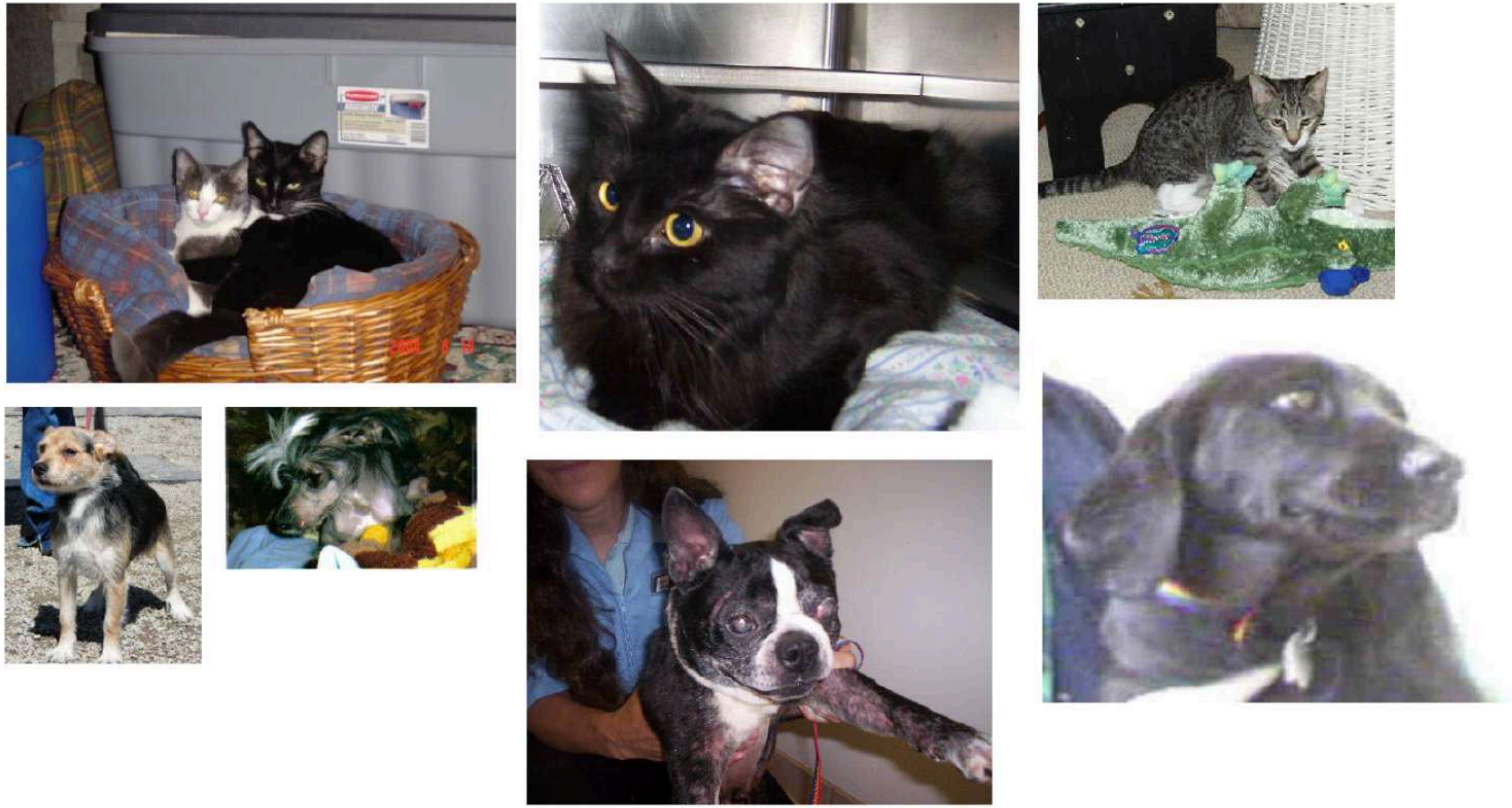
你必须知道的技巧

王树森

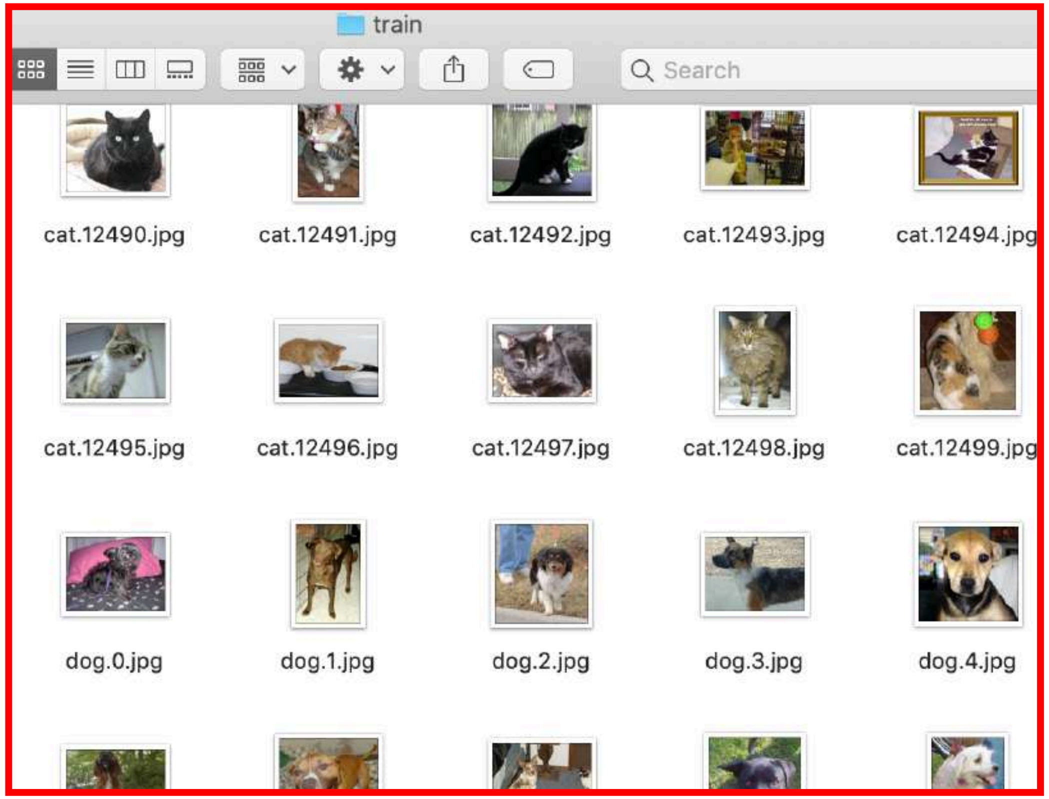
The Dogs vs. Cats Dataset

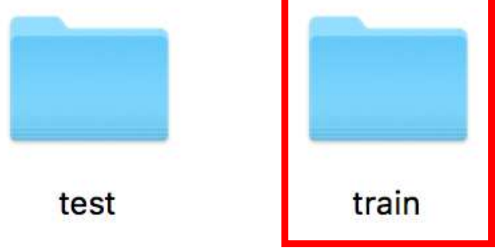
The Dogs vs. Cats Dataset



The Dogs vs. Cats Dataset

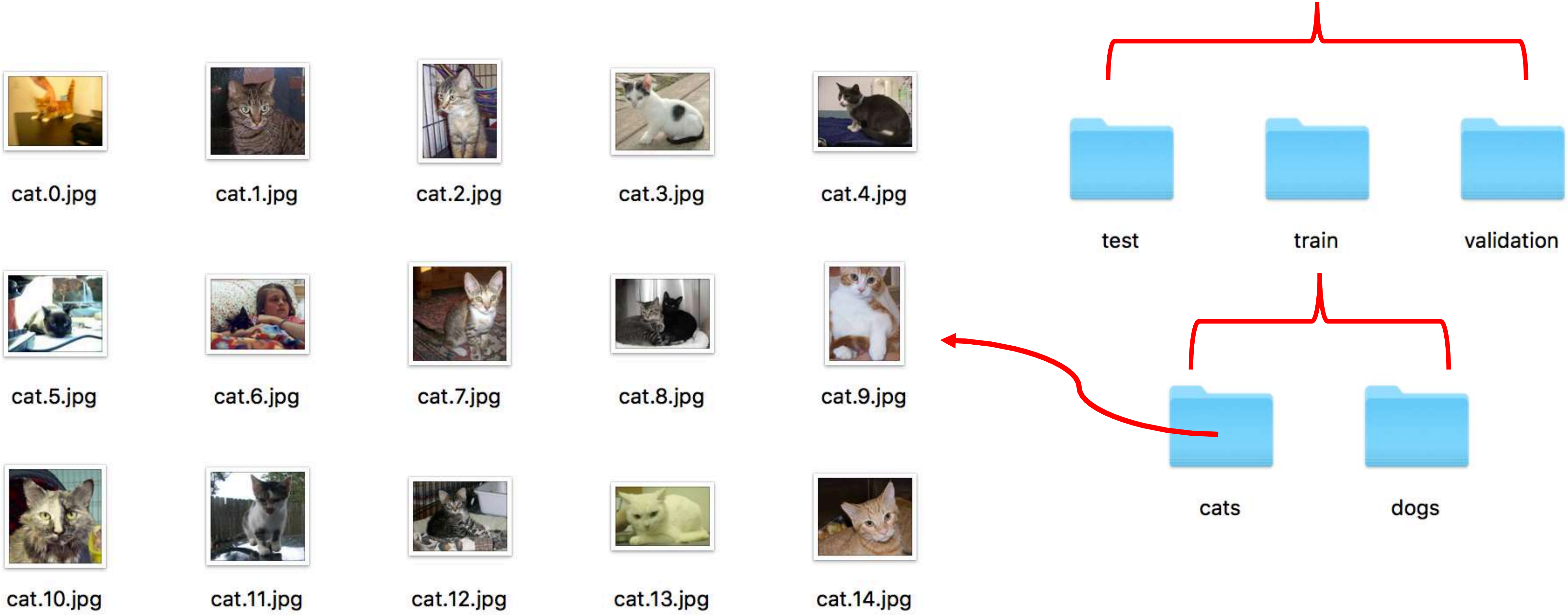
Download: https://www.kaggle.com/c/dogs-vs-cats/data





The Dogs vs. Cats Dataset

Download: https://www.kaggle.com/c/dogs-vs-cats/data



The Dogs vs. Cats Dataset

- 数据集有 25000 个样本.
- 使用一个子集:
  - 2000 images for 训练,
  - 1000 images for 验证,
  - 1000 images for 测试.

用 keras 实现一个CNN

1. Load and Process the Dataset

- 当前，文件以JPEG格式呈现
- 数据处理：
  1. 读取数据文件
  2. 把JPEG文件解码成三阶张量
  3. 把文件调整为相同大小 150×150×3
  4. 将像素值（在 0 到 255 之间）重新缩放到 [0, 1] 区间

1. Load and Process the Dataset

```
from keras.preprocessing.image import ImageDataGenerator

# All images will be rescaled by 1./255
train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)
```

1. Load and Process the Dataset

```
from keras.preprocessing.image import ImageDataGenerator

# All images will be rescaled by 1./255
train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        # This is the target directory
        train_dir,
        # All images will be resized to 150x150
        target_size=(150, 150),
        batch_size=20,
        # Since we use binary_crossentropy loss, we need binary labels
        class_mode='binary')
```

1. Load and Process the Dataset

```
for data_batch, labels_batch in train_generator:
    print('data batch shape:', data_batch.shape)
    print('labels batch shape:', labels_batch.shape)
    break
```

```
data batch shape: (20, 150, 150, 3)
labels batch shape: (20,)
```

1. Load and Process the Dataset

Disk → **Memory** → **GPU**

1. Load and Process the Dataset

Disk → **Memory** → **GPU**

2. Build the CNN

```
from keras import layers
from keras import models

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                        input_shape=(150, 150, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

2. Build the CNN

```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_1 (Conv2D)            (None, 148, 148, 32)      896       
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 74, 74, 32)        0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 72, 72, 64)        18496     
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 36, 36, 64)        0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 34, 34, 128)       73856     
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 17, 17, 128)       0         
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 15, 15, 128)       147584    
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 7, 7, 128)         0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 6272)              0         
_________________________________________________________________
dense_1 (Dense)              (None, 512)               3211776   
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 513       
=================================================================
Total params: 3,453,121
Trainable params: 3,453,121
Non-trainable params: 0
_________________________________________________________________
```

3. Train the CNN

指定：优化方法、学习率（LR）、损失函数和评估指标。

```
from keras import optimizers

model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-4),
              metrics=['acc'])
```

3. Train the CNN

```
history = model.fit_generator(
      train_generator,
      steps_per_epoch=100,
      epochs=30,
      validation_data=validation_generator,
      validation_steps=50)
```

3. Train the CNN

```
history = model.fit_generator(
        train_generator,
        steps_per_epoch=100,
        epochs=30,
        validation_data=validation_generator,
        validation_steps=50)
```

- Totally $n = 2000$ training samples.
- Batch size is $b = 20$.
- Thus $\frac{n}{b} = 100$ batches per epoch.

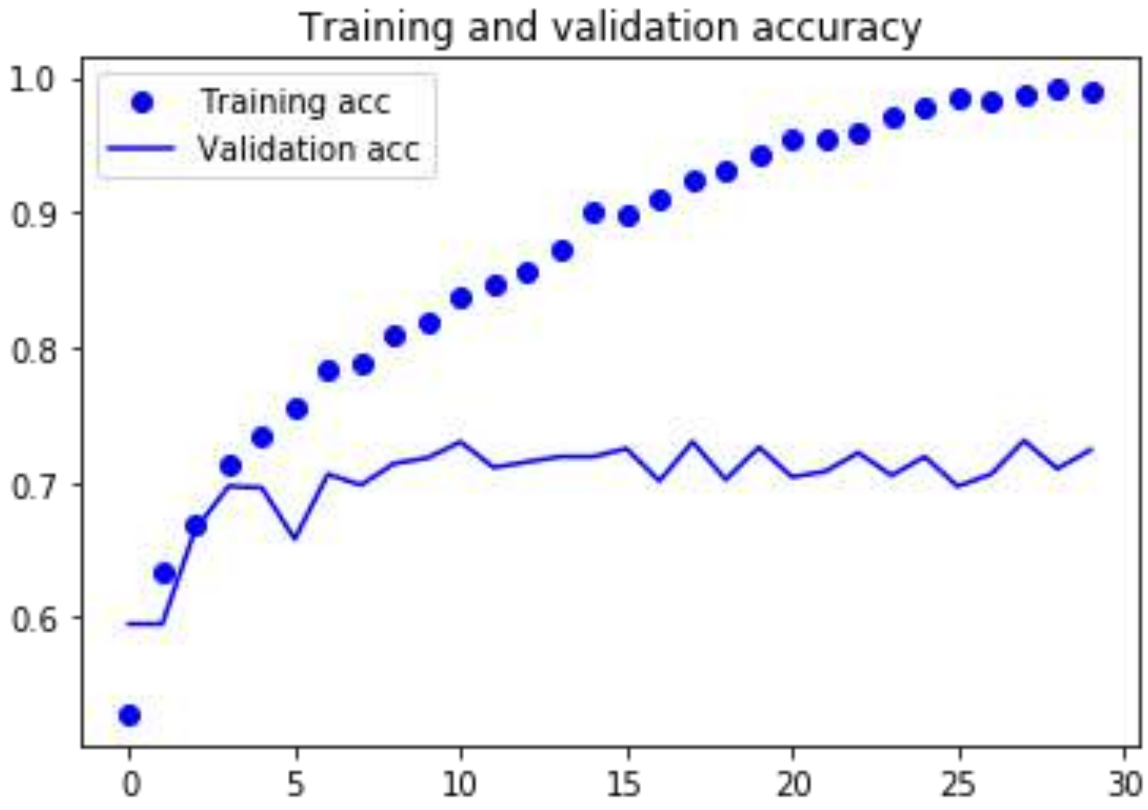
```
Epoch 1/30
100/100 [==============================] - 9s - loss: 0.6898 - acc: 0.5285 - val_loss: 0.6724 - val_acc: 0.5950
Epoch 2/30
100/100 [==============================] - 8s - loss: 0.6543 - acc: 0.6340 - val_loss: 0.6565 - val_acc: 0.5950
Epoch 3/30
100/100 [==============================] - 8s - loss: 0.6143 - acc: 0.6690 - val_loss: 0.6116 - val_acc: 0.6650
Epoch 4/30
100/100 [==============================] - 8s - loss: 0.5626 - acc: 0.7125 - val_loss: 0.5774 - val_acc: 0.6970
```

•
•
•

