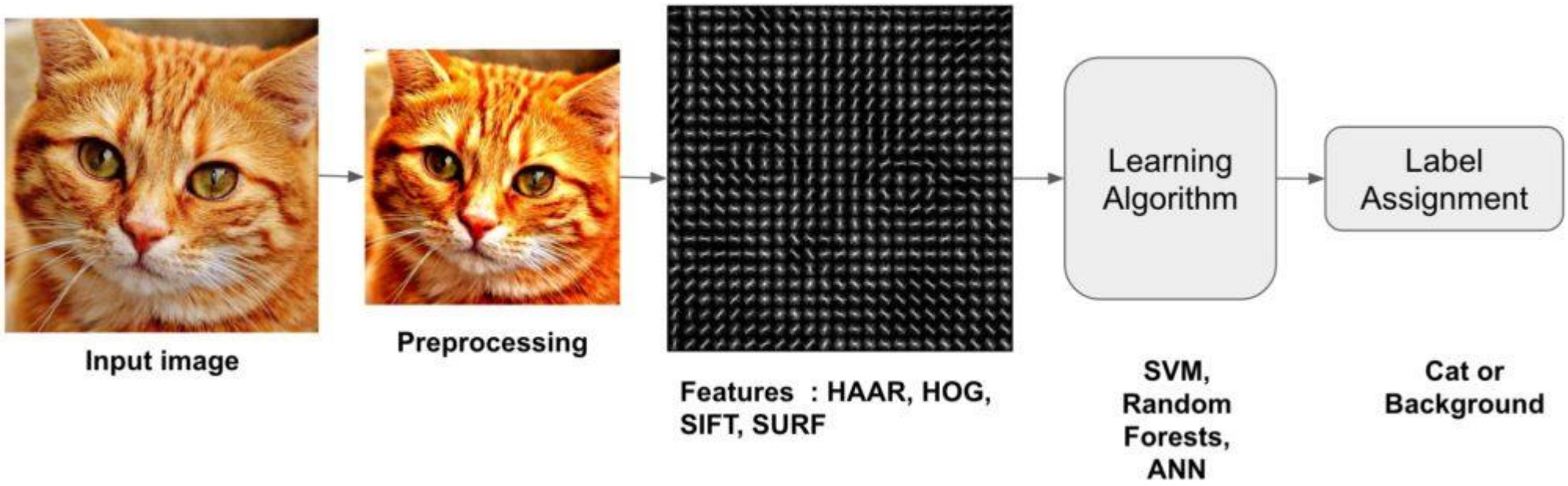
CNN Image Classification

PIRL, POSTECH

Hanul Roh

Recap: Traditional Image Classification

- A core task in Computer Vision



Data -> Feature extraction -> Classification

Recap: Traditional Image Classification

- A core task in Computer Vision

The classification performance depends on good feature design and extraction

Data -> Feature extraction -> Classification

Convolution Neural Network

- The building of an **integrated feature extraction and classification**
- **Learning to extract features** from given training datasets
  - Receptive field (Locality)
  - Parameter sharing
  - Convolution + Subsampling -> FC layers -> Output

