10.1 卷积神经网络

layers.Conv2D

```
In [1]: import tensorflow as tf
In [2]: from tensorflow.keras import layers

In [6]: layer=layers.Conv2D(4, kernel_size=5,strides=1,padding='valid')
In [8]: out=layer(x)
Out[9]: TensorShape([1, 28, 28, 4])

In [10]: layer=layers.Conv2D(4, kernel_size=5,strides=1,padding='same')
In [11]: out=layer(x)
Out[12]: TensorShape([1, 32, 32, 4])

In [13]: layer=layers.Conv2D(4, kernel_size=5,strides=2,padding='same')
In [14]: out=layer(x)
Out[15]: TensorShape([1, 16, 16, 4])

In [16]: layer.call(x).shape
Out[16]: TensorShape([1, 16, 16, 4])
```

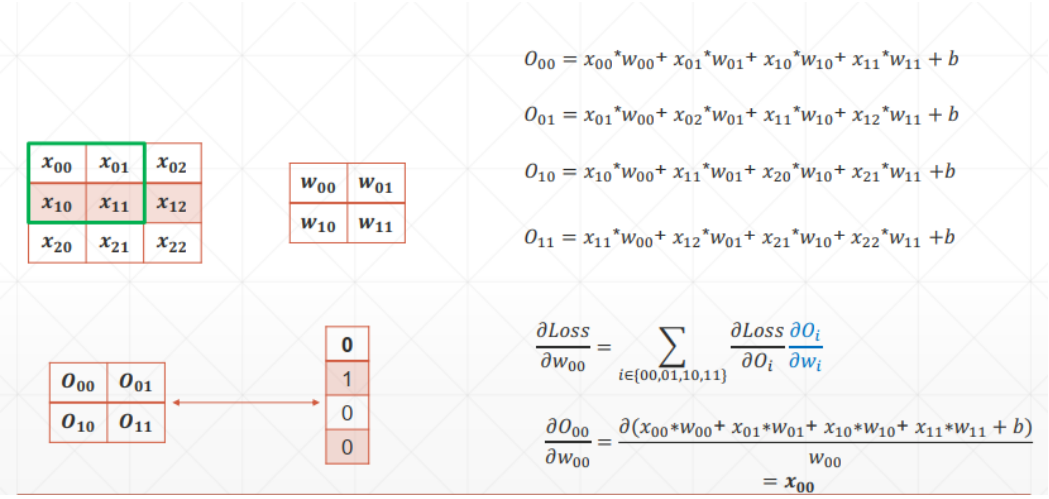
weight & bias

```
In [13]: layer=layers.Conv2D(4, kernel_size=5,strides=2,padding='same')
In [14]: out=layer(x)
Out[15]: TensorShape([1, 16, 16, 4])

In [17]: layer.kernel
<tf.Variable 'conv2d_3/kernel:0' shape=(5, 5, 3, 4) dtype=float32, numpy=
array([[[[-0.16160963,  0.04107726, -0.09828208, -0.00601757],
         [-0.02003701,  0.01415607, -0.07604317, -0.12557343],
         [-0.11157566,  0.1328298 ,  0.14624669, -0.04775226]], ...

In [18]: layer.bias
Out[18]: <tf.Variable 'conv2d_3/bias:0' shape=(4,) dtype=float32, numpy=array([0.,
0., 0., 0.], dtype=float32)>
```

nn.conv2d

```
In [21]: w=tf.random.normal([5,5,3,4])
In [22]: b=tf.zeros([4])
In [23]: x.shape
Out[23]: TensorShape([1, 32, 32, 3])

In [29]: out=tf.nn.conv2d(x, w, strides=1, padding='VALID')
Out[30]: TensorShape([1, 28, 28, 4])

In [31]: out = out + b
Out[32]: TensorShape([1, 28, 28, 4])

In [33]: out=tf.nn.conv2d(x,w,strides=2,padding='VALID')
Out[34]: TensorShape([1, 14, 14, 4])
```

卷积梯度的求导

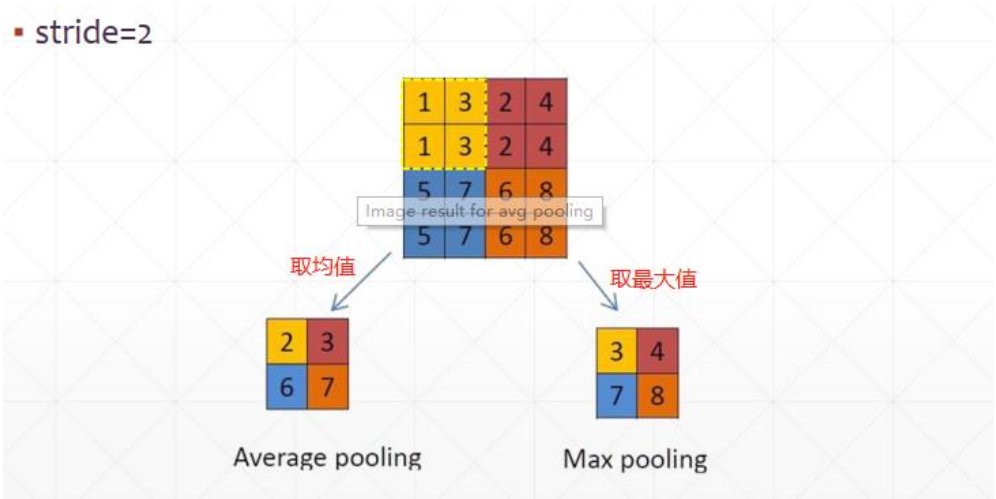


10.2 池化与采样

- ✓ Pooling ←→ 下采样
- ✓ upsample ←→ 上采样
- ✓ ReLU

下采样

- stride=2




```
In [36]: x # TensorShape([1, 14, 14, 4])

In [37]: pool=layers.MaxPool2D(2,strides=2)
In [38]: out=pool(x)
Out[39]: TensorShape([1, 7, 7, 4])

In [40]: pool=layers.MaxPool2D(3,strides=2)
In [41]: out=pool(x)
Out[42]: TensorShape([1, 6, 6, 4])

In [44]: out=tf.nn.max_pool2d(x, 2,strides=2,padding='VALID')
Out[45]: TensorShape([1, 7, 7, 4])
```

