机器学习纳米学位毕业项目

基于深度学习的猫狗种类识别

王颖

2018.09.13

目录

1. 定义

1.1 项目概览

深度学习是近十年来人工智能领域取得的重要突破。它在语音识别、自然语言处理、计算机视觉、图像与视频分析、多媒体等诸多领域的应用取得了巨大成功[1]。图像分类是根据图像的语义信息将不同类别图像区分开来，是计算机视觉中重要的基本问题，也是图像检测、图像分割、物体跟踪、行为分析等其他高层视觉任务的基础，图像分类在很多领域有广泛应用。

而猫狗大战是机器学习历史上一个很有趣的话题，也是大部分入门者入手处理的一个项目。通过 Kaggle 提供的大量带有标签的猫和狗图像，我们可以通过深度学习使得计算机从这些图像中学习并提取到猫狗图像特征，生成特定深度学习模型，进而可以对没有标签标注的猫狗图像进行分类处理。

本项目将基于 Keras 接口，通过 Xception 搭建一个识别猫狗种类的深度学习模型，基于 Kaggle 提供的数据集对模型进行训练及优化，最后将对新获取的猫狗图像进行分类测试，验证模型效果。

1.2 问题描述

“猫狗大战”是图像识别中的一个典型的二分类问题，通过深度学习网络基于标注数据训练一个分类模型，进而对不包含标签数据的猫狗图像进行分类预测。

1.3 评价指标

与 Kaggle 评分指标一致，我们选择 Log Loss，即对数损失作为我们的模型性能评价指标，即对数似然损失(Log-likelihood Loss)，在二分类情况下，对数损失函数的公式简化为：

$$-\frac{1}{N}\sum_{i=1}^{N}(y_i \log p_i + (1-y_i)\log(1-p_i))$$

y_i为输入实例 xi 的真实类别，p_i 为预测输入实例 xi 属于类别 1 的概率. 对所有样本的对数损失表示对每个样本的对数损失的平均值，对于完美的分类器，对数损失为 0[2]。

Log Loss 对应模型的预测及分类能力，当 Log Loss 较小时，代表模型的预测能力越好；当其值较大时，表示模型的分类预测能力越弱。

2. 分析

2.1 数据研究

本项目用于训练及测试的数据集来自 Kaggle 的【Dogs vs. Cats Redux: Kernels Edition】项目，数据集地址：https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition/data。

本数据集 Train 训练图像一共 25000 张，共分为 cat 和 dog 两类，其中带有标签的猫狗图像各有 12500 张；Test 测试图像一共 12500 张，不包含标签数据。

图 1 cat 示例图像

图 2 dog 示例图像

我们注意到，上面展示的猫狗图像的尺寸 size 不统一，通过训练图像的尺寸散点图分布，我们可以得知其尺寸范围，以及其异常值分布如图 3 所示。

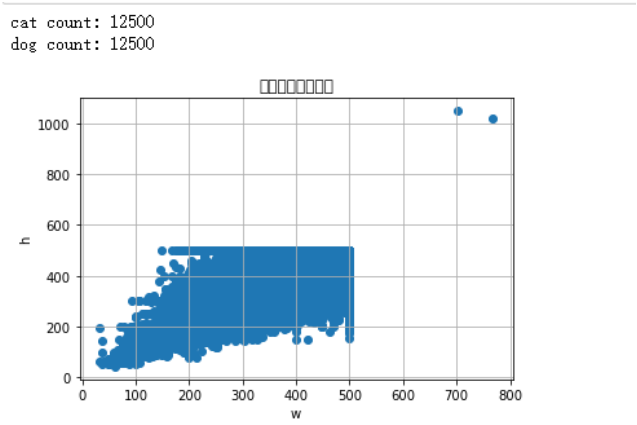

图 3 训练图像尺寸分布

我们发现存在两个异常值点，查阅发现其为

```
/home/sakulaki/code/udapro/dogcat/data/train/cat/cat.835.jpg
/home/sakulaki/code/udapro/dogcat/data/train/dog/dog.2317.jpg
```

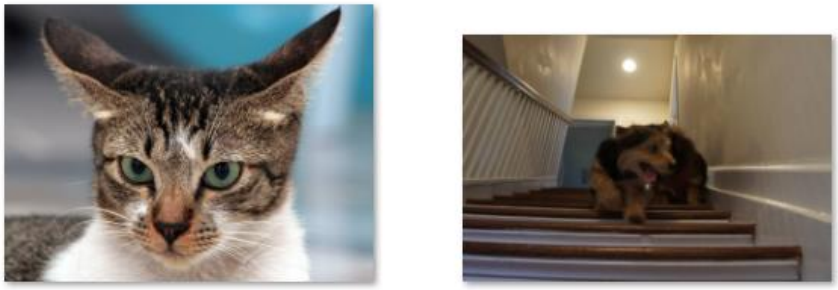

图 4 尺寸异常图像及路径

经查阅发现这个是正常图像，不可剔除。

手动查阅图像时，发现存在非猫非狗图像。如果我们想基于此数据集训练模型，需要对这些异常值进行剔除。我们引入 Xception 网络利用 imagenet 预分类模型对训练集进行分类，如果 topN 不含有‘dog’或‘cat’视为异常值。通过该模型的分类，我们识别出 33 个异常结果，如下图所示：

