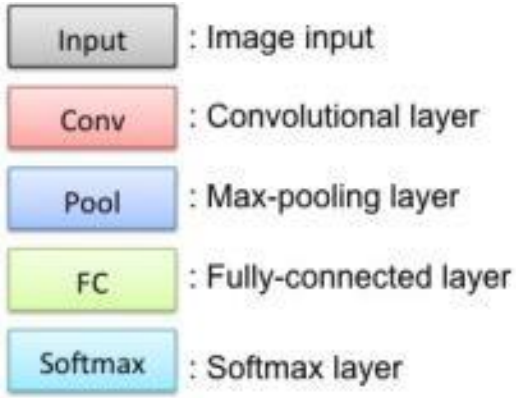
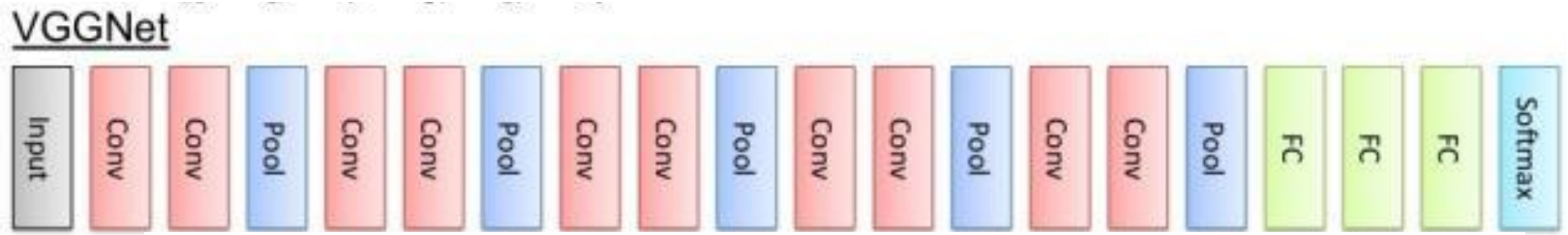
# 项目实践

- 数据准备
- VGG16的Tensorflow实现
  - 定义功能函数
  - 定义VGG16模型类
- VGG16模型复用
  - 微调
  - 载入权重
- 数据输入
- 模型重新训练与保存
- 预测

# 回顾VGG结构


Input
: Image input
Conv
: Convolutional layer
Pool
: Max-pooling layer
FC
: Fully-connected layer
Softmax
: Softmax layer


VGGNet
Input
Conv
Conv
Pool
Conv
Conv
Pool
Conv
Conv
Pool
Conv
Conv
Pool
Conv
Conv
Pool
FC
FC
FC
Softmax

# 定义功能函数



```
def conv(self,name, input_data, out_channel):
    in_channel = input_data.get_shape()[-1]
    with tf.variable_scope(name):
        kernel = tf.get_variable("weights", [3, 3, in_channel, out_channel], dtype=tf.float32)
        biases = tf.get_variable("biases", [out_channel], dtype=tf.float32)
        conv_res = tf.nn.conv2d(input_data, kernel, [1, 1, 1, 1], padding="SAME")
        res = tf.nn.bias_add(conv_res, biases)
        out = tf.nn.relu(res, name=name)
    return out
```

# 定义功能函数

```
def fc(self,name,input_data,out_channel):
    shape = input_data.get_shape().as_list()
    if len(shape) == 4:
        size = shape[-1] * shape[-2] * shape[-3]
    else:size = shape[1]
    input_data_flat = tf.reshape(input_data,[-1,size])
    with tf.variable_scope(name):
        weights = tf.get_variable(name="weights",shape=[size,out_channel],dtype=tf.float32)
        biases = tf.get_variable(name="biases",shape=[out_channel],dtype=tf.float32)
        res = tf.matmul(input_data_flat,weights)
        out = tf.nn.relu(tf.nn.bias_add(res,biases))
    return out
```

全连接层输入神经元个数

对数据进行展开操作

```
def maxpool(self,name,input_data):
    out = tf.nn.max_pool(input_data,[1,2,2,1],[1,2,2,1],padding="SAME",name=name)
    return out
```