```
Epoch 29/30
100/100 [==============================] - 8s - loss: 0.0375 - acc: 0.9915 - val_loss: 0.9987 - val_acc: 0.7100
Epoch 30/30
100/100 [==============================] - 8s - loss: 0.0387 - acc: 0.9895 - val_loss: 1.0139 - val_acc: 0.7240
```

4. Examine the Results

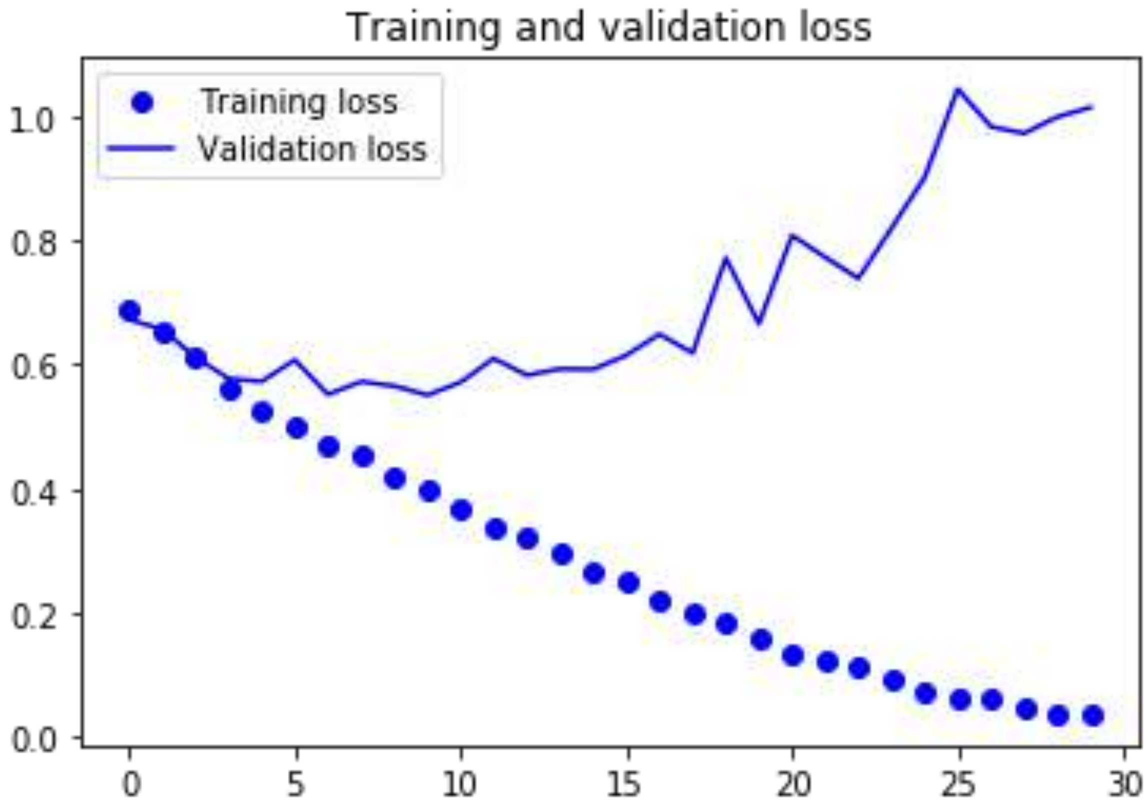
Plot the ***accuracy*** against ***epochs*** (1 epoch = 1 pass over the data).



- 训练准确率: 99.0%
- 测试准确率: 72.4%
- 好像过拟合了

4. Examine the Results

Plot the ***loss*** against ***epochs*** (1 epoch = 1 pass over the data).



- 训练损失函数持续下降
- 验证损失函数下降后上升

Why Overfitting?

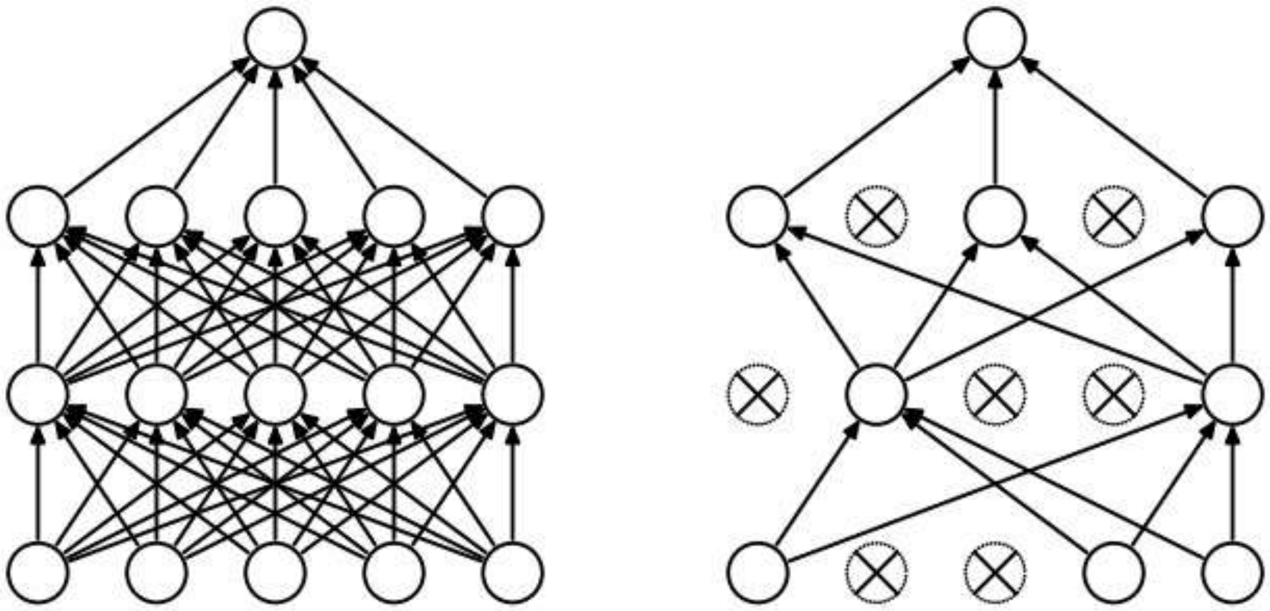
```
================================
Total params: 3,453,121
Trainable params: 3,453,121
Non-trainable params: 0
________________________________
```

超过 $3M$ 个参数; 但是只有 $2K$ 个训练样本.
过拟合并不意外

Trick 1: Dropout

Dropout: 基本概念

- Train
  - 在训练的每次迭代（1 次前向 + 1 次后向）中，随机屏蔽 50%（或任意百分比）的神经元。



(a) Standard Neural Net   (b) After applying dropout.

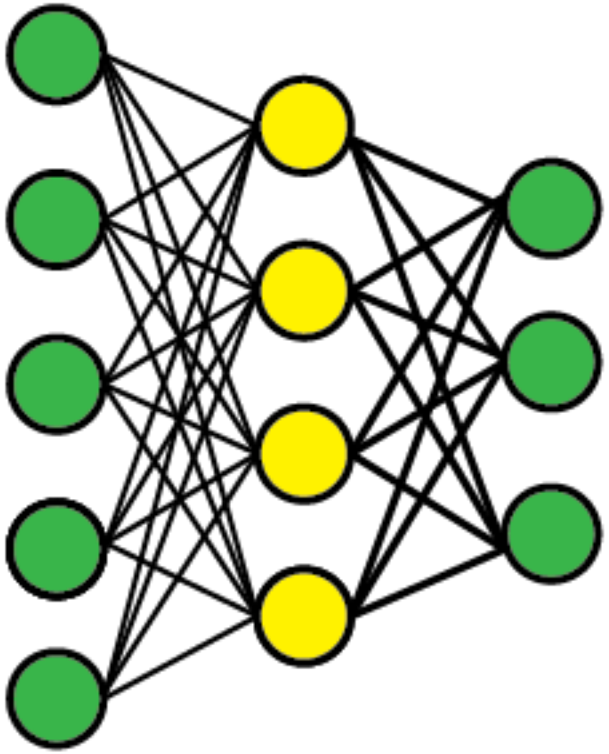
Dropout: 基本概念

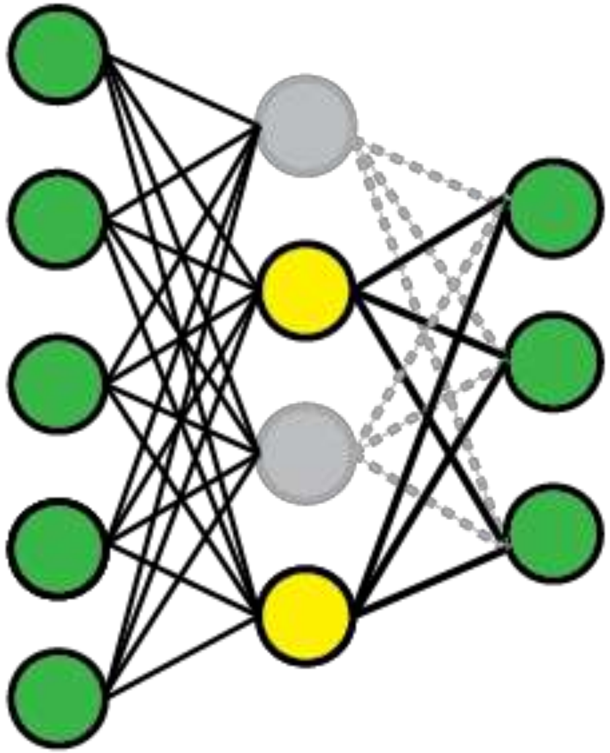
- Train
  - 在训练的每次迭代（1 次前向 + 1 次后向）中，随机屏蔽 50%（或任意百分比）的神经元。
- Prediction
  - 不使用 dropout
  - 使用所有的参数

Dropout: 实现

在这层执行dropout

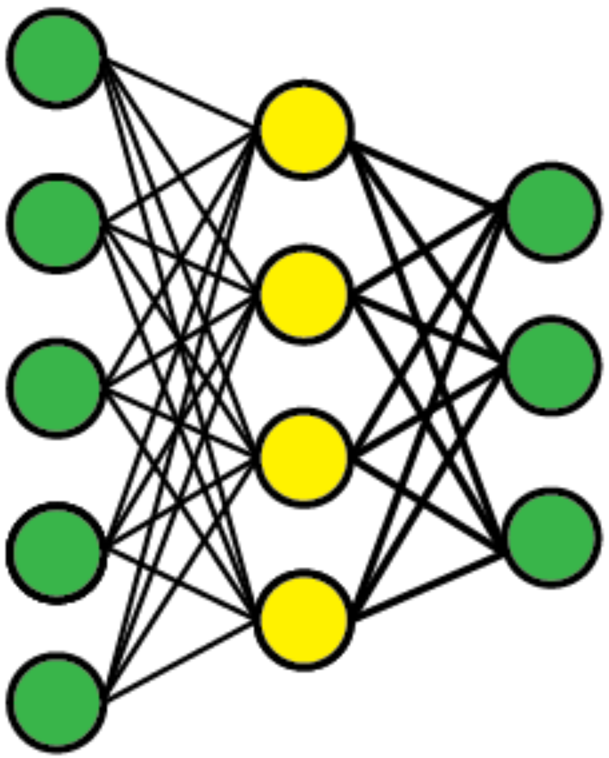
- **For a batch of training samples**
  - 随机选择50%的神经元
  - 把这些神经元设置为0
  - 将未被选中的神经元乘以 $\frac{1}{0.5} = 2.$



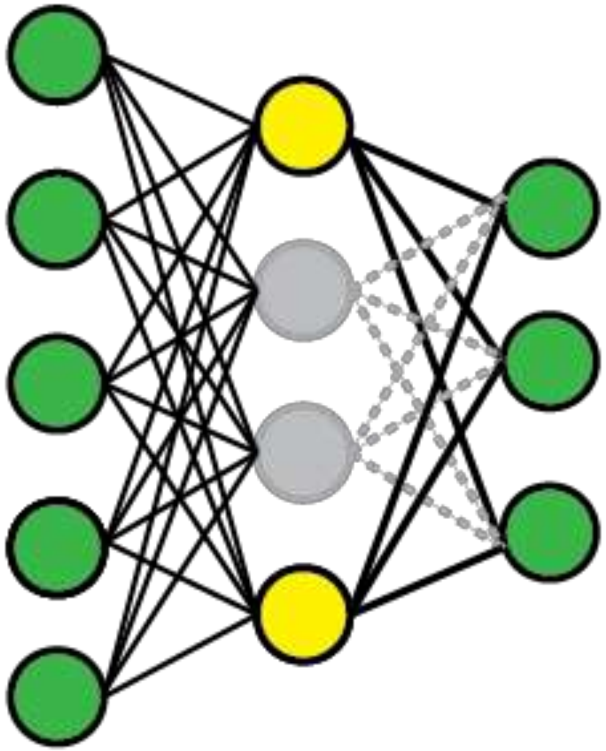


Dropout: 实现

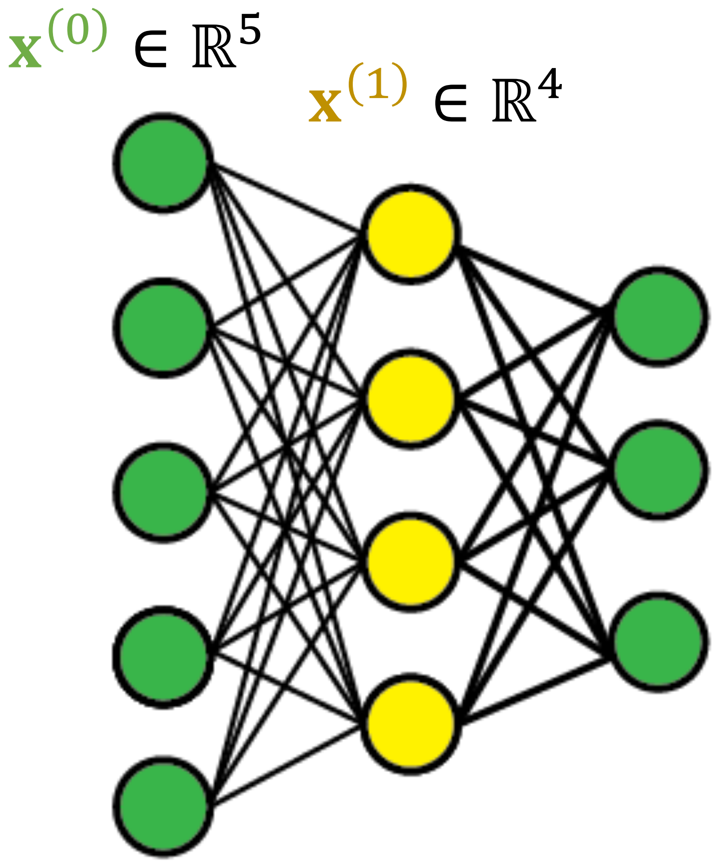
在这层执行dropout



- **For a batch of training samples…**
- 对于另一个批次，进行一次独立的随机抽样（即随机屏蔽神经元）



Dropout: 实现



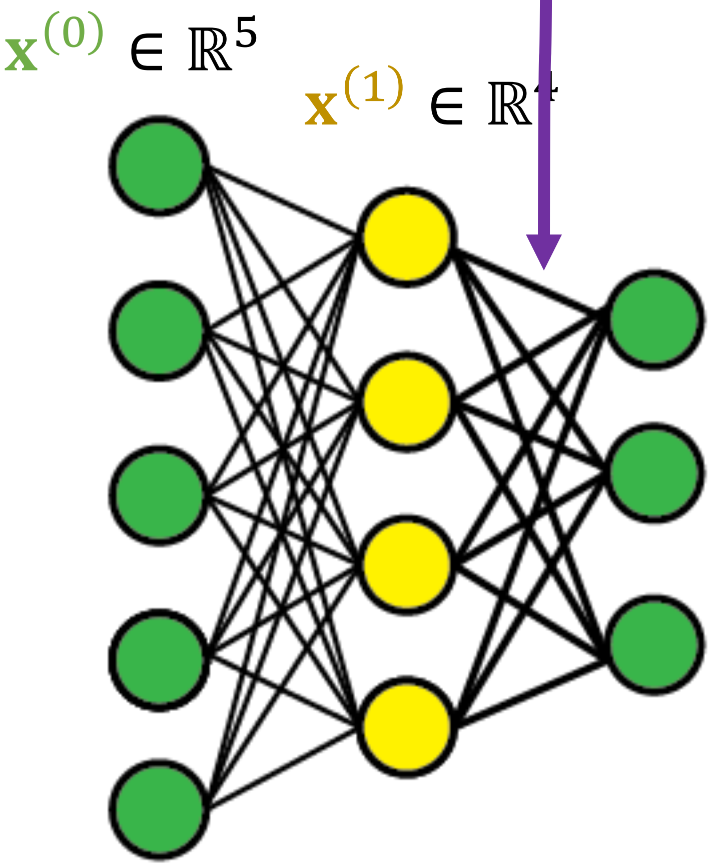
- **Input**: vector $\mathbf{x}^{(0)} \in \mathbb{R}^5$.

- $\mathbf{z}^{(1)} = \mathbf{W}^{(\mathbf{0})}\,\mathbf{x}^{(0)} \in \mathbb{R}^4$.
- $\mathbf{x}^{(1)} = \max\{\mathbf{0},\ \mathbf{z}^{(1)}\} \in \mathbb{R}^4$.

- $\mathbf{z}^{(2)} = \mathbf{W}^{(1)}\,\mathbf{x}^{(1)} \in \mathbb{R}^3$.
- **Output**: $\mathrm{SoftMax}\big(\mathbf{z}^{(2)}\big) \in \mathbb{R}^3$.

Dropout: 实现

这层执行正则化
(防止过拟合的技术就叫正则化)



- **Input**: vector $\mathbf{x}^{(0)} \in \mathbb{R}^5$.
- $\mathbf{z}^{(1)} = \mathbf{W}^{(0)} \mathbf{x}^{(0)} \in \mathbb{R}^4$.
- $\mathbf{x}^{(1)} = \max\{\mathbf{0}, \mathbf{z}^{(1)}\} \in \mathbb{R}^4$.
- Add a dropout layer
- $\mathbf{z}^{(2)} = \mathbf{W}^{(1)} \tilde{\mathbf{x}}^{(1)} \in \mathbb{R}^3$.
- **Output**: $\mathrm{SoftMax}\left(\mathbf{z}^{(2)}\right) \in \mathbb{R}^3$.

- $\mathbf{m} \in \mathbb{R}^4$ 是一个随机向量 (Each entry is 0 or 2, w.p. 50%.)
- Apply $\mathbf{m}$ to $\mathbf{x}^{(1)}$:

$$\tilde{\mathbf{x}}^{(1)} = \mathbf{m} \circ \mathbf{x}^{(1)}.$$

"$\circ$" is 逐元素乘法

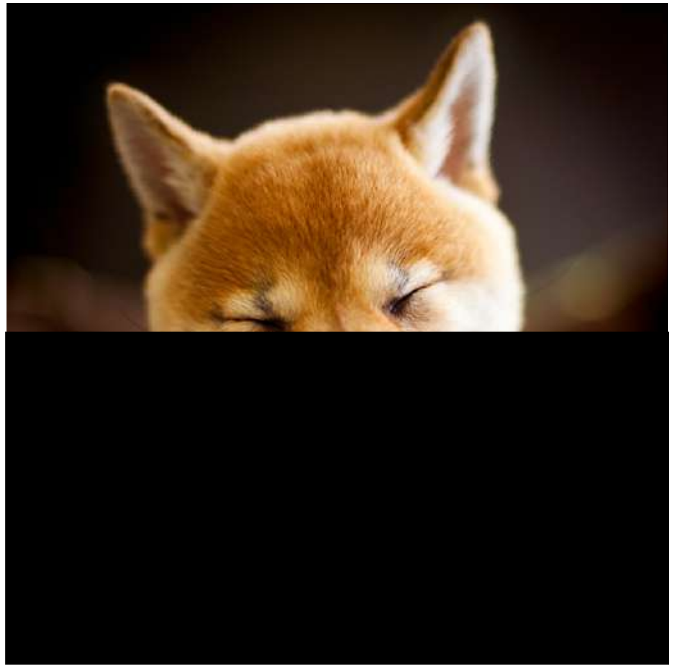
Keras’s Dropout Layer

- 仅在第一个全连接层之前使用 Dropout，以正则化第一个全连接层。
- 因为第一个全连接层有太多的可训练参数。

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                        input_shape=(150, 150, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dropout(0.5))
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-4),
              metrics=['acc'])
```

