机器学习（进阶）纳米学位毕业项目

---- 猫狗识别

郭澍存

2017/12/27

目录

1 定义

1.1 项目概述

毕业项目选择了猫狗大战，将训练一个猫狗图像识别二分类的模型。项目使用的模型是基于卷积神经网络的各种模型，如 Xception、InceptionResNetV2、InceptionV3 等.因为这些模型已经在 Imagenet 比赛上有很好的表现，所以毕业项目直接使用并学习这些模型。

项目使用的数据集全部来自 Kaggle，数据集地址 dogs-vs-cats。

1.2 问题陈述

数据集包含一个训练集和测试集，训练集包含 25000 张带标签的猫狗图片，其中猫狗各占一半，而测试集包含 12500 张图片。所有的猫狗图片都是从真实世界中采集而来，由于数据集的图片分辨率、光线条件、失焦模糊程度、品种类别等都不相同，这些会增加分类的难度。所以需要搭建一个较好的模型来对训练集的图片训练， 然后使用测试集验证模型的准确率，最后提交到 kaggle。

1.3 评价指标

提交的结果使用的是官方的评估标准 LogLoss，公式：

$$\text{LogLoss} = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)\right],$$

n 是测试集的总数量

$\hat{y}_i$是测试的图片为狗的概率

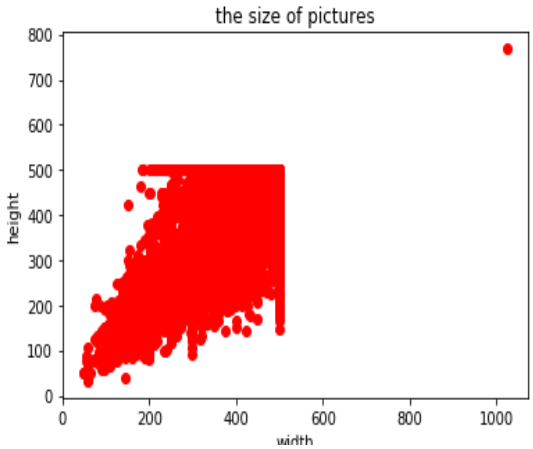
y_i是 1 时图片是狗，为 0 时是猫

2 分析

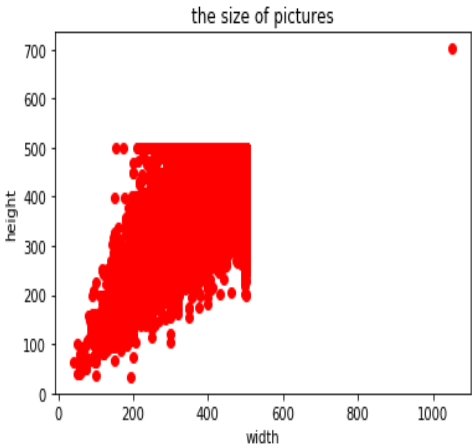
2.1 数据研究

数据集包含训练集和测试集， 其中训练集包含 25000 张猫狗图片，猫狗各有 12500 张，测试集有不带标签的 12500 张图片，需要训练分类。训练集中的每张图片的名字中含有图片的分类与序号。比如，命名为“cat.9.jpg”表明该图是猫，图片编号是 9。而测试集的图片只有序号，没有图片的分类信息。训练集的数据特征大致如下：

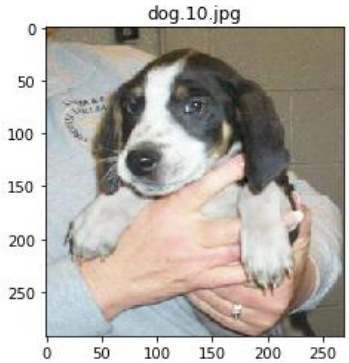
1. 猫狗照片数量相等
2. 图片的尺寸不一致， 可见猫狗的图片尺寸分布图,如下图的(cat)和(dog)。
3. 图片的场景不同，如下图的 cat.6 和 cat.11, dog.12490 和 dog.12491
4. 猫的图片存在异常值，如下图(a)中的 cat.5351、cat.5418、cat.7377、cat.8456、cat.7564、cat.10712、cat.11184、cat.9171 等
5. 狗的图片也存在异常值，如下图(b)中的 dog.2614、dog.4367、dog.5604、dog.8736、dog.9517、dog.10237、dog.10401、dog.10747、dog.10801、dog.12376 等