# 定义VGG-16模型类




Input
Conv
Conv
Pool
Conv
Conv
Pool
Conv
Conv
Pool
Conv
Conv
Pool
Conv
Conv
Pool
FC
FC
FC
Softmax
conv_layers
fc_layers

# 定义VGG-16模型类

```
class vgg16:

    def __init__(self, imgs):
        self.imgs = imgs
        self.convlayers()
        self.fc_layers()
        self.probs =self.fc8

    def fc_layers(self):

        self.fc6 = self.fc("fc1", self.pool5, 4096)
        self.fc7 = self.fc("fc2", self.fc6, 4096)
        self.fc8 = self.fc("fc3", self.fc7, n_class)
```

输出类别个数

数据输入

```
def convlayers(self):
    #conv1
    self.conv1_1 = self.conv("conv1re_1",self.imgs,64)
    self.conv1_2 = self.conv("conv1_2",self.conv1_1,64)
    self.pool1 = self.maxpool("poolre1",self.conv1_2)

    #conv2
    self.conv2_1 = self.conv("conv2_1",self.pool1,128)
    self.conv2_2 = self.conv("convwe2_2",self.conv2_1,128)
    self.pool2 = self.maxpool("pool2",self.conv2_2)

    #conv3
    self.conv3_1 = self.conv("conv3_1",self.pool2,256)
    self.conv3_2 = self.conv("convrwe3_2",self.conv3_1,256)
    self.conv3_3 = self.conv("convrew3_3",self.conv3_2,256)
    self.pool3 = self.maxpool("poolre3",self.conv3_3)

    #conv4
    self.conv4_1 = self.conv("conv4_1",self.pool3,512)
    self.conv4_2 = self.conv("convrwe4_2",self.conv4_1,512)
    self.conv4_3 = self.conv("conv4rwe_3",self.conv4_2,512)
    self.pool4 = self.maxpool("pool4",self.conv4_3)

    #conv5
    self.conv5_1 = self.conv("conv5_1",self.pool4,512)
    self.conv5_2 = self.conv("convrwe5_2",self.conv5_1,512)
    self.conv5_3 = self.conv("conv5_3",self.conv5_2,512)
    self.pool5 = self.maxpool("poorwel5",self.conv5_3)
```

# 项目实践

- 数据准备
- VGG16的Tensorflow实现
  - 定义功能函数
  - 定义VGG16模型类
- VGG16模型复用
  - 微调
  - 载入权重
- 数据输入
- 模型重新训练与保存
- 预测

# 微调（finetuining）



## (1) trainable参数变动

在进行Finetuning对模型重新训练时，对于部分不需要训练的层可以通过设置trainable=False来确保其在训练过程中不会被修改权值；

## (2) 全连接层的神经元个数

预训练的VGG是在ImageNet数据集上进行训练的，对1000个类别进行判定，若希望利用已训练模型用于其他分类任务，需要修改最后的全连接层 。

# 微调（finetuining）

浙江大学城市学院 ZHEJIANG UNIVERSITY CITY COLLEGE

## (1) trainable参数变动

```
class vgg16:

    def __init__(self, imgs):
        self.parameters = [] # 在类的初始化时加入全局列表，将所需共享的参数加载进来
        self.imgs = imgs
        self.convlayers()
        self.fc_layers()
        self.probs = tf.nn.softmax(self.fc8)# 输出每个属于各个类别的概率值
```

# 微调（finetuining）

## (1) trainable参数变动

```
def conv(self,name, input_data, out_channel, trainable=False): # trainable参数变动
    in_channel = input_data.get_shape()[-1]
    with tf.variable_scope(name):
        kernel = tf.get_variable("weights", [3, 3, in_channel, out_channel], dtype=tf.float32, trainable=False) # trainable参数变动
        biases = tf.get_variable("biases", [out_channel], dtype=tf.float32, trainable=False) # trainable参数变动
        conv_res = tf.nn.conv2d(input_data, kernel, [1, 1, 1, 1], padding="SAME")
        res = tf.nn.bias_add(conv_res, biases)
        out = tf.nn.relu(res, name=name)
    self.parameters += [kernel, biases] # 将卷积层定义的参数（kernel, biases）加入列表
    return out
```