Why Does Dropout Work?

- 在训练中，Dropout 强制网络根据部分特征进行决策。



Why Does Dropout Work?

- 在训练中，Dropout 强制网络根据部分特征进行决策。

- Dropout 是一种正则化机制 [1].
  - 缓解过拟合。
  - 类似于 L1 和 L2 范数正则化。
  - 但 Dropout 在经验上效果更好。

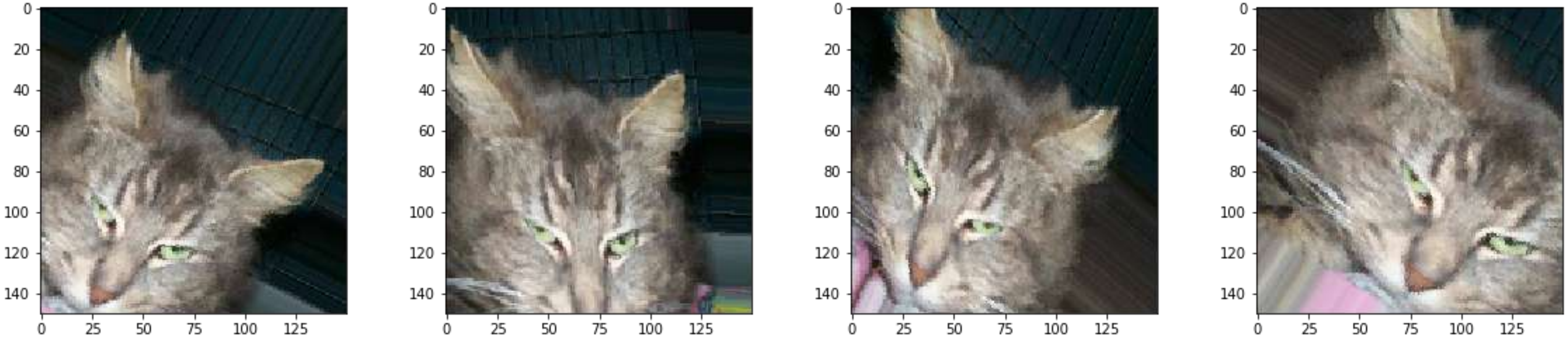
Reference:

1. Wager, Wang, & Liang. Dropout Training as Adaptive Regularization. In *NIPS*, 2013.

技巧 2：数据增强

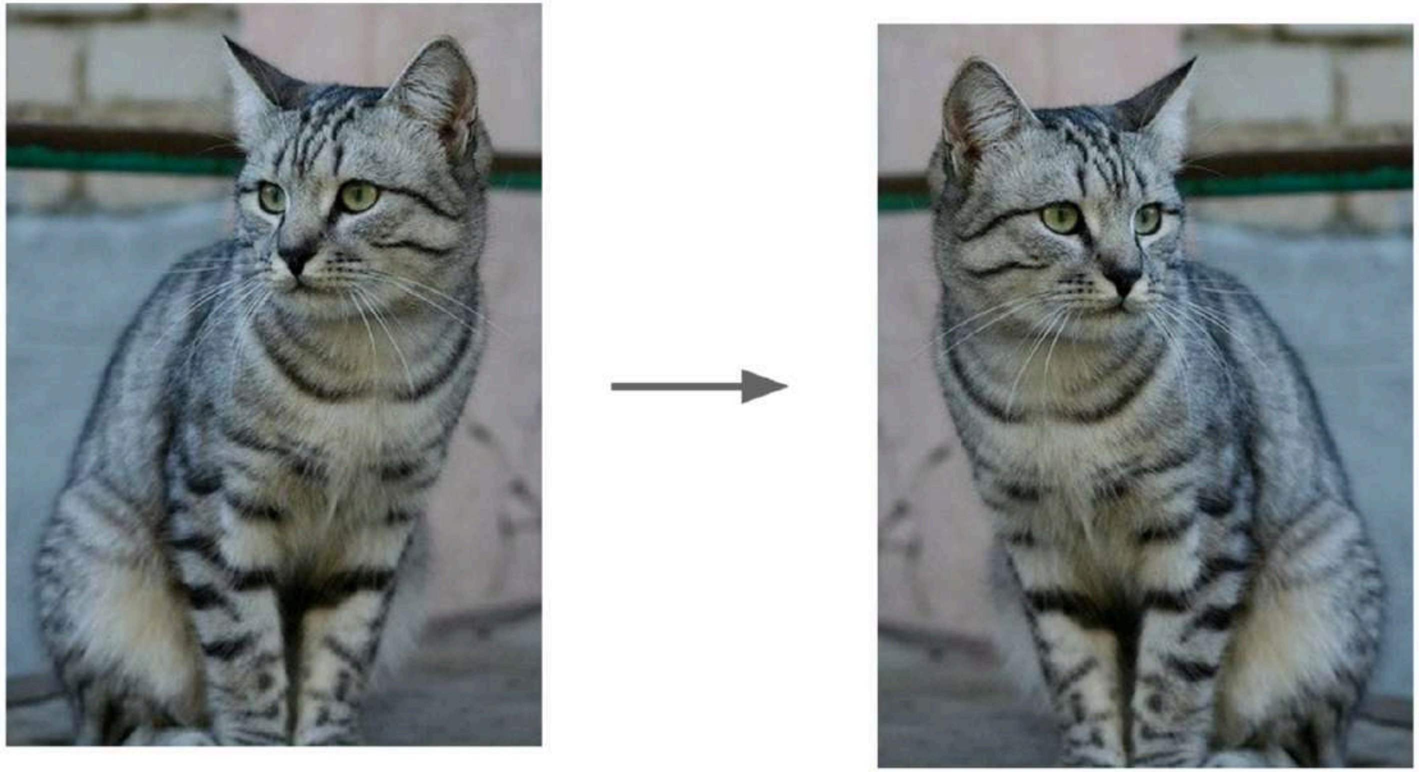
数据增强

- 数据增强：从现有训练数据生成更多训练样本。
- 例如，翻转、旋转、裁剪、平移、添加随机噪声。



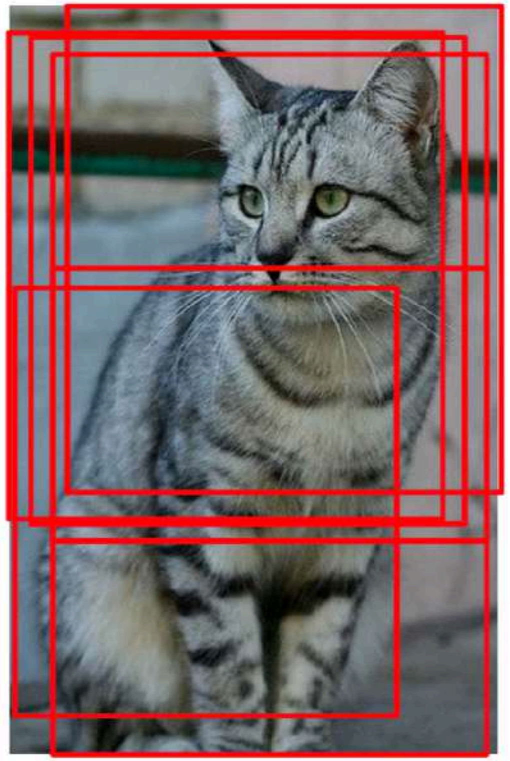
数据增强：样本

水平翻转



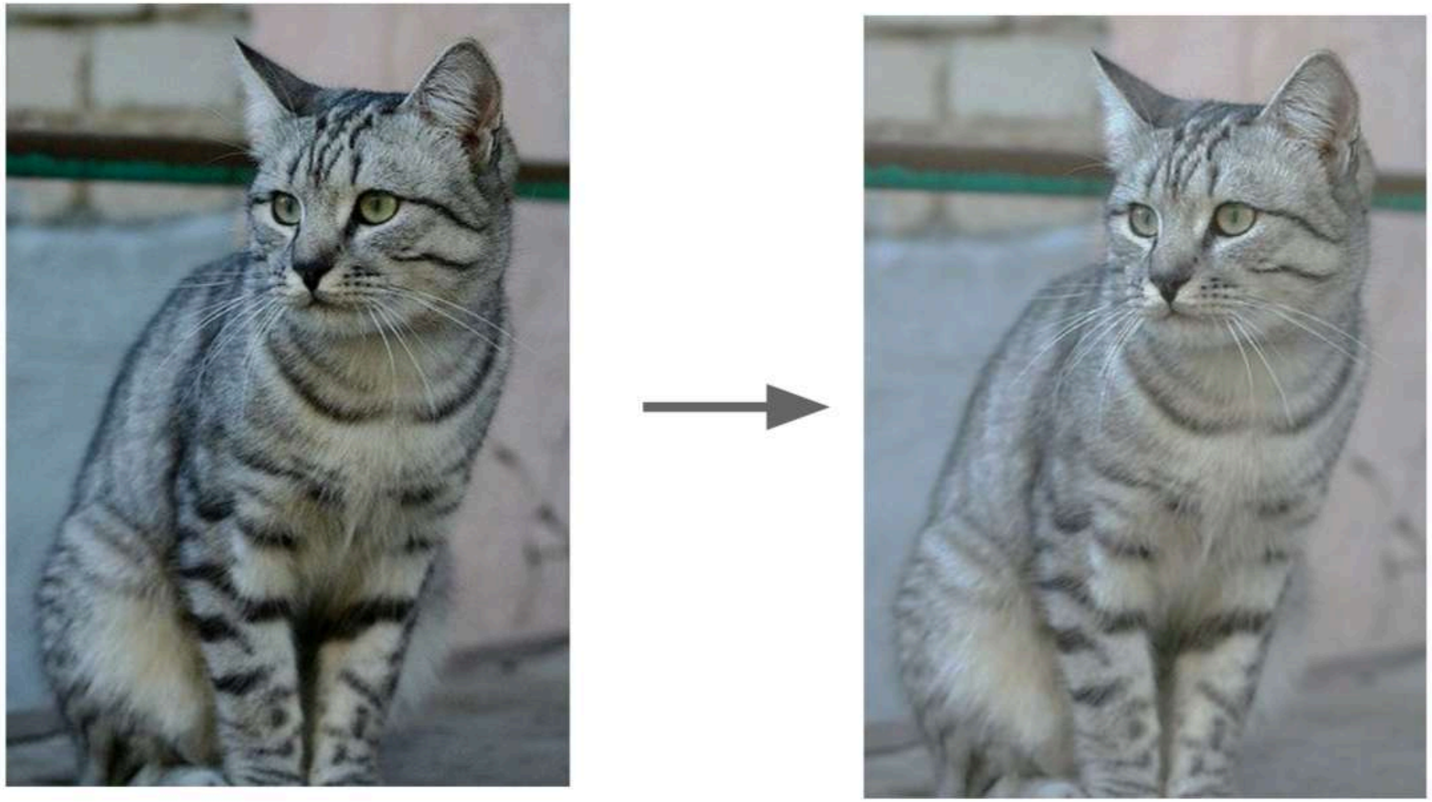
数据增强：样本

随机裁剪和缩放



数据增强：样本

颜色抖动（随机调整对比度和亮度）



Setup Data Augmentation Using Keras

```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,)
```

```
train_generator = train_datagen.flow_from_directory(
        # This is the target directory
        train_dir,
        # All images will be resized to 150x150
        target_size=(150, 150),
        batch_size=32,
        # Since we use binary_crossentropy loss, we need binary labels
        class_mode='binary')
```

Train the CNN

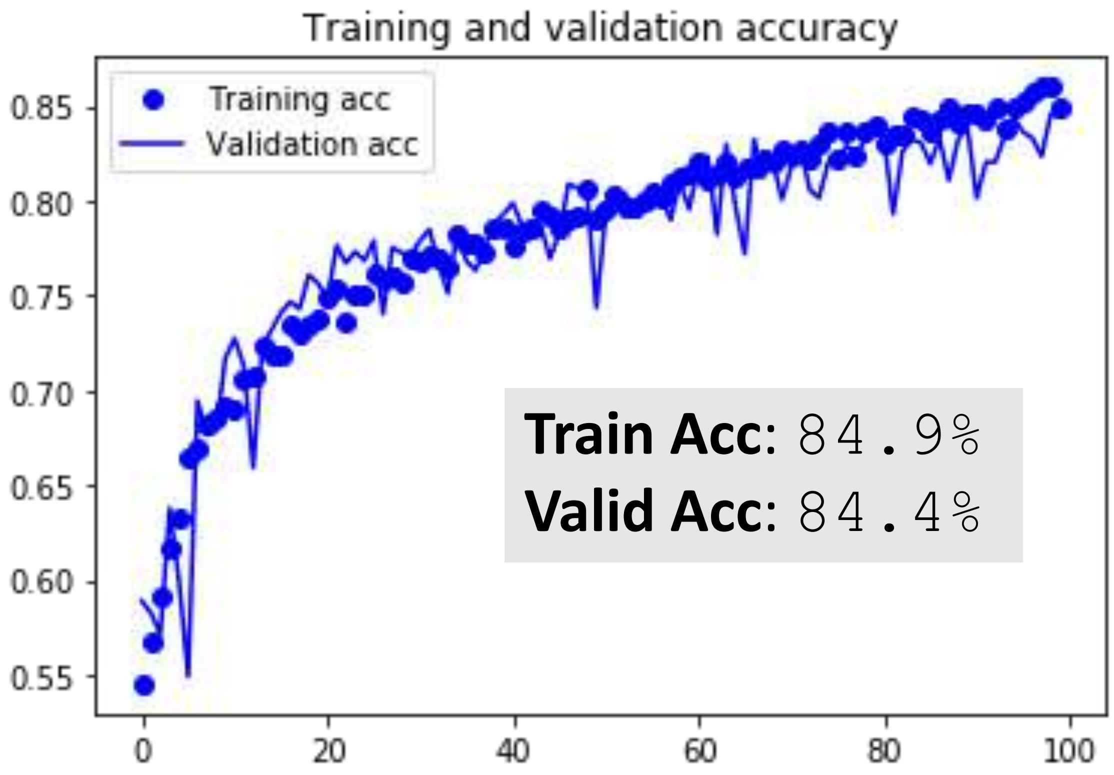
```
history = model.fit_generator(
      train_generator,
      steps_per_epoch=100,
      epochs=100,
      validation_data=validation_generator,
      validation_steps=50)
```

