Dogs vs. Cats Redux: Kernels Edition

项目概述

本项目是为完成 Kaggle 上的图片分类任务 Dogs vs. Cats Redux: Kernels Edition 而作，这是一个计算机视觉（Computer Vision）领域的问题。

计算机视觉（简称 CV）是人工智能领域的重要核心技术之一，广泛的商业化渠道和技术基础使其成为最热门的领域之一。伴随着技术成熟度的提高，人脸识别、物体识别等分类、分割算法精度的不断提升，计算机视觉技术广泛应用于安防、金融、互联网、物流、零售、医疗、制造业、农业等领域。

深度学习技术中的卷积神经网络（CNN）在计算机视觉领域取得了很大的成功，在 ImageNe 数据集上，许多成功的模型都是基于 CNN 的。CNN 网络对图片进行多次卷积和池化操作，提取图片特征，通过全连接神经网络输出样本类别。

本项目是计算机视觉技术的一个简单应用，以学习、研究、探讨 CV 技术为目的而展开，本文将具体介绍数据处理、模型搭建、算法及调参等细节。由于训练过程计算量较大，建议使用 GPU 加速完成训练。

问题陈述

猫狗识别是一个传统的二分类问题，该任务提供了 25000 张带标签的猫狗图片作为训练集，以及 12500 张无标签图片作为测试集，需要用到机器学习的方法来解决这个问题。

由于处理的数据是图片，CNN 网络能更好胜任这个工作。当前主流的 CNN 模型包括 VGG、ResNet、DenseNet、Inception 等系列的各种版本以及变种。CNN 史上的一个里程碑事件是 ResNet 模型的出现，ResNet 可以训练出更深的 CNN 模型，从而实现更高的准确度。而 DenseNet 是比 ResNet 更优的 CNN 模型。

综合计算资源、训练时间和模型表现等因素，我选择了在 ImageNet 中 Top-1 准确率(accuracy)较高的 Densenet-161 为基准模型来构建神经网络。

任务要求：输出测试集图片是狗的概率。即到达用神经网络识别猫狗的目的。

解决方案：通过 Densenet-161 提取样本特征，将特征输入全连接神经网络，最终得到图片分类概率。

期望结果：本文以模型准确率 99%以上，kaggle 评分小于 0.06127（Public Leaderboard 前 10%）为目标，探索切实可行的方法。

评价指标

1、 **评价指标——Cross Entropy Loss**

kaggle 官方的评分函数 log loss 实际为 ***binary cross entropy loss***（二元交叉熵损失函数）：

Submissions are scored on the log loss:

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

- n is the number of images in the test set
- $\hat{y}_i$ is the predicted probability of the image being a dog
- y_i is 1 if the image is a dog, 0 if cat
- *log*() is the natural (base e) logarithm

该函数常用于二分类问题，是一种度量概率距离的方式，loss 值越小说明模型拟合的分布越接近真实分布，模型表现越好，反之模型表现越差。

2、 **评价指标——Accuracy**

从应用角度考虑，我们还应评估模型的准确率——Accuracy。一般来说 loss 越小准确率越高，但也会存在偏差，所以实际的准确率将作为对模型的重要评估指标之一。

