CVDL 第一次作业

姓名：王帅　学号：1800012995　kaggle：iridescent black

选题：细粒度分类 GPU

首先理解一下细粒度分类，细粒度图像分类相对于传统图像分类而言，需要进行分类的图像中的可判别区域往往只是在图像中的很小一块区域内。在传统的图像分类中，无论图像中重要判别区域占整个图像的比重有多大，都只会对整张图片一并提取特征，这样一来，一旦一些对判定类别重要的区域占比较小时，大量与类别无关的背景信息就会被训练进去，增加了图片分类的难度，降低了分类精度。

引入细粒度图片分类，就是为了解决这一问题，如果我们关注图片中细小的差别，实现了更加精确的图像分类，就可以让网络对分类物体的特征提取更加有效。经过网上查找，我了解到目前可以用于细粒度分类的模型有 MACNN、RA-CNN 等。一开始设想尝试实现 MACNN 模型解决该问题，但听课过程中对 dendeNet 印象比较深刻，也想借此机会尝试一下该模型，故在这次作业中利用 denseNet 模型进行图片分类

基本训练过程

1. 模型实现

利用 keras 库实现 denseNet 网络构建，参考课上关于 denseNet 的 ppt 构筑网络

(1) bottleneck BN ReLU Conv(1x1) BN ReLU Conv(3x3)

(2) dense block 通过 keras 提供的 concatenate 函数，将各个 bottleneck 链接成一个 denseBlock

(3) denseNet 先对输入进行卷积，之后链接多个 dense Block，每个 block 之间通过 conv 和池化层相连，为了增强效果，我又在其中添加了一个 BN 层，最后模型尾端进行池化，和 softmax

2. 数据处理

a) 通过 opencv 库的 cv2.imread 函数将图像数据读入，然后将数据保存在一个 numpy 数组里，我在实验中尝试将数据经读取后存在一个.npy 的 numpy 数据格式文件中，这样下次再读入时，就不用再一次读取数据了，但是这样得到的.npy 文件很大，而且再次读入时还是耗时很长。

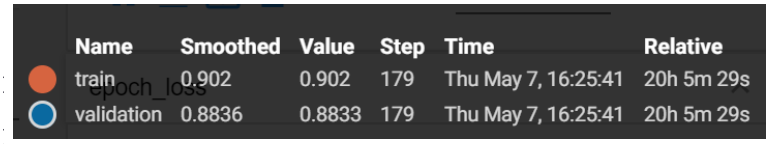
b) 我们将 csv 文件里保存的类型数据通过 keras 中提供的 keras.utils.to_categorical 函数处理成 onehot 向量，方便之后进行训练

3. 模型训练

由于 colab 需要科学上网，而且数据集很大，传到谷歌网盘上面耗时过长，我使用了国内的算力，平台是 mistgpu，使用 V100 进行训练，训练每个模型大概 20 个小时。

4. 数据可视化：

使用 Tensorboard 进行数据可视化，下图为每轮训练的准确度曲线



5. 利用得到的 densenet.h5 模型，加载模型对 test 图片进行预测

训练手段、遇到的问题

1. 数据增强：

a) 在数据处理阶段，将读入的数据进行预处理，令其减去均值再除以标准差

```
def color_prefit(x):
    x=x.astype('float32')
    for i in range(3):
        x[:,:,:,i]=(x[:,:,:,i]-np.mean(x[:,:,:,i]))/np.std(x[:,:,:,i])
    return x
```

b) 利用 keras 提供的 ImageDataGenerator 函数进行数据增强，

```
datapowerful=ImageDataGenerator(horizontal_flip=True,
                                width_shift_range=0.125,
                                 vertical_flip=True,
                                height_shift_range=0.125,
                                fill_mode='constant',
                                cval=0.)
```

尝试通过对图片进行随机水平翻转、随机上下翻转、左右平移、上下平移来进行数据增强。就我个人来看，由于处理的问题是细粒度分类，包含类别特征的像素可能位于靠近图片边界的位置，所以上下左右平移的范围不能过大。

