MACHINE LEARNING

Convolutional Neural Networks:
Deep Learning with Images

Contact Information

Instructor: Qi Hao

E-mail: hao.q@sustc.edu.cn

Office: Nanshan iPark A7 Room 906

Office Hours: M 2:00-4:00pm

Available other times by appointment or the open door policy

Office Phone: (0755) 8801-8537

QQ: 463715202 机器学习2018

Web:

http://hqlab.sustc.science/teaching/

Some cool projects

IMAGENET Large Scale Visual Recognition Challenge

Steel drum

The Image Classification Challenge:

1,000 object classes

1,431,167 images

Output:
Scale
T-shirt
<u>Steel drum</u>
Drumstick
Mud turtle

✔

Output:
Scale
T-shirt
Giant panda
Drumstick
Mud turtle

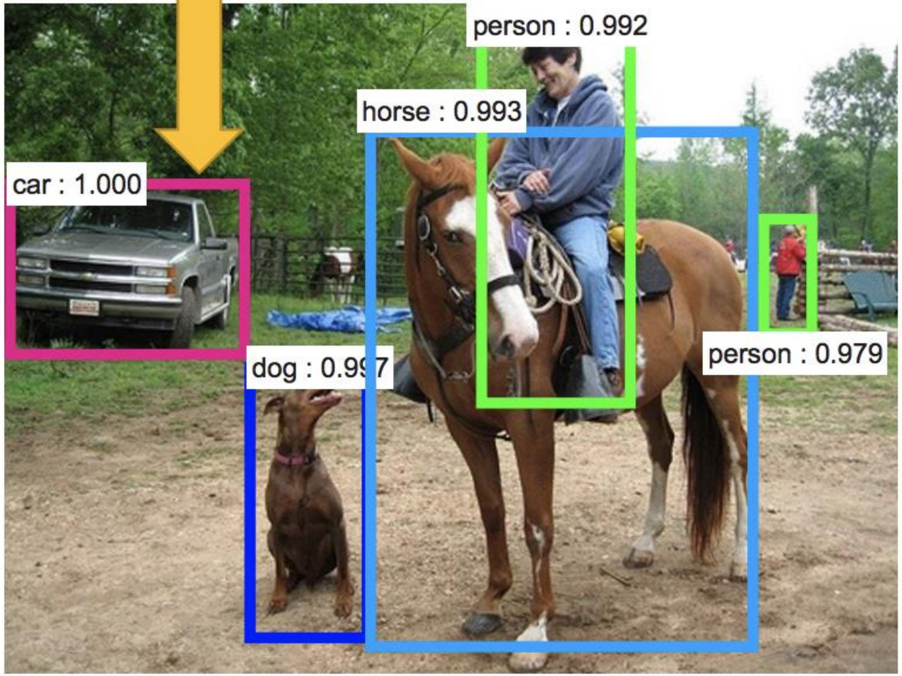
✗

Russakovsky et al. arXiv, 2014

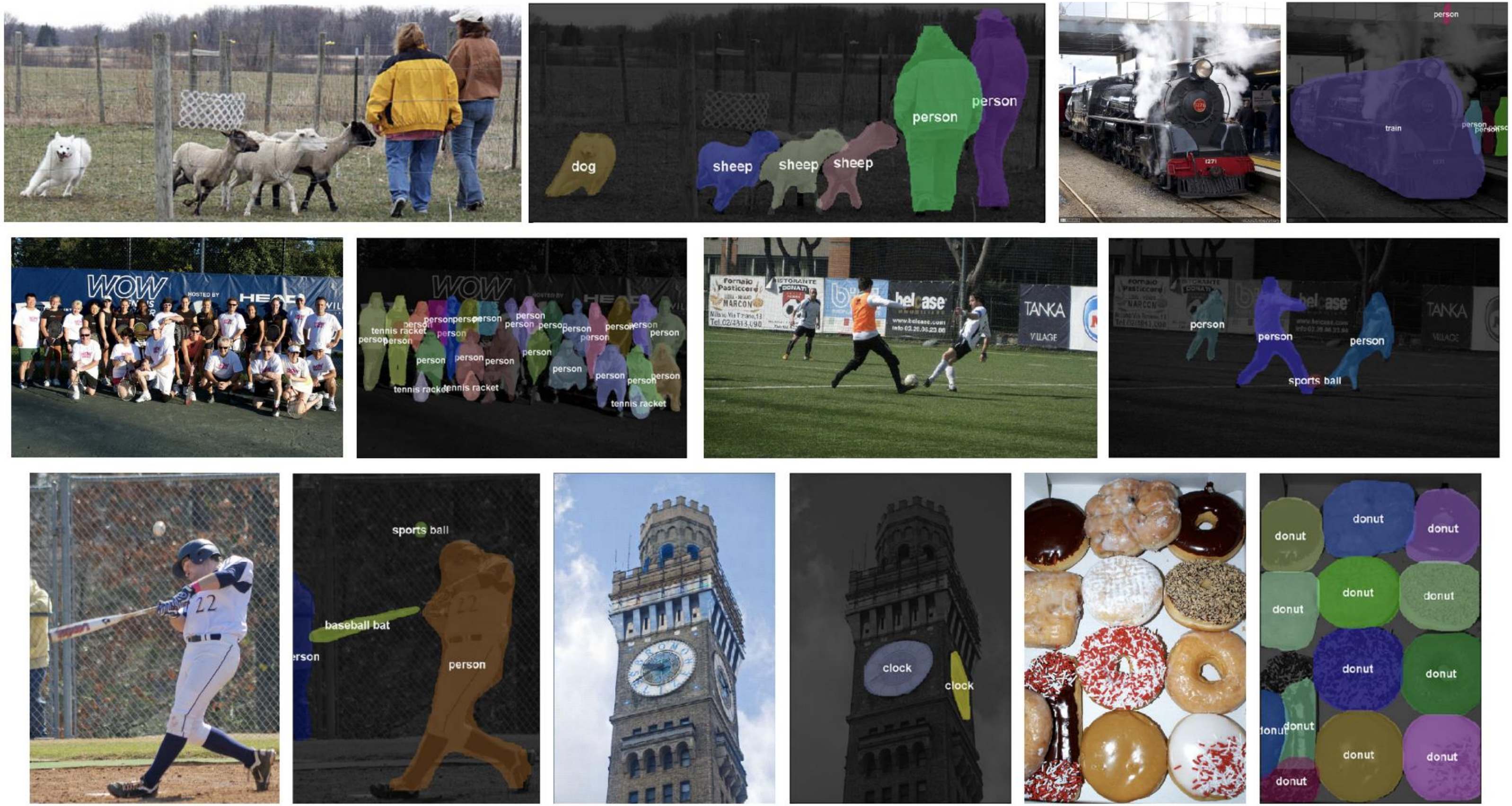
Object Detection = What, and Where

Localization
Where?

Recognition
What?

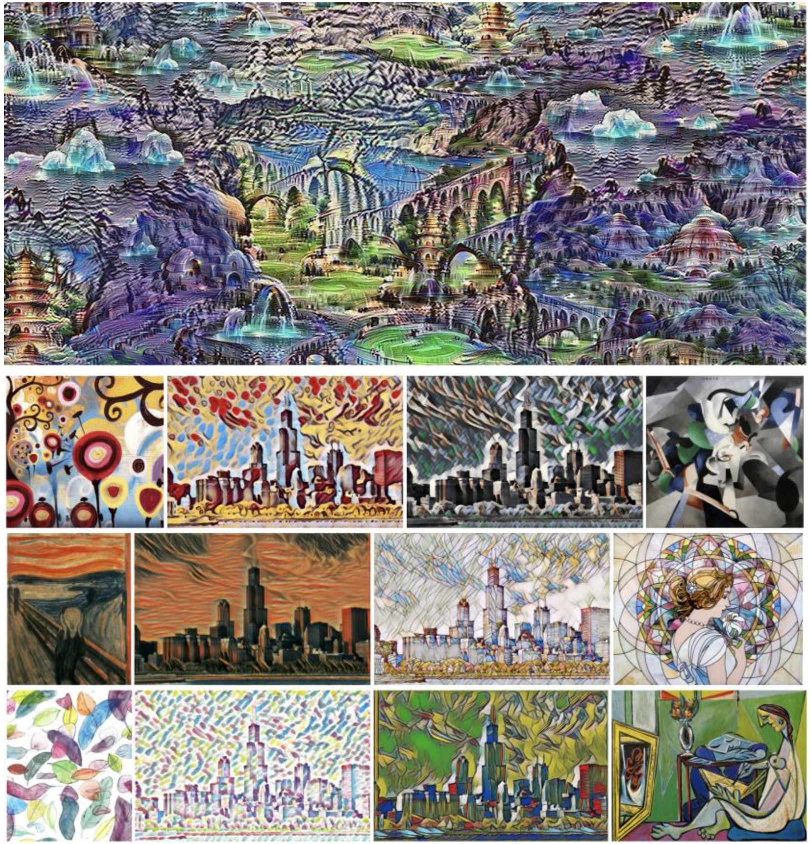


Object Segmentation



Pose Estimation

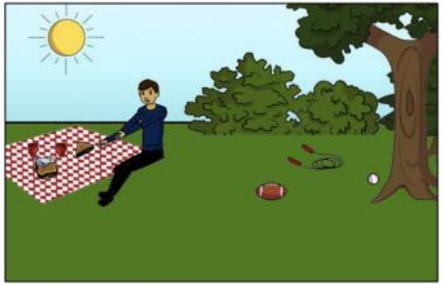
Art generation



Visual Question Answering

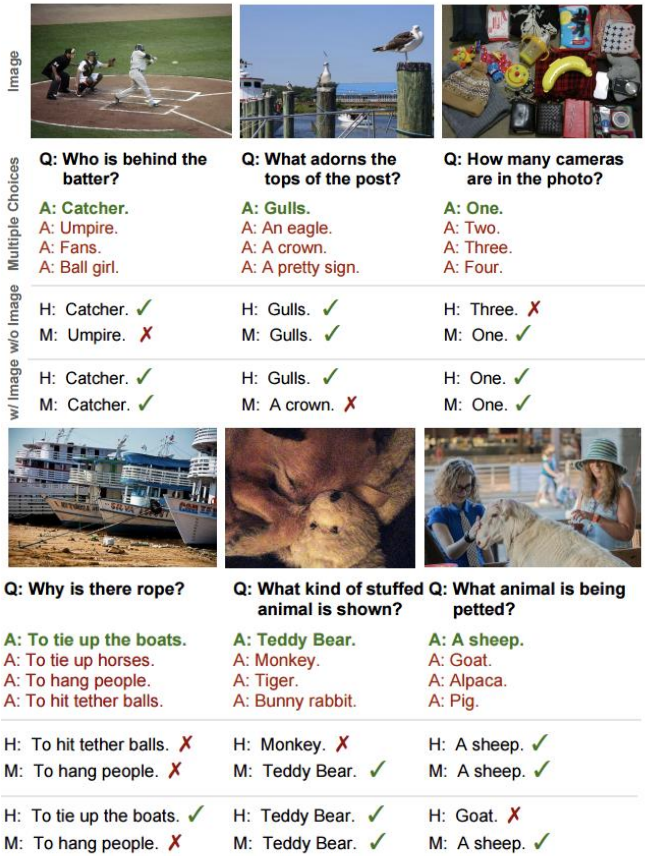
What color are her eyes?
What is the mustache made of?

