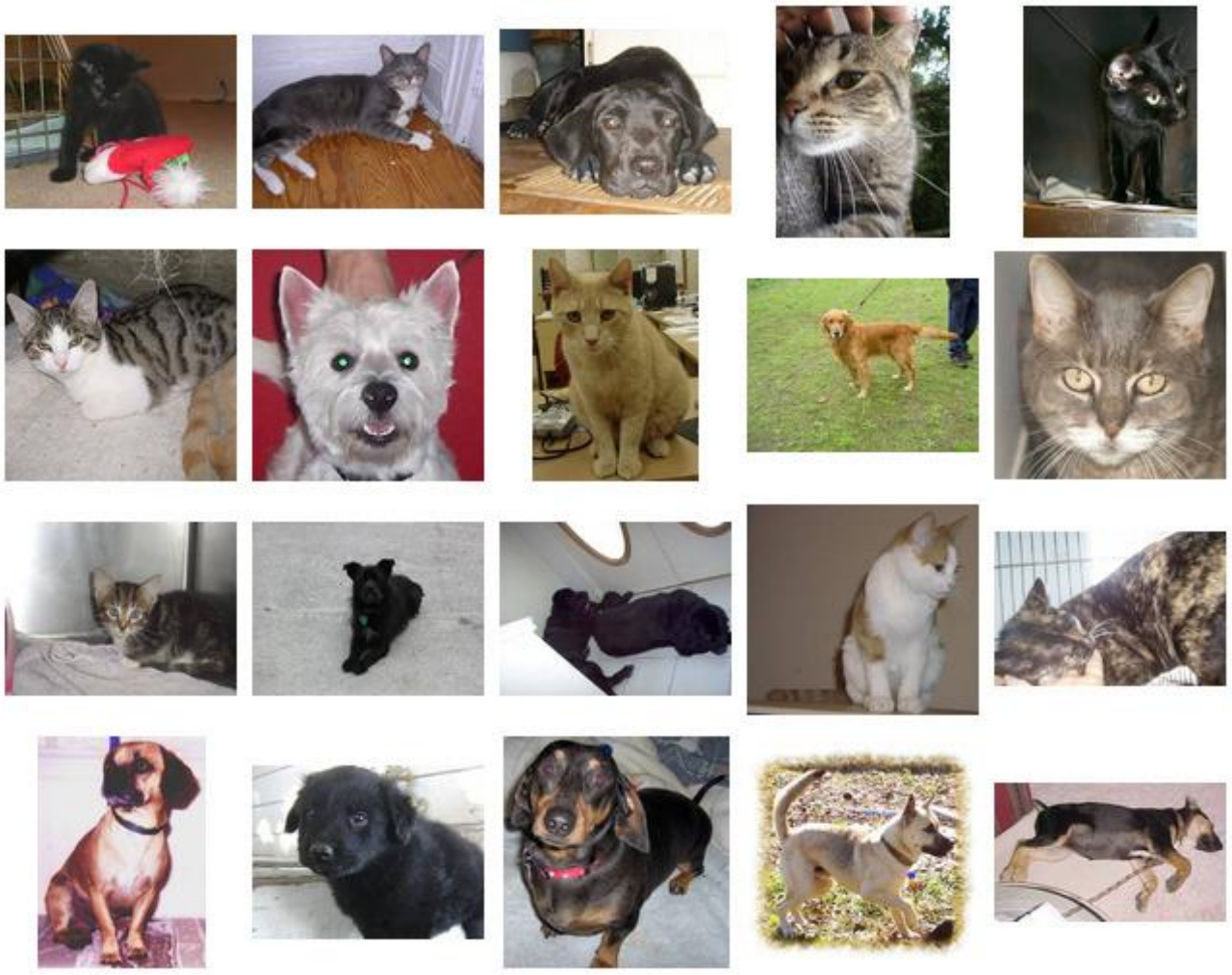
Tensorflow | Custom dataset

Kaggle dogs vs cats (download here)

X = 크기가 다양함
Y = 0 or 1

dog.1.jpg
dog.2.jpg
....

cat.1.jpg
cat.2.jpg
...