补充，关于卷积核后的图像尺寸计算，

1、 卷积后尺寸的计算

out_height = (in_height + 2*padding - filter_height) / strides[1] + 1
out_width = (in_width + 2*padding - filter_width) / strides[2] + 1

2、 卷积参数 *same* 和 *valid* 运算之后的维度计算

(1)same

out_height = ceil(float in_height)) / float(strides[1])
out_width = ceil(float(in_width)) / float(strides[2])

(2)valid

out_height = ceil(float(in_height - filter_height + 1)) / float(strides[1])
out_width = ceil(float(in_width - filter_width + 1)) / float(strides[2])

关于参数:

padding: SAME 和 VALID 两种形式

filter: [5,5,1,32] 表示 5*5 的卷积核，1 个 channel，32 个卷积核。

strides: [1,4,4,1] 表示横向和竖向的步长都为 4

上采样

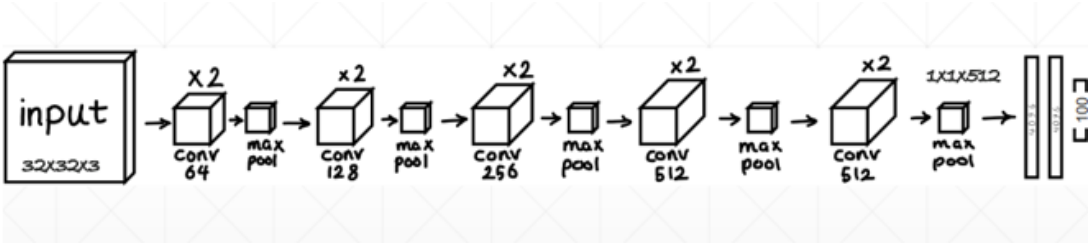
```
In [47]: x=tf.random.normal([1,7,7,4])

In [48]: layer=layers.UpSampling2D(size=3)
In [49]: out=layer(x)
Out[50]: TensorShape([1, 21, 21, 4])

In [51]: layer=layers.UpSampling2D(size=2)
In [52]: out=layer(x)
Out[53]: TensorShape([1, 14, 14, 4])
```

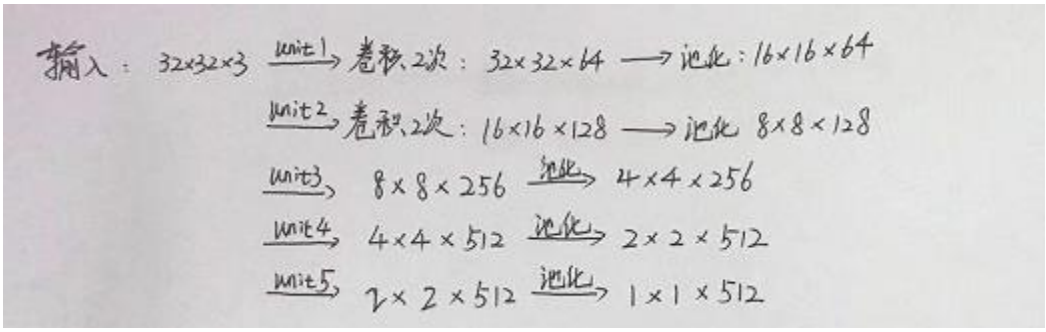
ReLU

```
In [55]: x=tf.random.normal([2,3])
<tf.Tensor: id=154, shape=(2, 3), dtype=float32, numpy=
array([[-1.533682  , -2.7053335 ,  0.36354962],
       [ 0.00713745,  0.69756126,  0.8053344 ]], dtype=float32)>

In [57]: tf.nn.relu(x)
<tf.Tensor: id=156, shape=(2, 3), dtype=float32, numpy=
array([[0.        , 0.        , 0.36354962],
       [0.00713745, 0.69756126, 0.8053344 ]], dtype=float32)>

In [59]: layers.ReLU()(x)
<tf.Tensor: id=158, shape=(2, 3), dtype=float32, numpy=
array([[0.        , 0.        , 0.36354962],
       [0.00713745, 0.69756126, 0.8053344 ]], dtype=float32)>
```