2. 学习率调整

学习率调整有利于提高模型精度、加快训练速度，在第一次训练过程中，我将总 epoch 数设为 300，每个 epoch 进行 400 次循环，分别在 100、200、300 个 epoch 时调整学习率，将其减小为原来的 10%，经过观察，这样训练时，到了改变学习率的前数十个 epoch 是，预测准确度增长几乎为 0，严重减慢了训练速度。

通过对首次训练，我将以后的训练总 epoch 数下降到了 185，学习率调整函数如下

```
def scheduler(epoch):
    if epoch<60:
        return 0.1
    if epoch<110:
        return 0.01
    if epoch<150:
        return 0.001
    else:
        return 0.0001
```

3. 损失函数的选择

由于一次模型训练时间过长(算力有点小贵),本次实验一共对两种损失函数进行尝试，

A） 交叉熵损失函数 categorical_crossentropy

B） KL 散度损失函数 kullback_leibler_divergence

经过实验，使用交叉熵的模型准确的达到了 90%，而使用 KL 散度的函数准确的只有 80%，在这个数据集上面，多分类交叉熵效果较好，KL 散度可能常用于逼近复杂函数，但是目前看来，它在图片分类上面的表现不是很好。

4. 正则化：

为降低过拟合的影响，我在卷积层中 l2 正则化，将 weight_decay 设置为 1e-4

```
def conv(x,out_filters,k_size):
    return Conv2D(filters=out_filters,
                  kernel_size=k_size,
                  strides=(1,1),
                  padding='same',
                  kernel_initializer='he_normal',
                  kernel_regularizer=regularizers.l2(weight_decay),
                  use_bias=False)(x)
```

5. 模型的调整：

由于数据集过大，模型也比较大，参数也比较多，在尝试使用一开始的每个 dense_block 中有 16 个 bottle_neck、batchsize=64 的模型发生了 OOM 错误，内存不够用了，为此，我对训练模型的大小和 batchsize 进行调整，将其调整为每个 dense_block 8 个 bottle_neck,batchsize=16，模型的长度缩小，一定程度上降低了模型对特征的理解能力。通过使用 mnist 数据集进行测试，batchsize 的大小一定程度上会影响模型训练的稳定性，由于 batch_size 调整的比较小，训练过程中经常会发生准确率的突然变化。

6. 针对内存不足的尝试：

由于数据集较大，训练数据占用内存比较多，因此会发生 OOM 此类错误，那么我们如何减小数据集大小的影响？

很显然，我们不用一次将整个数据集加载到内存中，通过将训练中为模型提供训练数据的 flow 函数替换为 flow_from_directory 函数，可以不同一次将全部数据读入内存，为满足函数的数据处理需要，要实现将不同类别图片存储在不同文件夹中。

```
model.fit_generator(datapowerful.flow(x_train,_train,batch_size=batch_size),
                    steps_per_epoch=iterations,
                    epochs=epochs,
                    callbacks=cbks,
                    validation_data=(x_valid,y_valid))
```

```
model.fit_generator(datapowerful.flow_from_directory(direction="./bird",
                     target_size=(224,224),class_mode='categorical',
                     batch_size=batch_size,featurewise_std_normalization=True,
                     rotation_range=0.1,width_shift_range=0.1,
                     height_shift_range=0.1),
                    steps_per_epoch=iterations,
                    epochs=epochs,
                    callbacks=cbks,
                    validation_data=(x_valid,y_valid))
```

7. 预测时出现的问题：

在一开始利用模型对数据进行预测时，得到的模型对图片的预测大多为类型中的两类，这当然是不符合实际结果并与模型准确率相悖的，经过思考，发现是模型对于训练数据和预测数据的处理并不一致，若要使预测与模型相符，在读入数据时，对测试数据也要进行相同的处理。