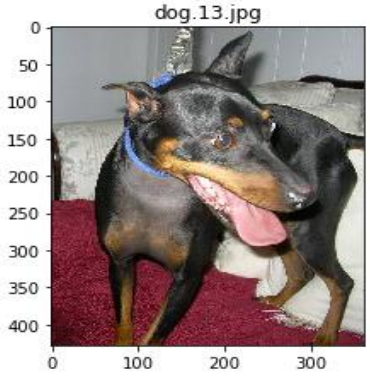


(cat)



(dog)





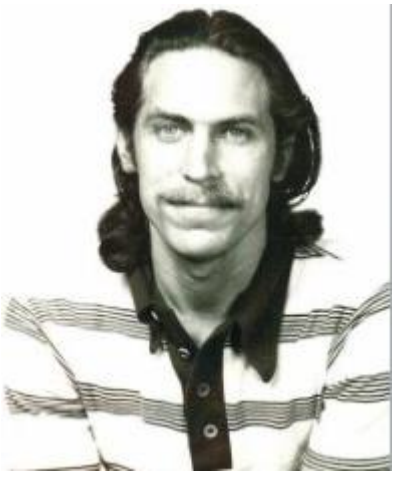
dog.12491

dog.12490

(a)猫的异常值图片示例

cat.5351

cat.5418



cat.7377

cat.8456

cat.7564

cat.10712

cat.11184

cat.9171

(b)狗的异常值图片示例

dog.2614

dog.8736

dog.9517

dog.10237

dog.10401

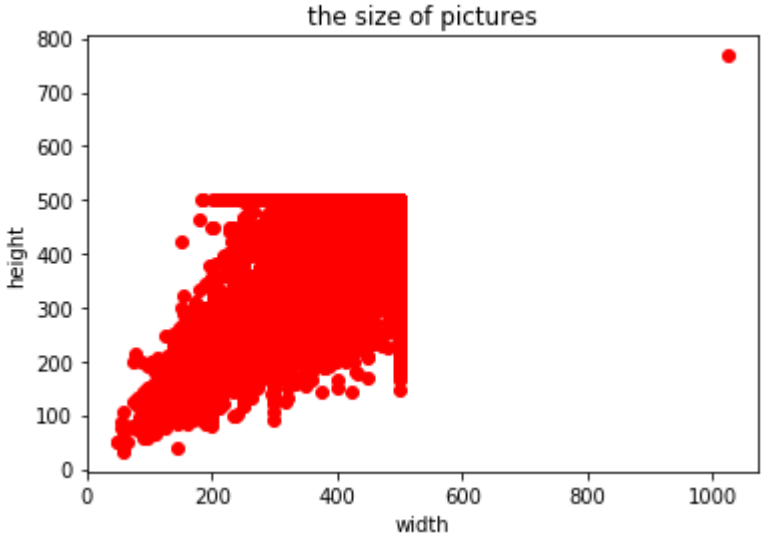
dog.10747

dog.10801

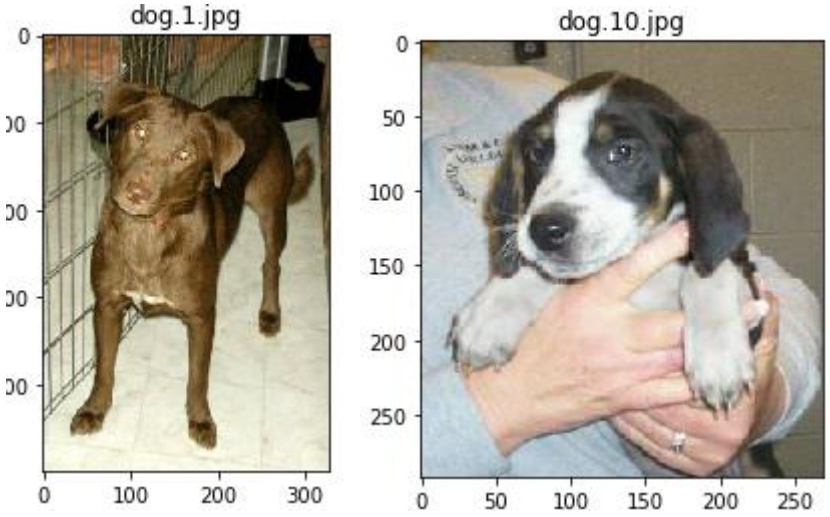
dog.12376

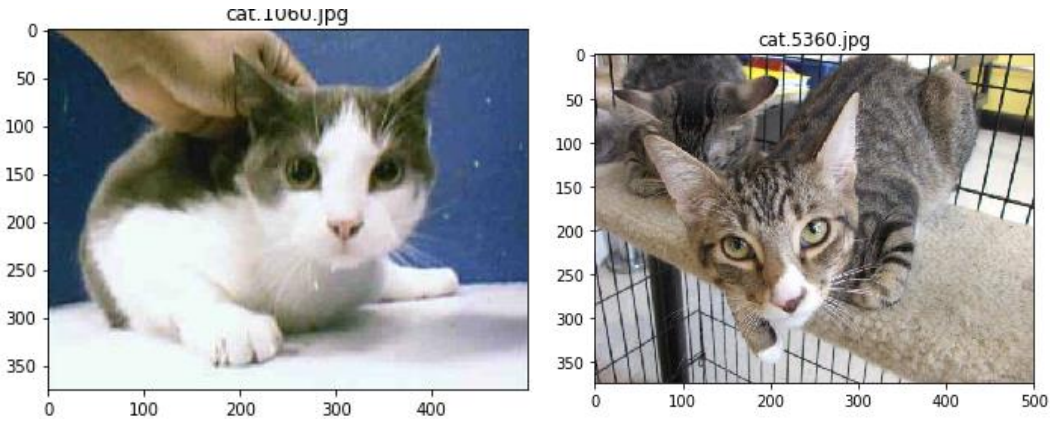
2.2 探索可视化

对数据集作初步探索，将猫狗图片分别放置 cat 和 dog 文件夹中并读取图片尺寸如下图(1)，可以发现训练集中有很少的部分图片尺寸是小于 100*100，个别的图片尺寸超过 700*1000 的，另外还有一些异常值的图片。随机的查看一部分训练集的图片，方便后续对数据集的处理，见下图。



(1)





2.3 算法与技术

猫狗识别是一个二元分类的问题。伴随着数据集的增加和计算能力的提升，特别是 GPU 计算能力的爆发性提升，使用卷积神经网络的深度学习便很好的解决了图片识别的问题。本次的项目中将采用 tensorflow 和 keras 来搭建学习模型，快速简单易上手。

卷积神经网路（Convolutional Neural Network）是一种前馈神经网络，其参数从输入层向输出层单向传播。卷积神经网路是由输入层、多个卷积层、池化层和顶端的全连接层组成。卷积神经网络输入的是原始数据，通过卷积池化和激活函数等计算真实值和预测值之间的损失，再依靠反向传播算法将损失值向前反馈来更新层的参数，如此往复直到模型收敛到预期的结果。其中隐藏层的激活函数使用 ReLU=max(0,x)，ReLU 函数其实是一个取最大值的函数，它有几大优点：解决了正区间梯度消失的问题；计算速度快，只需要判断输入是否大于 0；收敛速度远快于 sigmoid 和 tanh。