图 5 识别出的异常图像（部分）

我们将这些异常值从训练集中删除。删除后的训练图像分布如下图所示：

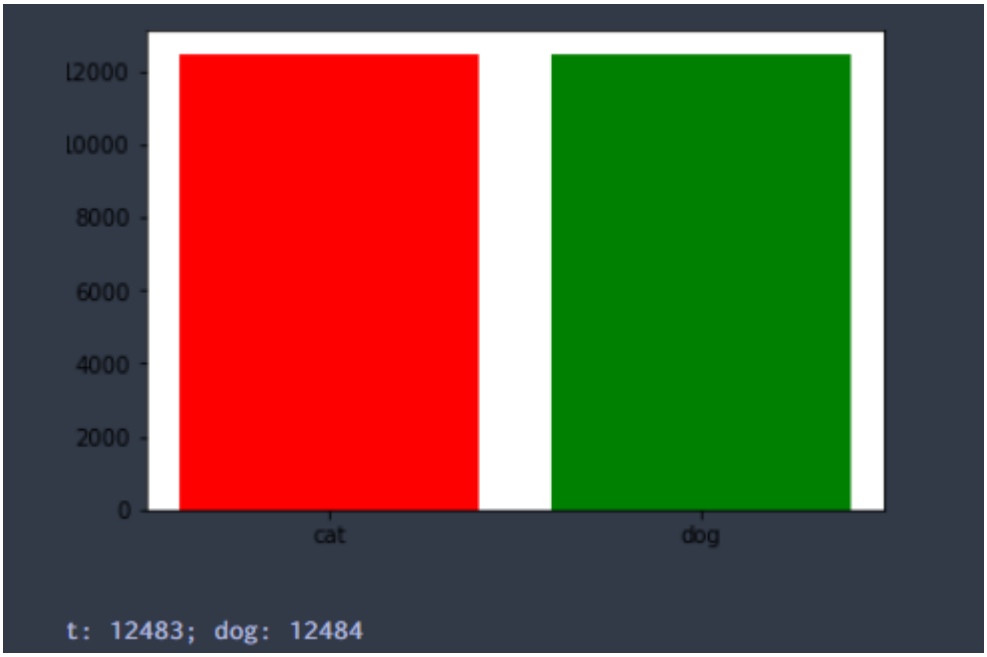

图 6 剔除非猫非狗图像后的训练图像分布

其中 cat 和 dog 图像分别有 12483 和 12484 张

2.2 算法简述

猫狗大战的本质是一个图像二分类问题。目前使用较为广泛的图像分类算法有 VGG、Darknet、Xception、DenseNet 等神经网络模型。

在本项目中，我们将使用 Python 的 Keras 接口，生成猫狗识别的二分类模型，利用 Fine-Tune Xception 深度神经网络模型，基于上文提及的 Kaggle Train 数据集进行训练。训练完成后我们将利用生成的二分类模型对测试集 Test 进行预测，并提交预测结果至 Kaggle 进行验证。

下面我们将对本项目引用到的关键技术名字做简要描述：

2.2.1 卷积神经网络

卷积神经网络（Convolutional Neural Network, CNN）是一种前馈神经网络，由一个或多个卷积层和顶端的全连通层（对应经典的神经网络）组成，同时

也包括关联权重和池化层（pooling layer）。这一结构使得卷积神经网络能够利用输入数据的二维结构。与其他深度学习结构相比，卷积神经网络在图像和语音识别方面能够给出更好的结果。这一模型也可以使用反向传播算法进行训练。相比较其他深度、前馈神经网络，卷积神经网络需要考量的参数更少，使之成为一种颇具吸引力的深度学习结构[4]。卷积神经网络一般包含卷积层、线性整流层、池化层和损失函数层：

卷积层：卷积运算的目的是提取输入的不同特征，多层网络能够从低级特征中迭代提取更为复杂的特征。

线性整流层：用于充当每层神经的记录函数，可以增强判定函数和整个神经网络的非线性特征。

池化层(Pooling Layer)：池化层通常会分别作用于每个输入的特征并减小其大小，池化层会不断地减小数据的空间大小，因此参数的数量和计算量也会下降，这在一定程度上也控制了过拟合。目前最常用形式的池化层是每隔 2 个元素从图像划分出 2 X 2 的区块，然后对每个区块中的 4 个数取最大值。这将会减少 75%的数据量，如图 3 所示：

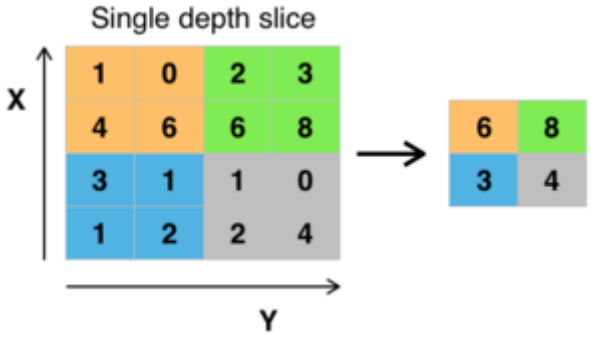

图 7 每隔 2 个元素进行的 2x2 最大池化

损失函数层：损失函数层（loss layer）用于决定训练过程如何来“惩罚”网络的预测结果和真实结果之间的差异，它通常是网络的最后一层[4]。

2.2.2 Fine-Tune

所谓 fine tune 就是用别人训练好的模型，加上我们自己的数据，来训练

新的模型。fine tune 相当于使用别人的模型的前几层，来提取浅层特征，然后在最后再落入我们自己的分类中。

fine tune 的好处在于不用完全重新训练模型，从而提高效率，因为一般新训练模型准确率都会从很低的值开始慢慢上升，但是 fine tune 能够让我们在比较少的迭代次数之后得到一个比较好的效果[5]。