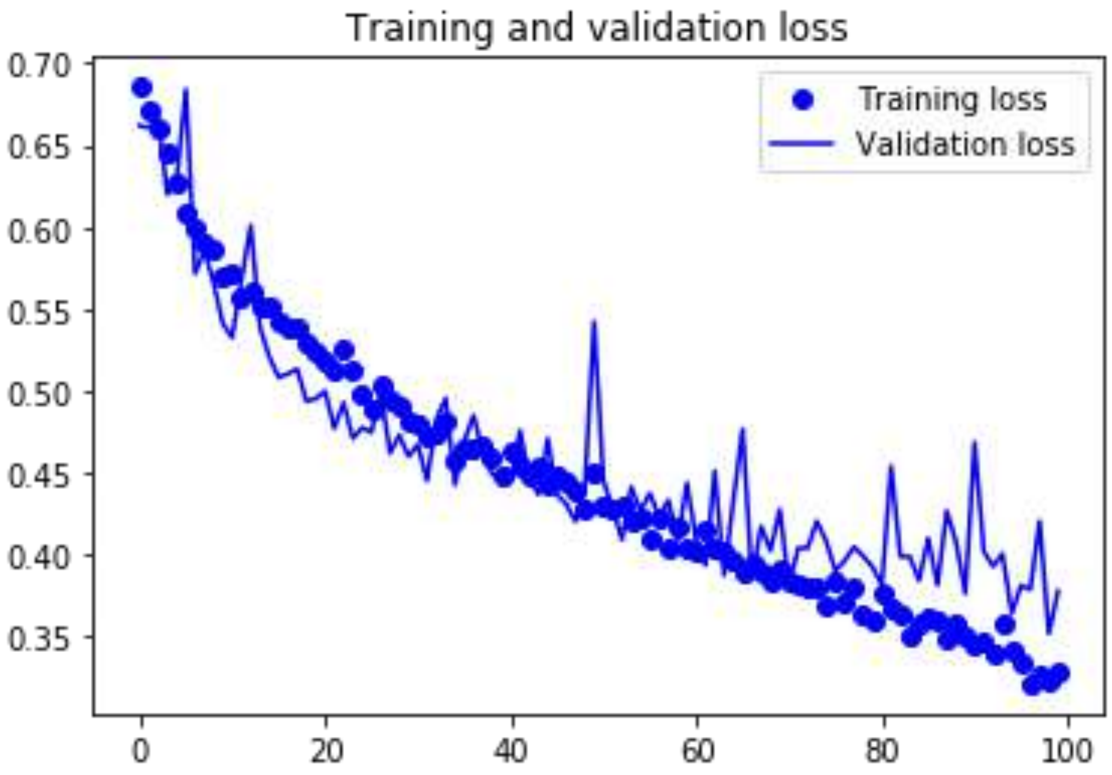
```
Epoch 1/100
100/100 [==============================] - 24s - loss: 0.6857 - acc: 0.5447 - val_loss: 0.6620 - val_acc: 0.5888
Epoch 2/100
100/100 [==============================] - 23s - loss: 0.6710 - acc: 0.5675 - val_loss: 0.6606 - val_acc: 0.5825
Epoch 3/100
100/100 [==============================] - 22s - loss: 0.6609 - acc: 0.5913 - val_loss: 0.6663 - val_acc: 0.5711
   •
   •
   •
Epoch 99/100
100/100 [==============================] - 22s - loss: 0.3255 - acc: 0.8581 - val_loss: 0.3518 - val_acc: 0.8460
Epoch 100/100
100/100 [==============================] - 22s - loss: 0.3280 - acc: 0.8491 - val_loss: 0.3776 - val_acc: 0.8439
```

Examine the Results

accuracy **against** ***epochs***

loss **against** ***epochs***





要点总结

- 要训练用于图像的卷积神经网络（ConvNet），始终使用数据增强.
- 它可以免费为你提供更多数据!

- 如果某一层有太多参数，在其之前添加一个 Gropout 层
- 正则化可以防止过拟合
- 训练速度会稍微变慢

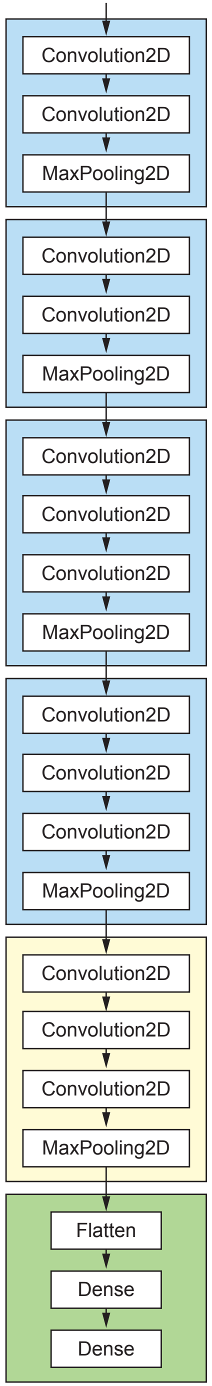
技巧3:预训练

Train a Deep Neural Network?