计算方法：Accuracy = 预测正确的样本数 / 样本总数

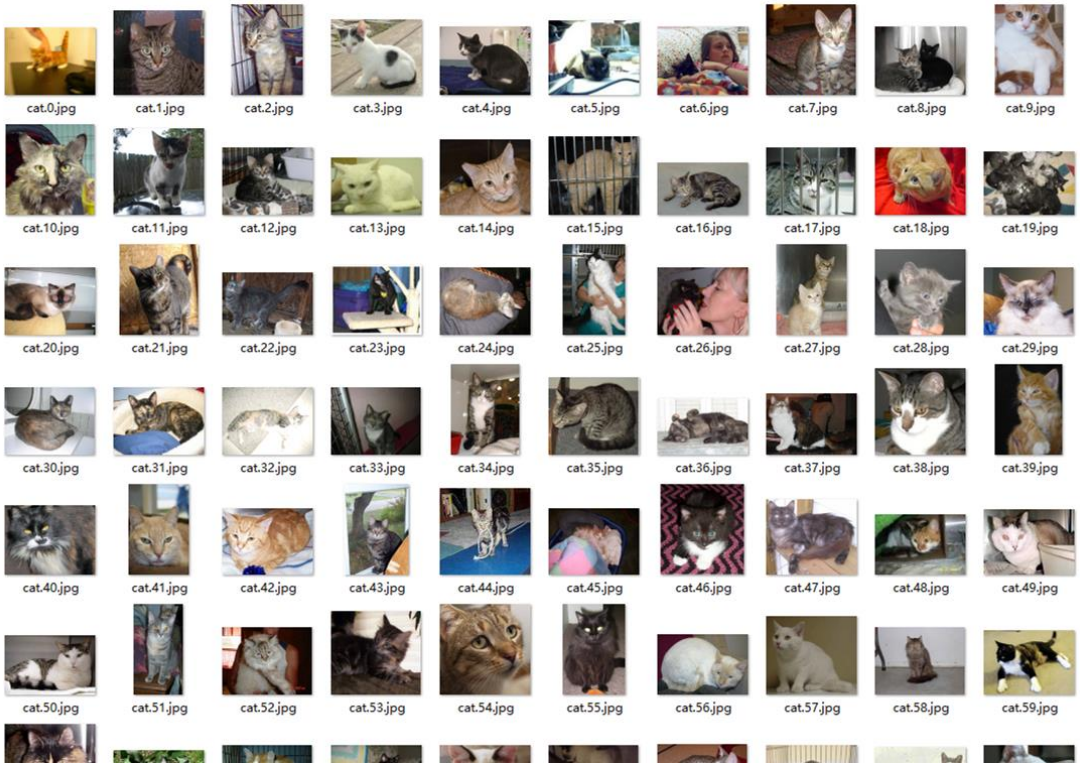
用预测结果与样本标签作比较，预测结果正确的样本占整体样本的比率即为准确率

Ⅰ．数据预处理

1、数据探索

1.1　人工观察训练集

基本上都是猫狗图片，图片的文件名即为标签。



1.2　获取训练集文件路径

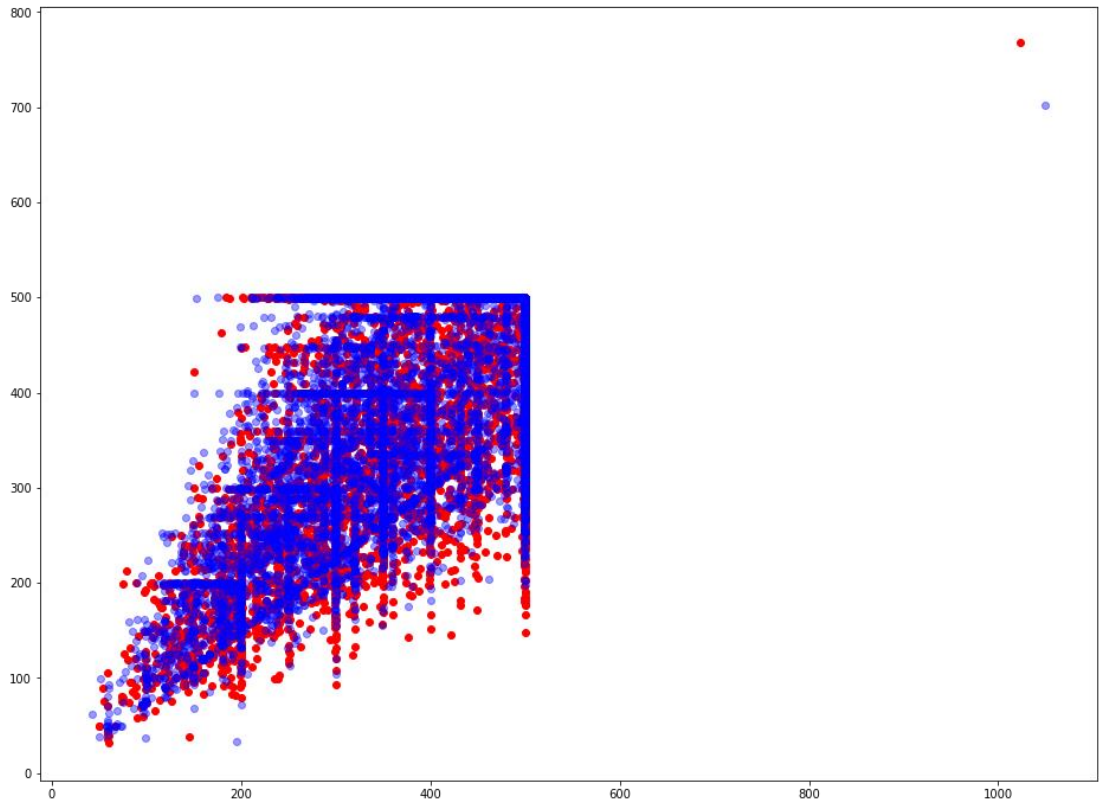
按猫狗标签分装到两个 list：dogs 和 cats。猫狗各各 12500 张。

```
data_dir='./dataset/' #数据目录
train_dir=os.path.join(data_dir,'train') #训练集文件夹
image_names=os.listdir(train_dir) #训练集文件名

#把 dog 和 cat 分装到两个列表 dogs 和 cats，并附加标签
dogs,cats=[],[]
for fn in image_names:
    dogs.append((os.path.join(train_dir,fn),1)) if 'dog' in fn else cats.append((os.path.join(train_dir,fn),
0))
```

1.3 可视化数据分布。

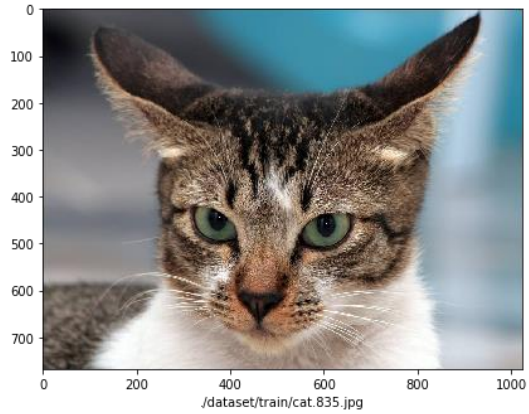
用散点图可视化图片尺寸，观察数据是否存在异常值。

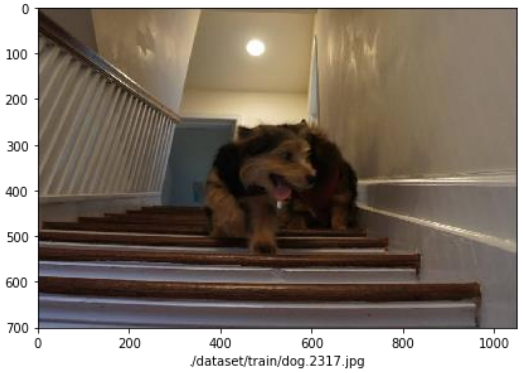


(红色为猫图片尺寸，蓝色为狗图片尺寸。)

在右上角，猫狗各有一个异常点

1.4 可视化尺寸异常的图片





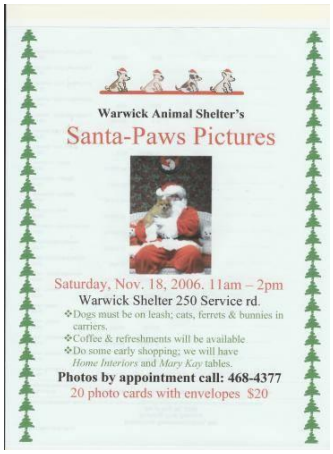
尺寸异常的图片其实都是正常的猫狗，Resize 后对模型训练没有影响。

1.5　人工再检查一遍训练集

发现其中存在一些构图异常或者标签与图片不符的样本，有些图片背景复杂、猫狗占比过小，或者同时存在猫和狗，或者根本就不是猫狗。这些异常图片混杂在训练集中会让模型难以拟合数据，或者对目标作出错误的预测。