项目中分别使用 Xception、InceptionResNetV2、inceptionV3 等模型。inceptionV3 的网络宽度是 299*299，不同于 VGG 的 244*244，同时 inceptionV3 提出了卷积分解，使用 N*1 的卷积级联既加速了计算又增加了网络深度，可以降低参数量，减轻过拟合，增加网络非线性的表达能力。因为小尺寸的滤波器可以提高网络的深度，进而让网络学习更加复杂的特征。将 Inception 模块和残差链接结合提出的 InceptionResNet 可以使得训练收敛更快，精度更高。Xception 是由 InceptionV3 的演化而来，和 InceptionV3 相比，Xception 的参数量有所下降，准确率也更高，在 Xception 中加入的类似 ResNet 的残差连接机制也显著加快了 Xception 的收敛过程并获得了显著更高的准确率。

权重初始化的目的是为了尽量避免随着网络层数的加深，而出现的梯度消失或者梯度爆炸的情况。目前常用的初始化方法有随机初始化(random initialization)、Xavier initialization、He initialization。由于随机初始化的弊端是一旦随机分布选择不当，就会导致网络优化陷入困境；而

Xavier initialization 可以解决随机初始化的问题，Xavier 初始化是保持输入和输出的方差一致，这样就避免了所有输出值都趋向于 0。但是它对于非线性函数不具备普适性。He initialization 可以解决非线性初始化的问题，要保持 variance 不变，只需要在 Xavier 的基础上再除以 2。其中 He init 的公式为：W=tf.Variable(npr.randn(node_in,node_out))/np.sqrt(node_in/2)。