2.2.3 Xception

Xception 表示「extreme inception」。和 Inception 架构一样，它重塑了我们看待神经网络的方式—尤其是卷积网络，它将 Inception 的原理推向了极致。Xception 的假设是：「跨通道的相关性和空间相关性是完全可分离的，最好不要联合映射它们。」

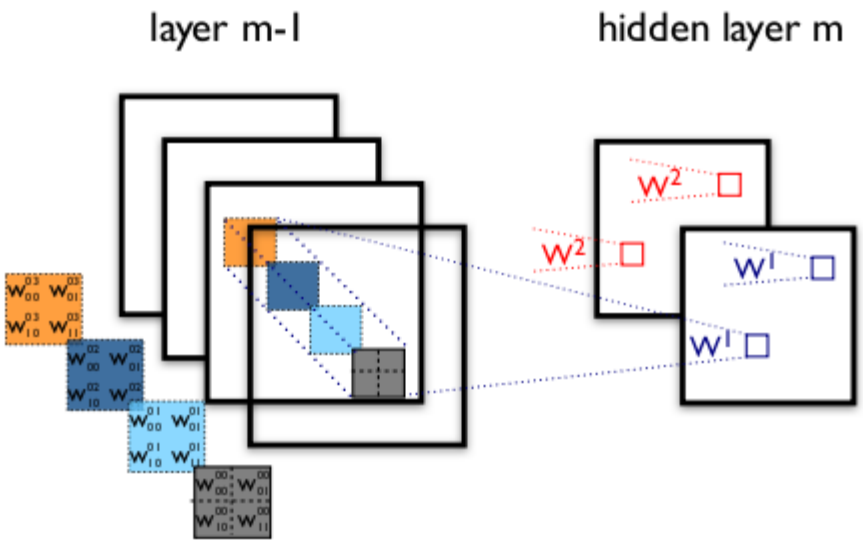

图 8 标准卷积层

在传统的卷积网络中，卷积层会同时寻找跨空间和跨深度的相关性。在上图中，过滤器同时考虑了一个空间维度（每个 2×2 的彩色方块）和一个跨通道或「深度」维度（4 个方块的堆叠）。在输入图像的输入层，这就相当于一个在所有 3 个 RGB 通道上查看一个 2×2 像素块的卷积过滤器。

在 Inception 中，我们开始将两者稍微分开。我们使用 1×1 的卷积将原始输入投射到多个分开的更小的输入空间，而且对于其中的每个输入空间，我们都使用一种不同类型的过滤器来对这些数据的更小的 3D 模块执行变换。Xception 更进一步。不再只是将输入数据分割成几个压缩的数据块，而是为每个输出通道单独映射空间相关性，然后再执行 1×1 的深度方面的卷积来获取跨通道的相关性[6]。