非猫非狗图片

猫狗太小



既有猫又有狗

2、分割训练集和验证集

为保证数据平衡，正式训练集也将由数量相等的猫和狗构成。从猫狗 list 各取 80%作为训练集，剩下 20%作为验证集。

分割后：训练集包含 10000 张猫和 10000 张狗，验证集包含 2500 张猫和 2500 张狗。

```
sample_size=0.8
#分割训练集和验证集
train_dogs,valid_dogs=train_test_split(dogs,train_size=sample_size,random_state=42,shuffle=True)
train_cats,valid_cats=train_test_split(cats,train_size=sample_size,random_state=42,shuffle=True)
train_sets=train_dogs+train_cats
valid_sets=valid_dogs+valid_cats
```

3、图片变换

3.1 transforms

使用 torchvision.transforms.Compose( )来组合图片变换函数。

训练集使用随机裁剪、翻转图片，一方面增加训练数据，另一方面减轻过拟合。

测试集则直接使用中心裁剪。

```
transform={
'train':transforms.Compose([transforms.Resize(256),
                        transforms.RandomCrop(224),
                        transforms.RandomHorizontalFlip(),
                        transforms.ToTensor(),
                        transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))]),
'test':transforms.Compose([transforms.Resize(224),
                       transforms.CenterCrop(224),
                       transforms.ToTensor(),
                       transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))])}
```

3.2 可视化 transform 后的图片

图片在输入神经网络之前的样子

4、定义批次变换函数

自定义函数 batch_transform，读取一个 batch 的文件路径，图片经过 transform 变换，拼接各图张量，返回 (batch,channel,height,width)，作为神经网络的输入。

```
def batch_transform(paths,transform):
    return torch.cat(tuple(transform(Image.open(path)).unsqueeze(0) for path in paths))
```

input： batch paths and transform

return：(batch,channel,height,width)

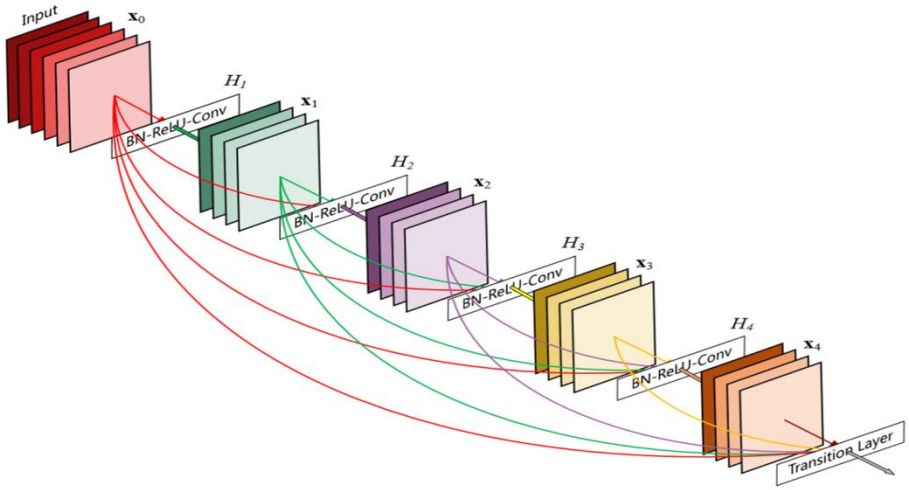
Ⅱ. 搭建网络

1、模型

1.1 基准模型的结构和组成

本项目使用 densenet-161 作为基准模型，属于 DenseNet 系列中的一个版本。

在介绍 DenseNet 之前不得不先讲一下 ResNet，ResNet 模型的核心是通过建立前面层与后面层之间的“短路连接”（shortcuts，skip connection），有助于训练过程中梯度的反向传播，从而能训练出更深的 CNN 网络。DenseNet 模型的基本思路与 ResNet 一致，但是它建立的是前面所有层与后面层的密集连接（dense connection），它的名称也是由此而来。DenseNet 的另一大特色是通过特征在 channel 上的连接来实现特征重用(feature reuse)。这些特点让 DenseNet 在参数和计算成本更少的情形下实现比 ResNet 更优的性能。