Tensorflow | Custom dataset

```
def get_files(data_path,datafolder):
    cats = []
    label_cats = []
    dogs = []
    label_dogs = []
    for file in datafolder:
        name = file.split('.')
        if name[0]=='cat':
            cats.append(os.path.join(data_path, file))
            label_cats.append(0)
        else:
            dogs.append(os.path.join(data_path, file))
            label_dogs.append(1)

    image_list = np.hstack((cats, dogs))
    label_list = np.hstack((label_cats, label_dogs))

    shuffle = list(zip(image_list, label_list))
    random.shuffle(shuffle)
    image_list, label_list = zip(*shuffle)
    return image_list, label_list
```

Tensorflow | Custom dataset

```
def get_batch(image, label, image_W, image_H, batch_size, capacity):
    image = tf.cast(image, tf.string)
    label = tf.cast(label, tf.int32)

    # make an input queue
    input_queue = tf.train.slice_input_producer([image, label])
    label = input_queue[1]
    image_contents = tf.read_file(input_queue[0])
    image = tf.image.decode_jpeg(image_contents, channels=3)

    # data argumentation
    image = tf.image.resize_image_with_crop_or_pad(image, image_W, image_H)


    image = tf.image.per_image_standardization(image)

    image_batch, label_batch = tf.train.batch([image, label],
                                              batch_size=batch_size,
                                              num_threads=64,
                                              capacity=capacity)
    label_batch = tf.one_hot(label_batch, depth=2)
    image_batch = tf.cast(image_batch, tf.float32)

    return image_batch, label_batch
```

Tensorflow | Custom dataset

Data augmentation (https://www.tensorflow.org/api_docs/python/tf/image)

`adjust_brightness(...)` : Adjust the brightness of RGB or Grayscale images.

`adjust_contrast(...)` : Adjust contrast of RGB or grayscale images.

`adjust_gamma(...)` : Perform

`adjust_hue(...)` : Adjust hue

`adjust_saturation(...)` : A

`central_crop(...)` : Crop the

`convert_image_dtype(...)` :

`crop_and_resize(...)` : Extr

`crop_to_bounding_box(...)`

`decode_and_crop_jpeg(...)`

`decode_bmp(...)` : Decode the first frame of a BMP-encoded image to a uint8 tensor.

`decode_gif(...)` : Decode the first frame of a GIF-en

`decode_image(...)` : Convenience function for `deco`

`decode_jpeg(...)` : Decode

`decode_png(...)` : Decode a

`draw_bounding_boxes(...`

`encode_jpeg(...)` : JPEG-e

`encode_png(...)` : PNG-enc

`extract_glimpse(...)` : Ex

`extract_jpeg_shape(...)`

`flip_left_right(...)`

`flip_up_down(...)` : Fl

`grayscale_to_rgb(...`

`hsv_to_rgb(...)` : Con

`is_jpeg(...)` : Conven

`non_max_suppression(`

`pad_to_bounding_box(`

`per_image_standardiz`

`random_brightness(..`

`random_contrast(...)`

`random_flip_up_down(...)` : Randomly flips an image vertically (ups

`random_hue(...)` : Adjust the hue of an RGB image by a random facto

`random_saturation(...)` : Adjust the saturation of an RGB image by

`resize_area(...)` : Resize `images` to `size` using area interpolation

`resize_bicubic(...)` : Resize `images` to `size` using bicubic interp

`resize_bilinear(...)` : Resize `images` to `size` using bilinear inter

`resize_image_with_crop_or_pad(...)` : Crops and/or pads an imag

`resize_images(...)` : Resize `images` to `size` using the specified `m`

`resize_nearest_neighbor(...)` : Resize `images` to `size` using ne

`rgb_to_grayscale(...)` : Converts one or more images from RGB to

`rgb_to_hsv(...)` : Converts one or more images from RGB to HSV.

`rgb_to_yiq(...)` : Converts one or more images from RGB to YIQ.

`rgb_to_yuv(...)` : Converts one or more images from RGB to YUV.

`rot90(...)` : Rotate image(s) counter-clockwise by 90 degrees.