实战

网络结构





代码：

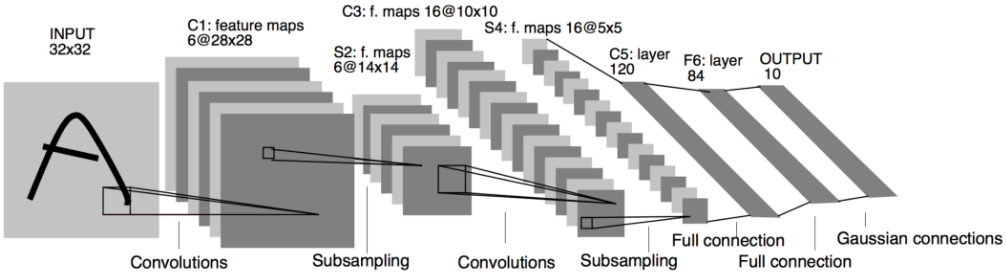
```
import tensorflow as tf
from tensorflow.keras import layers, optimizers,datasets,Sequential
tf.random.set_seed(2345)

# 5 unites of conv + maxpooling
conv_layers = [
    # unit 1
    layers.Conv2D(64, kernel_size=[3,3], padding='same', activation=tf.nn.relu),
    layers.Conv2D(64, kernel_size=[3,3], padding='same', activation=tf.nn.relu),
    layers.MaxPool2D(pool_size=[2,2], strides=2, padding='same'),

    # unit 2
    layers.Conv2D(128, kernel_size=[3,3], padding='same', activation=tf.nn.relu),
    layers.Conv2D(128, kernel_size=[3,3], padding='same', activation=tf.nn.relu),
    layers.MaxPool2D(pool_size=[2,2], strides=2, padding='same'),

    # unit 3
    layers.Conv2D(256, kernel_size=[3,3], padding='same', activation=tf.nn.relu),
    layers.Conv2D(256, kernel_size=[3,3], padding='same', activation=tf.nn.relu),
    layers.MaxPool2D(pool_size=[2,2], strides=2, padding='same'),

    # unit 4
    layers.Conv2D(512, kernel_size=[3,3], padding='same', activation=tf.n
n.relu),
    layers.Conv2D(512, kernel_size=[3,3], padding='same', activation=tf.n
n.relu),
```

```
    layers.MaxPool2D(pool_size=[2,2], strides=2, padding='same'),

    # unit 5
    layers.Conv2D(512, kernel_size=[3,3], padding='same', activation=tf.n
n.relu),
    layers.Conv2D(512, kernel_size=[3,3], padding='same', activation=tf.n
n.relu),
    layers.MaxPool2D(pool_size=[2,2], strides=2, padding='same')
 ]

# 数据预处理与加载数据集
def preprocess(x, y):
    x = tf.cast(x, dtype=tf.float32) / 255.
    y = tf.cast(y, dtype=tf.int32)
    return x,y
(x,y),(x_test,y_test) = datasets.cifar100.load_data()
y = tf.squeeze(y, axis=1)
y_test = tf.squeeze(y_test,  axis=1)

train_db = tf.data.Dataset.from_tensor_slices((x,y))
train_db = train_db.shuffle(1000).map(preprocess).batch(128)
test_db = tf.data.Dataset.from_tensor_slices((x,y))
test_db = test_db.map(preprocess).batch(64)

def main():
    # [b, 32, 32, 3]  ==> [b, 1, 1, 512]
    conv_net = Sequential(conv_layers)
    # conv_net.build(input_shape=[None, 32, 32,3])

    # 全连接层
    fc_net = Sequential([
        layers.Dense(256, activation=tf.nn.relu),
        layers.Dense(128, activation=tf.nn.relu),
        layers.Dense(100, activation=None)
    ])
    # 将上面两个网络给 build 一下
    conv_net.build(input_shape=[None, 32, 32, 3])
    fc_net.build(input_shape=[None, 512])

    # 设置优化器
    optimizer = optimizers.Adam(lr=1e-4)
    # 设置梯度参数
    variables = conv_net.trainable_variables + fc_net.trainable_varia
bles
```

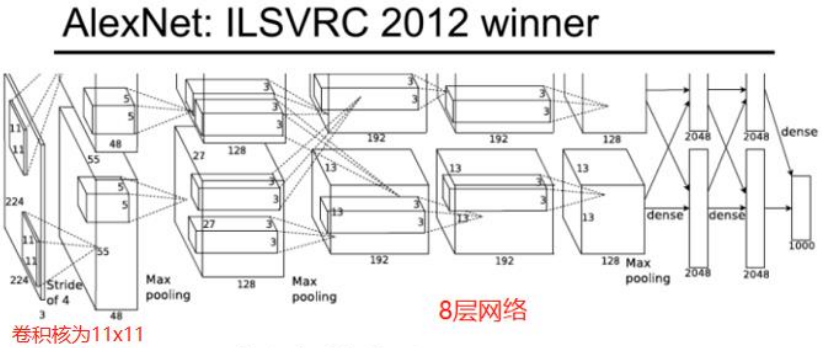
```
    for epoch in range(50):
        for step, (x,y) in enumerate(train_db):
            with tf.GradientTape() as tape:
                # [b, 32, 32, 3] => [b, 1, 1, 512]
                out = conv_net(x)
                # flatten to [b, 512]
                out = tf.reshape(out, [-1, 512])
                # [b, 512] ==> [b, 100]
                logits = fc_net(out)
                # 对 y 进行 one_hot 编码
                y_onehot = tf.one_hot(y, depth=100)
                # 计算损失
                loss = tf.losses.categorical_crossentropy(y_onehot, l
ogits, from_logits=True)
                loss = tf.reduce_mean(loss)

            grads = tape.gradient(loss, variables)
            optimizer.apply_gradients(zip(grads, variables))

            if step % 100 == 0:
                print(epoch, step, 'loss',  float(loss))

        total_num = 0
        total_correct = 0
        for x, y in test_db:
            out = conv_net(x)
            out = tf.reshape(out, [-1, 512])
            logits = fc_net(out)
            prob = tf.nn.softmax(logits, axis=1)
            pred = tf.argmax(prob, axis=1)
            pred = tf.cast(pred, dtype=tf.int32)

            correct = tf.cast(tf.equal(pred, y), dtype=tf.int32)
            correct = tf.reduce_sum(correct)

            total_num += x.shape[0]
            total_correct += int(correct)

        acc = total_correct / total_num
        print(epoch, 'acc:', acc)
main()
```

