1. 迁移学习和微调技术

- 实践中复杂模型与数据量的矛盾
  - 深度复杂模型（如VGG16，ResNet等）需要大量带标注的训练数据
  - 现实任务中难以收集大量带标注数据
    - 医学影像识别任务（X光片，CT等）
    - 遥感图像目标识别

1. 迁移学习和微调技术

- 深度卷积神经网络的层级特征
  - 低层：与数据类别无关的特征，如纹理、边缘等
  - 高层：与数据类别相关的高阶抽象特征

1. 迁移学习和微调技术

- 现有的大规模图像数据集：ImageNet
  - 1M以上的训练数据
  - 带有分类标注（1000类）和目标bounding box标注
  - 可以用于训练深度神经网络
- 卷积神经网络提取的ImageNet低层特征有潜力利用到其他数据集上

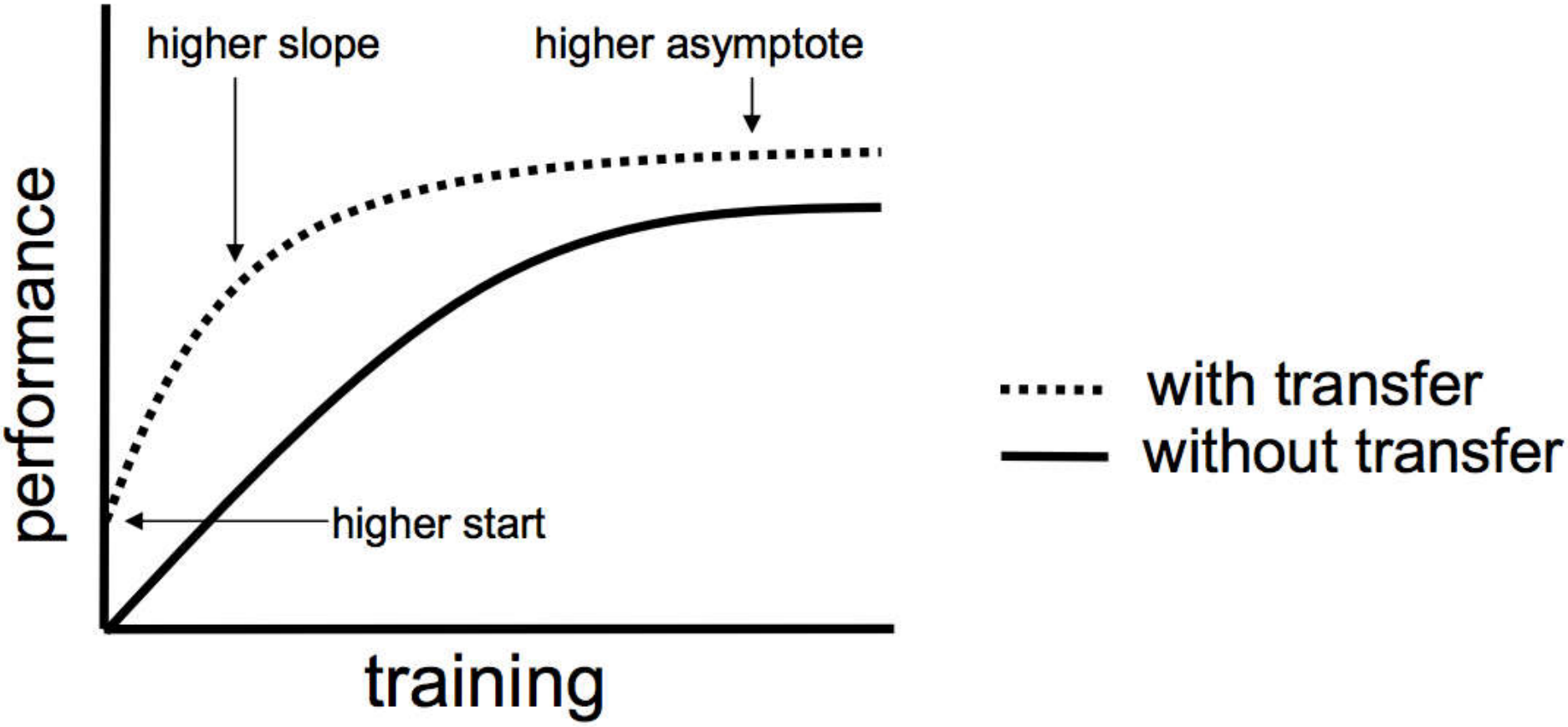
1. 迁移学习和微调技术

- 迁移学习技术：
  - 利用训练好的深度神经网络初始化其他任务的神经网络
  - 用新数据对已初始化的网络进行微调
  - 可以改善小数据集的训练效果

1. 迁移学习和微调技术

- **“迁移”的内涵：**
  - 让现有的模型算法经过微调之后即可应用于一个新的领域或功能（迁移）
  - 重点是预训练特征的迁移
  - 实践中体现为网络模型参数的迁移