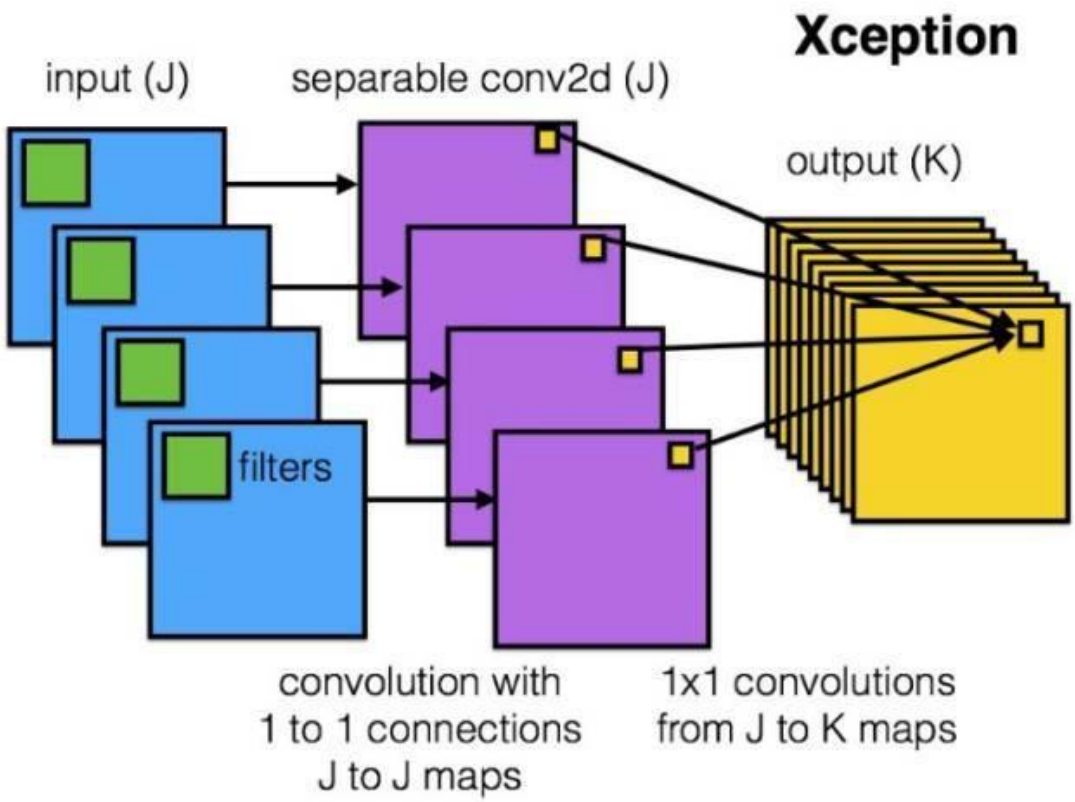

图 9 Xception 的卷积运算

2.3 基准模型与期望值

我们的基准模型采用 Xception，并预期可以在 Kagge 测试数据集上的得分能够进入前 10%，Log Loss 不高于 0.06149。

3. 解决方案

3.1 数据预处理

在本项目中，我们的预处理操作主要包括异常图像剔除、图像 shuffle、图像填充和图像归一化处理。

1. 异常图像剔除，已在 2.1 数据分析章节进行处理，主要操作为剔除非猫非狗图像，在本项目中，共找出异常图像共计 33 张，在输入模型前进行剔除。

2. 图像 shuffle，将输入训练模型的训练数据进行随机排列。在这里我们使用 python 的 random 模块对训练数据进行 shuffle。Shuffle 函数可以对输入的数组进行随机排列。

```
In [2]: # 获取图像路径
        train_image_path = "/home/sakulaki/code/udapro/dogcat/data/resized/"
        train_images_path = FilesScanner(train_image_path, suffix='.jpg').get_files()

        # 图像shuffle
        import random
        random.shuffle(train_images_path)
```

图 10 训练数据 shuffle 处理

3. 图像填充，将图像尺寸统一 resize 为 299*299*3。在这里我们使用 python 的 cv2 模块的 resize 方法对图像进行 resize 操作，使用默认的 interpolation=CV_INTER_LINEAR 双线性插值法对图像进行放缩，进而 resize 为统一尺寸。

```
img = cv2.imread(path)
img = img[:, :, ::-1]
img = cv2.resize(img, (299, 299))
```

图 11 Image Resize

4. 获取 shuffle 及 resize 后的图像数据及图像标签，并按照 9：1 的比例生成测试集与验证集。

```
for name in train_images_path:
    images.append(cv2.imread(name))
    if 'cat' in name:
        labels.append(0)
    else:
        labels.append(1)

X = np.asarray(images)
Y = np.asarray(labels)
```

```
4]:
# 划分训练集和验证集,将训练数据分为10份，9份作为训练集，剩下的一份作为测试集
boundary = int(len(train_images_path) / 10)

# 验证集
valid_X = X[:boundary]
valid_Y = Y[:boundary]
# 训练集
train_X = X[boundary:]
train_Y = Y[boundary:]
```

图 12 生成训练集与验证集

3.2 训练模型

3.2.1 构建模型

1. 图像的归一化处理

在训练数据输入模型之前，我们使用 xception 提供的 preprocess_input 接口对我们的图像进行归一化处理，对输入数据进行标准化。这里的处理操作主要为：在不改变数据分布的情况下，通过对像素值/255，将像素值压缩到[0,1]之间。

2. 图像增强处理

我们使用 keras 的 ImageDataGenerator 模块对我们的训练数据及验证数据进行增强处理，以扩展训练数据量。该函数提供的数据增强方法包含很多中，如旋转|反射变换(Rotation/reflection)、翻转变换(flip)、缩放变换(zoom)、尺度变换(scale)等，在本文主要使用了 shear-水平或垂直投影变换、zoom- 按照一定的比例放大或者缩小等方式来增强我们的训练数据。

```python
# 训练数据增强
train_datagen = ImageDataGenerator(preprocessing_function=xception.preprocess_input,
                                   shear_range=0.2,
                                   zoom_range=0.2,
                                   horizontal_flip=True)
```

图 13 训练数据增强处理

3. 执行过程

我们选用 Xception 作为我们的基础模型，模型初始化时加载 ImageNet 作为预训练权重，不包含 top 分类器。epochs 设置为 30，我们采用多 GPU 训练，batch_size 设置为 128。我们利用 keras 中的 EarlyStopping 回调函数，当 val_loss 经过 5 个训练轮数不再优化时停止训练（即 patience 设置为 5），并保存训练停止时的 epoch 训练后的 model。本次训练的过程中，我引入了 ModelCheckpoint，对每轮 epoch 的训练结果都可以进行存储。

In [7]:
```python
with tf.device('/cpu:0'):
    base_model = xception.Xception(weights='imagenet', input_shape = (299,299,3), include_to
    x = base_model.output

    # 二分类分类器
    predictions = Dense(1, activation='sigmoid')(x)

    # 冻结所有层，只训练top layer
    for layer in base_model.layers:
        layer.trainable = False

```

In [9]:
```python
# 绘制结果曲线
history = LossHistory(model)

# earlyStopping用于当检测值不在改善时终止训练
earlyStopping = callbacks.EarlyStopping(monitor='val_loss', patience=5, verbose=0, mode='aut

# 保存点保存模型
model_save_path = "model-{epoch:02d}-{val_loss:.2f}.hdf5"
checkpoint = ParallelModelCheckpoint(model, model_save_path)
#训练模型
parallel_model.fit_generator(train_data, steps_per_epoch=steps_per_epoch, epochs=epochs,
                    validation_data=valid_data,
                    validation_steps=valid_steps,
                    callbacks=[history, earlyStopping, checkpoint])
```

图 14 构建模型

我们采用 adam 作为优化器，lr 设置为 0.0001，fine-tune 模型。模型训练分两个部分，第一部分为训练 top 分类器，第二部分基于第一部分的最优结果，分