常用的优化器有随机梯度下降 SGD、Adagrad、Adadelta、Adam 等。因为 SGD 是根据 mini-batch 来迭代计算的梯度，然后对参数进行更新，所以选择合适的学习率比较困难,又容易收敛到局部最优。Adagrad 是对学习率进行了一个约束，自适应地为各个参数分配不同学习率的算法，适合处理稀疏梯度。缺点是其学习率是单调递减的，训练后期学习率非常小，需要手工设置一个全局的初始学习率等。Adadelta 是对 Adagrad 的扩展，能对学习率进行自适应约束，又进行了计算上的简化，训练时的加速效果不错，时间也很快，有效地克服 Adagrad 学习率收敛至零的缺点。通过比较，Adadelta 算法的表现效果通常不错，学习率自适应优化，不用手动调节。最后使用 Dropout 来防止过拟合，因为随机的丢弃一部分隐藏节点既加快了计算又减弱了神经元节点间的联合适应性，增强了泛化能力。

2.4 基准模型

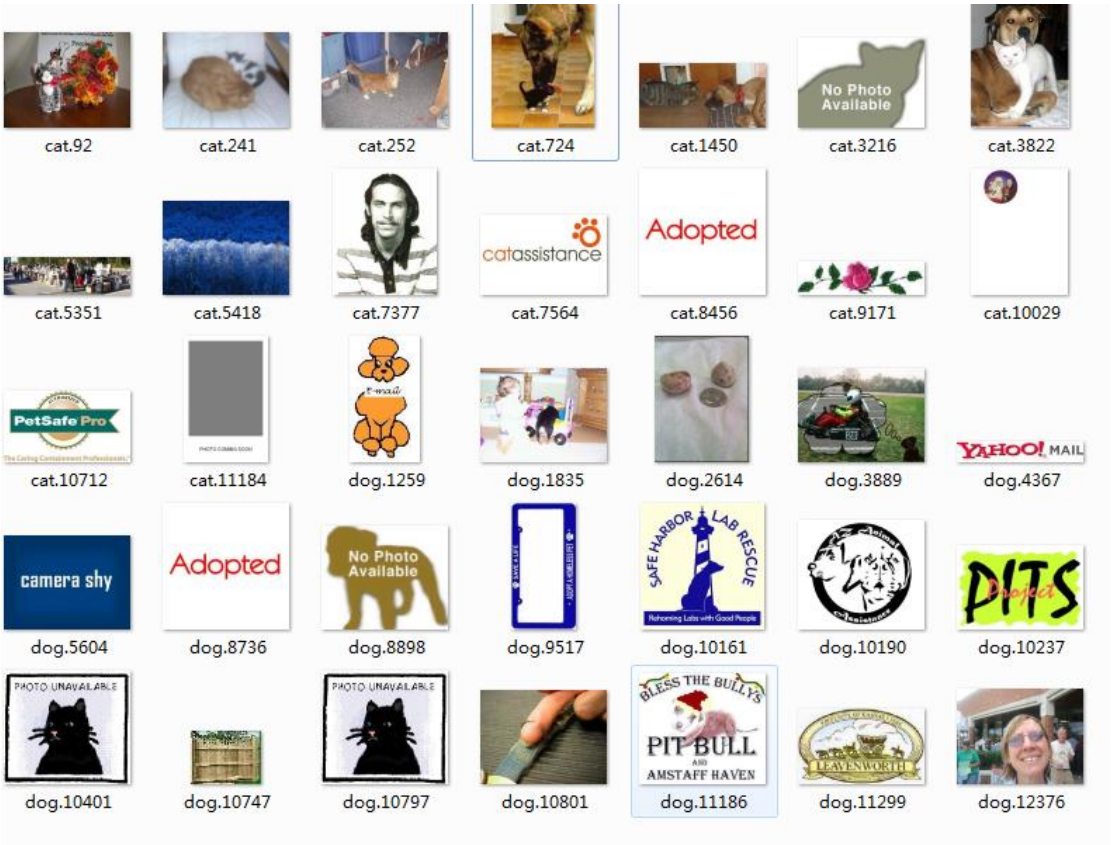
因为 Kaggle 的最好成绩 logloss 达到了 0.033，前 100 名的成绩是 0.057，而前 15 名的成绩是 0.04。所以我希望最终的目标成绩 logloss 能达到 0.04 以内。