1. 迁移学习和微调技术

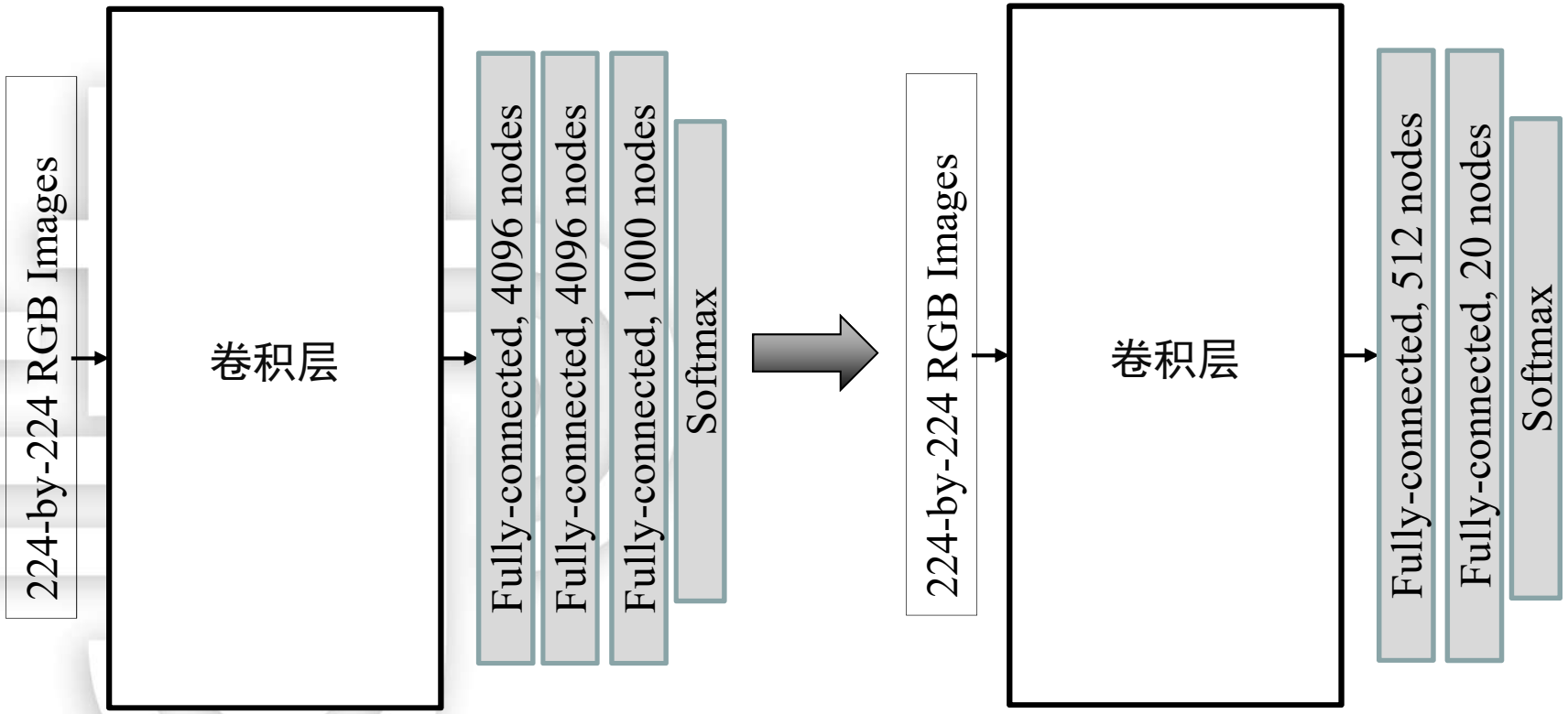

higher slope
higher asymptote
performance
higher start
training
with transfer
without transfer

1. 迁移学习和微调技术

- **迁移学习的应用场景：**
  - 同类型的复杂任务迁移到简单任务
  - 不同类型但是使用的低层卷积特征具有相关性的任务
- 基于ImageNet数据集训练的网络常用于迁移学习
  - 百万级的训练数据规模，1000个图像类别
  - 可以迁移至多数与自然图像相关的任务
  - VGG16，VGG19，GoogLeNet...

1. 迁移学习和微调技术

- 将VGG16网络（1000类分类任务）迁移到20类分类任务：



1. 迁移学习和微调技术

- **迁移学习的形式：**
  - 同类型任务的迁移
    - ImageNet分类任务迁移到小规模数据集分类任务
    - 彩色图像分类任务迁移到黑白图像分类任务
  - 不同类型任务的迁移
    - 图像分类任务迁移到目标检测识别任务（Faster-RCNN)
    - 图像分类任务迁移到回归任务

1. 迁移学习和微调技术

- 不适用迁移学习的情况：
  - 简单任务迁移到复杂任务
  - 类型差别过大，无法共享特征的任务
    - 图像识别与语音识别

1. 迁移学习和微调技术