How many slices of pizza are there?
Is this a vegetarian pizza?



Is this person expecting company?
What is just under the tree?

Does it appear to be rainy?
Does this person have 20/20 vision?



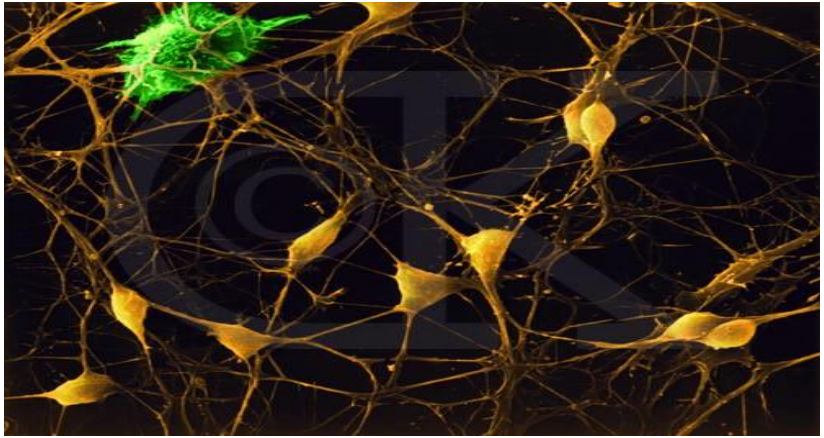
DNN： the tool to build cool projects like those!

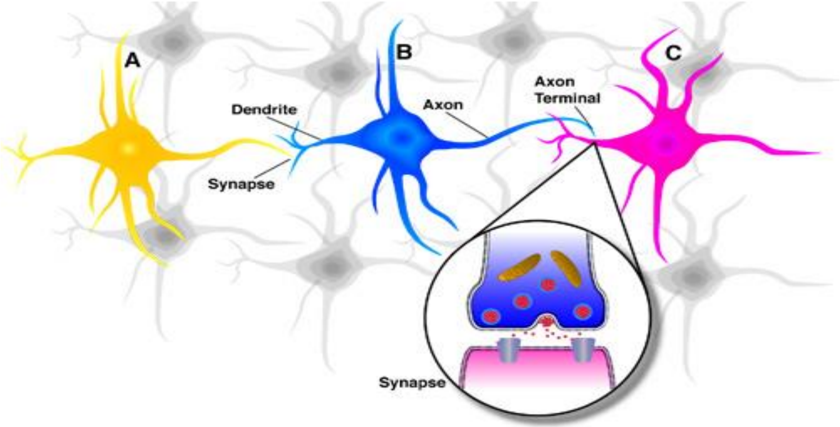
What are connectionist neural networks?

- Connectionism refers to a computer modeling approach to computation that is loosely based upon the architecture of the brain.
- Many different models, but all include:
  - Multiple, individual “nodes” or “units” that operate at the same time (in parallel)
  - A network that connects the nodes together
  - Information is stored in a distributed fashion among the links that connect the nodes
  - Learning can occur with gradual changes in connection strength

Neurons in the Brain

- Although heterogeneous, at a low level the brain is composed of neurons
  - A neuron receives input from other neurons (generally thousands) from its synapses
  - Inputs are approximately summed
  - When the input exceeds a threshold the neuron sends an electrical spike that travels that travels from the body, down the axon, to the next neuron(s)





The Neuron

- The neuron is the basic information processing unit of a NN. It consists of:

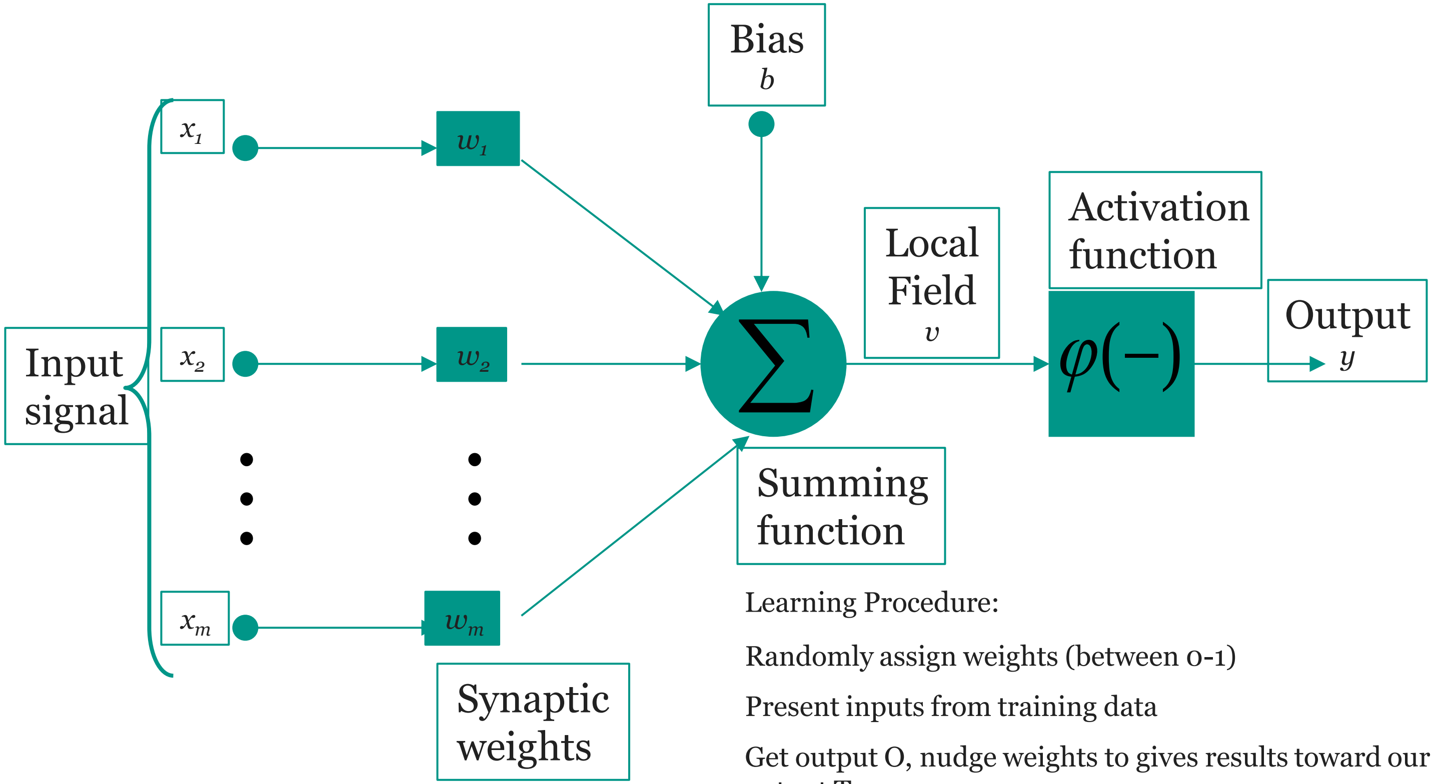
    - A set of synapses or connecting links, each link characterized by a weight: $W_1, W_2, \ldots, W_m$