经典卷积网络

LeNet-5



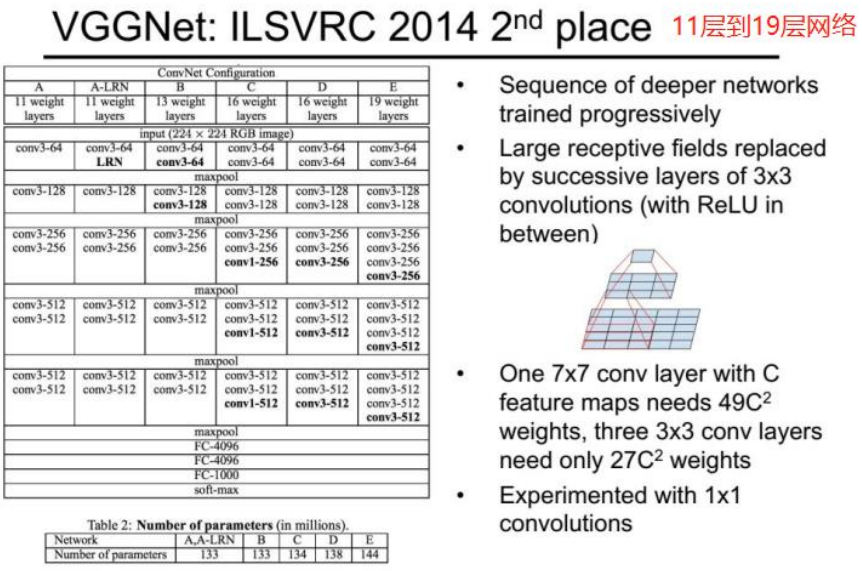
AlexNet



- Similar framework to LeNet but:
  - Max pooling, ReLU nonlinearity
  - More data and bigger model (7 hidden layers, 650K units, 60M params)
  - GPU implementation (50x speedup over CPU)
    - Trained on two GPUs for a week
  - Dropout regularization

A. Krizhevsky, I. Sutskever, and G. Hinton,
ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012

VGG



ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

- Sequence of deeper networks trained progressively
- Large receptive fields replaced by successive layers of 3x3 convolutions (with ReLU in between)

- One 7x7 conv layer with C feature maps needs $49C^2$ weights, three 3x3 conv layers need only $27C^2$ weights
- Experimented with 1x1 convolutions

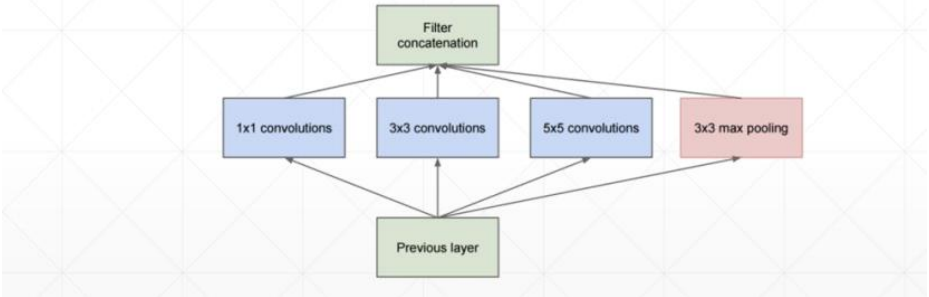
K. Simonyan and A. Zisserman,
Very Deep Convolutional Networks for Large-Scale Image Recognition, ICLR 2015

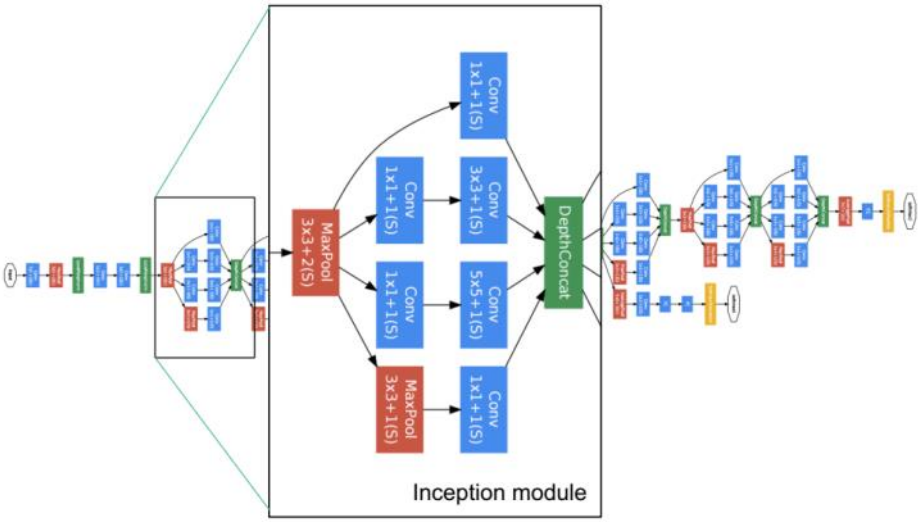
那么 1x1 的卷积核有什么作用呢？

- less computation
- 可以改变图片的通道数，从而改变维度，c_in >= c_out

GooLeNet

- 1st in 2014 ILSVRC
- 22 layers

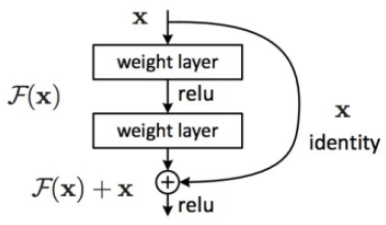




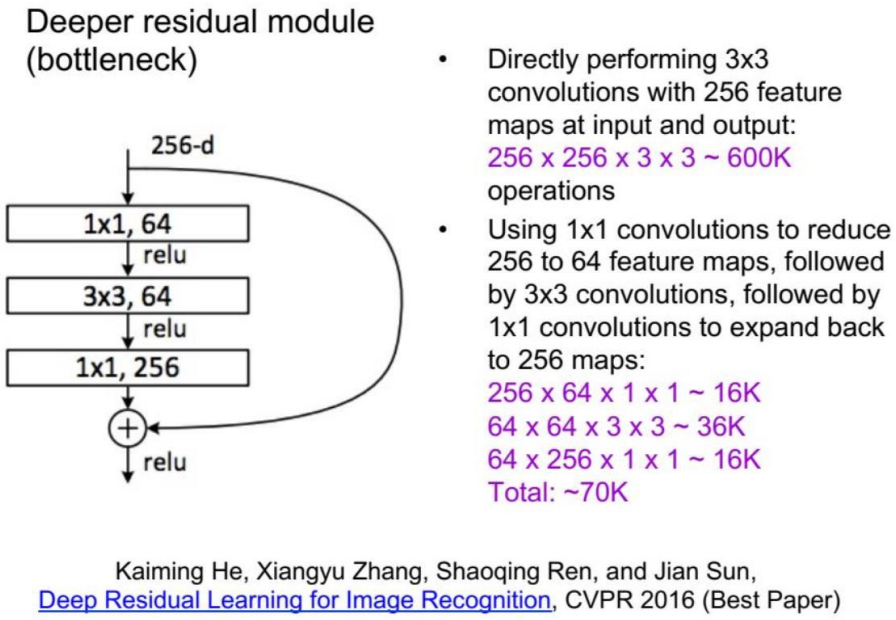
C. Szegedy et al., Going deeper with convolutions, CVPR 2015