为 4 个实验进行优化。

```
In [8]:
1  # 编译模型,采用多GPU进行
2  parallel_model = multi_gpu_model(model, gpus=2)
3  parallel_model.compile(optimizer=Adam(lr=0.0001), loss='binary_crossentropy', metrics=['accu
4
5  parallel_model.summary()
```

图 15 多 GPU 训练

第一步，只训练顶端的二分类器：

```
Epoch 00020: val_loss did not improve from 0.08850
Epoch 21/30
157/157 [==============================] - 203s 1s/step - loss: 0.0527 - acc: 0.9869 - val_loss: 0.0853 - val_acc: 0.9818

Epoch 00021: val_loss improved from 0.08850 to 0.08529, saving model to model-21-0.09.hdf5
Epoch 22/30
157/157 [==============================] - 204s 1s/step - loss: 0.0546 - acc: 0.9841 - val_loss: 0.0936 - val_acc: 0.9796

Epoch 00022: val_loss did not improve from 0.08529
Epoch 23/30
157/157 [==============================] - 205s 1s/step - loss: 0.0513 - acc: 0.9868 - val_loss: 0.0902 - val_acc: 0.9802

Epoch 00023: val_loss did not improve from 0.08529
Epoch 24/30
157/157 [==============================] - 205s 1s/step - loss: 0.0503 - acc: 0.9869 - val_loss: 0.0899 - val_acc: 0.9800

Epoch 00024: val_loss did not improve from 0.08529
Epoch 25/30
157/157 [==============================] - 203s 1s/step - loss: 0.0506 - acc: 0.9849 - val_loss: 0.0875 - val_acc: 0.9802

Epoch 00025: val_loss did not improve from 0.08529
Epoch 26/30
157/157 [==============================] - 205s 1s/step - loss: 0.0469 - acc: 0.9871 - val_loss: 0.0858 - val_acc: 0.9808

Epoch 00026: val_loss did not improve from 0.08529

<keras.callbacks.History at 0x7fe1d1b0b518>
```

图 16 顶端的二分类器训练过程

经过 26 个 epoch，共耗时 5330 秒，loss 与 accuracy 变化趋势如下所示：

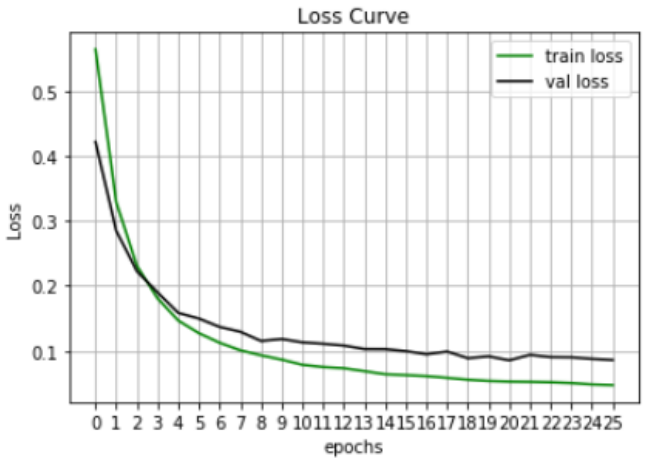

图 17 Loss 变化趋势

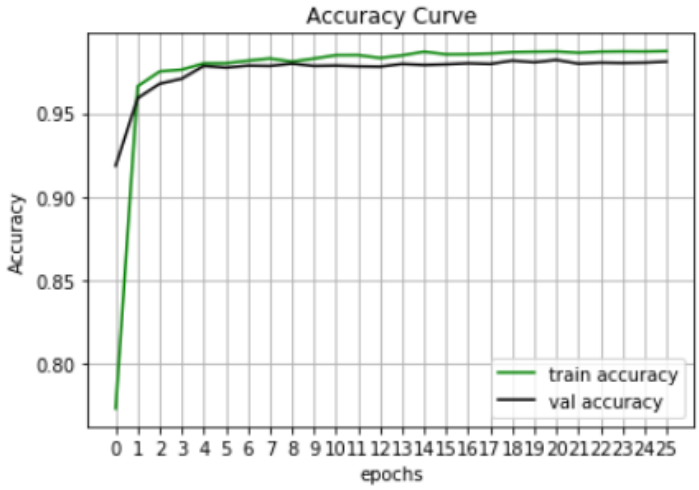

图 18 Accuracy 变化趋势

从以上训练结果可以看出，在 epoch=21 时，达到了 val_loss 最低，但是后面几轮的训练表现出了 val_loss 继续下降的趋势，分析，由于 epochs 的值设置过小，或 patient 值设置较小，导致训练提前结束，还没达到最好的训练效果，可能出现抖动，故继续以下实验

第二步，基于第一步的最优结果，分实验进行优化

实验一 选用上一步生成的最后一个模型 xception_finetune_top.h5，冻结所有层，继续训练，epoch=50

In [12]:

```
epochs = 50
with tf.device('/cpu:0'):
    del model
    model = load_model('xception_finetune_top.h5')

    # 冻结所有层，只训练top layer
    for layer in base_model.layers:
        layer.trainable = False

    model = Model(inputs=base_model.input, outputs=predictions)
```