3 方法

3.1 数据预处理

将训练集中的猫狗图片单独分开，并放置 cat 和 dog 文件夹中，从硬盘中读取图片和调整尺寸，将训练集进一步划分为训练集和验证集。

通过对图片进行分析，可以发现很多分辨率较低或模糊不清或异常的图片，我们需要把不合格的和尺寸太小模糊不清的图片移除，下面是其中一些不合格的图片。



此外，通过对图片分析，可以发现图片中猫狗的拍摄角度和背静光线等不尽相同，而且图片的尺寸比例也有差别。为了让模型尽量不受这些因素的干扰，需要对原始图片进行归一化、减均值、数据增强等操作，并把异常相关的图片移出训练集。

[17]:
```
# 移除不符合要求的图片
path_outlier = os.path.join(path, r'outlier')
if not os.path.isdir(path_outlier):
    os.mkdir(path_outlier)

# remove these pictures
for img_name in pick_cat:
    shutil.move((os.path.join(path_cat, img_name)), (os.path.join(path_outlier, img_name)))
```

3.2 实现

如果从零开始写一个卷积神经网络，需要精心设计网络层和层数，还要调整各种参数，所以为了简便直接使用了 keras 提供训练好的模型。单一的使用了 Xception、InceptionV3、InceptionResNetV2 模型，其中以 Xception 为例，调用定义的 xception 方法并训练模型：

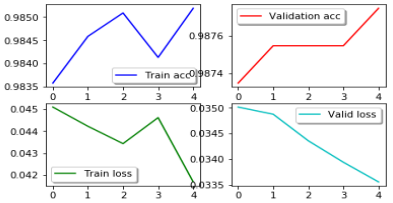
```
def xception(img_height, img_width=None, dropout = 0.25):
      if not img_width:
            img_width = img_height

      input_tensor = Input(shape=(img_height, img_width, 3))
      input_tensor = Lambda(xception_process)(input_tensor)
      base_model = Xception(input_tensor=input_tensor, weights='imagenet', include_top=False)

      for layers in base_model.layers:
            layers.trainable = False

      x = GlobalAveragePooling2D()(base_model.output)
      x = Dropout(dropout)(x)
      x = Dense(1, activation='sigmoid', kernel_initializer='he_normal')(x)

      model_xception = Model(inputs=base_model.input, outputs=x)
      model_xception.compile(optimizer='adadelta', loss='binary_crossentropy', metrics=['accuracy'])
      # model_xception.summary()
      print('Xception has %d layers.' % len(model_xception.layers))
      return model_xception
```

```
In [25]: model_xception = xception(299)

Xception has 136 layers.
```

```
In [27]: model_xception.fit(X_train_299, y_train_299, batch_size=32, epochs=10, validation_data=(X_valid_299, y_valid_299))

Train on 19913 samples, validate on 4979 samples
Epoch 1/10
19913/19913 [==============================] - 479s 24ms/step - loss: 0.1947 - acc: 0.9557 - val_loss: 0.0796 - val_acc: 0.9811
Epoch 2/10
19913/19913 [==============================] - 470s 24ms/step - loss: 0.0768 - acc: 0.9786 - val_loss: 0.0537 - val_acc: 0.9847
Epoch 3/10
19913/19913 [==============================] - 470s 24ms/step - loss: 0.0651 - acc: 0.9809 - val_loss: 0.0478 - val_acc: 0.9869
Epoch 4/10
19913/19913 [==============================] - 470s 24ms/step - loss: 0.0579 - acc: 0.9819 - val_loss: 0.0427 - val_acc: 0.9875
Epoch 5/10
19913/19913 [==============================] - 470s 24ms/step - loss: 0.0547 - acc: 0.9816 - val_loss: 0.0408 - val_acc: 0.9873
Epoch 6/10
19913/19913 [==============================] - 470s 24ms/step - loss: 0.0546 - acc: 0.9813 - val_loss: 0.0390 - val_acc: 0.9871
Epoch 7/10
19913/19913 [==============================] - 470s 24ms/step - loss: 0.0484 - acc: 0.9842 - val_loss: 0.0373 - val_acc: 0.9875
Epoch 8/10
19913/19913 [==============================] - 470s 24ms/step - loss: 0.0493 - acc: 0.9833 - val_loss: 0.0367 - val_acc: 0.9877
Epoch 9/10
19913/19913 [==============================] - 472s 24ms/step - loss: 0.0540 - acc: 0.9819 - val_loss: 0.0368 - val_acc: 0.9875
Epoch 10/10
19913/19913 [==============================] - 473s 24ms/step - loss: 0.0491 - acc: 0.9834 - val_loss: 0.0355 - val_acc: 0.9877

Out[27]: <keras.callbacks.History at 0x7fb46c3de5f8>
```