ResNet 与 DenseNet

- The residual module
  - Introduce *skip* or *shortcut* connections (existing before in various forms in literature)
  - Make it easy for network layers to represent the identity mapping
  - For some reason, need to skip at least two layers



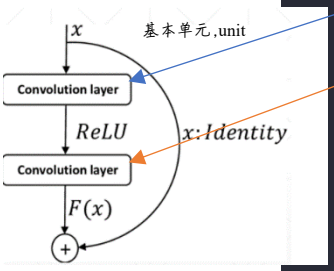
Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun,
Deep Residual Learning for Image Recognition, CVPR 2016 (Best Paper)



Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun,
Deep Residual Learning for Image Recognition, CVPR 2016 (Best Paper)

实现

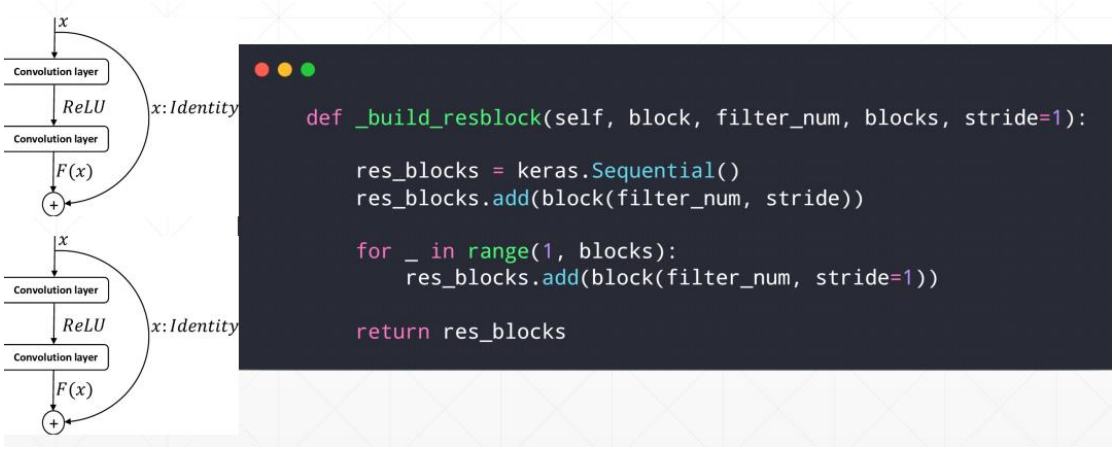
Basic Block




```python
class BasicBlock(layers.Layer):
    def __init__(self, filter_num, stride=1):
        super(BasicBlock, self).__init__()

        self.conv1 = layers.Conv2D(filter_num, (3, 3), strides=stride, padding='same')
        self.bn1 = layers.BatchNormalization()
        self.relu = layers.Activation('relu')
        self.conv2 = layers.Conv2D(filter_num, (3, 3), strides=1, padding='same')
        self.bn2 = layers.BatchNormalization()
        if stride != 1:
            self.downsample = Sequential()
            self.downsample.add(layers.Conv2D(filter_num, (1, 1), strides=stride))
            self.downsample.add(layers.BatchNormalization())
        else:
            self.downsample = lambda x: x

        self.stride = stride

    def call(self, inputs, training=None):
        residual = self.downsample(inputs)

        conv1 = self.conv1(inputs)
        bn1 = self.bn1(conv1)
        relu1 = self.relu(bn1)
        conv2 = self.conv2(relu1)
        bn2 = self.bn2(conv2)

        add = layers.add([bn2, residual])
        out = self.relu(add)
        return out
```