图 19 继续第一步的训练过程

由于我们引入了 earlyStopping，经过 14 个 epoch 训练就停止了，共耗时 2898 秒，loss 与 accuracy 变化趋势如下所示：

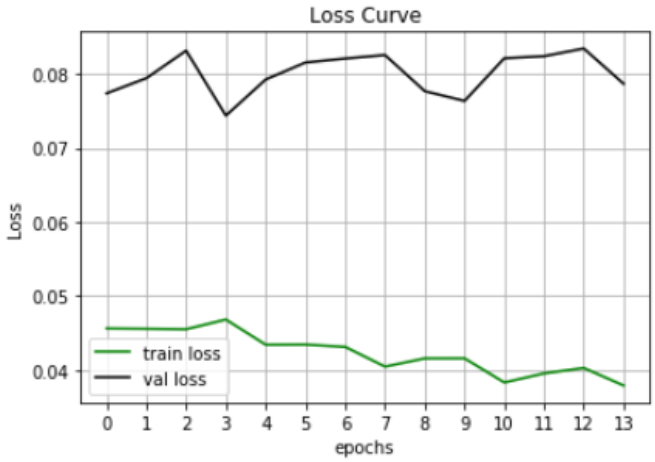

图 20 Loss 变化趋势

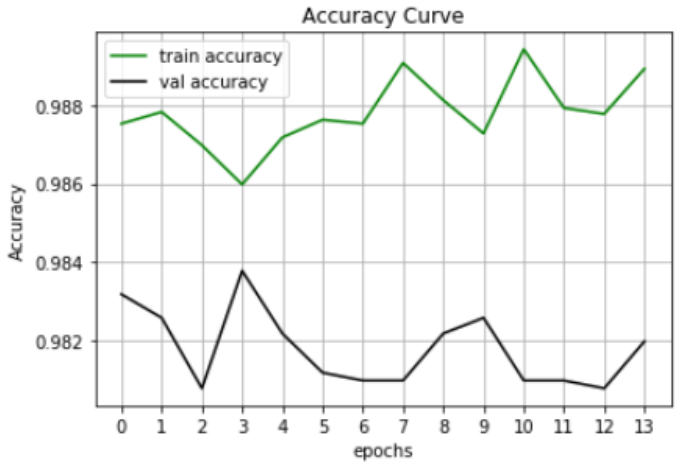

图 21 Accuracy 变化趋势

从实验一的训练结果来看，在 epoch=4 时，达到最低的 val_loss=0.0744,故选用 model-test-1-04-0.07.hdf5 作为后续实验的 base_model。

实验二 基于实验一结果，冻结前 12 个 block，开放最后两个 block

In [14]:

```
with tf.device('/cpu:0'):
    del model
    model = load_model('model-test-1-04-0.07.hdf5')

    for layer in model.layers[:116]:
        layer.trainable = False
    for layer in model.layers[116:]:
        layer.trainable = True
```

图 22 开放顶部 2 个 block

本次实验训练在第 16 轮停止，在第 6 轮的时候，val_loss 达到最低。本轮训练共耗时 3312 秒，loss 和 accuracy 曲线变化趋势如下所示。

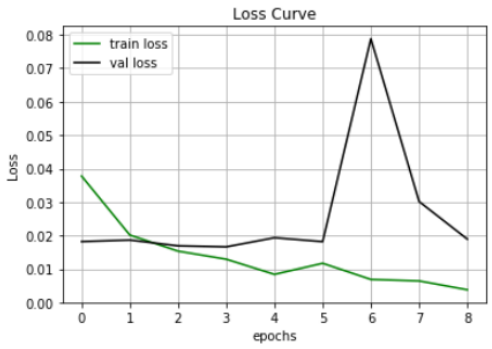

图 23 loss 变化趋势

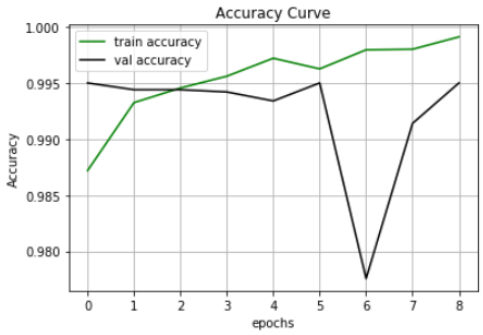

图 24 accuracy 变化趋势

从实验二的训练结果来看，在 epoch=6 时，达到最低的 val_loss=0.0165，生成模型 model-test-2-06-0.02.hdf5。

实验三 基于实验一结果，冻结前 10 个 block，开放最后 4 个 block

In [18]:
```
with tf.device('/cpu:0'):
    del model
    model = load_model('model-test-1-04-0.07.hdf5')

    for layer in model.layers[:96]:
        layer.trainable = False
    for layer in model.layers[96:]:
        layer.trainable = True
```

图 25 开放最后 4 个 block

本次实验训练在第 12 轮停止，在第 2 轮的时候，val_loss 达到最低。本轮训练共耗时 2544 秒，loss 和 accuracy 曲线变化趋势如下所示。