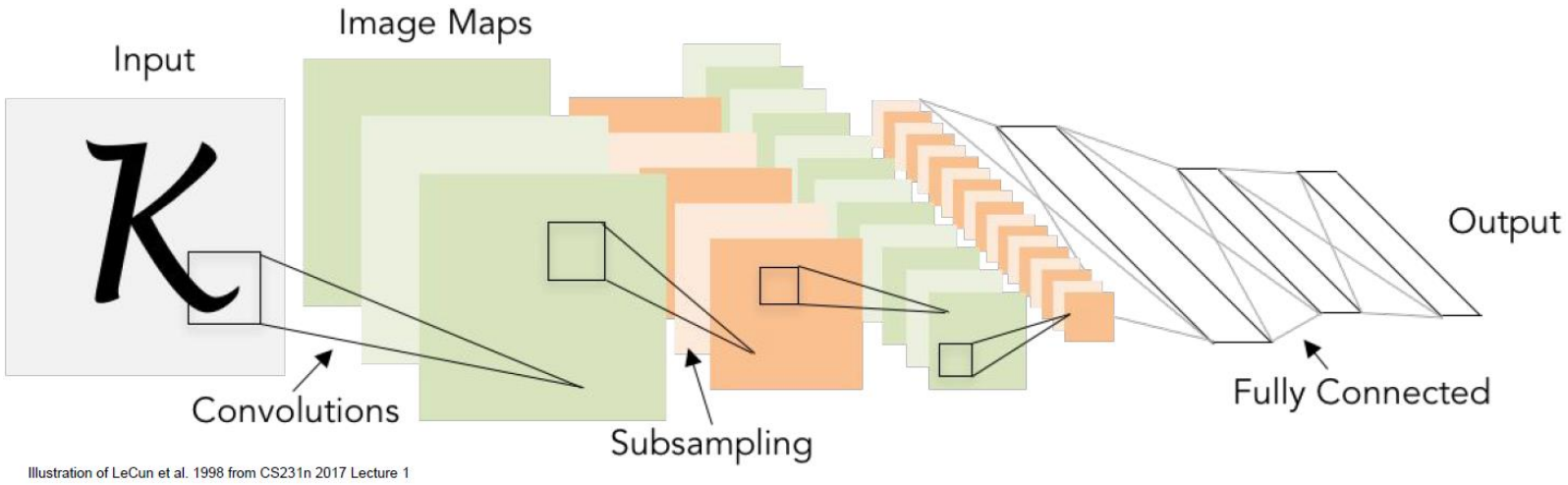


Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

ConvNet Architecture

Convolution Layer

Compute the output of neurons that are connected to local regions

Activation Layer

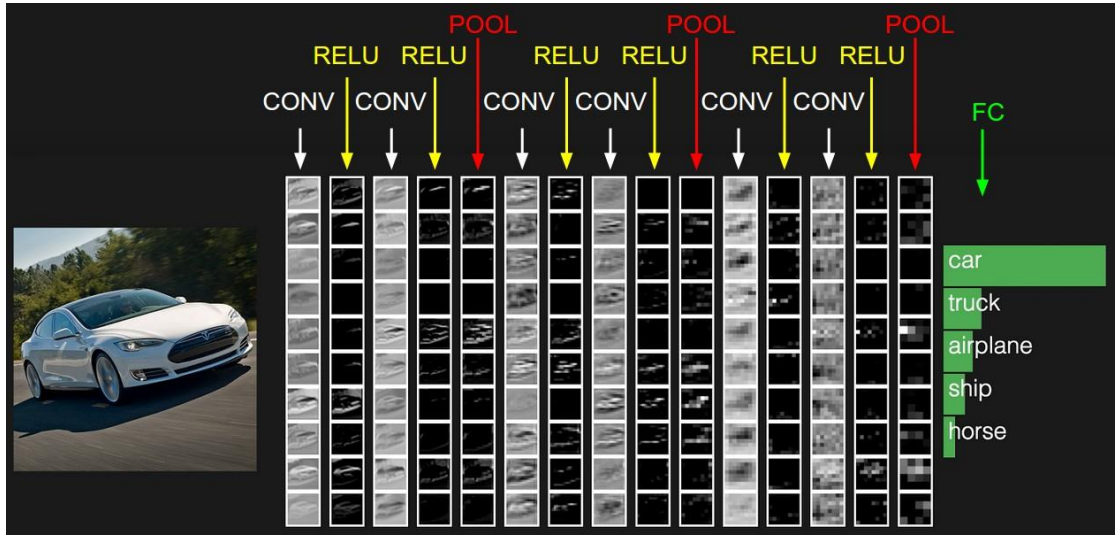
Elementwise activation function max(0, f(x)) : nonlinearity

Pooling Layer

Downsampling operation 2x2 input -> maximum value

Fully-connected Layer

Compute class scores resulting in volume size [1x1xN]



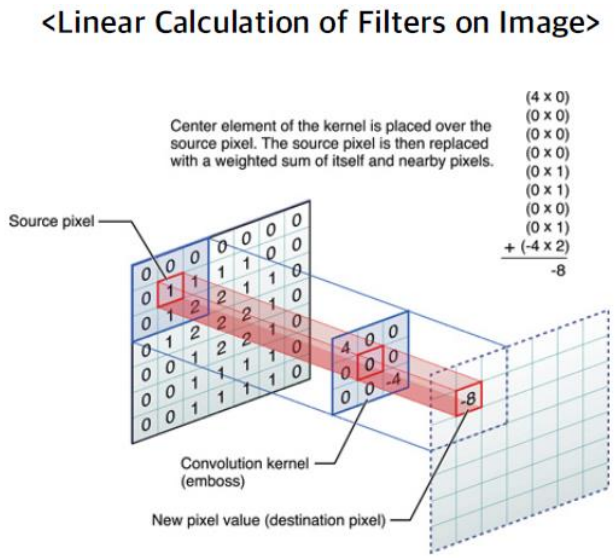
Convolution Operation

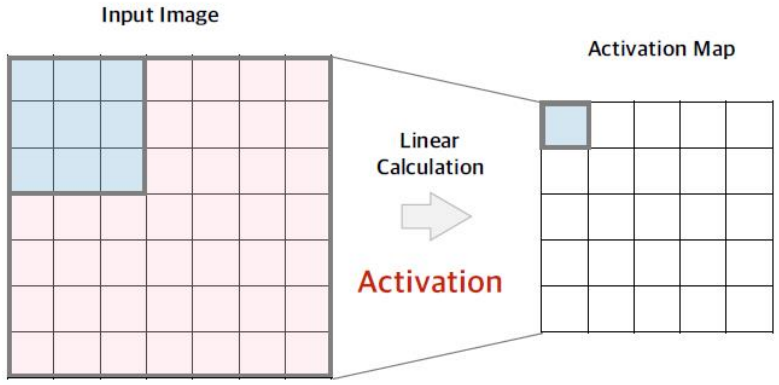
Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Convolution Layer

- Convolution layer demo here:

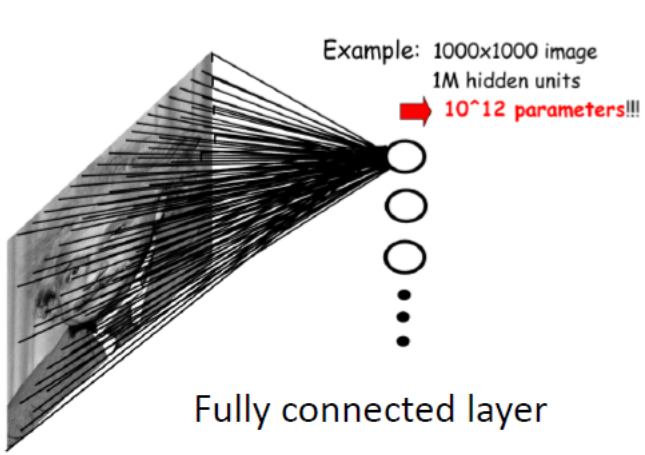
http://cs231n.github.io/convolutional-networks/#layers