- 深度神经网络中的迁移学习：
  - 以预训练模型的结构和参数为基础
  - 可以根据新数据集的情况适当调整网络结构
  - 对调整过的层随机初始化
  - 未调整的层直接使用预训练模型的参数初始化

1. 迁移学习和微调技术

- 微调的注意事项：
  - 低层特征更一般化，可采用较小学习率；高层特征与数据集相关性更高，可采用较大学习率
  - 微调所有层或仅微调高层：训练数据与原始数据相似且较少时，可以仅微调高层；训练数据量足够时，可以微调所有层

1. 迁移学习和微调技术

- **迁移学习编程范式：**
  - 设计包含预训练模型的计算图
  - 参数初始化
    - 利用预训练网络参数初始化部分层
    - 随机初始化部分层
  - 开始微调训练：多种不同的学习策略

1. 迁移学习和微调技术

- **迁移学习实例：**
  - 任务：训练分类器对Pascal VOC 2012数据集中的图像进行分类
  - 特点：
    - 数据集规模小（训练集：5717；测试集：5823）
    - 共有20个不同的种类
    - 图像大小和质量与ImageNet数据集接近

1. 迁移学习和微调技术

- 实验方案：
  - 1. 不使用迁移学习，对模型随机初始化
  - 2. 使用迁移学习
    - 方法1. 仅更新全连接层权值
    - 方法2. 采用相对较大的学习率训练全连接层，相对较小的学习率微调卷积层
    - 方法3. 采用相同的较大学习率微调所有层