观测 vac_loss 的变化，当其基本稳定了停止训练，然后预测结果并提交。

```
%matplotlib inline
import matplotlib.pyplot as plt

fig, ax = plt.subplots(2,2)
ax = ax.flatten()
his_model = model_xception.history
history = his_model.history

ax[0].plot(history['acc'],color='b',label="Train acc")
legend = ax[0].legend(loc='best', shadow=True)

ax[1].plot(history['val_acc'],color='r',label="Validation acc")
legend = ax[1].legend(loc='best', shadow=True)

ax[2].plot(history['loss'],color='g',label="Train loss")
legend = ax[2].legend(loc='best', shadow=True)

ax[3].plot(history['val_loss'],color='c',label="Valid loss")
legend = ax[3].legend(loc='best', shadow=True)
```



最终结果提交到 kaggle，分数为 0.0539，排名 90 左右，所以成绩不是很好，还有提升的空间。

3.3 改进

由于只使用单一的模型，最好的成绩只能达到前 100 名以内。因此使用多模型融合的提取特征和 fine-tune 可以进一步的提升模型性能。以 Xception 模型为例，通过 fine-tune 来解冻输出层，来进一步的提升结果表现。首先定义一个 fine-tune 方法：

```
def finetune(ft_model, preprocess, X_train, y_train, X_valid, y_valid, img_height, freeze_layer, batch_size=64, epochs=1, weights=None):
    # Preprocess: Standardization
    x = Input(shape=(img_height, img_height, 3))
    x = Lambda(preprocess)(x)

    # Base Model
    base_model = ft_model(input_tensor=x, weights='imagenet', include_top=False)
    for layer in base_model.layers:
        layer.trainable = True
    for layer in base_model.layers[:freeze_layer]:
        layer.trainable = False

    x = GlobalAveragePooling2D()(base_model.output)
    x = Dropout(0.25)(x)
    x = Dense(1, activation='sigmoid', kernel_initializer='he_normal')(x)

    model = Model(inputs=base_model.input, outputs=x, name='Transfer_Learning')
    model.compile(optimizer='adadelta', loss='binary_crossentropy', metrics=['accuracy'])
    print('Trainable: %d, Non-Trainable: %d' % get_params_element(model))

    if weights:
        model.load_weights(weights)

    logs_file = 'finetune-%s-{val_loss:.4f}.h5'%str(ft_model.__name__)
    path = os.getcwd()
    path_logs = os.path.join(path, logs_file)

    early_stop = EarlyStopping(monitor='val_loss', patience=5)
    # learning_rate_reduction = ReduceLROnPlateau(monitor='val_loss', patience=10, verbose=0,epsilon=0.0001,factor=0.5, min_lr=0)
    model_check = ModelCheckpoint(path_logs, monitor='val_loss', save_best_only=True)

    loss_history = LossHistory()

    model.fit(x=X_train, y=y_train, batch_size=batch_size, epochs=epochs,
              validation_data=(X_valid, y_valid), callbacks=[loss_history, model_check])
    return model
```

依次从输出层到输入层的方向来解冻，解冻并训练几轮后观察下 loss 和 accuracy 的情况。

```
finetune(Xception, xception_process, X_train_299, y_train_299, X_valid_299, y_valid_299, 299, 115, batch_size=64, epochs=1)
```

```
Trainable: 6791889, Non-Trainable: 14071640
Train on 20000 samples, validate on 5000 samples
Epoch 1/1
20000/20000 [==============================] - 565s 28ms/step - loss: 0.0463 - acc: 0.9845 - val_loss: 0.0215 - val_acc: 0.9922
```

```
log_file = 'finetune-Xception-0.0215.h5'
finetune(Xception, xception_process, X_train_299, y_train_299, X_valid_299, y_valid_299, 299, freeze_layer=85, weights=log_file)
```

```
Trainable: 11633817, Non-Trainable: 9229712
Train on 20000 samples, validate on 5000 samples
Epoch 1/1
20000/20000 [==============================] - 712s 36ms/step - loss: 0.0168 - acc: 0.9940 - val_loss: 0.0190 - val_acc: 0.9934
```

```
log_file = 'finetune-Xception-0.0190.h5'
model_Xception = finetune(Xception, xception_process, X_train_299, y_train_299, X_valid_299, y_valid_299, 299, freeze_layer=65, weights=log_fil
```

```
Trainable: 14861769, Non-Trainable: 6001760
Train on 20000 samples, validate on 5000 samples
Epoch 1/1
20000/20000 [==============================] - 812s 41ms/step - loss: 0.0065 - acc: 0.9979 - val_loss: 0.0191 - val_acc: 0.9940
```