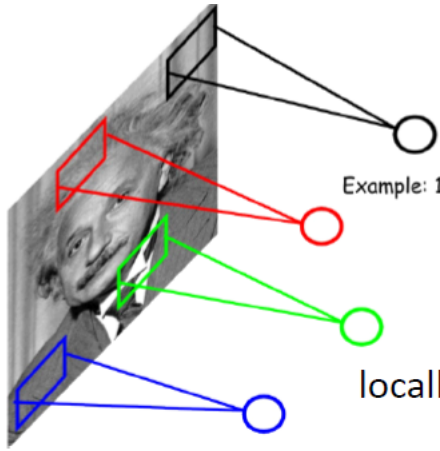


(Recall) In this example,
only 9 value of weights are used
to make activation map! —> Weight Sharing

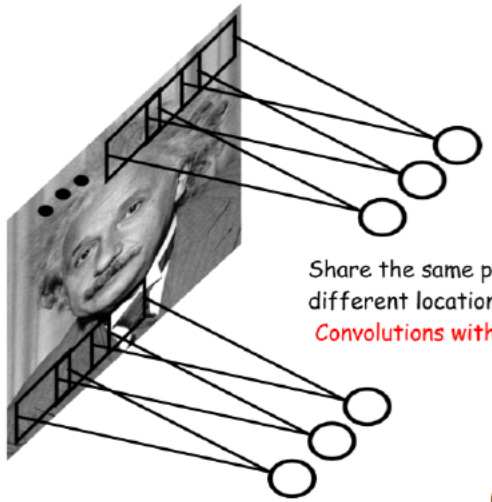
Convolution Layer

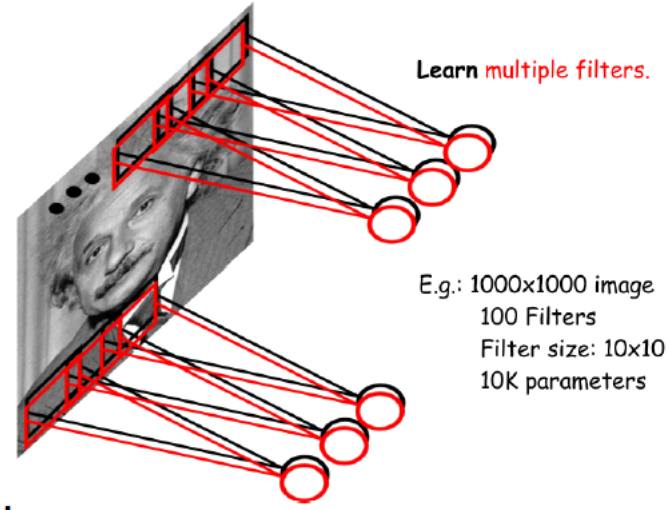


Fully connected layer



locally connected layer





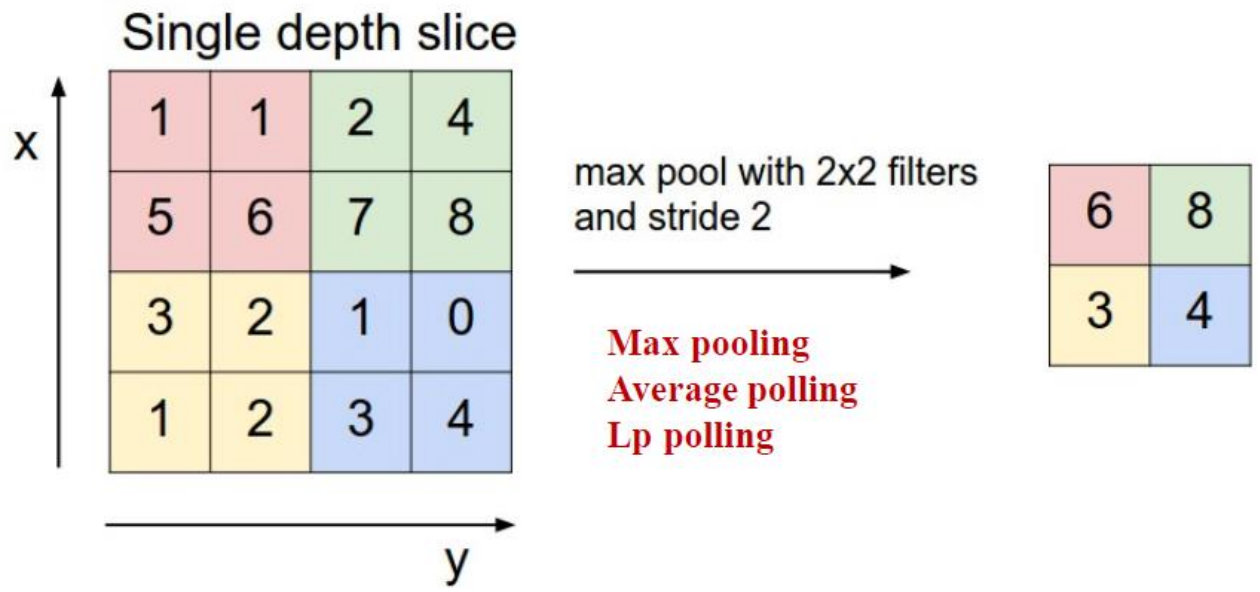
Convolutional layer

Convolution Layer

- Local Connection
  - Restricting the network architecture through the use of local connections known as receptive fields
- Parameter Sharing
  - All neurons in a feature share the same weights
  - Reduce the number of free parameters in the input images