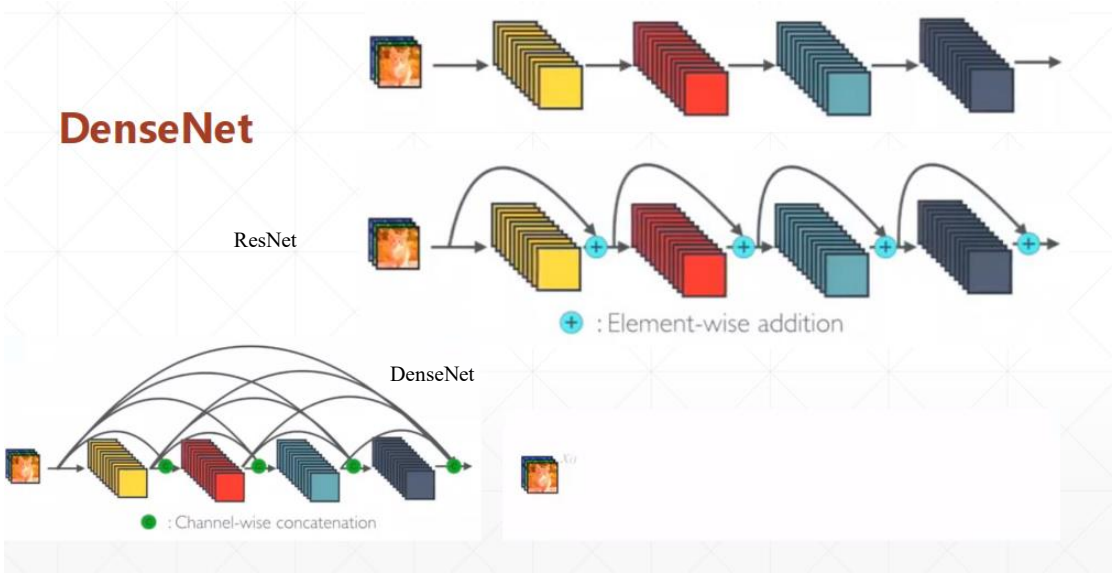
ResBlock

将 BasicBlock 堆叠起来，形成 ResBlock。



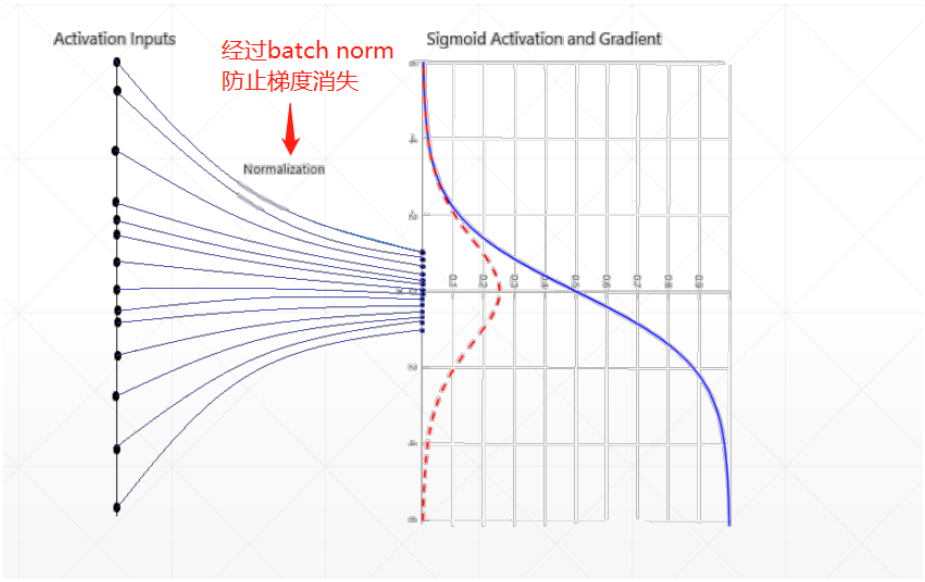

```
def _build_resblock(self, block, filter_num, blocks, stride=1):

    res_blocks = keras.Sequential()
    res_blocks.add(block(filter_num, stride))

    for _ in range(1, blocks):
        res_blocks.add(block(filter_num, stride=1))

    return res_blocks
```

DenseNet



Batch Norm

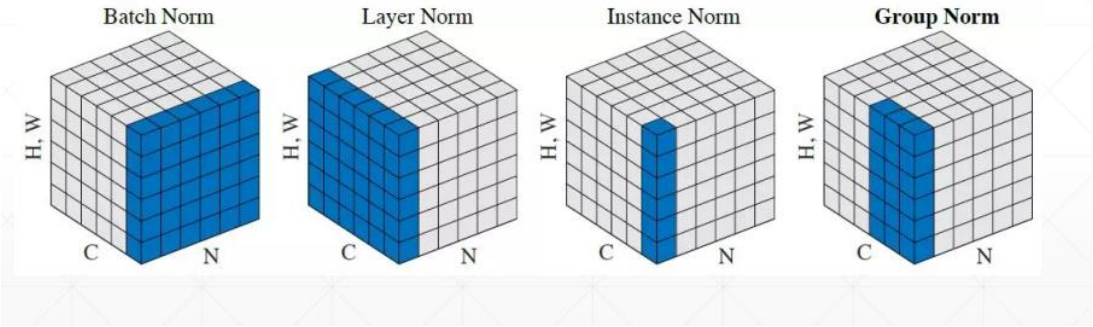
直观的解释：

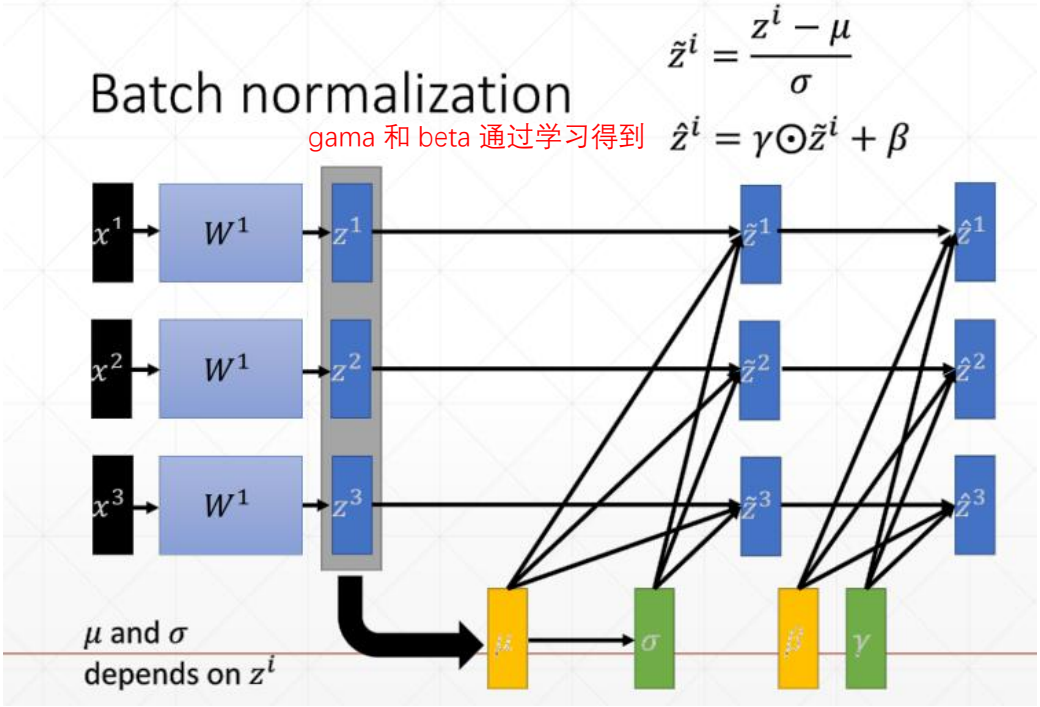


- Image Normalization

```
def normalize(x, mean,std):
  # x: [b, h, w, c]
  x = x - mean
  x = x / std
  return x
```

```
(mean=[0.485, 0.456, 0.406],
 std=[0.229, 0.224, 0.225])
```