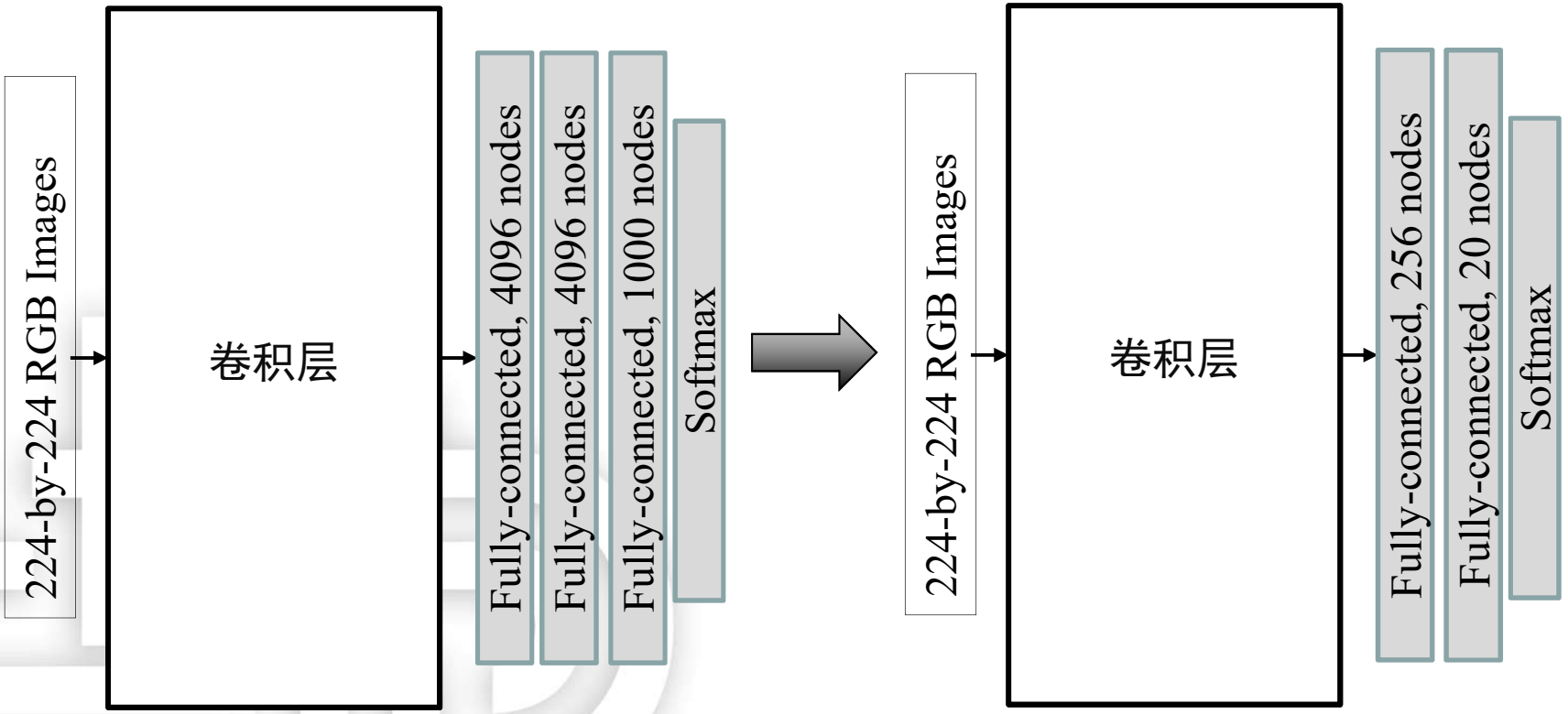
1. 迁移学习和微调技术

- 1. 不使用迁移学习：
  - 网络结构与VGG16相似，将输出层节点数改为20，并减小了全连接层的规模
  - 随机初始化所有网络参数
  - 实验结果：
    - 训练集分类错误率0%
    - 测试集分类错误率30.9%
    - 由于可调参数过多而数据量太少，出现严重的过拟合

1. 迁移学习和微调技术

- 2. 使用迁移学习：
  - 采用在ImageNet数据集上训练好的VGG16网络作为基础
  - 设计包含预训练模型的计算图，并进行参数初始化：
    - 输出层替换为20个节点，减小全连接层规模
    - 其他层与VGG16结构一致
    - 随机初始化全连接层和输出层，其他层用VGG16的模型参数初始化