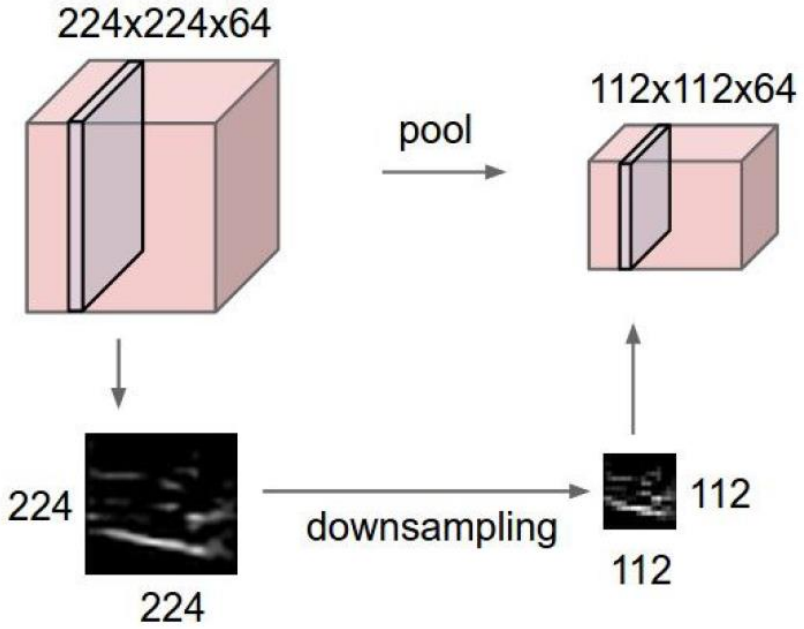
Pooling Layer

- Reduce previous feature map size from [2x2] -> 1 value
  - Mean pooling, **Max pooling**, and so on
- Make the representations smaller and more manageable
- Operates over each activation map **independently**



Pooling Layer

- Reduce previous feature map size from [2x2] -> 1 value
  - Mean pooling, **Max pooling**, and so on
- Make the representations smaller and more manageable
- Operates over each activation map **independently**