    - An adder function (linear combiner) which computes the weighted sum of the inputs: $$\mathbf{u} = \sum_{j=1}^{\mathbf{m}} \mathbf{w_j x_j}$$

    - Activation function (squashing function) $\boxed{\varphi}$ for limiting the amplitude of the output of the neuron.

$$y = \varphi(u + b)$$

The Neuron



Learning Procedure:

Randomly assign weights (between 0-1)

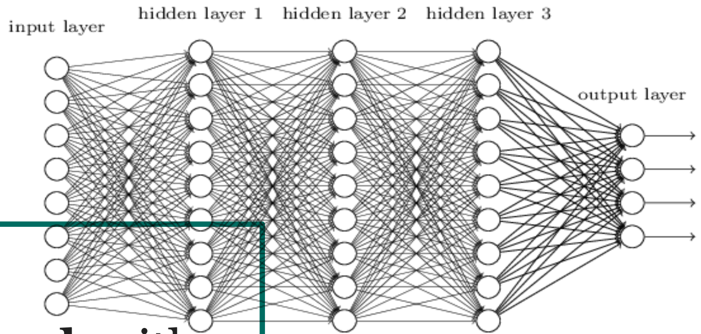
Present inputs from training data

Get output O, nudge weights to gives results toward our desired output T

Repeat; stop when no errors, or enough epochs completed

So, 1. **what exactly is deep learning** ?

And, 2. **why is it generally better** than other methods on image, speech and certain other types of data?



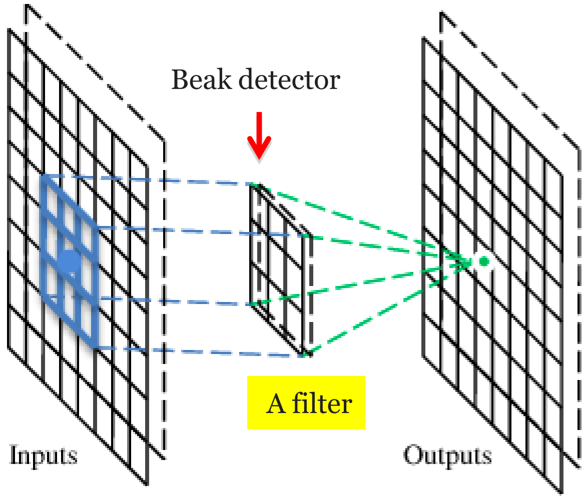
The short answers

- **‘Deep Learning’ means** using a **neural network** with **<u>several layers of nodes</u>** between input and output

- **the series of layers between input & output do feature identification and processing in a series of stages, just as our brains seem to.**

A convolutional layer

A CNN is a neural network with some convolutional layers (and some other layers). A convolutional layer has a number of filters that does convolutional operation.