# 微调（finetuining）

浙江大学城市学院 ZHEJIANG UNIVERSITY CITY COLLEGE

## (1) trainable参数变动

```
def fc(self,name,input_data,out_channel, trainable=True): # trainable参数变动
    shape = input_data.get_shape().as_list()
    if len(shape) == 4:
        size = shape[-1] * shape[-2] * shape[-3]
    else:size = shape[1]
    input_data_flat = tf.reshape(input_data,[-1,size])
    with tf.variable_scope(name):
        weights = tf.get_variable(name="weights",shape=[size,out_channel],dtype=tf.float32, trainable=trainable) # trainable参数变动
        biases = tf.get_variable(name="biases",shape=[out_channel],dtype=tf.float32, trainable=trainable) # trainable参数变动
        res = tf.matmul(input_data_flat,weights)
        out = tf.nn.relu(tf.nn.bias_add(res,biases))
    self.parameters += [weights, biases] # 将全连接层定义的参数（weights, biases）加入列表
    return out
```

```
def convlayers(self):
    #conv1
    self.conv1_1 = self.conv("conv1re_1",self.imgs,64,trainable=False)# trainable参数变动
    self.conv1_2 = self.conv("conv1_2",self.conv1_1,64,trainable=False)# trainable参数变动
    self.pool1 = self.maxpool("poolre1",self.conv1_2)

    #conv2
    self.conv2_1 = self.conv("conv2_1",self.pool1,128,trainable=False)# trainable参数变动
    self.conv2_2 = self.conv("convwe2_2",self.conv2_1,128,trainable=False)# trainable参数变动
    self.pool2 = self.maxpool("pool2",self.conv2_2)

    #conv3
    self.conv3_1 = self.conv("conv3_1",self.pool2,256,trainable=False)# trainable参数变动
    self.conv3_2 = self.conv("convrwe3_2",self.conv3_1,256,trainable=False)# trainable参数变动
    self.conv3_3 = self.conv("convrew3_3",self.conv3_2,256,trainable=False)# trainable参数变动
    self.pool3 = self.maxpool("poolre3",self.conv3_3)

    #conv4
    self.conv4_1 = self.conv("conv4_1",self.pool3,512,trainable=False)# trainable参数变动
    self.conv4_2 = self.conv("convrwe4_2",self.conv4_1,512,trainable=False)# trainable参数变动
    self.conv4_3 = self.conv("conv4rwe_3",self.conv4_2,512,trainable=False)# trainable参数变动
    self.pool4 = self.maxpool("pool4",self.conv4_3)

    #conv5
    self.conv5_1 = self.conv("conv5_1",self.pool4,512,trainable=False)# trainable参数变动
    self.conv5_2 = self.conv("convrwe5_2",self.conv5_1,512,trainable=False)# trainable参数变动
    self.conv5_3 = self.conv("conv5_3",self.conv5_2,512,trainable=False)# trainable参数变动
    self.pool5 = self.maxpool("poorwel5",self.conv5_3)
```

# 微调（finetuining）



## (2) 全连接层的神经元个数

预训练的VGG是在ImageNet数据集上进行训练的，对1000个类别进行判定，若希望利用已训练模型用于其他分类任务，需要修改最后的全连接层 。

```
def fc_layers(self):

    self.fc6 = self.fc("fc1", self.pool5, 4096, trainable=False)# trainable参数变动
    self.fc7 = self.fc("fc2", self.fc6, 4096, trainable=False) # trainable参数变动
    self.fc8 = self.fc("fc3", self.fc7, 2, trainable=True) #fc8正是我们需要训练的，因此trainable=True；2是n_class
```

重新训练fc8

# 微调（finetuining）



**无需调整：**

- maxpool()
- saver()

# 载入权重

```
def load_weights(self, weight_file, sess): # 这个函数将获取的权重载入VGG模型中
    weights = np.load(weight_file)
    keys = sorted(weights.keys())
    for i, k in enumerate(keys):
        if i not in [30, 31]:# 剔除不需载入的层
            sess.run(self.parameters[i].assign(weights[k]))
    print("----------weights loaded---------------")
```

复用权值

'./vgg16/vgg16_weights.npz'

- 权重文件：https://www.cs.toronto.edu/~frossard/vgg16/vgg16_weights.npz
- 分类文件： https://www.cs.toronto.edu/~frossard/vgg16/imagenet_classes.py