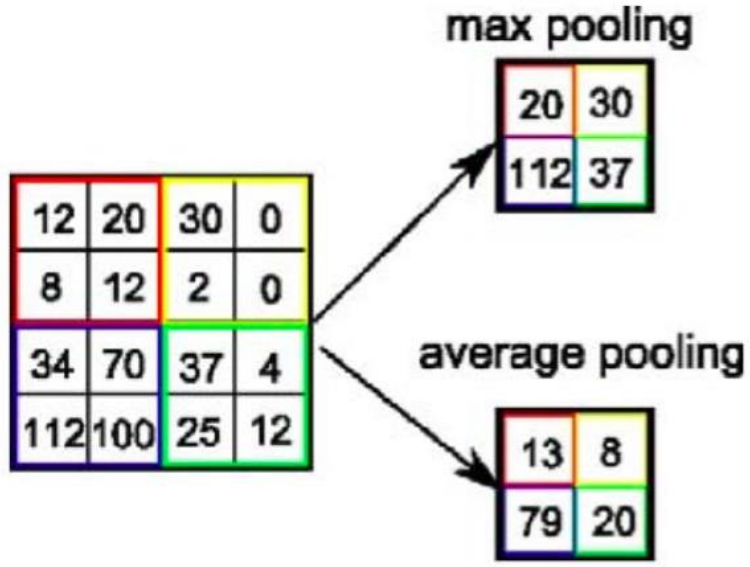
Pooling Layer

Subsampling

- Overlapping vs. non-overlapping
- Max pooling vs. average pooling

Translation invariance

- Pooling을 하면 작은 변화에는 둔감해짐
- 따라서, 이에 강인한 특징을 생성할 수 있음



Fully-connected Layer

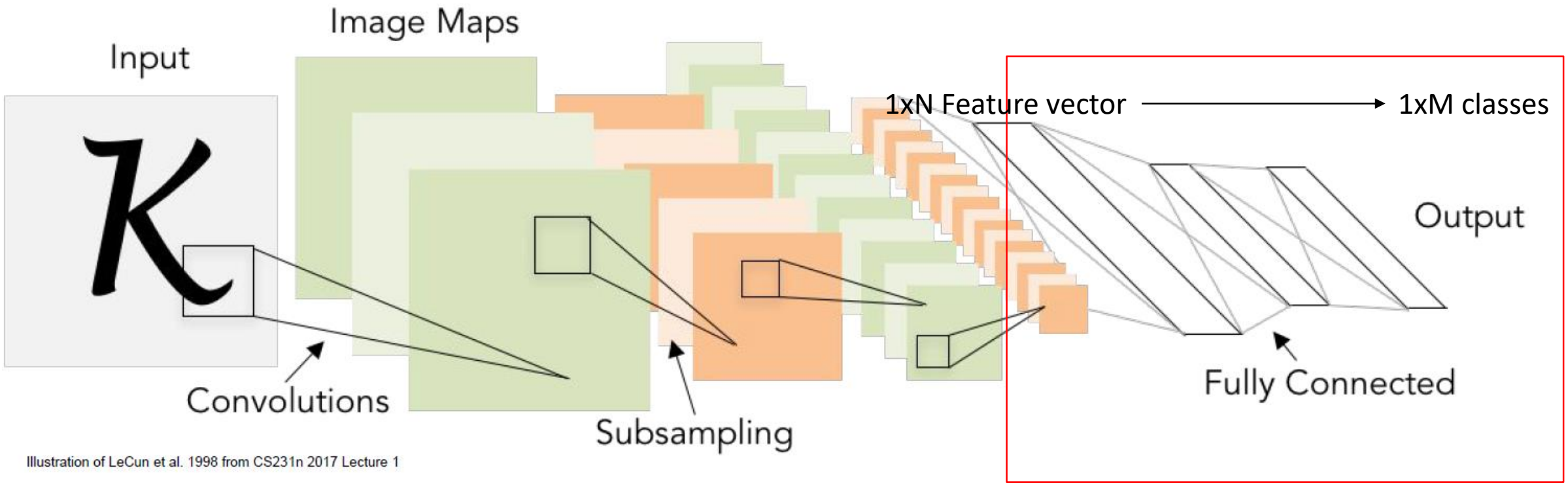
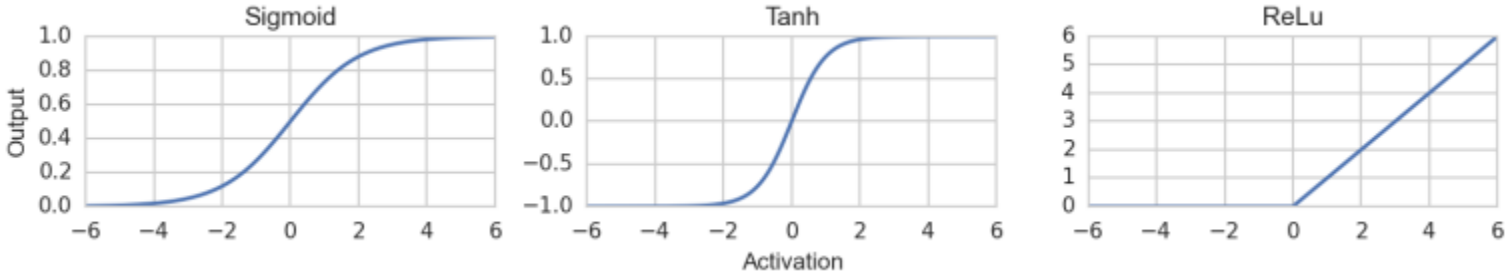
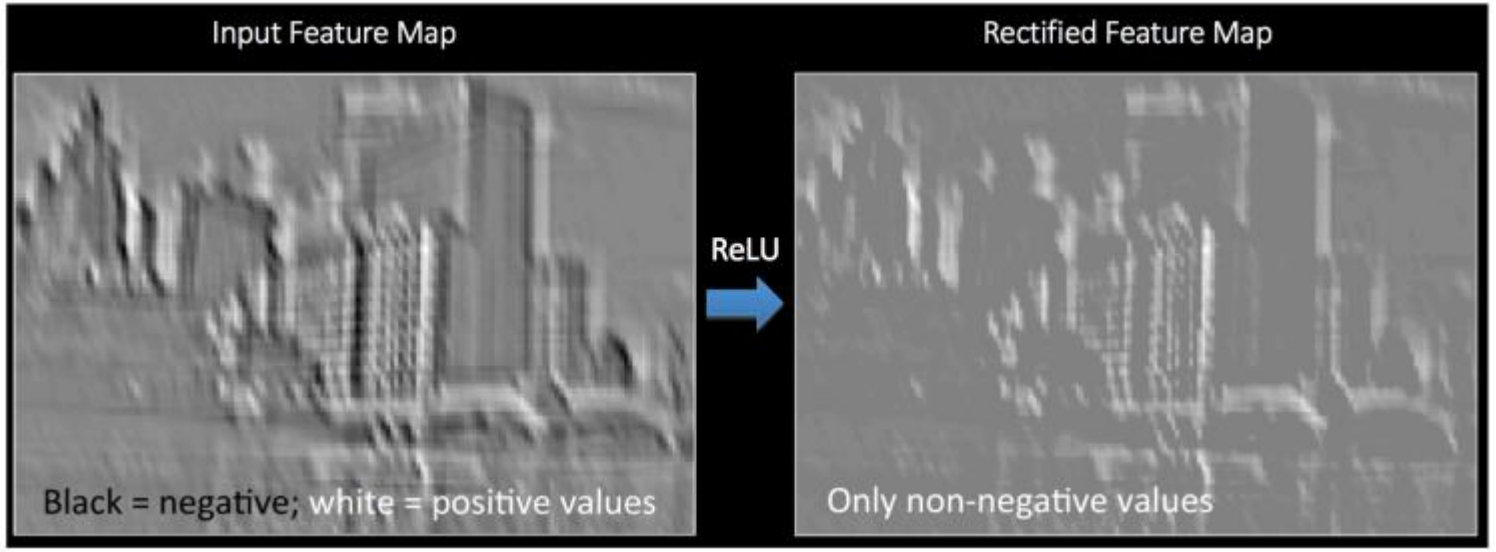


Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

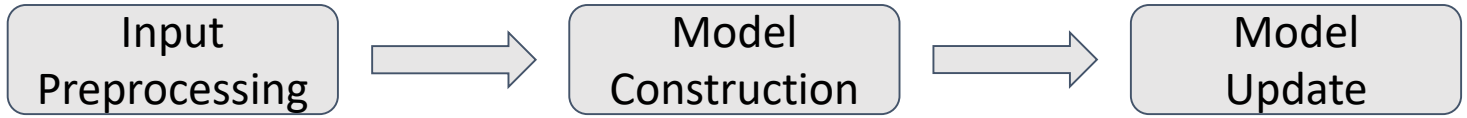
Activation Function

- Gradient vanishing problem -> ReLu!