1. 迁移学习和微调技术



- 卷积层不变
- 全连接层减少为一层，节点数变为256
- 输出层改为20个节点

1. 迁移学习和微调技术

- 加载VGG16模型

```
def __init__(self, vgg16_npy_path=None, restore_from=None):
    self.data_dict = np.load(vgg16_npy_path, encoding='latin1').item()   # np.load:
    numpy方法中的函数，用于加载文件。vgg16_npy_path为预训练模型路径

    self.tfx = tf.placeholder(tf.float32, [None, 224, 224, 3])
    self.tfy = tf.placeholder(tf.float32, [None, 20])
    # 定义输入输出的占位符

    conv1_1 = self.conv_layer(tfx, "conv1_1")
    conv1_2 = self.conv_layer(conv1_1, "conv1_2")
    pool1 = self.max_pool(conv1_2, 'pool1')
    ......
    conv5_1 = self.conv_layer(pool4, "conv5_1")
    conv5_2 = self.conv_layer(conv5_1, "conv5_2")
    conv5_3 = self.conv_layer(conv5_2, "conv5_3")
    pool5 = self.max_pool(conv5_3, 'pool5')
    # 利用VGG16的参数初始化所有卷积层。conv_layer方法的定义见下页
```

1. 迁移学习和微调技术

- 加载VGG16模型

```
def __init__(self, vgg16_npy_path=None, restore_from=None):
        self.data_dict = np.load(vgg16_npy_path, encoding='latin1').item()   # np.load:
        numpy方法中的函数，用于加载文件。vgg16_npy_path为预训练模型路径

        self.tfx = tf.placeholder(tf.float32, [None, 224, 224, 3])
        self.tfy = tf.placeholder(tf.float32, [None, 20])
        # 定义输入输出的占位符

        conv1_1 = self.conv_layer(tfx, "conv1_1")
        conv1_2 = self.conv_layer(conv1_1, "conv1_2")
        pool1 = self.max_pool(conv1_2, 'pool1')
        ......
        conv5_1 = self.conv_layer(pool4, "conv5_1")
        conv5_2 = self.conv_layer(conv5_1, "conv5_2")
        conv5_3 = self.conv_layer(conv5_2, "conv5_3")
        pool5 = self.max_pool(conv5_3, 'pool5')
        # 利用VGG16的参数初始化所有卷积层。conv_layer方法的定义见下页
```

1. 迁移学习和微调技术

- 加载VGG16模型

```
def __init__(self, vgg16_npy_path=None, restore_from=None):
        self.data_dict = np.load(vgg16_npy_path, encoding='latin1').item()   # np.load:
        numpy方法中的函数，用于加载文件。vgg16_npy_path为预训练模型路径

        self.tfx = tf.placeholder(tf.float32, [None, 224, 224, 3])
        self.tfy = tf.placeholder(tf.float32, [None, 20])
        # 定义输入输出的占位符

        conv1_1 = self.conv_layer(tfx, "conv1_1")
        conv1_2 = self.conv_layer(conv1_1, "conv1_2")
        pool1 = self.max_pool(conv1_2, 'pool1')
        ......
        conv5_1 = self.conv_layer(pool4, "conv5_1")
        conv5_2 = self.conv_layer(conv5_1, "conv5_2")
        conv5_3 = self.conv_layer(conv5_2, "conv5_3")
        pool5 = self.max_pool(conv5_3, 'pool5')
        # 利用VGG16的参数初始化所有卷积层。conv_layer方法的定义见下页
```

1. 迁移学习和微调技术

- 加载VGG16模型：重要方法

```
def conv_layer(self, bottom, name):
        with tf.variable_scope(name):
            filt = self.get_conv_filter(name)

            conv = tf.nn.conv2d(bottom, filt, [1, 1, 1, 1], padding='SAME')
            conv_biases = self.get_bias(name)
            bias = tf.nn.bias_add(conv, conv_biases)

            relu = tf.nn.relu(bias)
            return relu

def get_conv_filter(self, name):
        return tf.constant(self.data_dict[name][0], name="filter")

def get_bias(self, name):
        return tf.constant(self.data_dict[name][1], name="biases")
# 从预训练模型获取的参数均为常数，因此不会被重新随机初始化
```