相比 ResNet，DenseNet 提出了一个更激进的密集连接机制：即互相连接所有的层，具体来说就是每个层都会接受其前面所有层作为其额外的输入。图 1 为 ResNet 网络的连接机制，作为对比，图 2 为 DenseNet 的密集连接机制。可以看到，ResNet 是每个层与前面的某层（一般是 2~3 层）短路连接在一起，连接方式是通过元素级相加。而在 DenseNet 中，每个层都会与前面所有层在 channel 维度上连接（concat）在一起，并作为下一层的输入。对于一个 L 层的网络，DenseNet 共包含 L(L+1)/2 个连接，相比 ResNet，这是一种密集连接。而且 DenseNet 是直接 concat 来自不同层的特征图，这可以实现特征重用，提升效率。

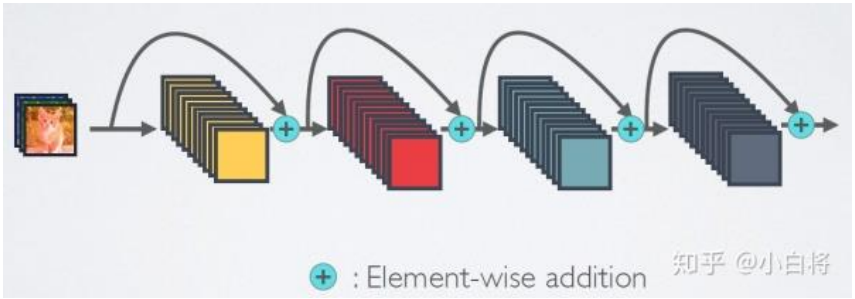


图 1 ResNet 网络的短路连接机制（其中+代表的是元素级相加操作）

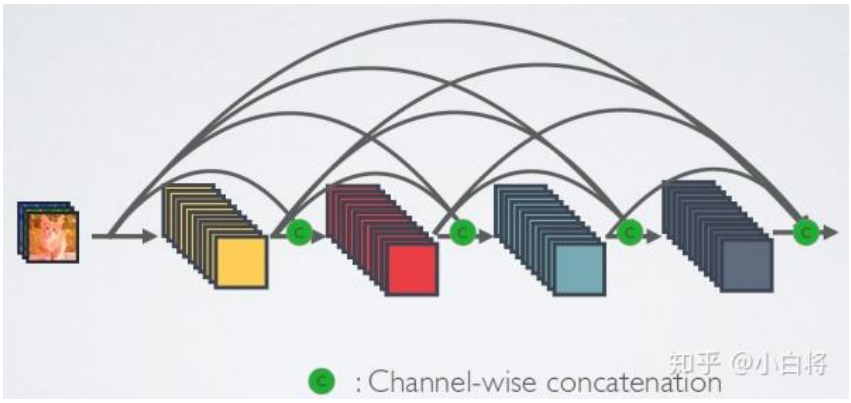
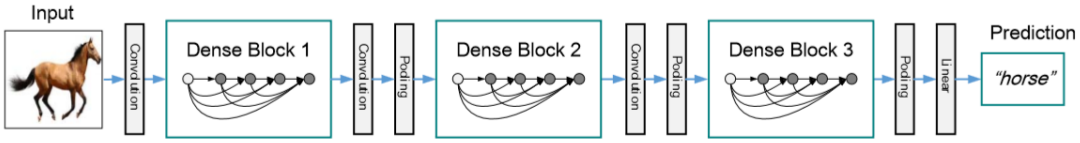