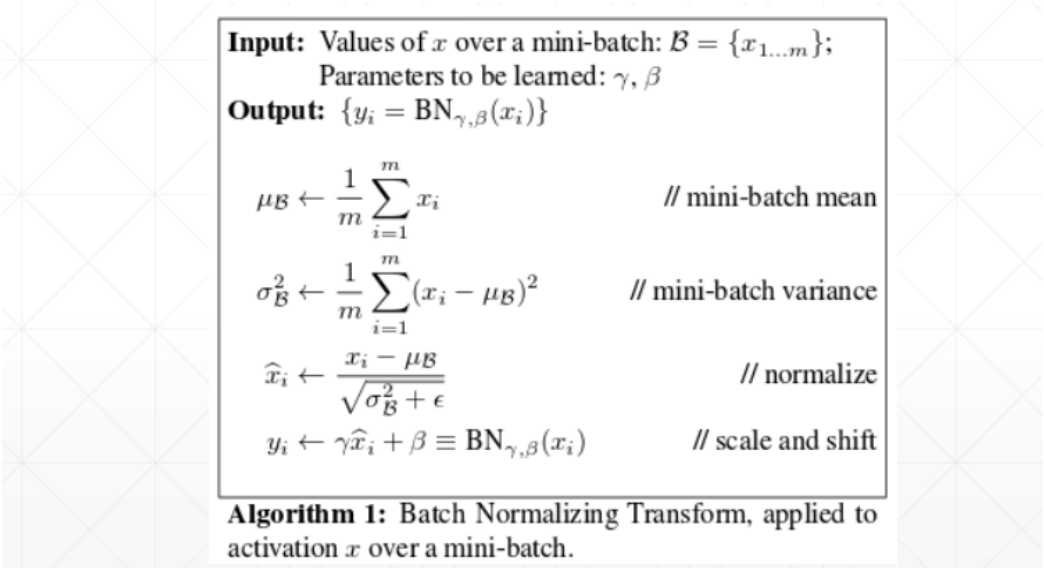


layers.BatchNormalization

- net = layers.BatchNormalization()
  - axis=-1,
  - center=True,
  - scale=True
  - trainable=True
- net(x, training=None)

```
In [3]: net=layers.BatchNormalization()
In [5]: x=tf.random.normal([2,3])
In [6]: out=net(x)

In [7]: net.trainable_variables
[<tf.Variable 'batch_normalization/gamma:0' shape=(3,) dtype=float32, numpy=array([1., 1., 1.],
dtype=float32)>,
 <tf.Variable 'batch_normalization/beta:0' shape=(3,) dtype=float32, numpy=array([0., 0., 0.],
dtype=float32)>]

In [8]: net.variables
[<tf.Variable 'batch_normalization/gamma:0' shape=(3,) dtype=float32, numpy=array([1., 1., 1.],
dtype=float32)>,
 <tf.Variable 'batch_normalization/beta:0' shape=(3,) dtype=float32, numpy=array([0., 0., 0.],
dtype=float32)>,
 <tf.Variable 'batch_normalization/moving_mean:0' shape=(3,) dtype=float32, numpy=array([0., 0.,
0.], dtype=float32)>,
 <tf.Variable 'batch_normalization/moving_variance:0' shape=(3,) dtype=float32, numpy=array([1.,
1., 1.], dtype=float32)>]
```

规范化写法



Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

```
In [15]: x=tf.random.normal([2,4,4,3],mean=1.,stddev=0.5)
In [16]: net=layers.BatchNormalization(axis=3)
In [17]: out=net(x)

In [18]: net.variables
[<tf.Variable 'batch_normalization_2/gamma:0' shape=(3,) dtype=float32, numpy=array([1., 1.,
1.], dtype=float32)>,
 <tf.Variable 'batch_normalization_2/beta:0' shape=(3,) dtype=float32, numpy=array([0., 0., 0.],
dtype=float32)>,
 <tf.Variable 'batch_normalization_2/moving_mean:0' shape=(3,) dtype=float32, numpy=array([0.,
0., 0.], dtype=float32)>,
 <tf.Variable 'batch_normalization_2/moving_variance:0' shape=(3,) dtype=float32,
numpy=array([1., 1., 1.], dtype=float32)>]
```

反向传播

```
for i in range(10):
    with tf.GradientTape() as tape:
        out = net(x, training=True)
        loss = tf.reduce_mean(tf.pow(out,2)) - 1
    grads = tape.gradient(loss, net.trainable_variables)
    optimizer.apply_gradients(zip(grads, net.trainable_variables))

backward(10 steps):
  [<tf.Variable 'batch_normalization/gamma:0' shape=(3,) dtype=float32, numpy=array([0.93549937,
0.9356556 , 0.9355564 ], dtype=float32)>,
   <tf.Variable 'batch_normalization/beta:0' shape=(3,) dtype=float32, numpy=array([ 1.3411044e-
09,  1.3411045e-08, -4.0978188e-10], dtype=float32)>...]
```