CNN Image Classification



Data loader

- Load batch data
- Preprocess the data
  - normalization
  - random crop

Model Evaluation

Measure the accuracy

- # of correct examples / # of total examples

Forward path

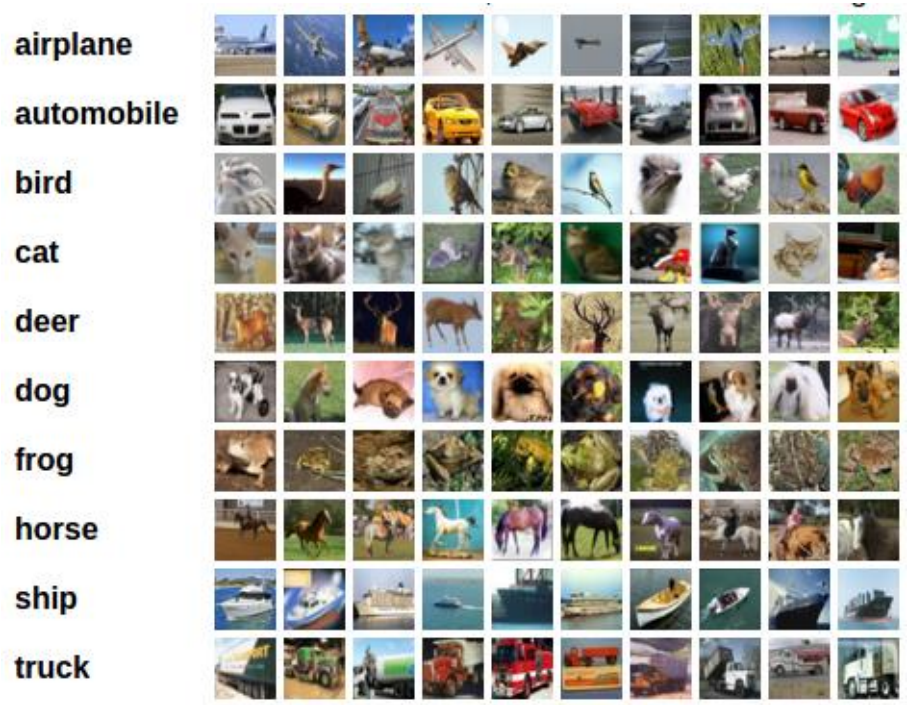
- Build input placeholders
- Build network
  - convolution
  - linear (fc)
  - pooling
  - dropout
  - activation function
- Build loss
  - cross-entropy

Backward path

- Compute gradient
  - gradient clip
- Update model
  - SGD, Adam, Ad agrad, etc
  - Weight decay

Input Preprocessing

- CIFAR-10 Dataset
  - **10 classes** - airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck
  - **32x32 color images** with 6,000 images per class (50,000 training / 10,000 test)
  - **1x3072 vector** shape per each image

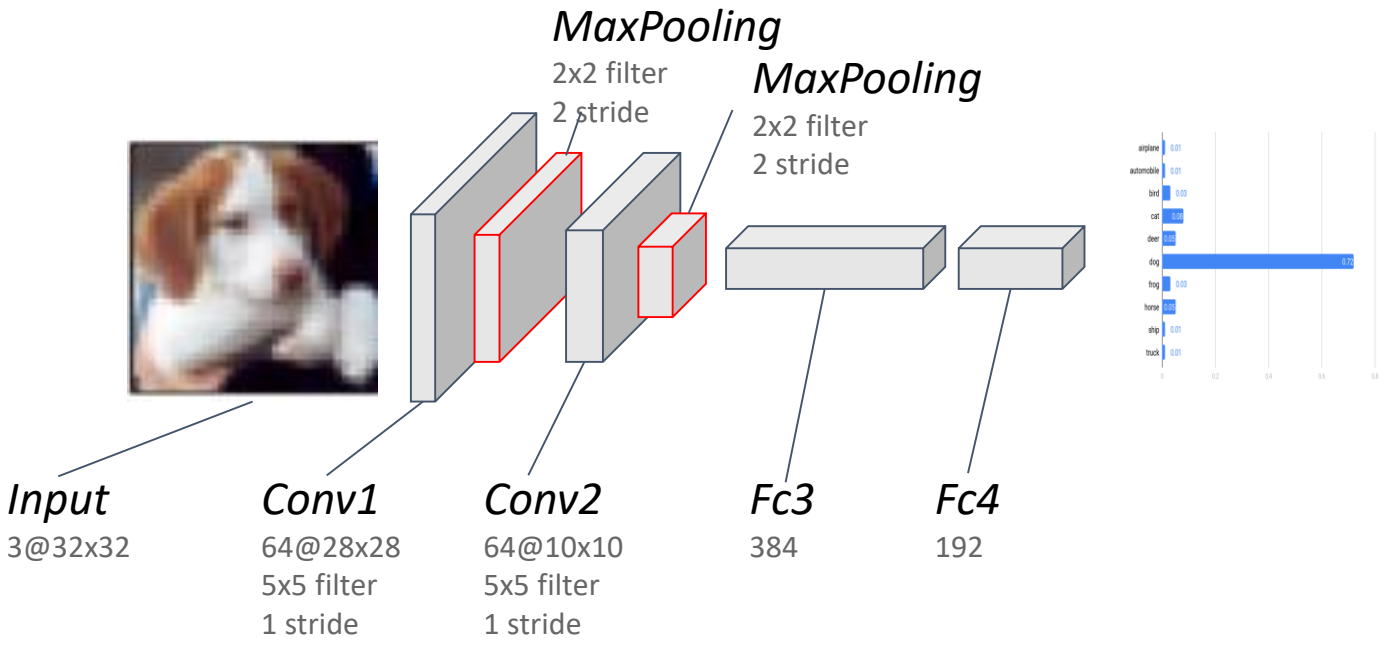


Input Preprocessing

- Data loader (cifar10_loader.py) does
  - Reshape a given vector of 3,072 size into a tensor of 3x32x32 size (RGB channels, width, height)
  - Transpose (channel, width, height) into (width, height, channel)
  - Load data of batch size using get_batch()

Model Construction

- In this practice,
  - 2 convolution layers + 2 fully connected layers + classification layer



Model Construction

- How to implement the layers?