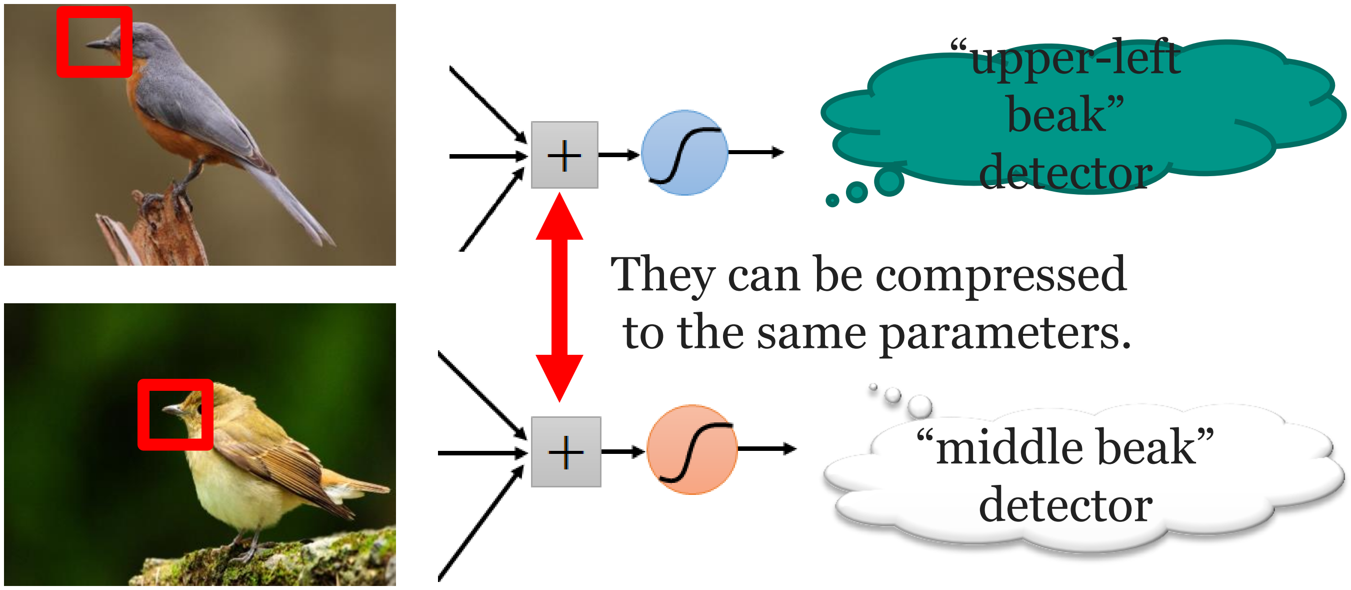


Consider learning an image:

- Some patterns are much smaller than the whole image

Can represent a small region with fewer parameters

Same pattern appears in different places:
They can be compressed!
What about training a lot of such “small” detectors
and each detector must “move around”.



Convolution

These are the network parameters to be learned.

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

⋮ ⋮

Each filter detects a small pattern (3 x 3).

Convolution

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

Dot product

3 -1

Convolution

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

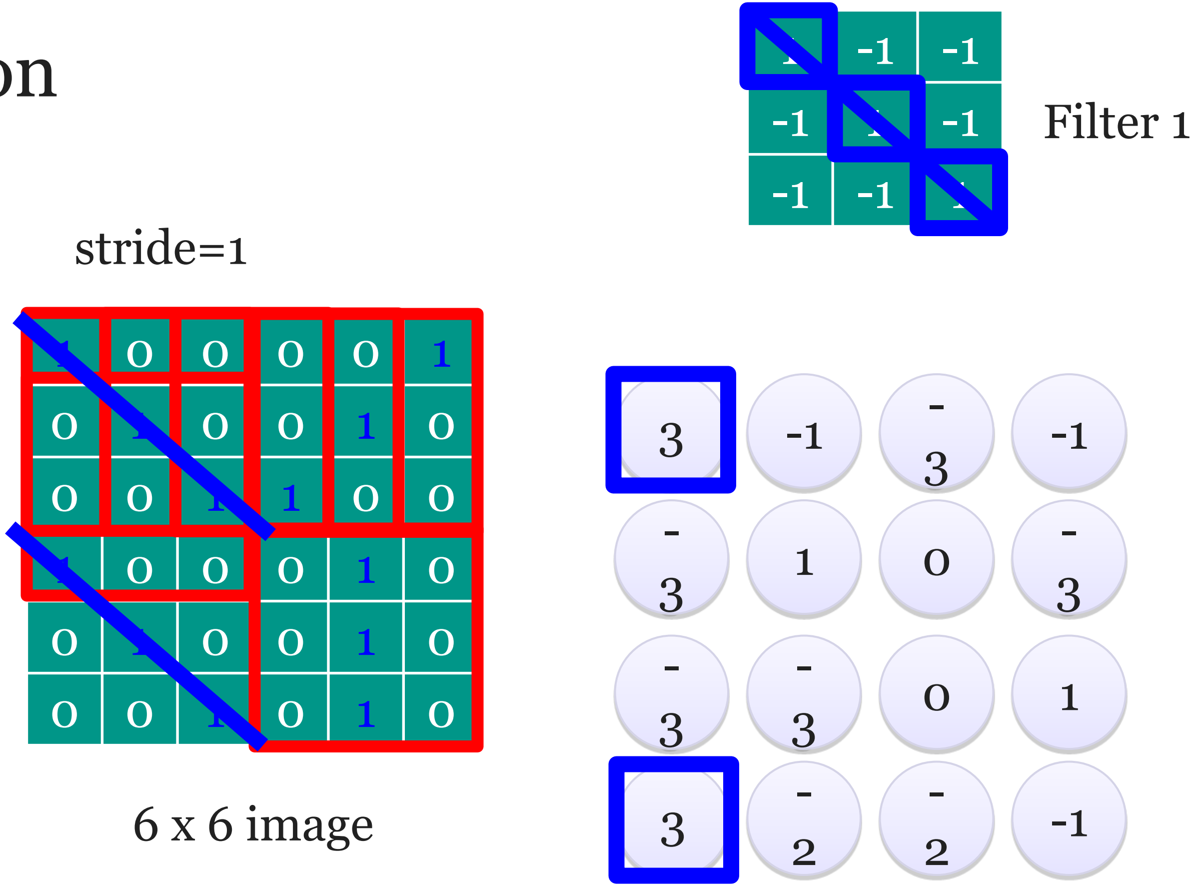
If stride=2

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

3 -3

Convolution



Convolution

-1	1	-1
-1	1	-1
-1	1	-1

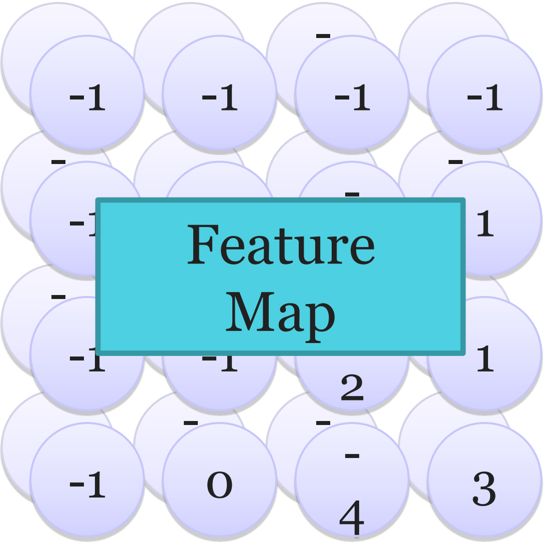
Filter 2

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

Repeat this for each filter



Two 4 x 4 images
Forming 2 x 4 x 4 matrix

Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

Common settings:

K = (powers of 2, e.g. 32, 64, 128, 512)

- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 5, S = 2, P = ? (whatever fits)
- F = 1, S = 1, P = 0

tf.layers.conv2d

```
conv2d(
    inputs,
    filters,
    kernel_size,
    strides=(1, 1),
    padding='valid',
    data_format='channels_last',
    dilation_rate=(1, 1),
    activation=None,
    use_bias=True,
    kernel_initializer=None,
    bias_initializer=tf.zeros_initializer(),
    kernel_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    bias_constraint=None,
    trainable=True,
    name=None,
    reuse=None
)
```

- `inputs`: Tensor input.
- `filters`: Integer, the dimensionality of the output space (i.e. the number of filters in the convolution).
- `kernel_size`: An integer or tuple/list of 2 integers, specifying the height and width of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions.
- `strides`: An integer or tuple/list of 2 integers, specifying the strides of the convolution along the height and width. Can be a single integer to specify the same value for all spatial dimensions. Specifying any stride value != 1 is incompatible with specifying any `dilation_rate` value != 1.
- `padding`: One of `"valid"` or `"same"` (case-insensitive).

Defined in `tensorflow/python/layers/convolutional.py`.

Functional interface for the 2D convolution layer.

This layer creates a convolution kernel that is convolved (actually cross-correlated) with the layer input to produce a tensor of outputs. If `use_bias` is True (and a `bias_initializer` is provided), a bias vector is created and added to the outputs. Finally, if `activation` is not `None`, it is applied to the outputs as well.

Convolution layer in tensorflow

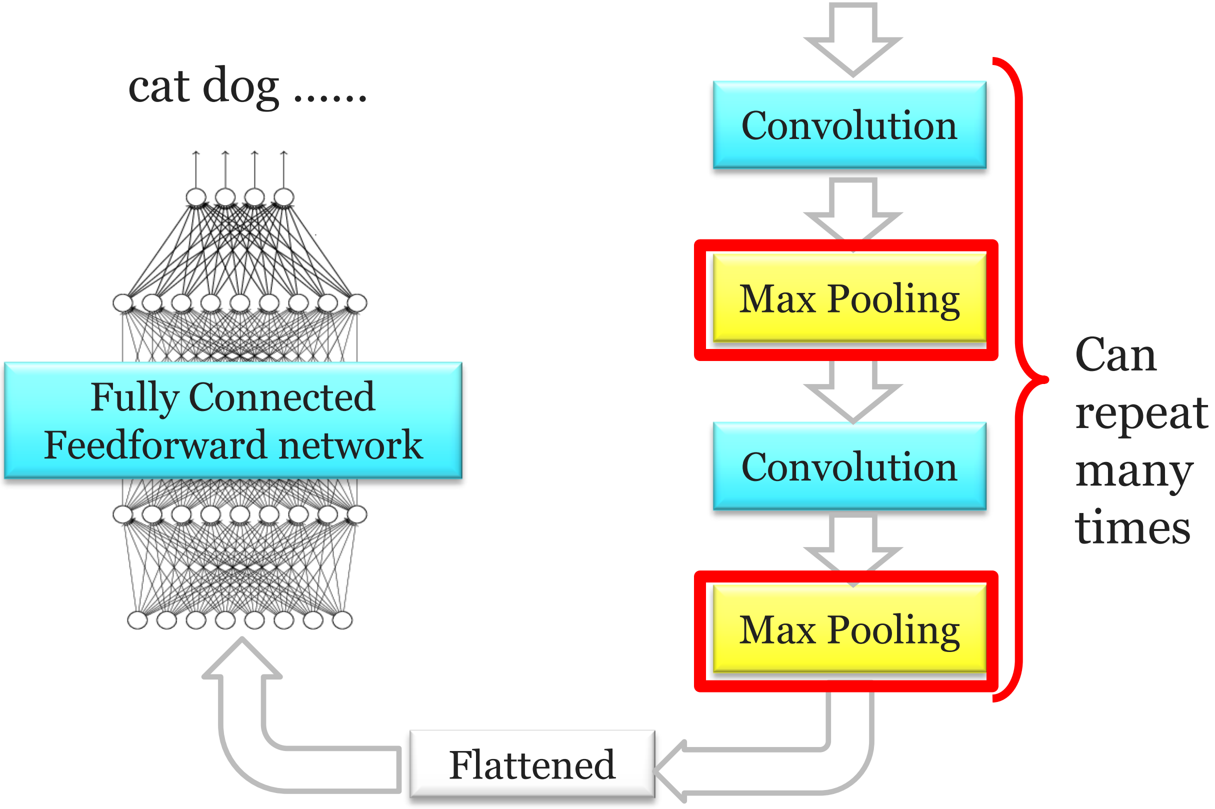
We will be using tf.nn.conv2d for the convolutional layer. A common practice is to group convolutional layer and non-linearity together, which we will do in this case. We will create a method conv_relu that can be used for both convolutional layers.

```
def conv_relu(inputs, filters, k_size, stride, padding, scope_name):
    with tf.variable_scope(scope_name, reuse=tf.AUTO_REUSE) as scope:
        in_channels = inputs.shape[-1]
        kernel = tf.get_variable('kernel', [k_size, k_size,
in_channels, filters],

initializer=tf.truncated_normal_initializer())
        biases = tf.get_variable('biases', [filters],

initializer=tf.random_normal_initializer())
        conv = tf.nn.conv2d(inputs, kernel, strides=[1, stride,
stride, 1], padding=padding)
    return tf.nn.relu(conv + biases, name=scope.name)
```