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                        input_shape=(150, 150, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dropout(0.5))
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-4),
              metrics=['acc'])
```

- 我们已经训练了一个具有 4 个卷积层和 2 个全连接层的神经网络
- 相对较浅

Train a Deep Neural Network?



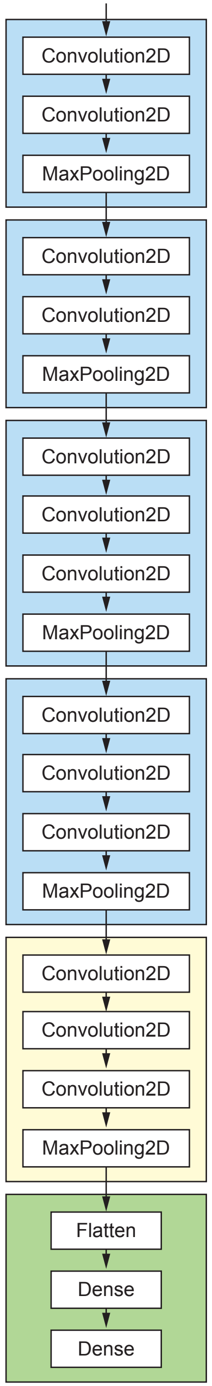
VGG16 网络

我们可以训练一个深层神经网络吗？

- 难度很大
  - 训练的参数数量非常大
  - 深层网络表达能力非常强
  - 我们只有 2555 个训练样本
- 单纯地训练一个深层网络肯定会导致过拟合

解决方案：预训练

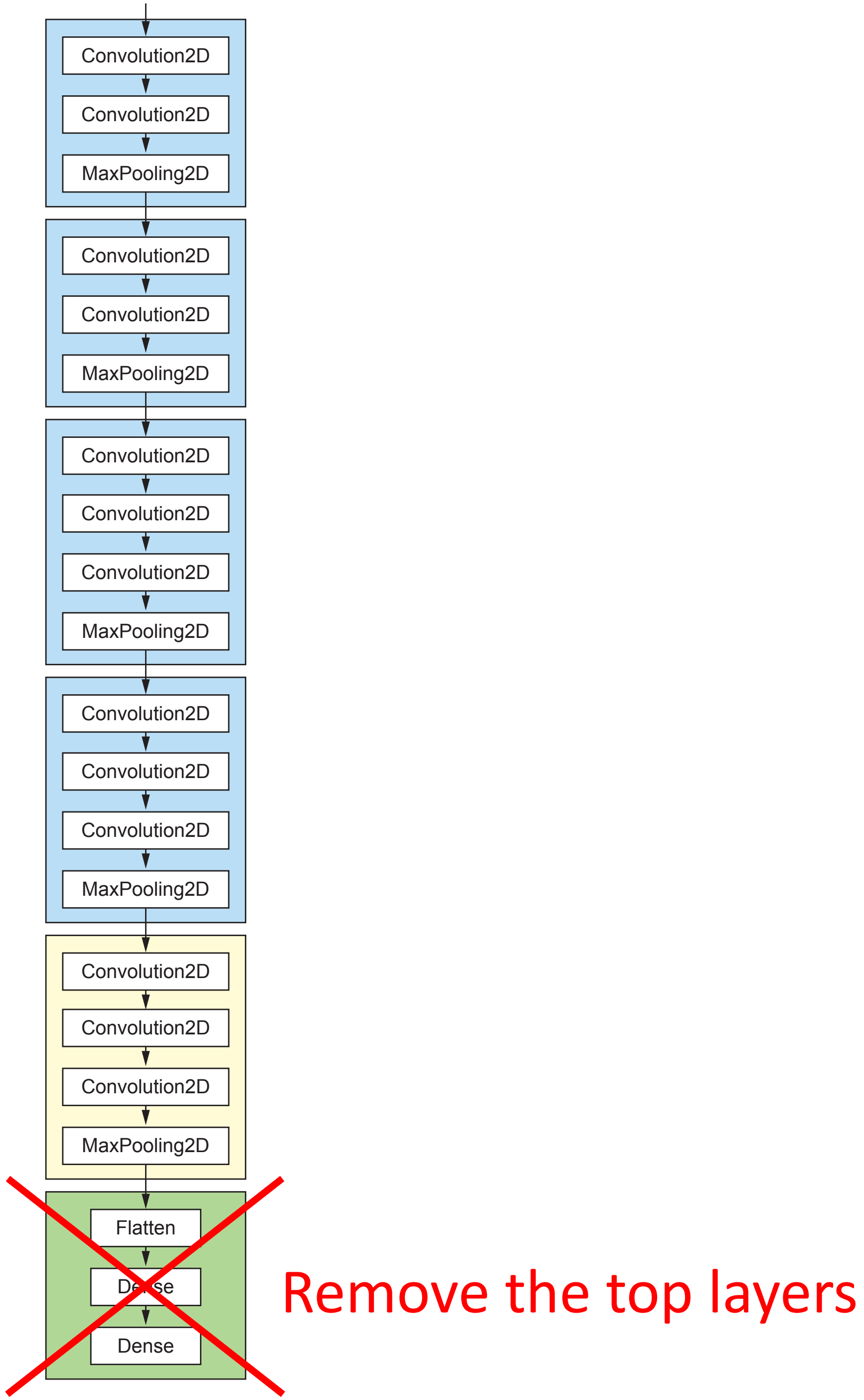
预训练



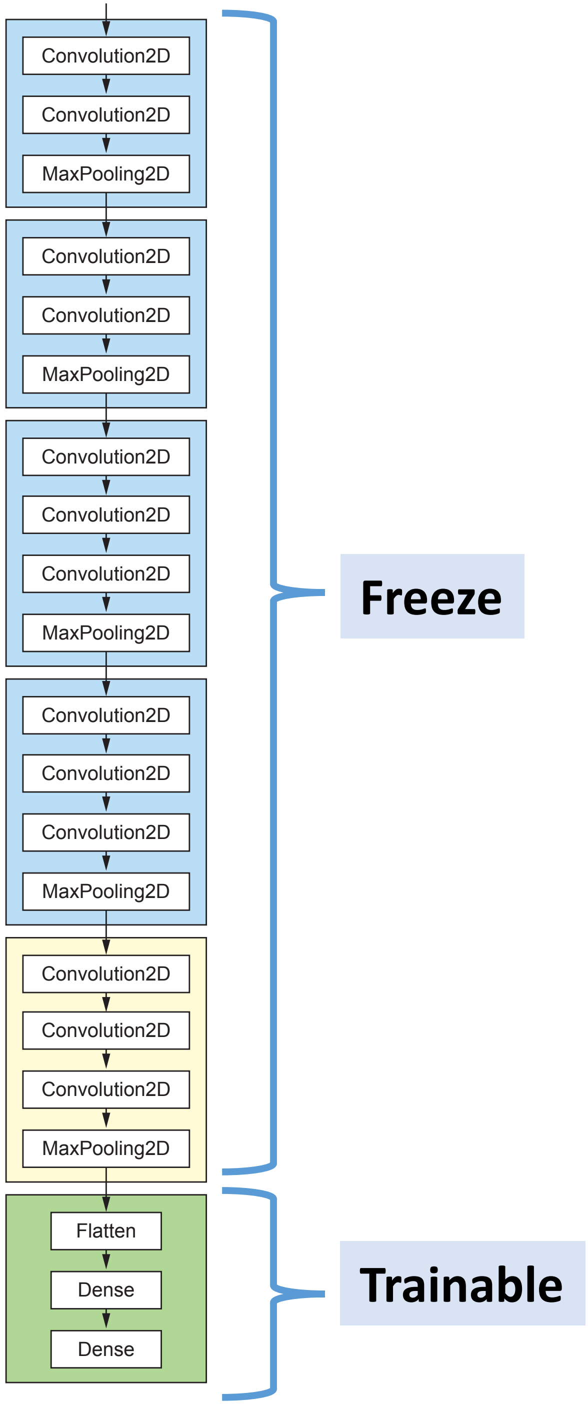
VGG16 网络

1. 在大规模数据集上预训练一个深层网络，例如，ImageNet（1400 万张带标签的图像）。

预训练

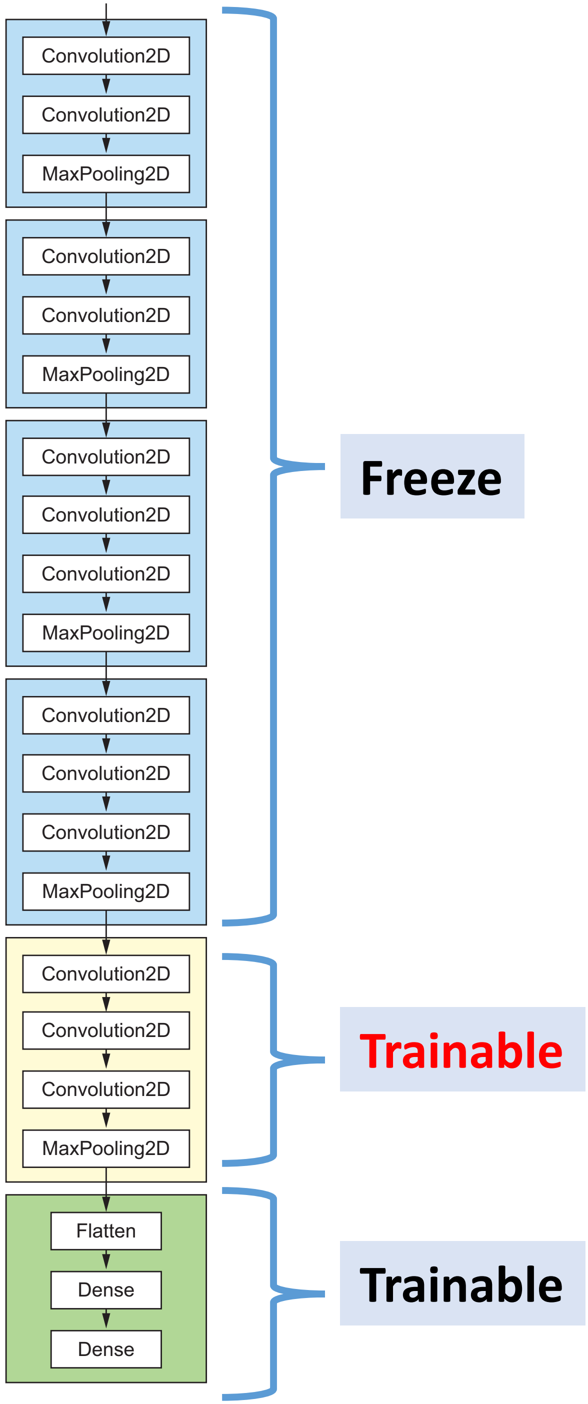


1. 在大规模数据集上预训练一个深层网络，例如，ImageNet（1i55 万张带标签的图像）。
2. 为什么只移除最顶层？
   - 不同的输出形状和激活函数
   - 新的分类器专用于某种场景



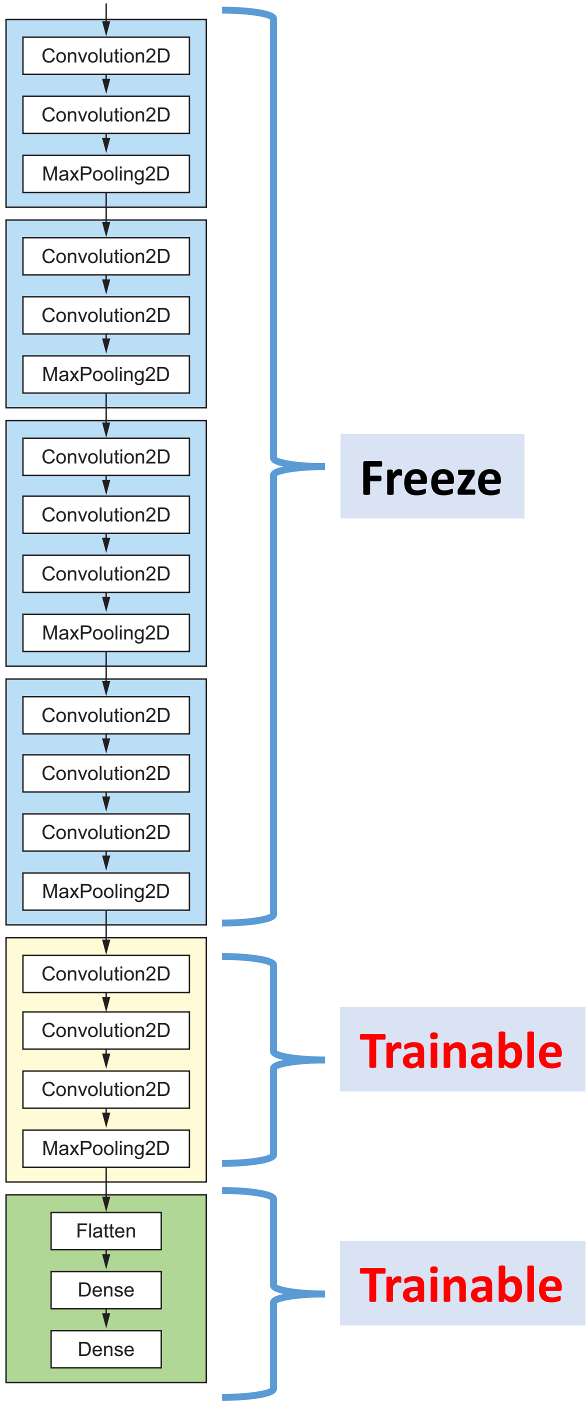
预训练

1. 在大规模数据集上预训练一个深层网络，例如，ImageNet（1400 万张带标签的图像）。
2. 移除最顶层
3. 搭建新的最顶层 (随机初始化).
4. 冻结其他层; 只训练最顶层



预训练

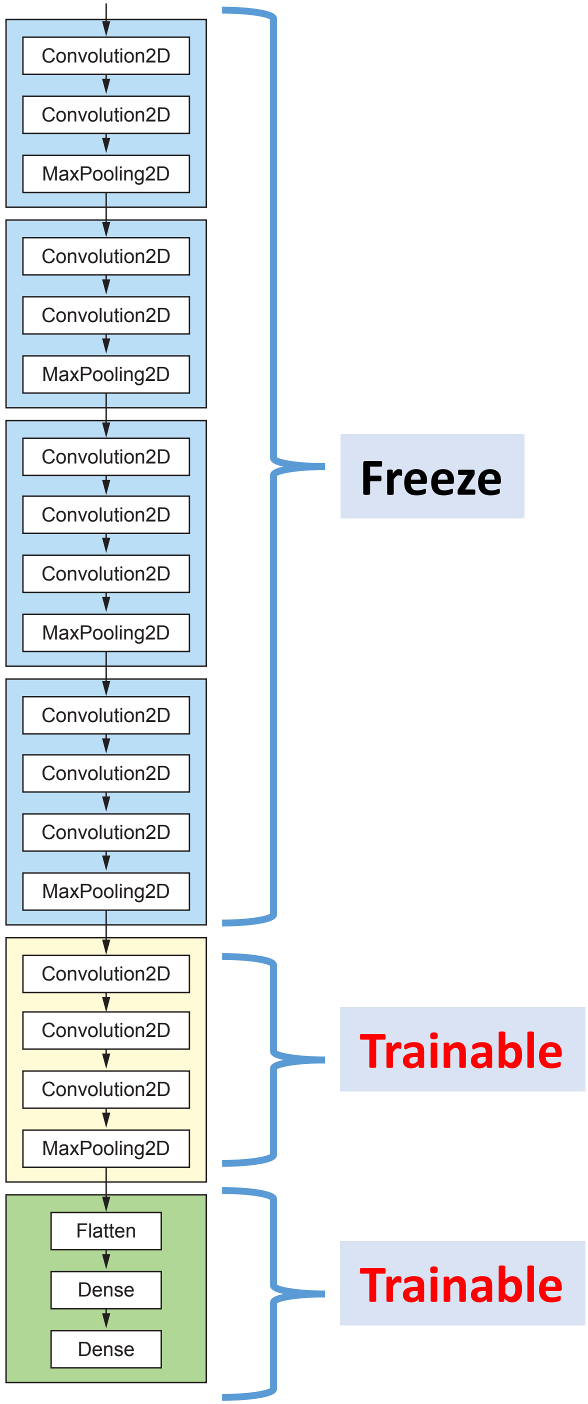
1. 在大规模数据集上预训练一个深层网络，例如，ImageNet（1455 万张带标签的图像）。
2. 移除最顶层
3. 搭建新的最顶层 (随机初始化).
4. 冻结其他层; 只训练最顶层
5. 可选: **微调** 高层卷积层



预训练

1. 在大规模数据集上预训练一个深层网络，例如，ImageNet（1455 万张带标签的图像）。
2. 移除最顶层
3. 搭建新的最顶层 (随机初始化).
4. 冻结其他层; 只训练最顶层
5. 可选: 微调 高层卷积层

问：步骤 4 & 5 是否能合并?



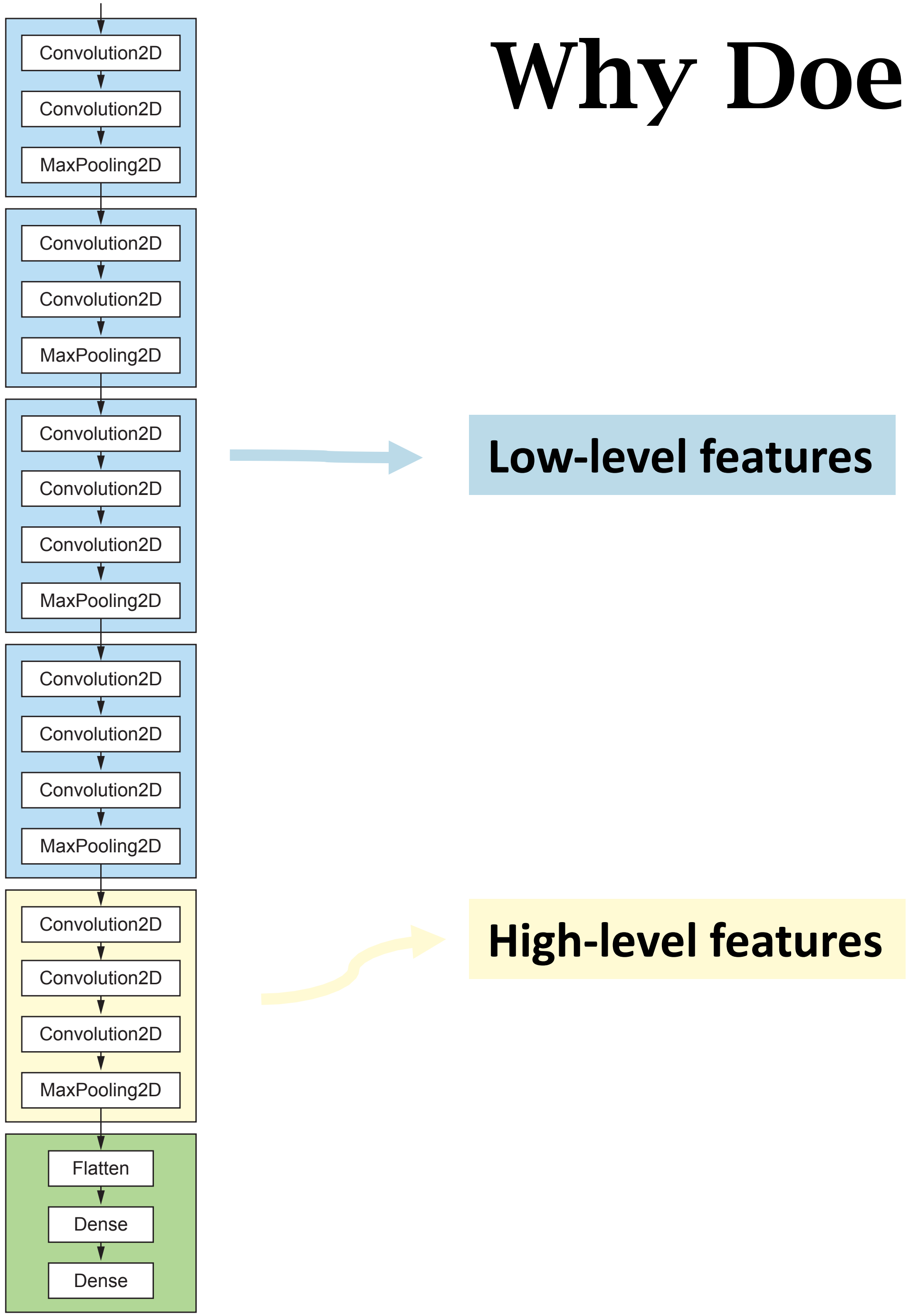
预训练

不能合并步骤 4 和 5 是

如果顶层是随机的，初始梯度会很大。
大的梯度会破坏卷积层。
因此，在训练顶层后再训练卷积层。

问：步骤 4 & 5 是否能合并？

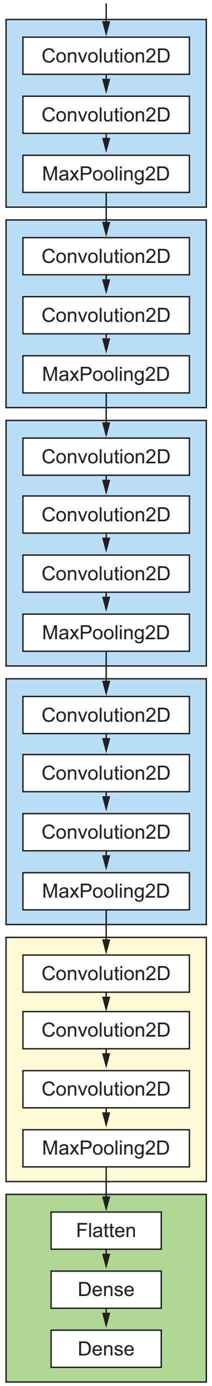
Why Does Pretraining Work?



- *卷积层*

  用于特征提取
  - 从 ImageNet 学习到的低级特征**低级特征** (边缘, 形状, 模式, 等等.) 对其他图像问题很有效
  - 从 ImageNet 学习到的 **高级特征** 也有用，但效果较差

Why Does Pretraining Work?



Low-level features

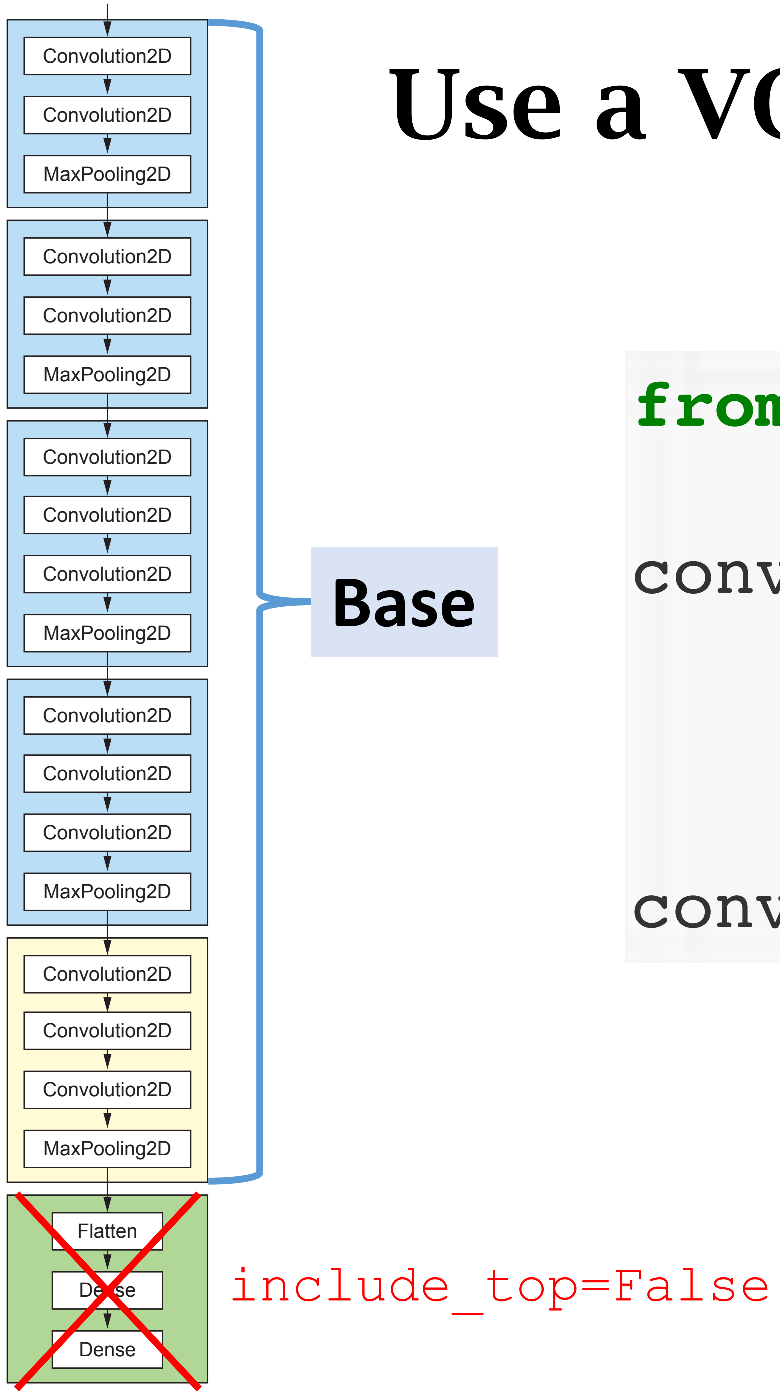
High-level features

Classifier (e.g., Softmax, Logistic, SVM, etc.)

- *卷积层*
  用于特征提取
  - 从 ImageNet 学习到的低级特征**低级特征** (边缘, 形状, 模式, 等等.) 对其他图像问题很有效
  - 从 ImageNet 学习到的 **高级特征** 也有用，但效果较差
- 将 *高层的全连接层* 视为一个分类器，它以提取的特征作为输入

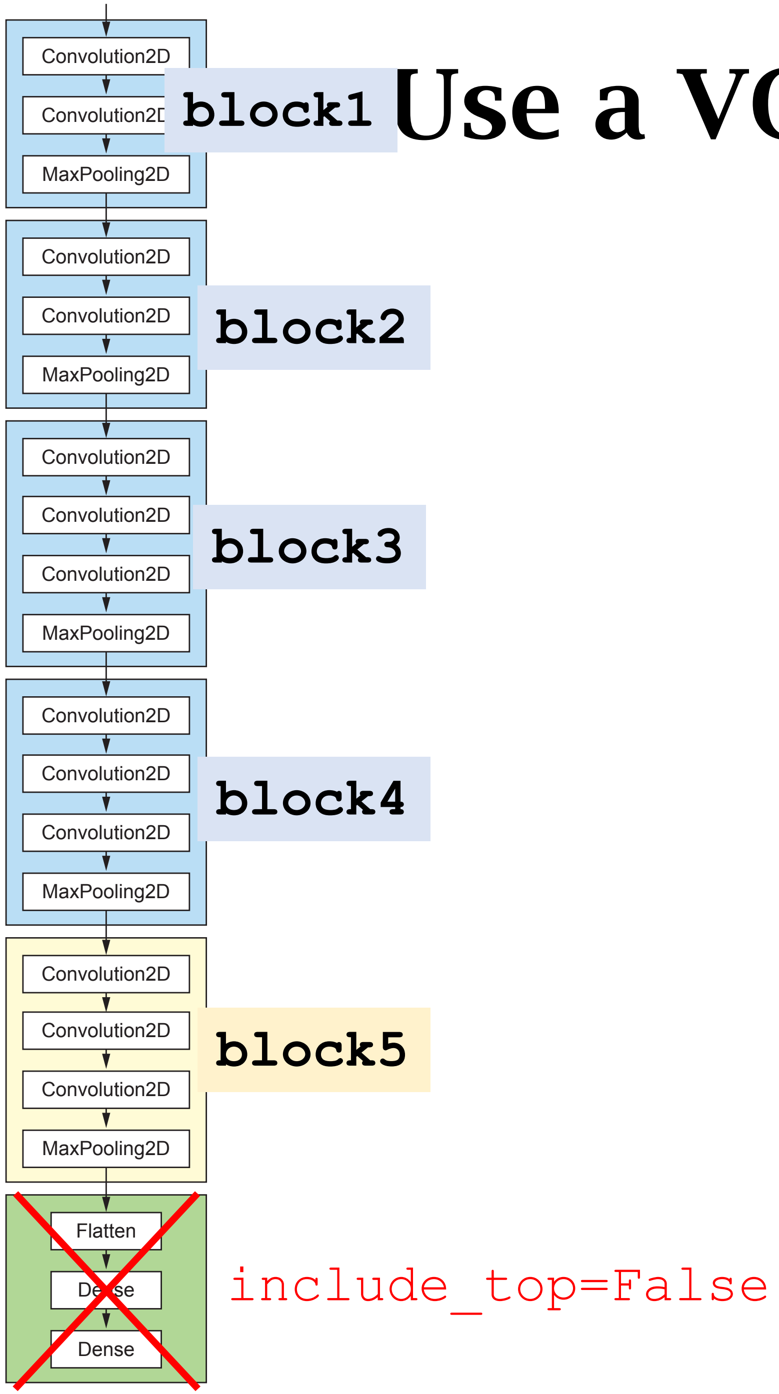
可训练参数更少，越不容易过拟合。

Use a VGG16 Net Pretrained on ImageNet



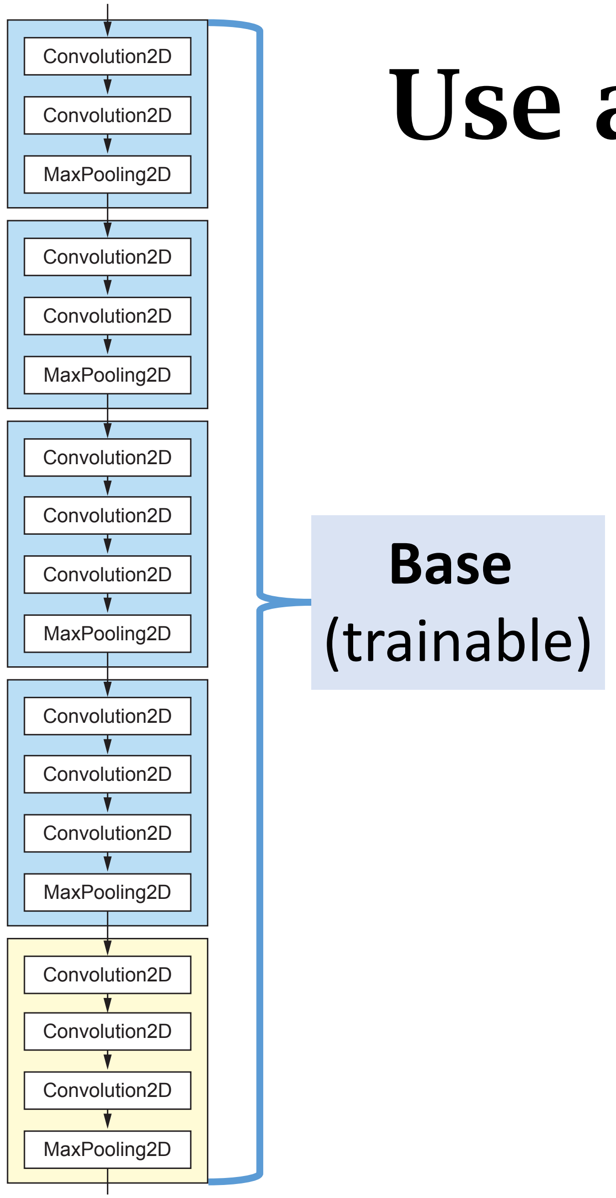