Convolutional Layer - tf.nn.conv2d()

```
# Create variables
filter = tf.get_variable("weights", [filer_size,filter_size, inp_dim, out_dim])
bias = tf.get_variable("biases", [out_dim])

# Compute convolution
conv = tf.nn.conv2d(input, filter)
conv = tf.nn.bias_add(conv, bias)
```

Activation function (relu, tanh)?

```
conv = tf.nn.relu(conv)
conv = tf.tanh(conv)
```

Max pooling?

```
# kernel size 2 and stride 2
pool = tf.nn.max_pool(conv, ksize=[1,2,2,1], strides=[1,2,2,1])
```

Fully Connected Layer - tf.matmul()

```
# Create variables
weight = tf.get_variable("weights", [inp_dim, out_dim])
bias = tf.get_variable("biases", [out_dim])

# Compute convolution
fc = tf.matmul(input, weight)
fc = tf.nn.bias_add(fc, bias)
```

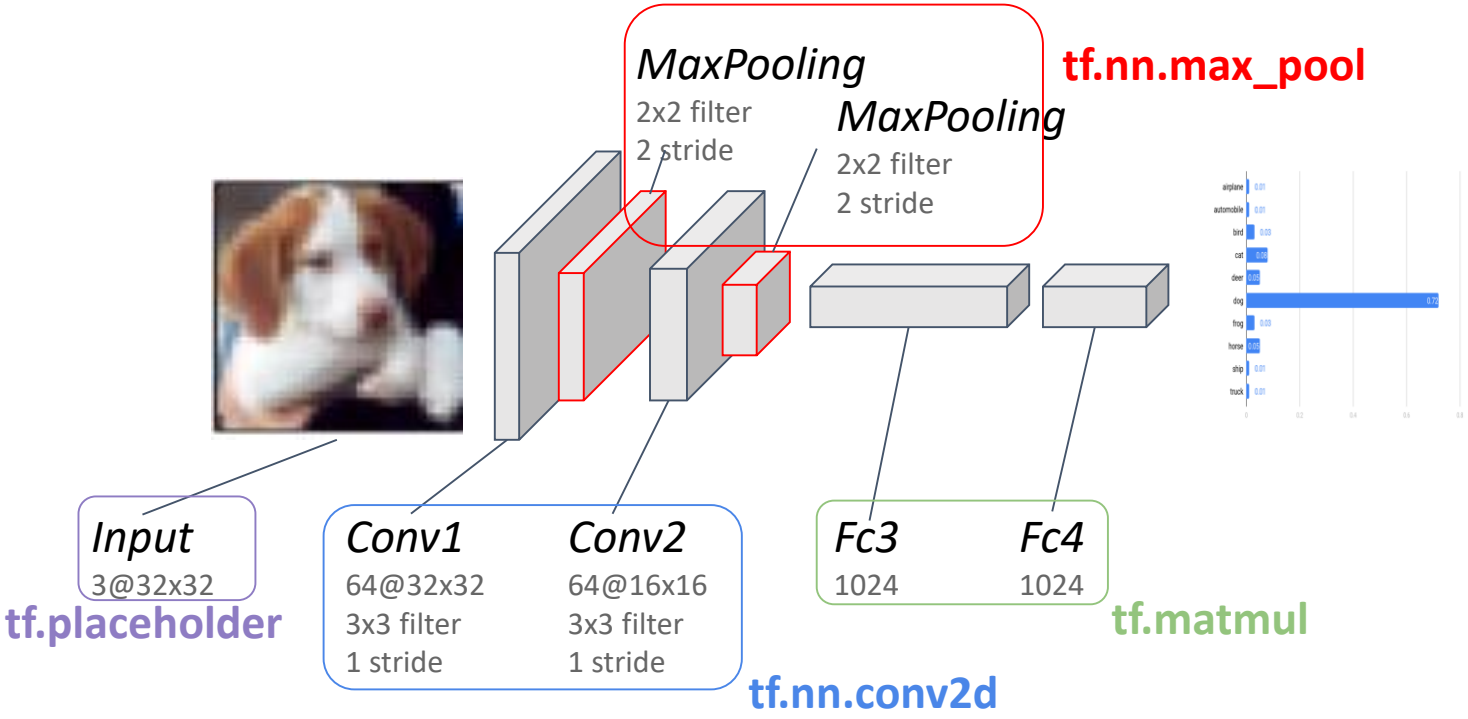
```
fc = tf.nn.relu(fc)
fc = tf.tanh(fc)
```

Dropout?

```
drop_fc = tf.nn.dropout(fc, keep_prob)
```

Model Construction

- How to implement the layers in TensorFlow?



Model Construction

- We should always define variables manually? **No**
- There are three libraries based on TensorFlow which are **simple and concise**!

```
# Create variables
filter = tf.get_variable("weights", [filer_size,filter_size,
inp_dim, out_dim])
bias = tf.get_variable("biases", [out_dim])

# Compute convolution
conv = tf.nn.conv2d(input, filter)
conv = tf.nn.bias_add(conv, bias)

# Apply activation function
conv = tf.nn.relu(conv)
conv = tf.tanh(conv)

# Apply dropout
drop_conv = tf.nn.dropout(conv, keep_prob)
```