The whole CNN



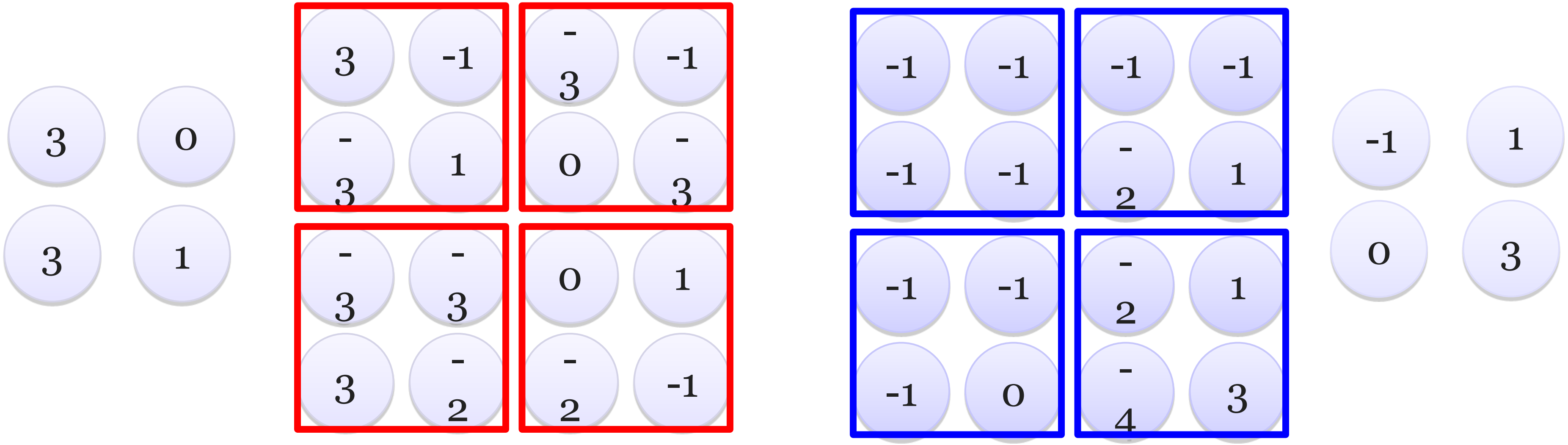
Max Pooling

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2



Why Pooling

- Subsampling pixels will not change the object

bird

Subsampling

bird

We can subsample the pixels to make image smaller

fewer parameters to characterize the image

Max Pooling

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

Common settings:

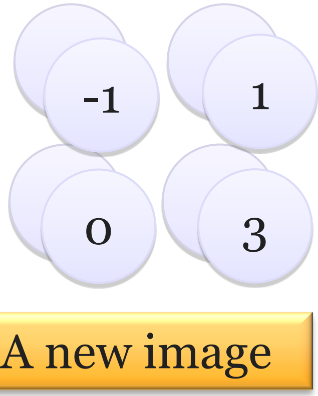
F = 2, S = 2
F = 3, S = 2

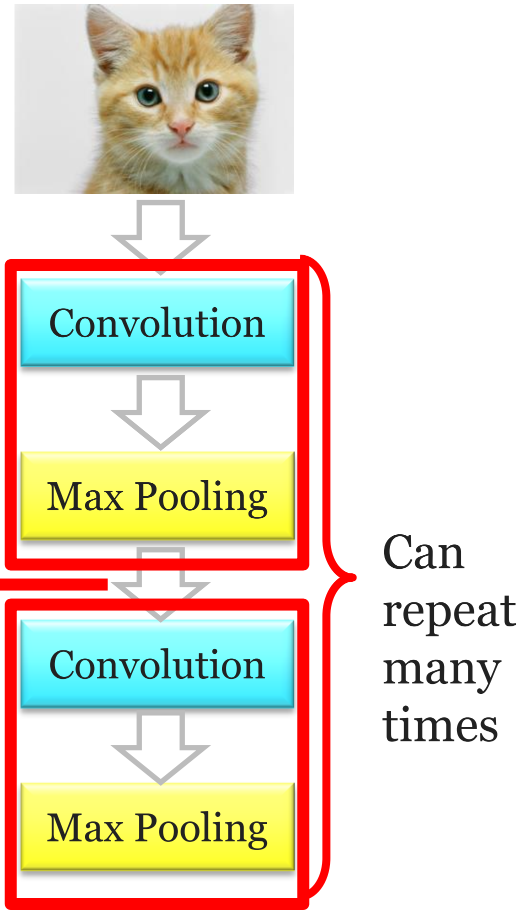
pooling layer in tensorflow

We will be using tf.nn.max_pool for the max pooling layer. We will create a method maxpool that can be used for both convolutional layers.

```
def maxpool(inputs, ksize, stride, padding='VALID', scope_name='pool'):
    with tf.variable_scope(scope_name, reuse=tf.AUTO_REUSE) as scope:
        pool = tf.nn.max_pool(inputs, ksize=[1, ksize, ksize, 1],
                              strides=[1, stride, stride, 1], padding=padding)
    return pool
```