```
from keras.applications import VGG16

conv_base = VGG16(weights='imagenet',
                  include_top=False,
                  input_shape=(150, 150, 3))

conv_base.summary()
```

Use a VGG16 Net Pretrained on ImageNet



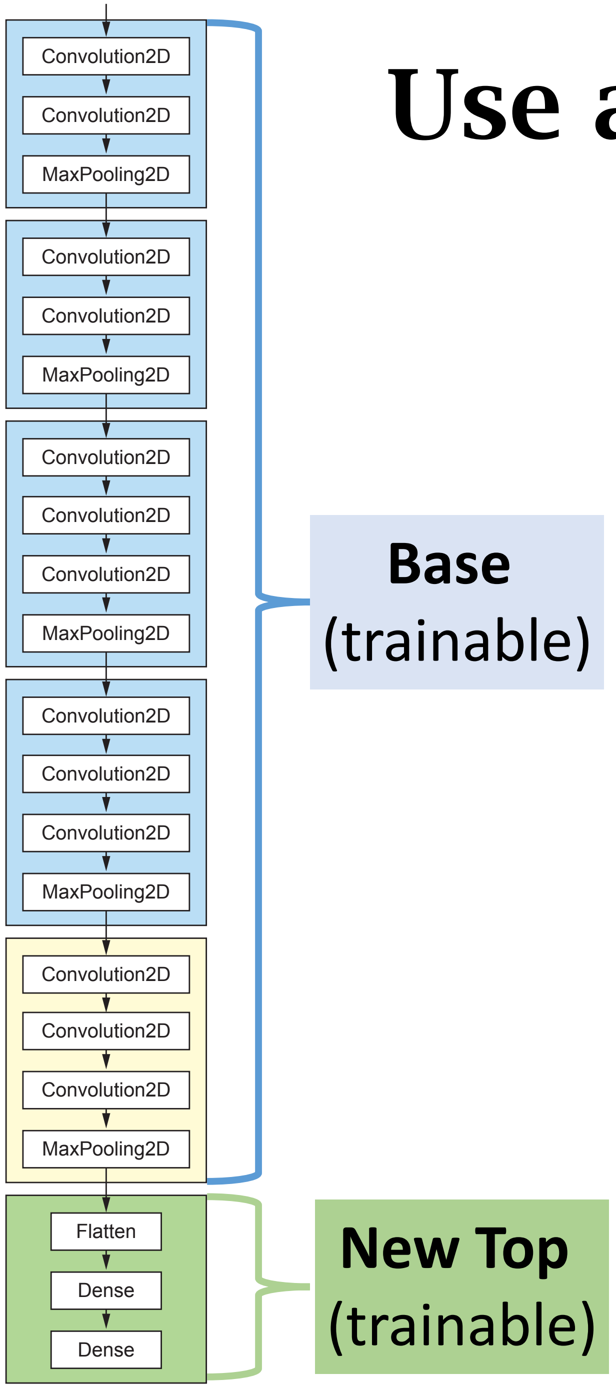

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         (None, 150, 150, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 150, 150, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 150, 150, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 75, 75, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 75, 75, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 75, 75, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 37, 37, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 37, 37, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 37, 37, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 37, 37, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 18, 18, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 18, 18, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 18, 18, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 18, 18, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 9, 9, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 4, 4, 512)         0
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
```

Use a VGG16 Net Pretrained on ImageNet



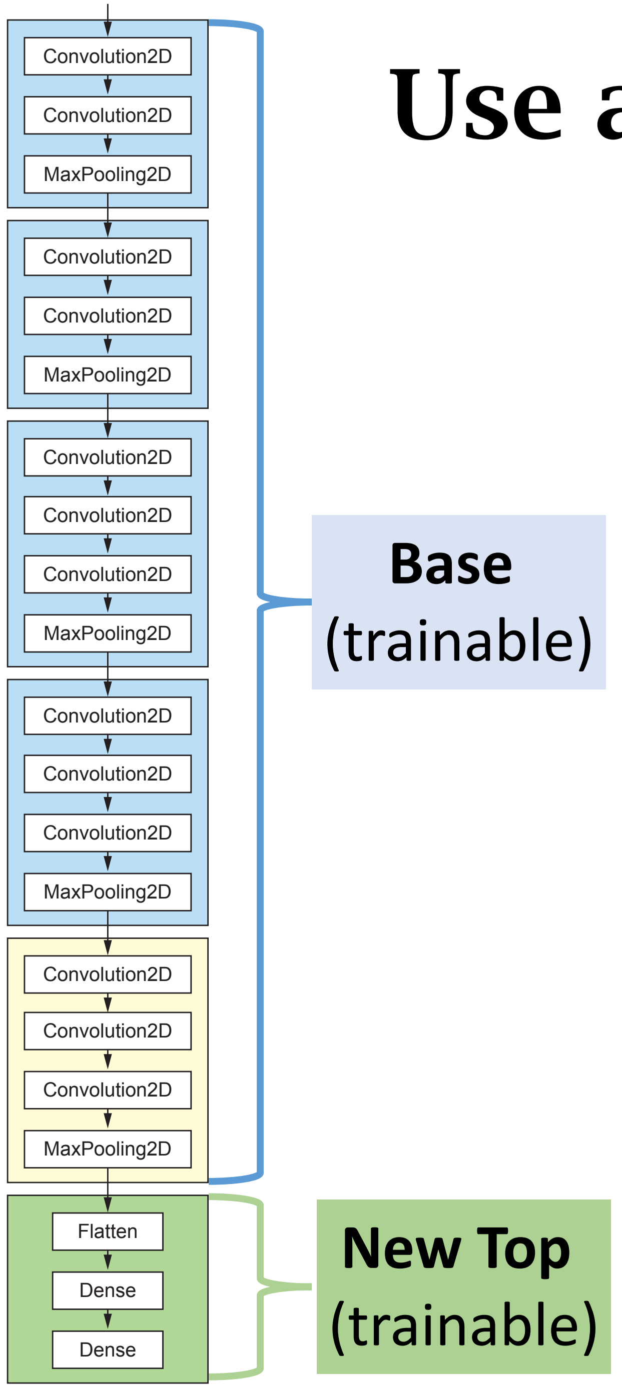

```
from keras import models
from keras import layers

model = models.Sequential()
model.add(conv_base)
```

Use a VGG16 Net Pretrained on ImageNet



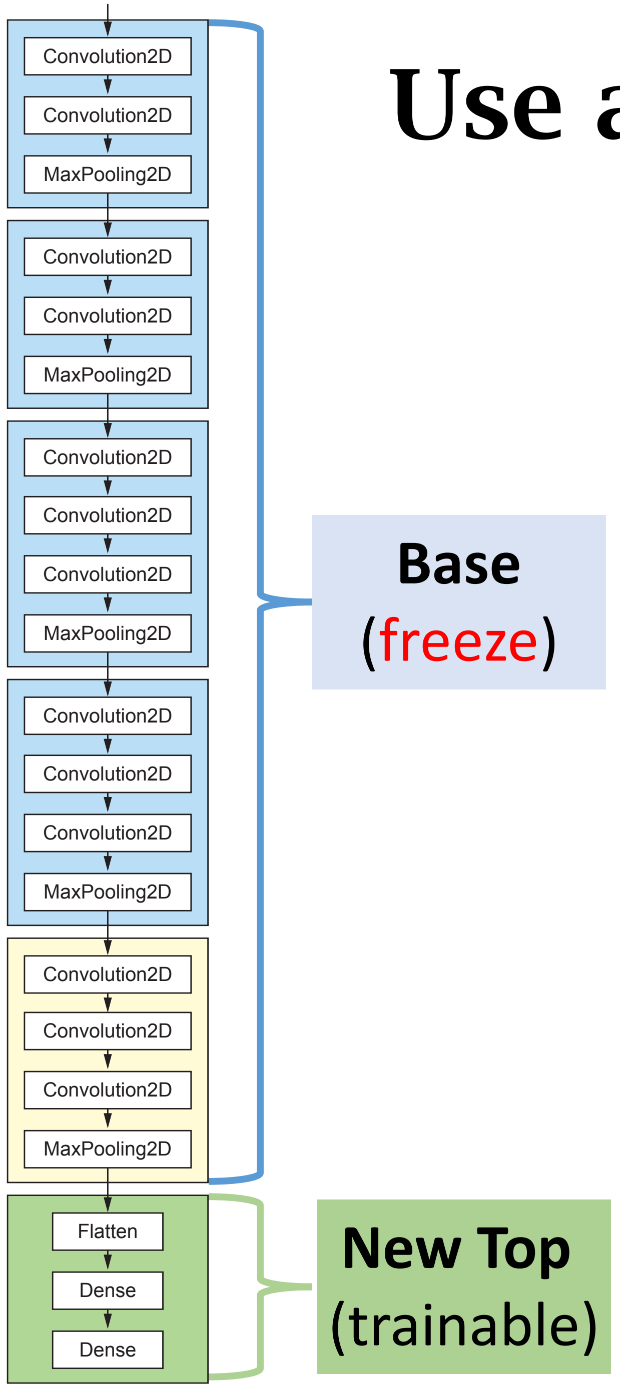

```python
from keras import models
from keras import layers

model = models.Sequential()
model.add(conv_base)
model.add(layers.Flatten())
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

Use a VGG16 Net Pretrained on ImageNet




```
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 4, 4, 512)         14714688
_________________________________________________________________
flatten_1 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               2097408
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 257
=================================================================
Total params: 16,812,353
Trainable params: 16,812,353
Non-trainable params: 0
```

Use a VGG16 Net Pretrained on ImageNet

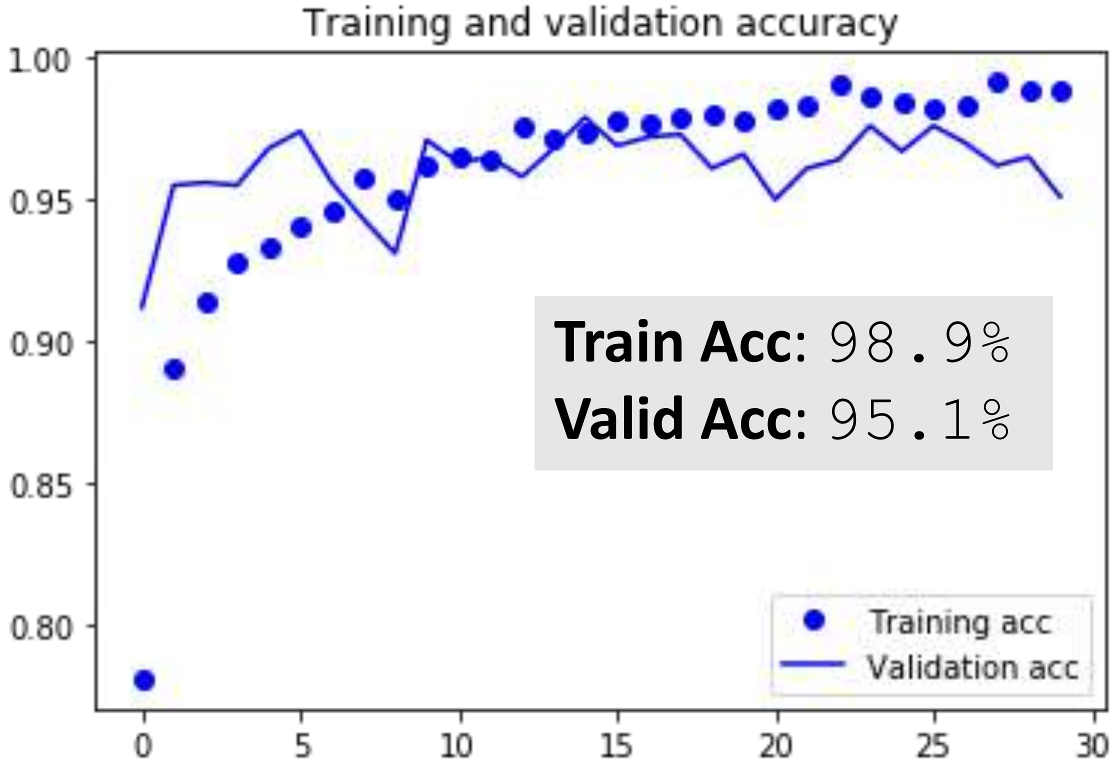