`sample_distorted_bounding_box(...)` : Generate a single randomly

Tensorflow | Convolutional layer

```
with tf.variable_scope('conv1') as scope:
  weights = tf.get_variable('weights',
                            shape=[3, 3, 3, 16],
                            dtype=tf.float32,
                            initializer=tf.truncated_normal_initializer(stddev=0.1, dtype=tf.float32))
  biases = tf.get_variable('biases',
                           shape=[16],
                           dtype=tf.float32,
                           initializer=tf.constant_initializer(0.1))
  conv = tf.nn.conv2d(images, weights, strides=[1, 1, 1, 1], padding='SAME')
  pre_activation = tf.nn.bias_add(conv, biases)
  conv1 = tf.nn.relu(pre_activation, name=scope.name)

# pool1
with tf.variable_scope('pooling1_lrn') as scope:
  pool1 = tf.nn.max_pool(conv1, ksize=[1, 3, 3, 1], strides=[1, 2, 2, 1],
                         padding='SAME', name='pooling1')


# conv2
with tf.variable_scope('conv2') as scope:
  weights = tf.get_variable('weights',
                            shape=[3, 3, 16, 16],
                            dtype=tf.float32,
                            initializer=tf.truncated_normal_initializer(stddev=0.1, dtype=tf.float32))
  biases = tf.get_variable('biases',
                           shape=[16],
                           dtype=tf.float32,
                           initializer=tf.constant_initializer(0.1))
  conv = tf.nn.conv2d(pool1, weights, strides=[1, 1, 1, 1], padding='SAME')
  pre_activation = tf.nn.bias_add(conv, biases)
  conv2 = tf.nn.relu(pre_activation, name='conv2')

# pool2 a
with tf.variable_scope('pooling2_lrn') as scope:
  pool2 = tf.nn.max_pool(conv2, ksize=[1, 3, 3, 1], strides=[1, 1, 1, 1],
                         padding='SAME', name='pooling2')
```

Tensorflow | Convolutional layer

```
# local3
with tf.variable_scope('local3') as scope:
  reshape = tf.reshape(pool2, shape=[batch_size, -1])
  dim = reshape.get_shape()[1].value
  weights = tf.get_variable('weights',
                            shape=[dim, 128],
                            dtype=tf.float32,
                            initializer=tf.truncated_normal_initializer(stddev=0.005, dtype=tf.float32))
  biases = tf.get_variable('biases',
                           shape=[128],
                           dtype=tf.float32,
                           initializer=tf.constant_initializer(0.1))
  local3 = tf.nn.relu(tf.matmul(reshape, weights)+biases, name=scope.name)

  # local4
with tf.variable_scope('local4') as scope:
  weights = tf.get_variable('weights',
                            shape=[128, 128],
                            dtype=tf.float32,
                            initializer=tf.truncated_normal_initializer(stddev=0.005, dtype=tf.float32))
  biases = tf.get_variable('biases',
                           shape=[128],
                           dtype=tf.float32,
                           initializer=tf.constant_initializer(0.1))
  local4 = tf.nn.relu(tf.matmul(local3, weights)+biases, name='local4')

# softmax
with tf.variable_scope('softmax_linear') as scope:
  weights = tf.get_variable('softmax_linear',
                            shape=[128, n_classes],
                            dtype=tf.float32,
                            initializer=tf.truncated_normal_initializer(stddev=0.005, dtype=tf.float32))
  biases = tf.get_variable('biases',
                           shape=[n_classes],
                           dtype=tf.float32,
                           initializer=tf.constant_initializer(0.1))
  softmax_linear = tf.add(tf.matmul(local4, weights), biases, name='softmax_linear')
```

Tensorflow | Custom dataset

```
def _run():
    train, train_label = get_files(data_path=data_path,datafolder=datafolder)

    train_batch, train_label_batch = get_batch(train,
                                               train_label,
                                               img_W,
                                               img_H,
                                               batch_size,
                                               batch_size*2)
    train_logits = inference(train_batch, batch_size, n_classes)
    train_loss = losses(train_logits, train_label_batch)
    train_op = trainning(train_loss, learning_rate)
    train__acc = evaluation(train_logits, train_label_batch)

    summary_op = tf.summary.merge_all()
    sess = tf.Session()
    train_writer = tf.summary.FileWriter(logs_train_dir, sess.graph)
    saver = tf.train.Saver()

    sess.run(tf.global_variables_initializer())
    coord = tf.train.Coordinator()
    threads = tf.train.start_queue_runners(sess=sess, coord=coord)

    try:
        for step in np.arange(10000):
            if coord.should_stop():
                break
            _, tra_loss, tra_acc = sess.run([train_op, train_loss, train__acc])
```

Tensorflow | 요약

- ConvNet 기반 MNIST 분석
- 나만의 데이터를 이용한 데이터 분석
- 다양한 방식의 코드 분석

To be continued..