1. 迁移学习和微调技术

- 加载VGG16模型：重要方法

```
def conv_layer(self, bottom, name):
        with tf.variable_scope(name):
            filt = self.get_conv_filter(name)

            conv = tf.nn.conv2d(bottom, filt, [1, 1, 1, 1], padding='SAME')
            conv_biases = self.get_bias(name)
            bias = tf.nn.bias_add(conv, conv_biases)

            relu = tf.nn.relu(bias)
            return relu

def get_conv_filter(self, name):
        return tf.constant(self.data_dict[name][0], name="filter")

def get_bias(self, name):
        return tf.constant(self.data_dict[name][1], name="biases")
# 从预训练模型获取的参数均为常数，因此不会被重新随机初始化
```

1. 迁移学习和微调技术

- 自定义全连接层和输出层，并进行随机初始化

```
self.flatten = tf.reshape(pool5, [-1, 7*7*512])
self.fc6 = tf.layers.dense(self.flatten, 256, tf.nn.relu, name='fc6')
self.out = tf.layers.dense(self.fc6, 20, name='out')
# 重定义全连接层和输出层。输出层改为20个节点

self.sess = tf.Session()

self.loss =        tf.nn.sparse_softmax_cross_entropy_with_logits(labels=self.tfy,
predictions=self.out) # 使用交叉熵目标函数，配合softmax函数实现分类功
能

self.train_op = tf.train.RMSPropOptimizer(0.0005).minimize(self.loss)
#定义训练方法
self.sess.run(tf.global_variables_initializer())
# 初始化全连接层和输出层(仅对非constant的参数起作用）
```

1. 迁移学习和微调技术

- 自定义全连接层和输出层，并进行随机初始化

```
self.flatten = tf.reshape(pool5, [-1, 7*7*512])
self.fc6 = tf.layers.dense(self.flatten, 256, tf.nn.relu, name='fc6')
self.out = tf.layers.dense(self.fc6, 20, name='out')
# 重定义全连接层和输出层。输出层改为20个节点

self.sess = tf.Session()

self.loss =
    tf.nn.sparse_softmax_cross_entropy_with_logits(labels=self.tfy,
    predictions=self.out) # 使用交叉熵目标函数，配合softmax函数实现分类功
    能

self.train_op = tf.train.RMSPropOptimizer(0.0005).minimize(self.loss)
#定义训练方法
self.sess.run(tf.global_variables_initializer())
# 初始化全连接层和输出层(仅对非constant的参数起作用）
```

1. 迁移学习和微调技术

- 自定义全连接层和输出层，并进行随机初始化

```
self.flatten = tf.reshape(pool5, [-1, 7*7*512])
self.fc6 = tf.layers.dense(self.flatten, 256, tf.nn.relu, name='fc6')
self.out = tf.layers.dense(self.fc6, 20, name='out')
# 重定义全连接层和输出层。输出层改为20个节点

self.sess = tf.Session()

self.loss =
    tf.nn.sparse_softmax_cross_entropy_with_logits(labels=self.tfy,
    predictions=self.out) # 使用交叉熵目标函数，配合softmax函数实现分类功
    能

self.train_op = tf.train.RMSPropOptimizer(0.0005).minimize(self.loss)
#定义训练方法
self.sess.run(tf.global_variables_initializer())
# 初始化全连接层和输出层(仅对非constant的参数起作用）
```

1. 迁移学习和微调技术

- 开始微调训练：
  - 方法1. 仅更新全连接层权值，卷积层权值保持不变
  - 方法2. 采用相对较大的学习率（0.0005）训练全连接层，相对较小的学习率微调卷积层（0.0001）
  - 方法3. 采用相同的较大学习率（0.0005）微调所有层

1. 迁移学习和微调技术

- 测试集分类误差：

	Top1误差	Top5误差
不使用迁移学习	30.9%	4.2%
迁移学习方法1	19.1%	1.79%
迁移学习方法2	20.4%	2.08%
迁移学习方法3	16.7%	1.53%

- 使用迁移学习可以大大提高分类性能