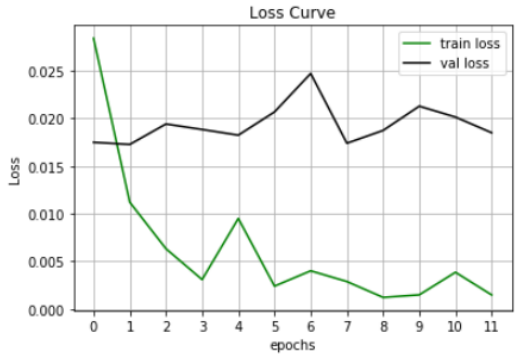

图 26 loss 变化趋势

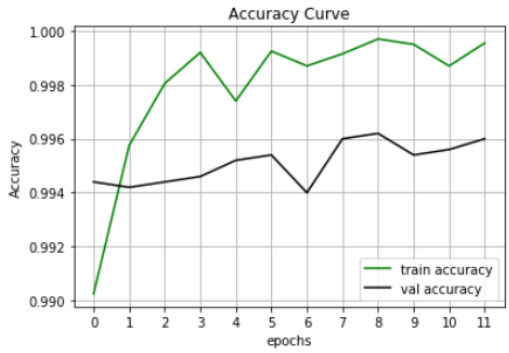

图 27 Accuracy 变化趋势

从实验三的训练结果来看，在 epoch=2 时，达到最低的 val_loss=0.0173，生成模型 model-test-3-02-0.02.hdf5。

实验四 基于实验一结果，使用 SGD 优化器，lr=0.0001，momentum=0.9，冻结前 10 个 block，一次性开放最后 4 个 block。

```
In [*]:
with tf.device('/cpu:0'):
    del model
    model = load_model('model-test-1-04-0.07.hdf5')

    for layer in model.layers[:96]:
        layer.trainable = False
    for layer in model.layers[96:]:
        layer.trainable = True

# 编译模型,采用多GPU进行
parallel_model = multi_gpu_model(model, gpus=2)

from keras.optimizers import SGD
parallel_model.compile(optimizer=SGD(lr=0.0001, momentum=0.9), loss='binary_crossentropy', m
```

图 28 采用 SGD 优化器，开放最后 4 个 block

本次实验训练在第 50 轮停止，尚未达到 val_loss 最低，因为资源所限，所以

在 epoch=50 停止，如果 epoch 设置更大的话，应该还有更好的表现。本轮训练共耗时 10550 秒，loss 和 accuracy 曲线变化趋势如下所示。

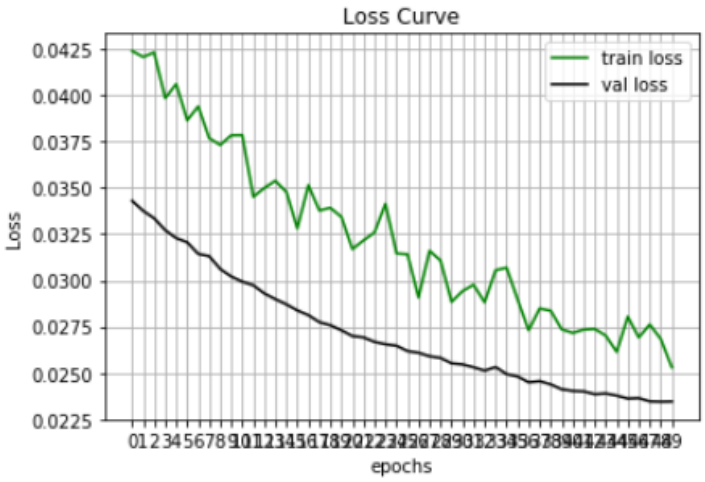

图 29 loss 变化趋势

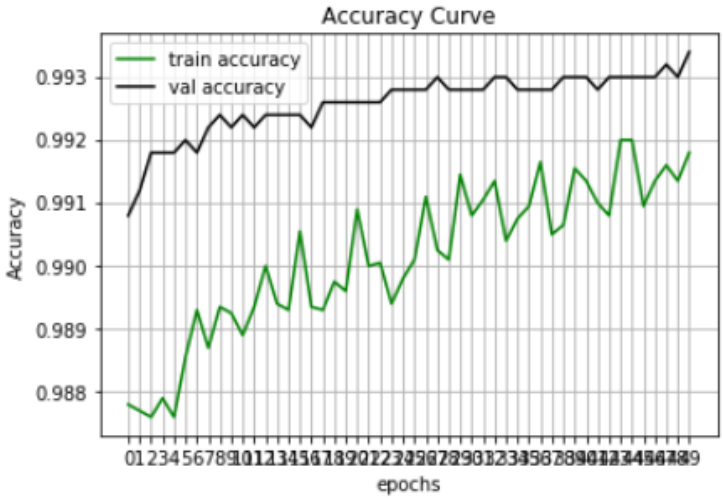

图 30 Accuracy 变化趋势

从实验四的训练结果来看，虽然训练过程中有震荡，但是 val_loss 有一直降低的趋势，不难看出，采用了 SGC 优化器后，模型的收敛速度变慢。实验四生成模型 model-test-4-49-0.02.hdf5。