前向传播

```
In [15]: x=tf.random.normal([2,4,4,3],mean=1.,stddev=0.5)
In [16]: net=layers.BatchNormalization(axis=3)
In [19]: out=net(x,training=True)
In [20]: net.variables
[<tf.Variable 'batch_normalization_2/gamma:0' shape=(3,) dtype=float32, numpy=array([1., 1.,
1.], dtype=float32)>,
 <tf.Variable 'batch_normalization_2/beta:0' shape=(3,) dtype=float32, numpy=array([0., 0., 0.],
dtype=float32)>,
 <tf.Variable 'batch_normalization_2/moving_mean:0' shape=(3,) dtype=float32,
numpy=array([0.00992113, 0.00976678, 0.00942467], dtype=float32)>,
 <tf.Variable 'batch_normalization_2/moving_variance:0' shape=(3,) dtype=float32,
numpy=array([0.9923687 , 0.9918039 , 0.99186534], dtype=float32)>]

In [21]: for i in range(100):out=net(x,training=True)
In [22]: net.variables
[<tf.Variable 'batch_normalization_2/gamma:0' shape=(3,) dtype=float32, numpy=array([1., 1.,
1.], dtype=float32)>,
 <tf.Variable 'batch_normalization_2/beta:0' shape=(3,) dtype=float32, numpy=array([0., 0., 0.],
dtype=float32)>,
 <tf.Variable 'batch_normalization_2/moving_mean:0' shape=(3,) dtype=float32,
numpy=array([0.63259894, 0.6227575 , 0.60094345], dtype=float32)>,
 <tf.Variable 'batch_normalization_2/moving_variance:0' shape=(3,) dtype=float32,
numpy=array([0.5134074, 0.4773918, 0.4813124], dtype=float32)>]
```

实战：

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, Sequential

class BasicBlock(layers.Layer):

    def __init__(self, filter_num, stride=1):
        super(BasicBlock, self).__init__()

        self.conv1 = layers.Conv2D(filter_num, (3,3), strides=stride,
 padding='same')
        self.bn1 = layers.BatchNormalization()
        self.relu = layers.Activation('relu')

        self.conv2 = layers.Conv2D(filter_num, (3,3), strides=1, padd
ing='same')
        self.bn2 = layers.BatchNormalization()

        if stride != 1:
            self.downsample = Sequential()
            self.downsample.add(layers.Conv2D(filter_num, (1,1), stri
des=stride))
        else:
            self.downsample = lambda x:x
```

```
    def call(self, inputs, training=None):
        # [b, h, w, c]
        out = self.conv1(inputs)
        out = self.bn1(out)
        out = self.relu(out)

        out = self.conv2(out)
        out = self.bn2(out)

        identity = self.downsample(inputs)

        output = layers.add([out, identity])
        # output = self.relu(output)  等价于下面的
        output = tf.nn.relu(output)

        return output

class ResNet(keras.Model):

    def __init__(self, layer_dims, num_classes=100):
        super(ResNet, self).__init__()

        self.stem = Sequential([layers.Conv2D(64,[3,3],strides=(1,1))
,
                                layers.BatchNormalization(),
                                layers.Activation('relu'),
                                layers.MaxPool2D(pool_size=(2,2), str
ides=[1,1], padding='same')])

        self.layer1 = self.build_resblock(64, layer_dims[0])
        self.layer2 = self.build_resblock(128, layer_dims[1], stride=
2)
        self.layer3 = self.build_resblock(256, layer_dims[2], stride=
2)
        self.layer4 = self.build_resblock(512, layer_dims[3], stride=
2)


        self.avgpool = layers.GlobalAveragePooling2D()
        self.fc = layers.Dense(num_classes)

    def call(self, inputs, training=None):
        x = self.stem(inputs)
        x = self.layer1(x)
        x = self.layer2(x)
```

```
        x = self.layer3(x)
        x = self.layer4(x)

        x = self.avgpool(x)
        x = self.fc(x)

        return x

    def build_resblock(self, filter_num, blocks, stride=1):

        res_blocks = Sequential()
        res_blocks.add(BasicBlock(filter_num, stride))

        for _ in range(1, blocks):
            res_blocks.add(BasicBlock(filter_num, stride=1))

        return res_blocks

def resnet18():
    return ResNet([2, 2, 2, 2])

def resnet34():
    return ResNet([3, 4, 6, 3])
```

主代码

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
import tensorflow as tf
from tensorflow.keras import layers, optimizers,datasets,Sequential
from resnet import resnet18
tf.random.set_seed(2345)

# 数据预处理与加载数据集
def preprocess(x, y):
    x = tf.cast(x, dtype=tf.float32) / 255.
    y = tf.cast(y, dtype=tf.int32)
    return x,y
(x,y),(x_test,y_test) = datasets.cifar100.load_data()
y = tf.squeeze(y, axis=1)
y_test = tf.squeeze(y_test,  axis=1)

train_db = tf.data.Dataset.from_tensor_slices((x,y))
train_db = train_db.shuffle(1000).map(preprocess).batch(128)
test_db = tf.data.Dataset.from_tensor_slices((x,y))
test_db = test_db.map(preprocess).batch(64)
```

```
def main():
    # [b, 32, 32, 3]  ==> [b, 1, 1, 512]
    model = resnet18()
    model.build(input_shape=(None, 32, 32, 3))
    optimizer = optimizers.Adam(lr=1e-3)

    for epoch in range(50):
        for step, (x,y) in enumerate(train_db):
            with tf.GradientTape() as tape:
                # [b, 512] ==> [b, 100]
                logits = model(x)
                # 对 y 进行 one_hot 编码
                y_onehot = tf.one_hot(y, depth=100)
                # 计算损失
                loss = tf.losses.categorical_crossentropy(y_onehot, l
ogits, from_logits=True)
                loss = tf.reduce_mean(loss)

            grads = tape.gradient(loss, model.trainable_variables)
            optimizer.apply_gradients(zip(grads, model.trainable_vari
ables))

            if step % 100 == 0:
                print(epoch, step, 'loss',  float(loss))

        total_num = 0
        total_correct = 0
        for x, y in test_db:
            logits = model(out)
            prob = tf.nn.softmax(logits, axis=1)
            pred = tf.argmax(prob, axis=1)
            pred = tf.cast(pred, dtype=tf.int32)

            correct = tf.cast(tf.equal(pred, y), dtype=tf.int32)
            correct = tf.reduce_sum(correct)

            total_num += x.shape[0]
            total_correct += int(correct)

        acc = total_correct / total_num
        print(epoch, 'acc:', acc)

main()
```