可以看到，经过 fine-tune 后的训练，结果有一定幅度的提升。最终提交的结果成绩是 0.04496，排名为前 30 以内。

多模型融合提取特征也可以大幅度的提升模型性能，通过融合 Xception、 InceptionV3、InceptionResNetV2 模型来提取特征。先定义特征提取的方法并执行

```
def get_features(ft_model, preprocess, img_height, train_data, test_data, train_label=None, batch_size=64):
    inputs = Input(shape=(img_height, img_height, 3))
    in_tensor = Lambda(preprocess)(inputs)
    base_model = ft_model(input_tensor=in_tensor, weights='imagenet', include_top=False)
    x = GlobalAveragePooling2D()(base_model.output)
    model = Model(inputs=inputs, outputs=x)
    train_feature = model.predict(train_data, batch_size=batch_size)
    test_feature = model.predict(test_data, batch_size=batch_size)

    with h5py.File("feature_%s.h5" % ft_model.__name__, 'w') as f:
        f.create_dataset('train', data=train_feature)
        f.create_dataset('test', data=test_feature)
        f.create_dataset('label', data=train_label)
    return train_feature, test_feature
```

```
train_InceptionV3, test_InceptionV3 = get_features(InceptionV3, inception_v3_process, 299, X_train_299, X_test_299,train_label=y_train_299)

train_Xception, test_Xception = get_features(Xception, xception_process, 299, X_train_299, X_test_299, train_label=y_train_299)

train_InceptionResNetV2, test_InceptionResNetV2 = get_features(InceptionResNetV2, inception_resnet2_process, 299,
                                                               X_train_299, X_test_299, train_label=y_train_299)

# train_ResNet50, test_ResNet50 = get_features(ResNet50, renet50_process, 224, X_train_224, X_test_224, train_label=y_train_224)
```

```
# fetch_file = ['feature_Xception.h5', 'feature_InceptionV3.h5', 'feature_InceptionResNetV2.h5', 'feature_ResNet50.h5']

train, test= [], []
for fetchfile in ['feature_Xception.h5', 'feature_InceptionV3.h5', 'feature_InceptionResNetV2.h5']:
    with h5py.File(fetchfile, 'r') as h:
        train.append(np.array(h['train']))
        test.append(np.array(h['test']))
        label = np.array(h['label'])
```

载入提取的特征向量，并且将它们合并，划分训练集和验证集。构建好模型后设置 Dense 和 Dropout

```
# 划分训练集和验证集
X_ft_train, X_ft_valid, y_ft_train, y_ft_valid = train_test_split(fetch_feature_train, y_train_299, test_size=0.2, random_state=42)
```

```
inputs = Input(shape=(X_ft_train.shape[1],))

x = Dense(128, activation = "relu")(inputs)
x = Dropout(0.5)(x)
# x = Dropout(0.25)(inputs)
x = Dense(1, activation='sigmoid', kernel_initializer='he_normal')(x)
fetch_feature_model = Model(inputs=inputs, outputs=x)
fetch_feature_model.compile(optimizer='adadelta', loss='binary_crossentropy', metrics=['accuracy'])

logs_file = 'ft_extract_features-{val_loss:.4f}.h5'
path = os.getcwd()
path_logs = os.path.join(path, logs_file)

early_stop = EarlyStopping(monitor='val_loss', patience=5)
# feature_loss = LossHistory()
# learning_rate_reduction = ReduceLROnPlateau(monitor='val_loss', patience=10, verbose=0,epsilon=0.0001,factor=0.1, min_lr=0)
model_check = ModelCheckpoint(path_logs, monitor='val_loss', save_best_only=True)

fetch_feature_model.fit(X_ft_train, y_ft_train, batch_size=64, epochs=5,
              validation_data=(X_ft_valid, y_ft_valid), callbacks=[early_stop, model_check])
```

```
Train on 20000 samples, validate on 5000 samples
Epoch 1/5
20000/20000 [==============================] - 3s 134us/step - loss: 0.0433 - acc: 0.9847 - val_loss: 0.0153 - val_acc: 0.9944
Epoch 2/5
20000/20000 [==============================] - 2s 100us/step - loss: 0.0196 - acc: 0.9945 - val_loss: 0.0131 - val_acc: 0.9946
Epoch 3/5
20000/20000 [==============================] - 2s 99us/step - loss: 0.0155 - acc: 0.9949 - val_loss: 0.0164 - val_acc: 0.9938
Epoch 4/5
20000/20000 [==============================] - 2s 99us/step - loss: 0.0138 - acc: 0.9955 - val_loss: 0.0186 - val_acc: 0.9932
Epoch 5/5
20000/20000 [==============================] - 2s 101us/step - loss: 0.0129 - acc: 0.9961 - val_loss: 0.0126 - val_acc: 0.9952
```