3.3 测试集测试

我们基于 3.2 生成的 5 个模型分别对测试集进行测试，测试结果提交 kaggle，分模型得分结果如下：

1. model-21-0.09.hdf5， score=0.09701

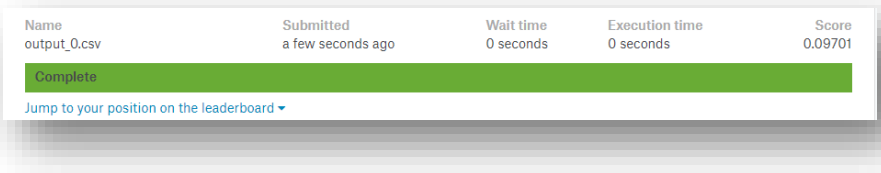

图 31 score of model-21-0.09.hdf5

2. model-test-1-04-0.07.hdf5， score=0.08627

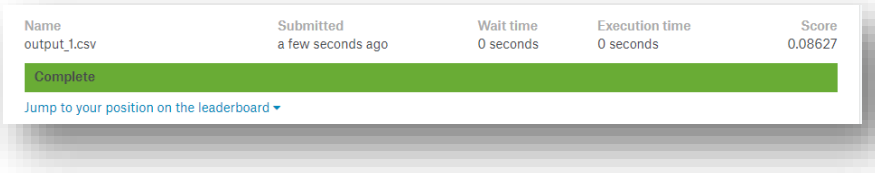

图 32 score of model-test-1-04-0.07.hdf5

3. model-test-2-06-0.02.hdf5， score=0.03855

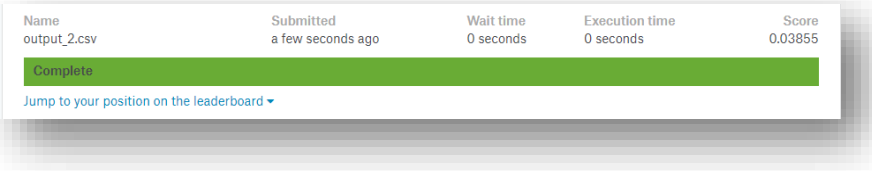

图 33 score of model-test-2-06-0.02.hdf5

4. model-test-3-02-0.02.hdf5， score=0.03889

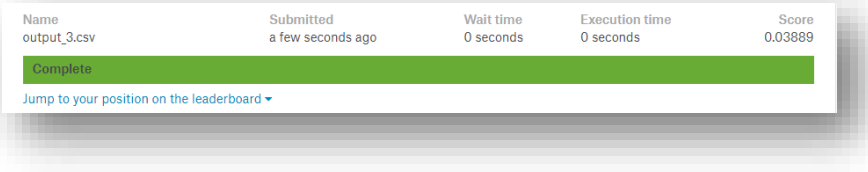

图 34 score of model-test-3-02-0.02.hdf5

5. model-test-4-49-0.02.hdf5， score=0.04156

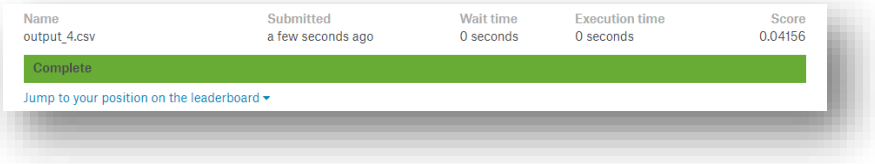

图 35 score of model-test-4-49-0.02.hdf5

如上实验结果，实验三的实验效果最好，实验四较次之，在 kaggle 上的排名达到了 11~13 名左右，这是我这四五次作业提交中最好的成绩了。实验五应该随着 epoch 的增加会有更好的表现。

4. 优化方向

经过四次作业提交，在评阅老师的指导下，也在自己的努力下，目前对机器学习的概念和应用有了更深一步的了解和使用。

在本次作业提交中，虽然结果能达到作业要求，但这应该不是我目前能到达的最好的结果，限于时间和经历，在作业上只能以目前的成果进行提交，但是对于这个项目，在以后的周末中我会继续投入精力进行优化，发现这个不断探索的过程真的是很有趣儿。

在本次作业中，有对别人项目及说明的参考，我加入了自己的理解和自己目前能力，引入了 adam / SGC 优化器，自定义了 lr 和 batch_size，也使用了多 GPU 进行训练，相比单个 GPU，速度简直快的飞起。后面的话，可以在开放的 block 层数上，以及不同的优化器还有 base_model 都可以使用别的可选项进行，应该也会有不同的结果，这是我以后周末的任务方向了。

感谢评阅老师的耐心和细心指导，这个作业，收获贼大。

5. 结论

基于我们目前的训练参数以及条件限制，使用了 xception 作为，分别采用了 adam / SGD 作为优化器，通过四次不同的实验，分别生成了不同的 model。基于这 5 个不同的 model，分别对测试集进行预测，取得了不同的

score，其中最优模型为。虽然结果尚可接受，但是优化空间还很大。毕业不是学习的截至，而是另一端学习的开始。我会继续努力的，感谢各位老师！

参考资料：

[1] 王晓刚，图像识别中的深度学习《中国计算机学会通讯》第 8 期《专题》，[Online] Available:

https://www.cnblogs.com/nowornever-L/p/6392577.html (2017.02.13)

[2] 对数损失函数(Logarithmic Loss Function)的原理和 Python 实现，[Online] Available: http://www.cnblogs.com/klchang/p/9217551.html(2018.06.23)

[3] 凌蓝风， 毕业设计 Dogs vs Cats For Udacity P7 (异常值检验) ，[Online] Available: https://zhuanlan.zhihu.com/p/34068451(2018.05.21)

[3] jiandanjinxin，图像分类，[Online] Available:

https://www.jianshu.com/p/9870ab8743be(2017.12.19)

[4] 卷积神经网络，[Online] Available:

https://zh.wikipedia.org/wiki/卷积神经网络(2018.06.23)

[5] 深度学习笔记（六）finetune ，[Online] Available:

https://blog.csdn.net/xuanyuyt/article/details/59174613(2017.03.01)

[6] 无需数学背景，读懂 ResNet、Inception 和 Xception 三大变革性架构，[Online] Available: https://www.sohu.com/a/166062301_465914(2017.08.20)

[7] Xception: Deep Learning with Depthwise Separable Convolutions, [Online] Available: https://arxiv.org/abs/1610.02357 (2017.04.04)

[8] Keras, [Online] Available: https://keras-cn.readthedocs.io/en/latest/