```
conv_base.trainable = False
model.summary()
```

```
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 4, 4, 512)         14714688
_________________________________________________________________
flatten_1 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               2097408
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 257
=================================================================
Total params: 16,812,353
Trainable params: 2,097,665
Non-trainable params: 14,714,688
```

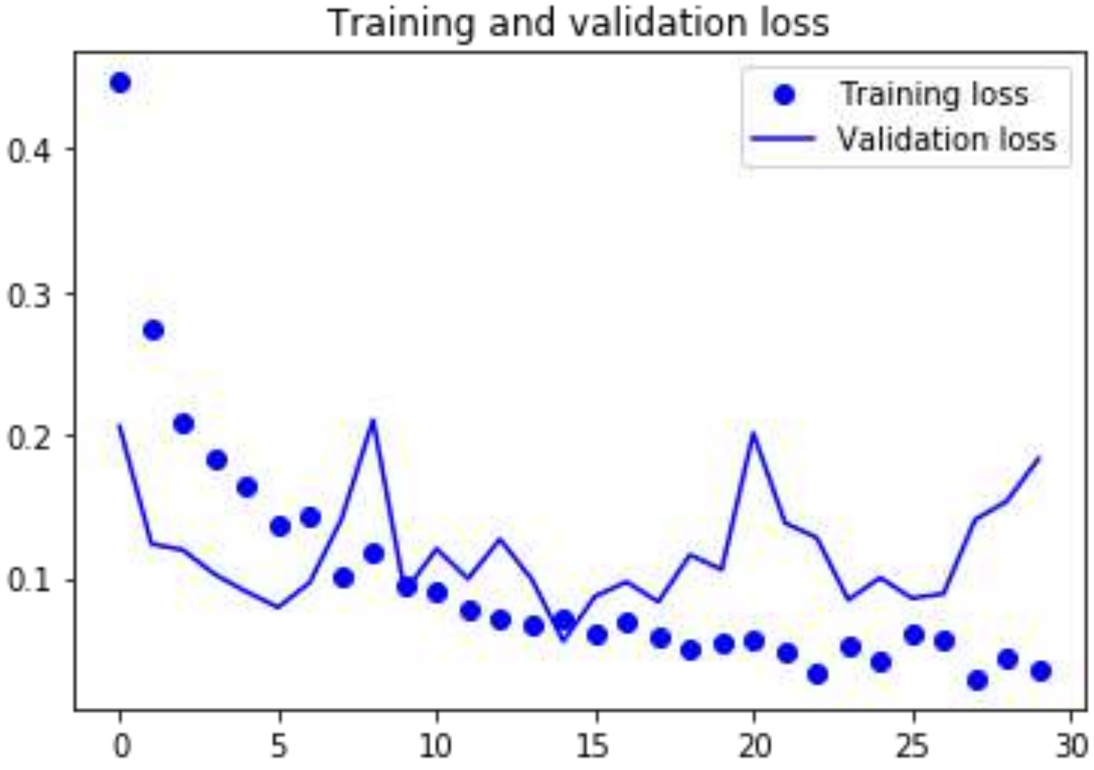
After Training the New Top

accuracy **against** ***epochs***

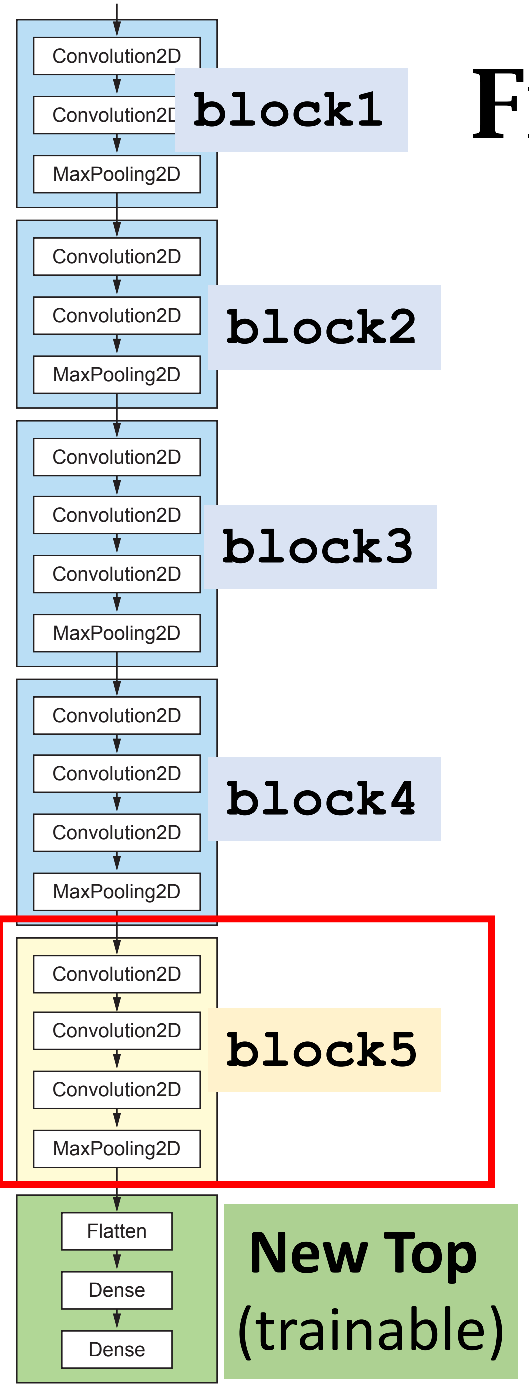


Train Acc: `98.9%`
Valid Acc: `95.1%`

loss **against** ***epochs***

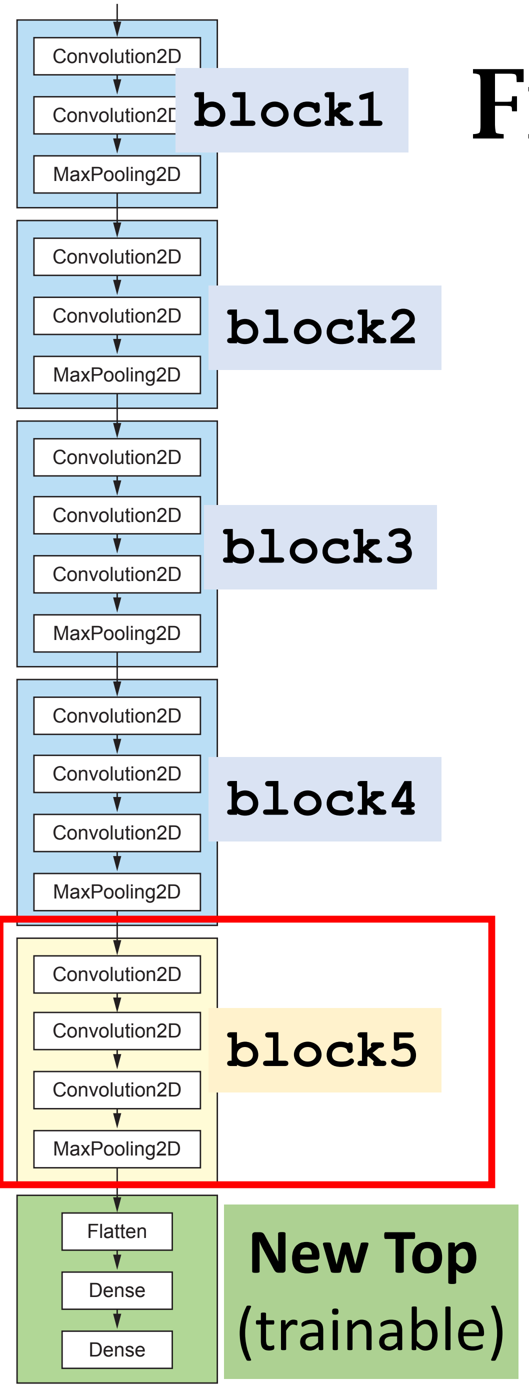


Fine Tuning the Top Conv Layers

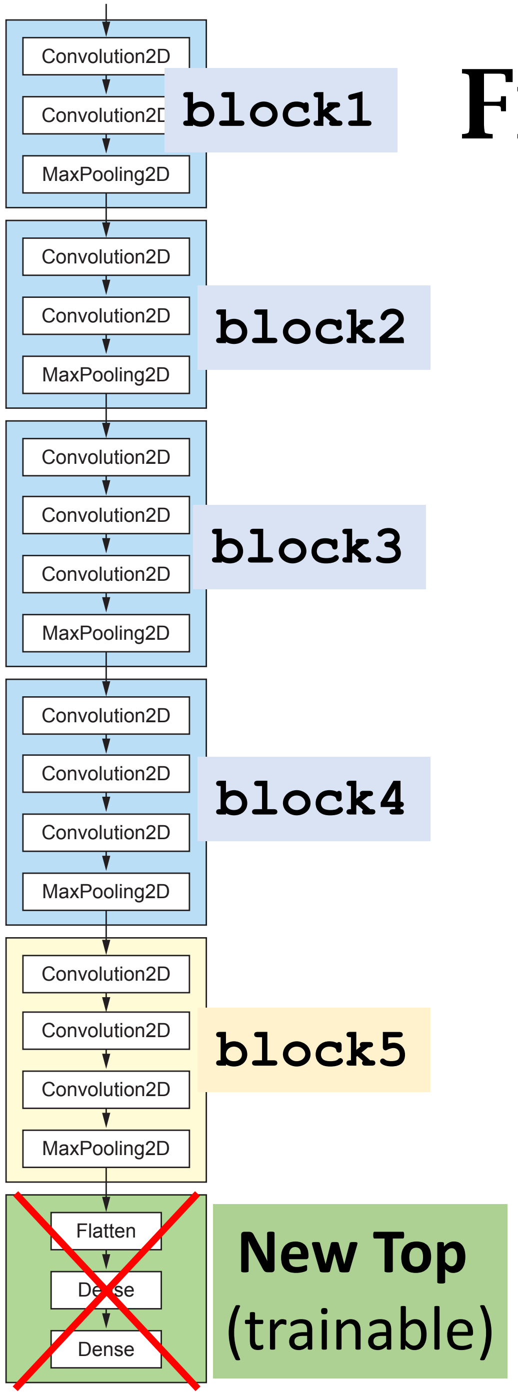



```
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         (None, 150, 150, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 150, 150, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 150, 150, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 75, 75, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 75, 75, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 75, 75, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 37, 37, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 37, 37, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 37, 37, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 37, 37, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 18, 18, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 18, 18, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 18, 18, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 18, 18, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 9, 9, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 4, 4, 512)         0
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
```

Fine Tuning the Top Conv Layers




```
trainable_layer_names = ['block5_conv1', 'block5_conv2',
                         'block5_conv3', 'block5_pool']
conv_base.trainable = True

for layer in conv_base.layers:
    if layer.name in trainable_layer_names:
        layer.trainable = True
    else:
        layer.trainable = False
```



Fine Tuning the Top Conv Layers

```
model.summary()
```

```
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 4, 4, 512)         14714688
_________________________________________________________________
flatten_1 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               2097408
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 257
=================================================================
Total params: 16,812,353
Trainable params: 9,177,089
Non-trainable params: 7,635,264
_________________________________________________________________
```

Fine Tuning the Top Conv Layers

Re-compile before training

```
model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-5),
              metrics=['acc'])
```

Fine Tuning the Top Conv Layers

```python
history = model.fit_generator(
      train_generator,
      steps_per_epoch=100,
      epochs=100,
      validation_data=validation_generator,
      validation_steps=50)
```

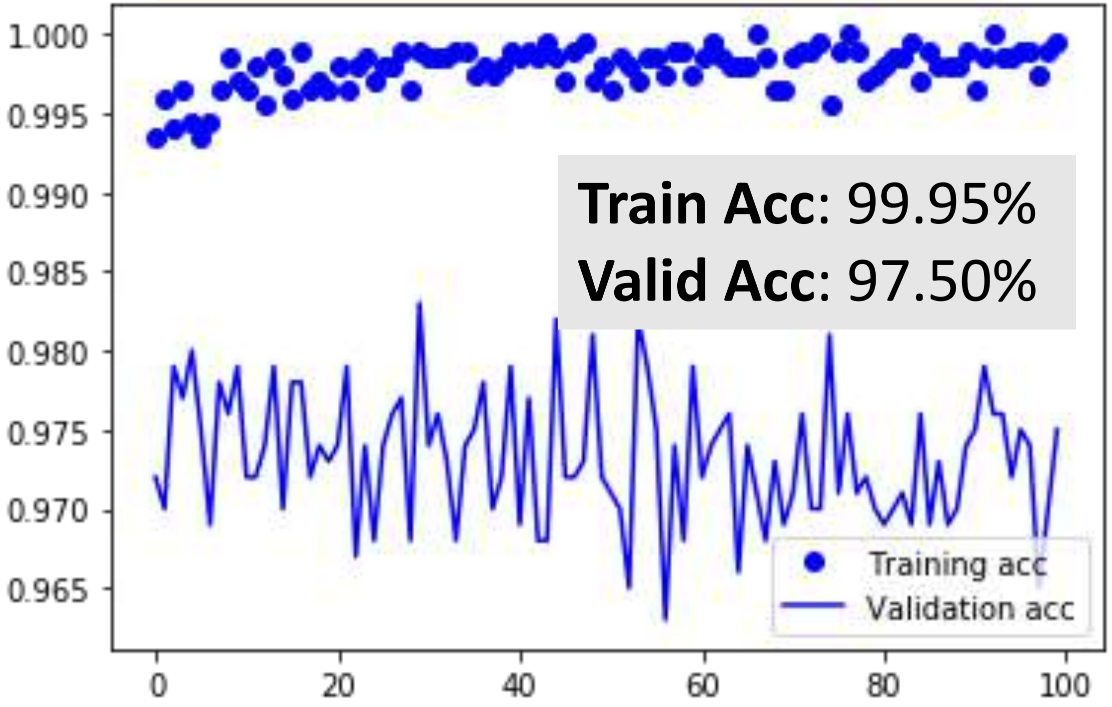
```
Epoch 1/100
100/100 [==============================] - 32s - loss: 0.0215 - acc: 0.9935 - val_loss: 0.0980 - val_acc: 0.9720
Epoch 2/100
100/100 [==============================] - 32s - loss: 0.0131 - acc: 0.9960 - val_loss: 0.1247 - val_acc: 0.9700
Epoch 3/100
100/100 [==============================] - 32s - loss: 0.0140 - acc: 0.9940 - val_loss: 0.1044 - val_acc: 0.9790
```

-
-
-

```
Epoch 99/100
100/100 [==============================] - 33s - loss: 0.0060 - acc: 0.9990 - val_loss: 0.2242 - val_acc: 0.9700
Epoch 100/100
100/100 [==============================] - 33s - loss: 0.0010 - acc: 0.9995 - val_loss: 0.2403 - val_acc: 0.9750
```

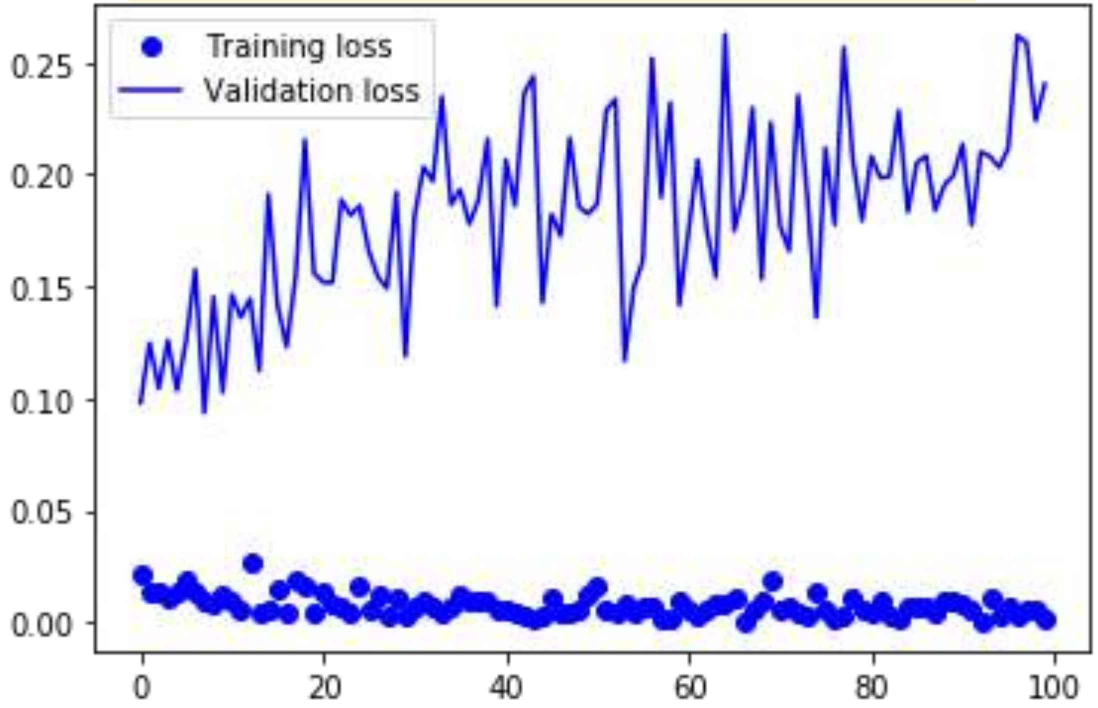
Fine Tuning the Top Conv Layers

accuracy against ***epochs***



Train Acc: 99.95%
Valid Acc: 97.50%

loss against ***epochs***



Fine Tuning the Top Conv Layers

Evaluate the model on the test set

```
test_generator = test_datagen.flow_from_directory(
        test_dir,
        target_size=(150, 150),
        batch_size=20,
        class_mode='binary')

test_loss, test_acc = model.evaluate_generator(test_generator, steps=50)
print('test acc:', test_acc)
```