The whole CNN

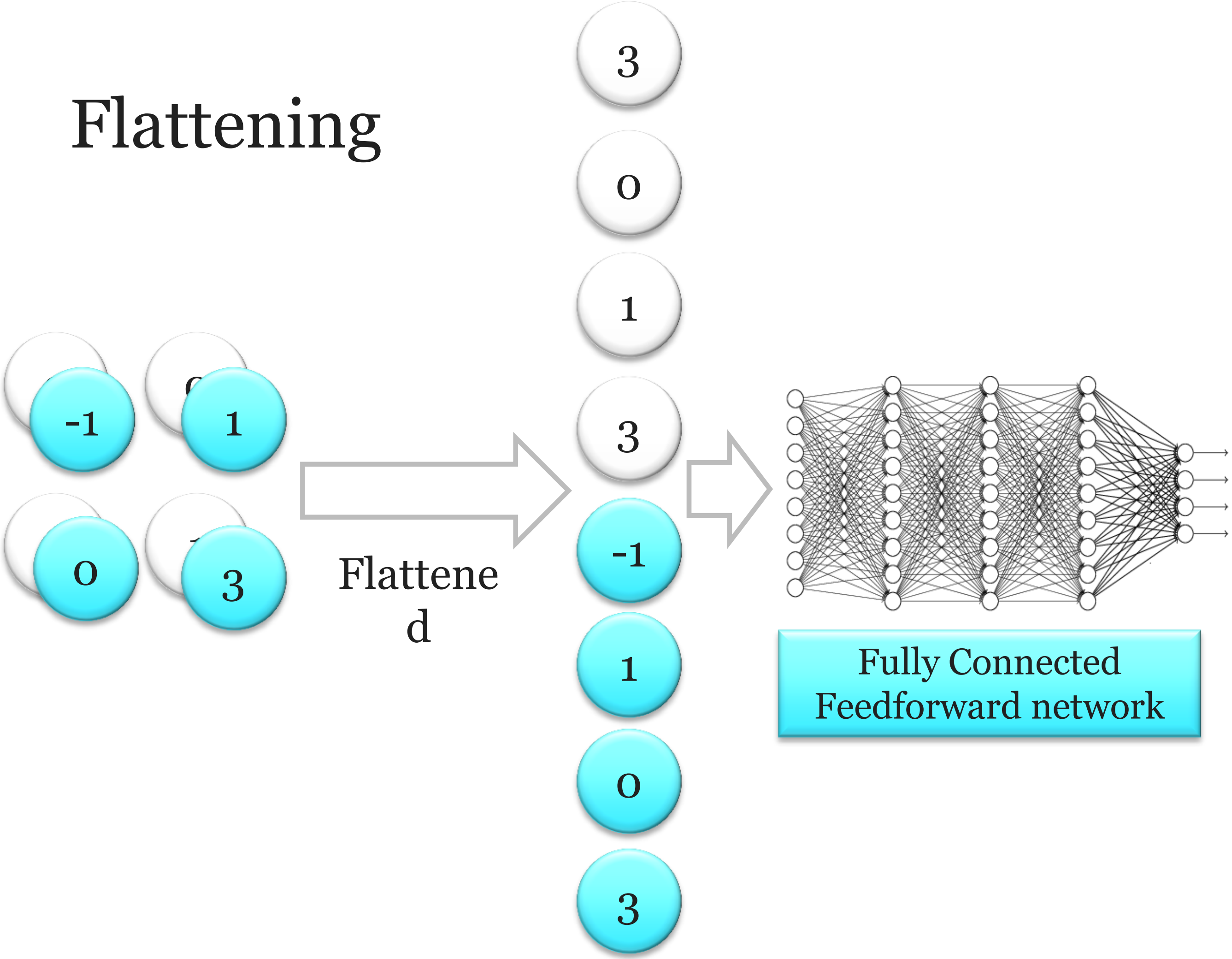




Smaller than the original image

The number of channels is the number of filters

Flattening


-1
1
0
3
Flattene
d
3
0
1
3
-1
1
0
3
Fully Connected
Feedforward network

Fully connected in tensorflow

We should be pretty familiar with the fully connected layer by now, as we have been using it for all of our models. Fully connected, or dense, layer is called so because every node in the layer is connected to every node in the preceding layer. Convolutional layers are only locally connected.

```
def fully_connected(inputs, out_dim, scope_name='fc'):
    with tf.variable_scope(scope_name, reuse=tf.AUTO_REUSE) as scope:
        in_dim = inputs.shape[-1]
        w = tf.get_variable('weights', [in_dim, out_dim],
                            initializer=tf.truncated_normal_initializer())
        b = tf.get_variable('biases', [out_dim],
                            initializer=tf.constant_initializer(0.0))
        out = tf.matmul(inputs, w) + b
    return out
```

With those building blocks (layers), we can easily construct our model.

```
def inference(self):
        conv1 = conv_relu(inputs=self.img,
                        filters=32,
                        k_size=5,
                        stride=1,
                        padding='SAME',
                        scope_name='conv1')
        pool1 = maxpool(conv1, 2, 2, 'VALID', 'pool1')
        conv2 = conv_relu(inputs=pool1,
                        filters=64,
                        k_size=5,
                        stride=1,
                        padding='SAME',
                        scope_name='conv2')
        pool2 = maxpool(conv2, 2, 2, 'VALID', 'pool2')
        feature_dim = pool2.shape[1] * pool2.shape[2] * pool2.shape[3]
        pool2 = tf.reshape(pool2, [-1, feature_dim])
        fc = tf.nn.relu(fully_connected(pool2, 1024, 'fc'))
        dropout = tf.layers.dropout(fc, self.keep_prob, training=self.training, 
name='dropout')
                self.logits = fully_connected(dropout, self.n_classes, 'logits')
```

During training, we alternate between training an epoch and evaluating the accuracy on the test set. We will track both the training loss and test accuracy on Tensorboard.

```
def eval(self):
        '''          Count the number of right predictions in a batch        '''
       with tf.name_scope('predict'):
           preds = tf.nn.softmax(self.logits)
           correct_preds = tf.equal(tf.argmax(preds, 1), tf.argmax(self.label, 1))
           self.accuracy = tf.reduce_sum(tf.cast(correct_preds, tf.float32))
```