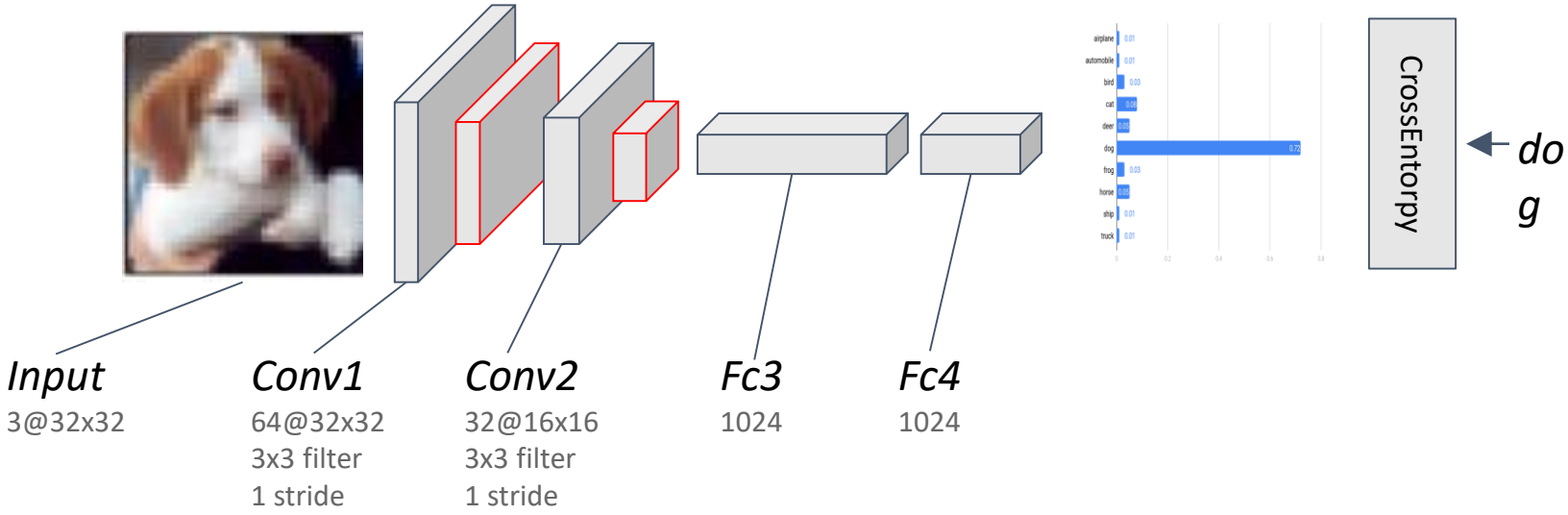
wrapper fun
ction

1. TFLearn
   - tf.learn.layers.conv.conv_2d
     ()
2. Keras
   - tf.contrib.keras.layers.Conv
     2D()
3. TF-Slim
   - tf.contrib.slim.conv2d()

Note that we do not cover above libraries in this course

Model Update - Learning

- Define loss function (cross-entropy loss)
- Define optimizer (Adam)



Model Update - Learning

- How to define loss function and optimizer in TensorFlow?

Loss Function

```
# Compute cross-entropy loss
cross_entropy =     tf.nn.sparse_softmax_cross_entropy_with_logits(labels, logits)
loss = tf.reduce_mean(cross_entropy)
```

- cross-entropy with softmax
  - Softmax: $S(y_i) = \frac{e^{y_i}}{\sum_{k=1} e^{y_k}}$ cross-entropy: $H_{y'}(y) = -\sum_i y'_i \log(\text{softmax}(y_i))$

Now, we can train model with following line

```
sess = tf.Session()
sess.run(fetches=train_op, feed_dict=inputs)
```

Optimizer

```
# Create optimizer
optimizer = tf.train.AdamOptimizer(learning_rate)

# Two methods exists
# 1. compute gradients and update model
grads_vars = optimizer.compute_gradients(loss)
train_op = optimizer.apply_gradients(grads_vars)

# 2. Update model with one function
train_op = optimizer.minimize(loss)
```

* other optimizers?
  - tf.train.GradientDescentOptimizer()
  - tf.train.RMSPropOptimizer()
  - radOptimizer()

Misc

How to show the variables defined in the network (graph)?

```
for var in tf.global_variables():
    print(var)
```

* global_variables() returns the variables that are created by tf.Variable() or tf.get_variable()

* We can show operations using tf.get_default_graph().get_operations()

```
Tensor("conv1/weights/read:0", shape=(5, 5, 3, 64), dtype=float32)
Tensor("conv1/biases/read:0", shape=(64,), dtype=float32)
Tensor("conv2/weights/read:0", shape=(5, 5, 64, 64), dtype=float32)
Tensor("conv2/biases/read:0", shape=(64,), dtype=float32)
Tensor("fc3/weights/read:0", shape=(1600, 384), dtype=float32)
Tensor("fc3/biases/read:0", shape=(384,), dtype=float32)
Tensor("fc4/weights/read:0", shape=(384, 192), dtype=float32)
Tensor("fc4/biases/read:0", shape=(192,), dtype=float32)
Tensor("fc5/weights/read:0", shape=(192, 10), dtype=float32)
Tensor("fc5/biases/read:0", shape=(10,), dtype=float32)
```

How to access weight parameters?

```
sess = tf.Session()
g = tf.get_default_graph()
w_tensor = g.get_tensor_by_name("fc5/weights/read:0")
w = w_tensor.eval(session=sess)
print(w.shape)
print(w[0, :10])
```

```
# create session
# get the current graph model
# obtain a tensor variable using the name
# get the tensor by running session
```

```
(192, 10)
[ 0.00417436  0.01035099  0.00198316  0.00643856  0.00888029  0.00903297
 -0.0122347  -0.01635222 -0.01234471  0.00342033]
[ 0.00417436  0.01035099  0.00198316  0.00643856  0.00888029  0.00903297
 -0.0122347  -0.01635222 -0.01234471  0.00342033]
```

Check the code image_classification.ipynb

- Train the model and evaluate it

Exercises

- Implement simple CNN architecture and learn CIFAR-10 datasets

SimpleCNN
9 weight layers
Input: CIFAR-10 32x32 RGB image
Conv3x3-64 Conv3x3-64
Maxpool
Conv3x3-128 Conv3x3-128 Conv3x3-128
Maxpool
FC-1024
FC-1024
FC-10
Soft-max