训练完模型之后，loss 和 val_loss 都有一定的提升。调整参数多训练几轮之后，把预测的结果提交到 Kaggle ，结果基本稳定在 0.0383 左右。最终的成绩排名大概是前 15 名以内。

4 结果

4.1 模型评估与验证

采用的是三个模型(Xception、 InceptionV3、InceptionResNetV2)融合提取特征的方法作为最终的项目方案，提取特征后就可以训练模型了，在模型中添加了全连接层 Dense，激活函数是 relu，并添加 Dropout 层来防止过拟合。最终提交的成绩是 0.03826。

各模型训练的结果：

Model	Epochs	Logloss	Ranking
InceptionV3	10	0.07607	>220
Xception	10	0.05390	90
inceptionResNetV2	10	0.06508	<150
Xception 特征提取	8	0.04496	<30
模型融合	10	0.03826	<12

4.2 理由分析

经过不断的测试，可以看到多模型融合的效果是最好的。三种模型(Xception、 InceptionV3、InceptionResNetV2)融合的训练效果好于 fine-tune，也更好于单一模型。因为这三种模型的准确率都比较高，默认输入图片的大小都是 299*299，所以将它们综合到一起，可以很好的获取图片的信息。

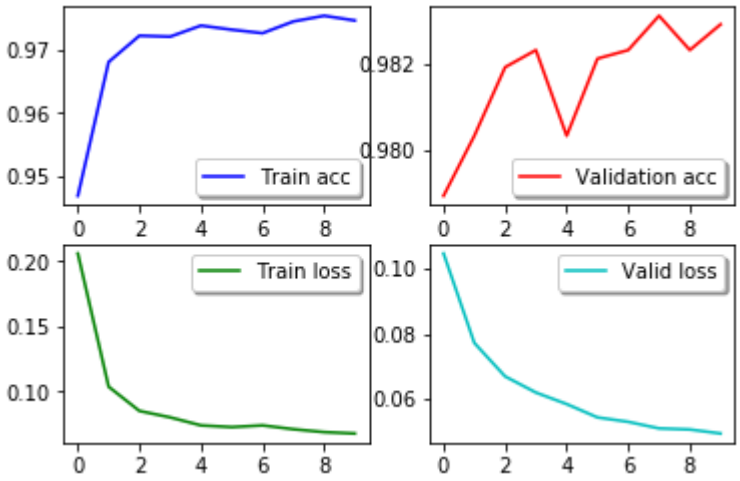
5 结论

5.1 总结

整个项目的操作步骤大致分为：数据预处理、使用预训练的模型提取特征向量、构建模型并训练、使用测试集来预测结果并提交。

预测结果要有好的表现，其中最重要的是提取特征向量。对于特征向量的提取，直接使用了预训练好的模型，既能节省时间又能得到较好的结果。通过不同的模型组合来训练，三种模型(Xception、 InceptionV3、InceptionResNetV2)的效果是最佳的。训练模型时设置了 Dropout 以一定的概率忽略隐藏层节点来防止过拟合，最终发现在 0.25 时的表现最好。

模型训练的 loss 变化曲线：



5.2 改进

如果想进一步的提高训练模型的准确率，可以在多模型融合特征的基础上进行图片增强，或者添加额外的数据集，或者使用其他的模型，并不断的调整参数和超参数来获取更好的结果。

6 参考文献

[1] http://keras-cn.readthedocs.io/en/latest/

[2] https://arxiv.org/abs/1610.02357_Xception

[3] https://arxiv.org/abs/1512.00567_Rethinking_Inception

[4] https://arxiv.org/abs/1602.07261_Inception-ResNet

[5] https://arxiv.org/abs/1512.03385_Deep_Residual_Learning

[6] https://en.wikipedia.org/wiki/Convolutional_neural_network

[7] https://cs231n.github.io/

[8] https://deeplearning4j.org/convolutionalnetwork

[9] https://www.tensorflow.org/

[10] https://github.com/fchollet/deep-learning-models