```
Found 1000 images belonging to 2 classes.
test acc: 0.967999992371
```

Summary of the Results

- 小型 ConvNet 网络(4 个卷积层 + 2 个全连接层), 带有 3.5M 个参数.
  - Training accuracy: `99.0%`
  - Validation accuracy: `72.4%`
- 小型 ConvNet 网络 + 1 个 dropout 层 + 数据增强.
  - Training accuracy: `84.9%`
  - Validation accuracy: `84.4%`
- 大型 VGG16 网络， 通过大量图像数据集的预训练 (训练最顶层)
  - Training accuracy: `98.9%`
  - Validation accuracy: `95.1%`
- 大型 VGG16 网络， 通过大量图像数据集的预训练 (顶层卷积层微调)
  - Training accuracy: `99.95%`
  - Validation accuracy: `97.5%`

评估方法

评估方法

- Varying data (bagging).
  - Fit a VGG16 network on a subset of data ➔ Model 1
  - Fit a VGG16 network on a subset of data ➔ Model 2
  - Fit a VGG16 network on a subset of data ➔ Model 3

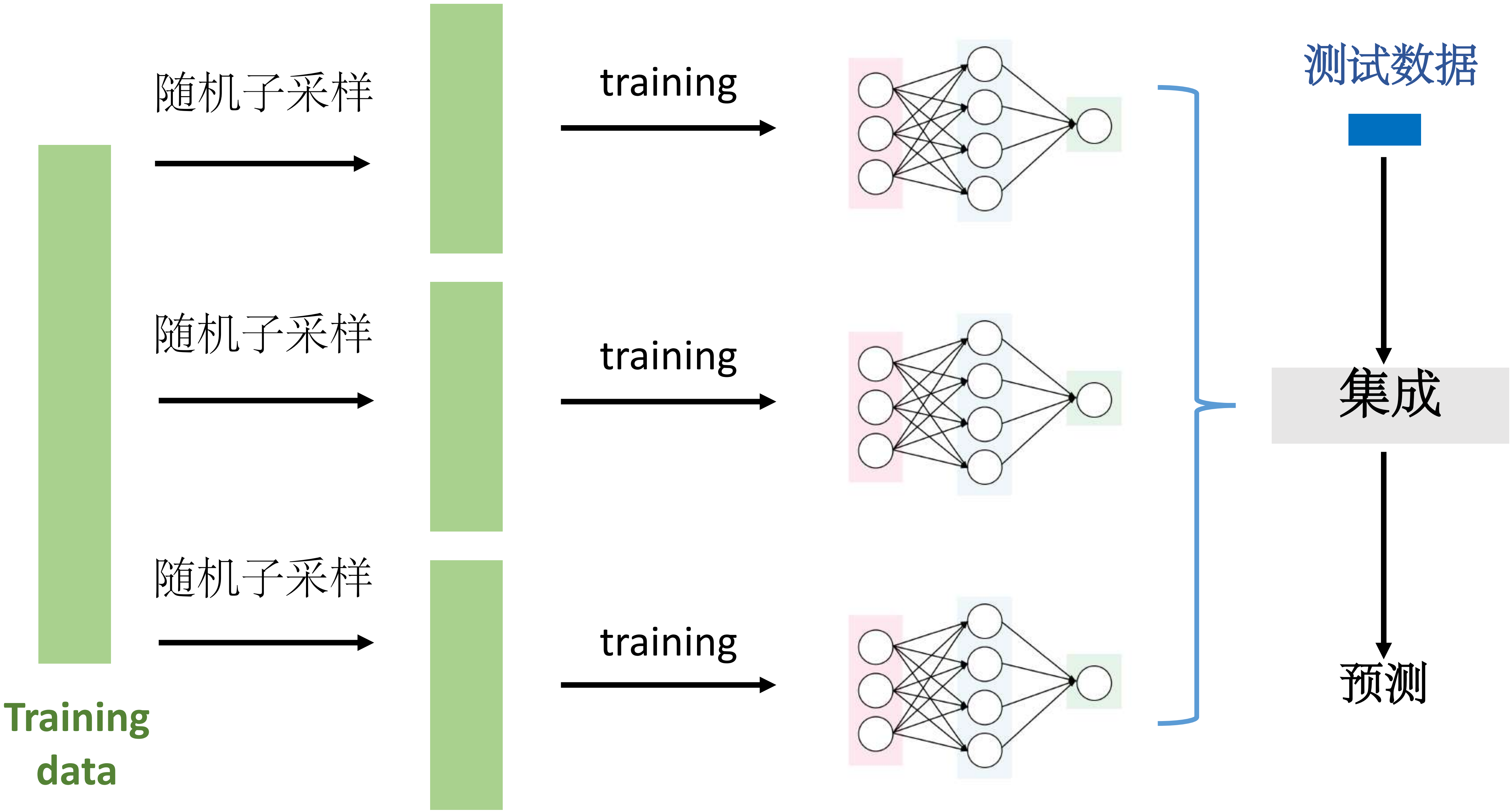
一种集成学习方法，通过在训练数据上进行有放回的随机抽样（bootstrap），生成多个子集，然后在每个子集上训练一个模型，最后将这些模型的结果聚合（例如投票或平均）。

集成方法

- Varying data (bagging).
  - Fit a VGG16 network on a subset of data ➔ Model 1 ➔ pred 1
  - Fit a VGG16 network on a subset of data ➔ Model 2 ➔ pred 2
  - Fit a VGG16 network on a subset of data ➔ Model 3 ➔ pred 3

Vote

Bagging（也称为 Bootstrap Aggregating，引导聚合）



集成方法

- Varying data (bagging).
  - Fit a VGG16 network on a subset of data ➔ Model 1 ➔ pred 1
  - Fit a VGG16 network on a subset of data ➔ Model 2 ➔ pred 2
  - Fit a VGG16 network on a subset of data ➔ Model 3 ➔ pred 3

  Vote

- 模型多样化
  - 不同的网络结构
  - 不同的随机初始化
  - 不同优化算法

为什么才用集成方案？

- 深度神经网络非常 不稳定

对超参数敏感

为什么才用集成方案？

- 深度神经网络非常 不稳定 并且 随机.

对超参数敏感

随机初始化

随机梯度下降算法

不同的局部最优解

为什么才用集成方案?

- 深度神经网络非常 不稳定 并且 随机.

对超参数敏感

随机初始化

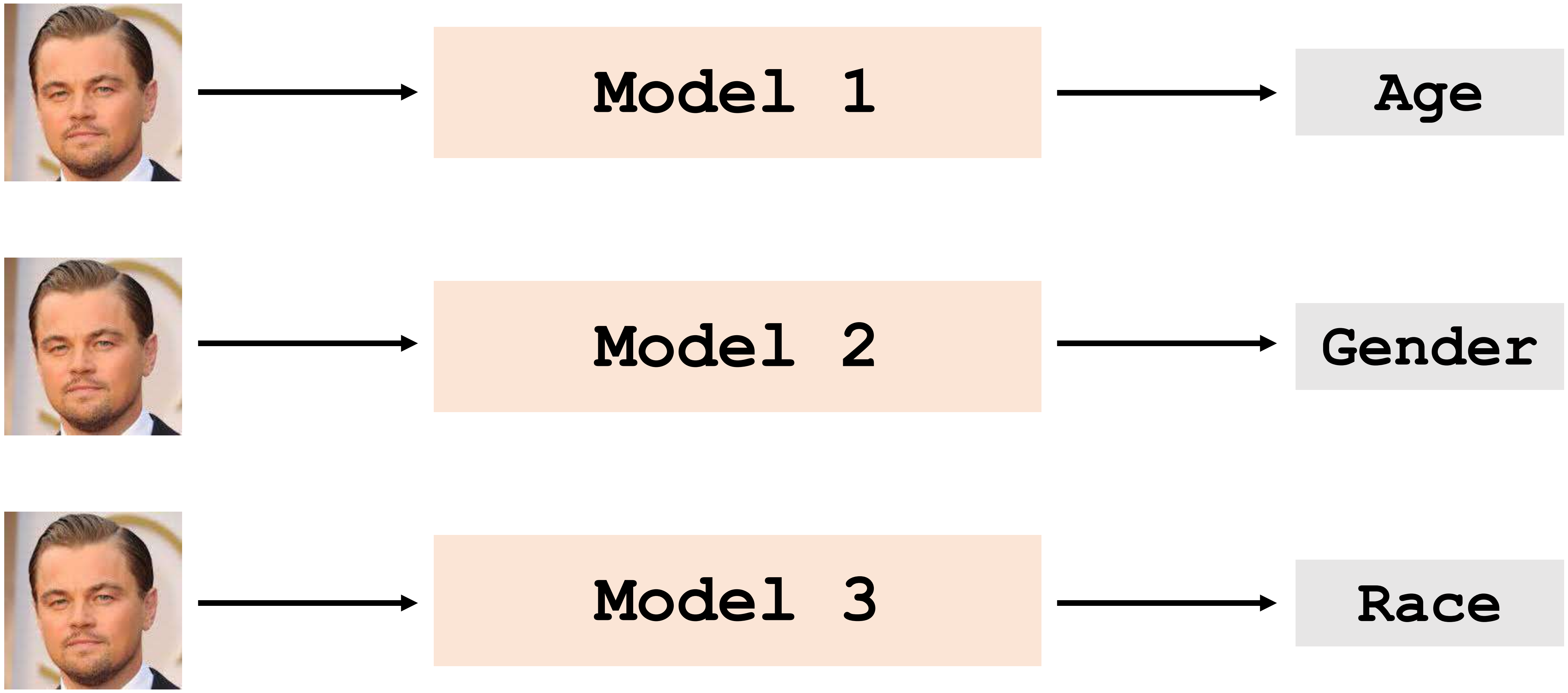
随机梯度下降算法

不同的局部最优解

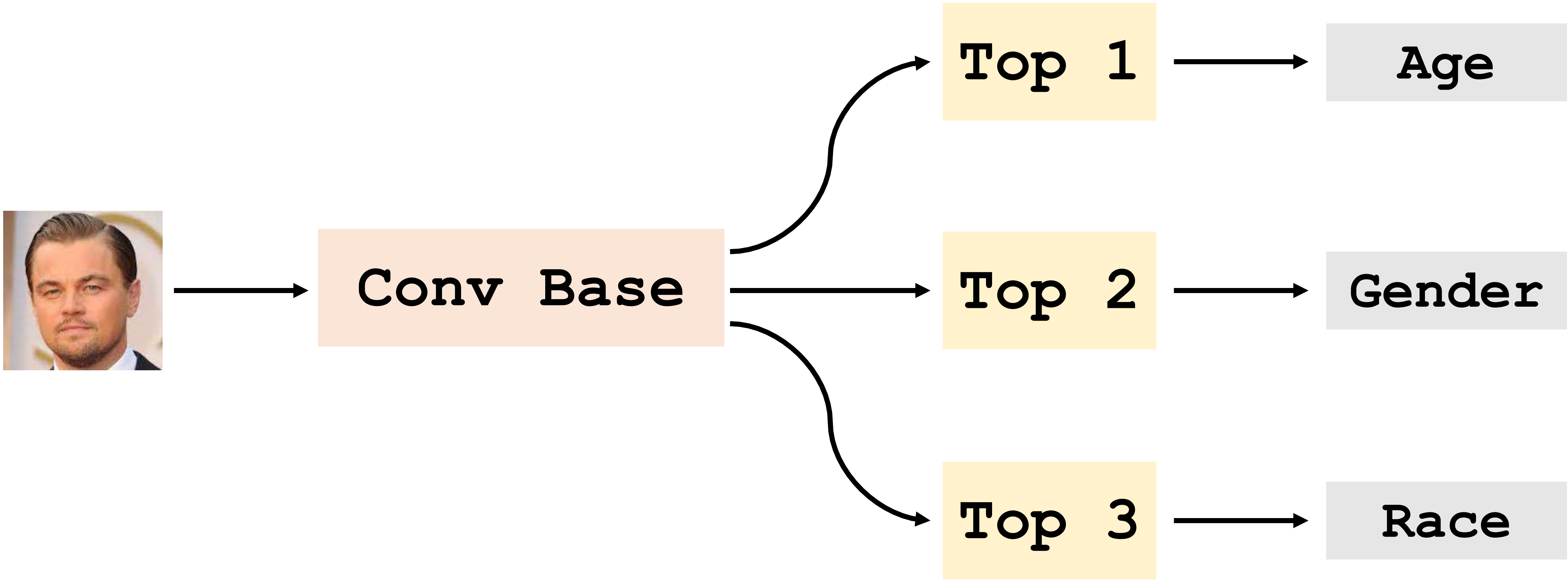
- 集成方法减少方差。

多任务学习

参数不共享



多任务学习



多任务学习

$$\text{Loss1} = (\text{Age\_Label} - \text{Age\_Pred})^2$$

回归

$$\text{Loss2} = \text{dist}(\text{Gender\_Label}, \text{Gender\_Pred})$$

二分类

$$\text{Loss3} = \text{dist}(\text{Race\_Label}, \text{Race\_Pred})$$

多元分类

- 目标函数: $\text{Loss1} + \lambda \cdot \text{Loss2} + \gamma \cdot \text{Loss3}.$

多任务学习

$$\text{Loss1} = (\text{Age\_Label} - \text{Age\_Pred})^2$$

回归

$$\text{Loss2} = \text{dist}(\text{Gender\_Label}, \text{Gender\_Pred})$$

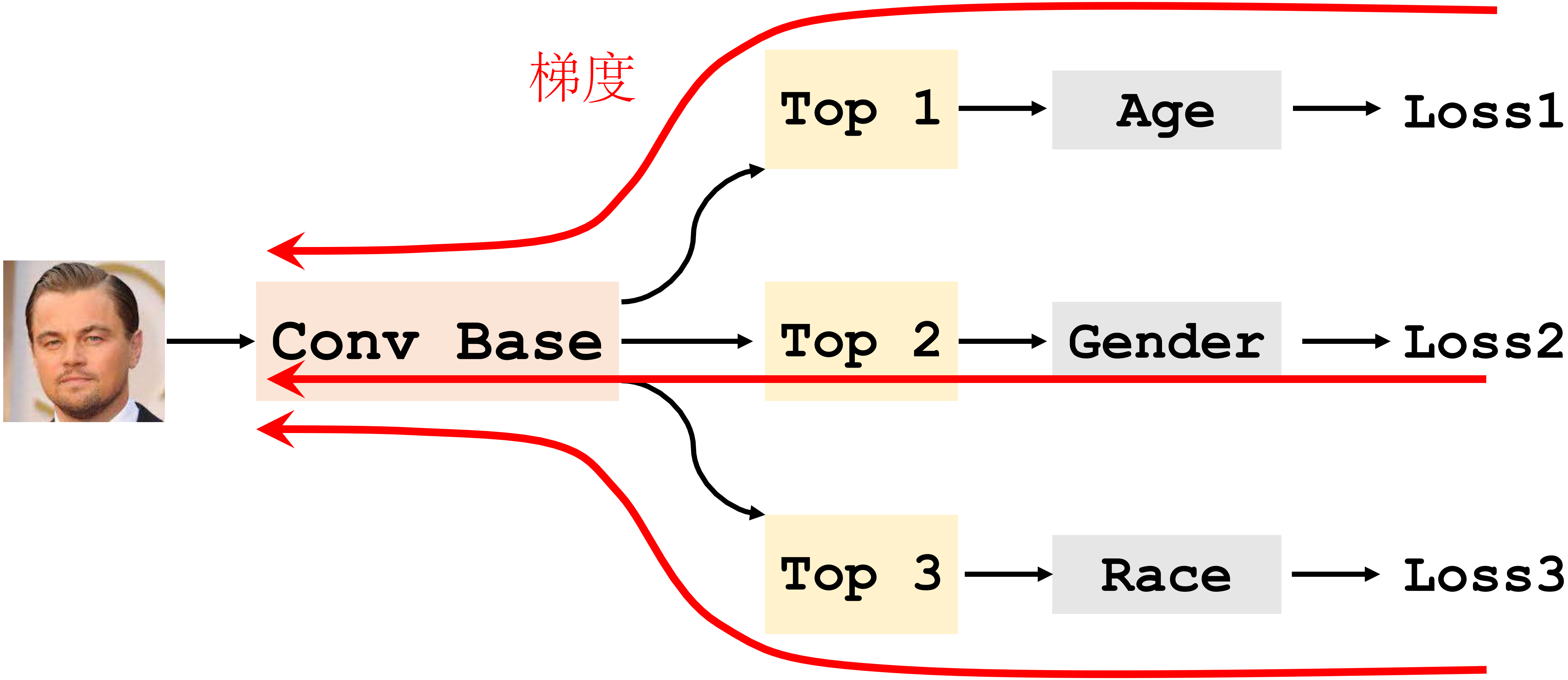
二分类

$$\text{Loss3} = \text{dist}(\text{Race\_Label}, \text{Race\_Pred})$$

多元分类

- 目标函数: $\text{Loss1} + \lambda \cdot \text{Loss2} + \gamma \cdot \text{Loss3}.$
  - 为什么使用这两个超参数?
  - Loss1 大约为 10。
  - Loss2 和 Loss3 大约为 0.1。
  - 如果不进行缩放，卷积基将由年龄任务主导。

多任务学习



梯度
Conv Base
Top 1
Top 2
Top 3
Age
Gender
Race
Loss1
Loss2
Loss3

总结

提升泛化能力的技巧

- 技巧 1：Dropout 正则化。
- 技巧 2：数据增强。
- 技巧 3：预训练。
- 技巧 4：集成方法。
- 技巧 5：多任务学习。

其他提升泛化能力的技巧

在每层（通常是卷积层或全连接层）之后，标准化层的输入（或激活值），使均值为0，方差为1，然后再进行线性变换。

技巧1：batch 标准化。

技巧2：梯度注入（Google Inception Net）。

技巧3：跳跃连接（ResNet）。