图 2 DenseNet 网络的密集连接机制（其中 c 代表的是 channel 级连接操作）

DenseNet 在 L 层的输出，会连接前面所有层作为输入：

$$\mathbf{x}_\ell = H_\ell([\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{\ell-1}]),$$

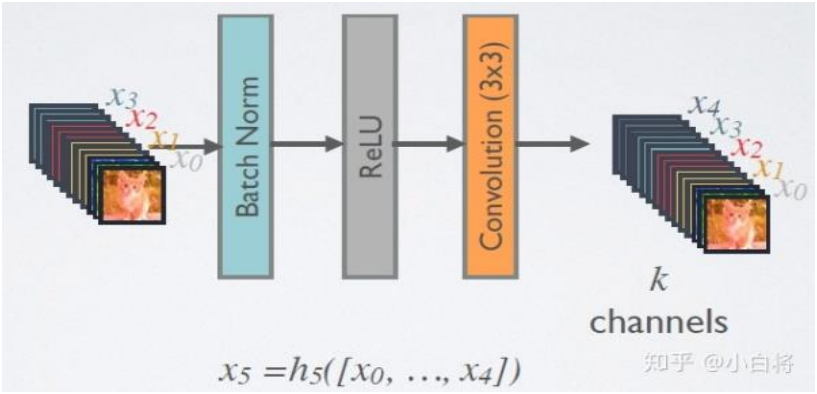
H 代表非线性转化函数（non-liear transformation），其可能包括一系列的 BN(Batch Normalization)，ReLU，Pooling 及 Conv 操作。



DenseNet 网络结构

CNN 网络一般要经过 Pooling 或者 stride > 1 的 Conv 来降低特征图的大小，而 DenseNet 的密集连接方式需要特征图大小保持一致。为了解决这个问题，DenseNet 网络中使用 DenseBlock+Transition 的结构，其中 DenseBlock 是包含很多层的模块，每个层的特征图大小相同，层与层之间采用密集连接方式。而 Transition 模块是连接两个相邻的 DenseBlock，并且通过 Pooling 使特征图大小降低。

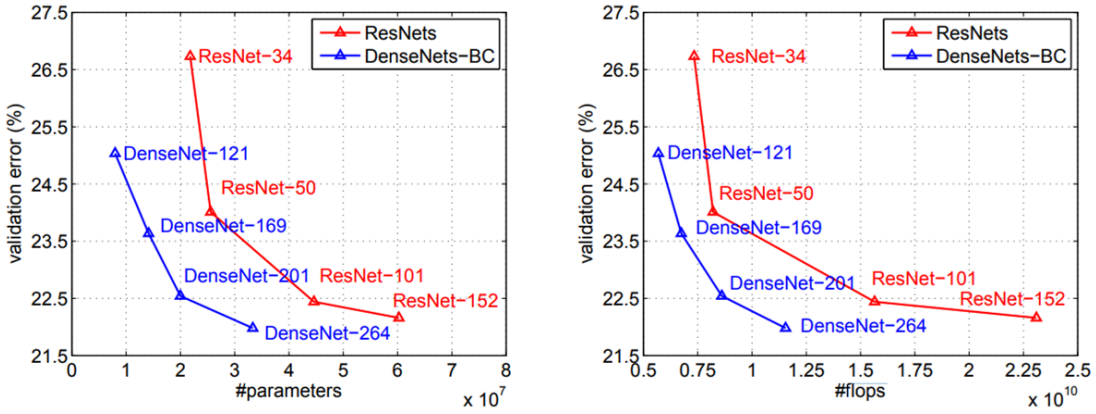
与 ResNet 不同，所有 DenseBlock 中各个层卷积之后均输出 k 个特征图，即得到的特征图的 channel 数为 k ，或者说采用 k 个卷积核。 k 在 DenseNet 称为 growth rate，这是一个超参数。一般情况下使用较小的 k （比如 12），就可以得到较佳的性能。假定输入层的特征图的 channel 数为 k0 ，那么 l 层输入的 channel 数为 k0+k(l-1) 。本案例所使用的的 densenet-161 是 k = 48 的结构。



Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	[1 × 1 conv, 3 × 3 conv] × 6	[1 × 1 conv, 3 × 3 conv] × 6	[1 × 1 conv, 3 × 3 conv] × 6	[1 × 1 conv, 3 × 3 conv] × 6
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	[1 × 1 conv, 3 × 3 conv] × 12	[1 × 1 conv, 3 × 3 conv] × 12	[1 × 1 conv, 3 × 3 conv] × 12	[1 × 1 conv, 3 × 3 conv] × 12
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	[1 × 1 conv, 3 × 3 conv] × 24	[1 × 1 conv, 3 × 3 conv] × 32	[1 × 1 conv, 3 × 3 conv] × 48	[1 × 1 conv, 3 × 3 conv] × 64
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	[1 × 1 conv, 3 × 3 conv] × 16	[1 × 1 conv, 3 × 3 conv] × 32	[1 × 1 conv, 3 × 3 conv] × 32	[1 × 1 conv, 3 × 3 conv] × 48
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

DenseNet architectures for ImageNet

DenseNet 具有缓解梯度消失问题，加强特征传播，鼓励特征重用，并大大减少参数的数量等优点。在 top-1 error 相当的情况下 DenseNet 的参数量和计算量几乎都是 Resnet 的一半。



Comparison of the DenseNets and ResNets top-1 error rates

1.2 模型修改

从 torchvision 的 models 库中下载预训练的 densenet-161：

```
model=models.densenet161(pretrained=True)
```

冻结除 classifier 以外所有层的参数，修改 classifier 为最简单的全连接结构 nn.Linear(in_features,2)，输出是维度为 2 的向量，分别对应两个类别。模型是用 ImageNet 的图片训练的，而 ImageNet 的 1000 个分类中本来就有猫和狗，这里可以直接使用预训练权重提取图片特征来训练 classifier。

```
for param in model.parameters():
    param.requires_grad=False
in_features=model.classifier.in_features
model.classifier=nn.Linear(in_features,2)
```

2、损失函数

原题使用 ***log loss*** 作为评分函数，本项目用 CrossEntropyLoss( )作为损失函数，loss 计算方式将与原题保持一致。

$$\text{loss}(x, class) = -\log\left(\exp(x[class]) / \sum_j \exp(x[j])\right)$$

x 为预测结果， class 为标签对应的类别

该函数对模型输出计算 softmax 后，用标签对应类别的概率计算负对数似然函数(negative log-likelihood Loss, NLLLoss)，与 LogLoss 是一样的计算效果

3、优化器

后面我会训练两个模型，将会用到两种优化器：

3.1 Adam 优化算法：

算法解释：

Adam：adaptive moment estimation（自适应矩估计），Adam 是一种对随机目标函数执行一阶梯度优化的算法，该算法基于适应性低阶矩估计，通过计算梯度的一阶矩估计和二阶矩估计而为不同的参数设计独立的自适应性学习率，是需要资源更少、令模型收敛更快的最优化算法。

优化器使用：

第一个模型使用 Adam 自动调整学习率，只训练 classifier， 模型的准确率可能会稍差一点，但是可以帮我们筛出一些异常图片。

```
optimizer=torch.optim.Adam(model.classifier.parameters( ))
```

3.2 随机梯度下降(SGD)优化算法：

算法解释：

SGD：Stochastic Gradient Descent，随机梯度下降是每次迭代使用一个样本来对参数进行更新。

优点：

由于不是在全部训练数据上的 loss，而是在每轮迭代中，随机优化某一条训练数据上的 loss，这样每一轮参数的更新速度大大加快。

缺点：

① SGD 无法做到线性收敛，单样本更新参数使得模型准确率下降。

② 可能会收敛到局部最优解，因为单个样本并不能代表全体样本的梯度。

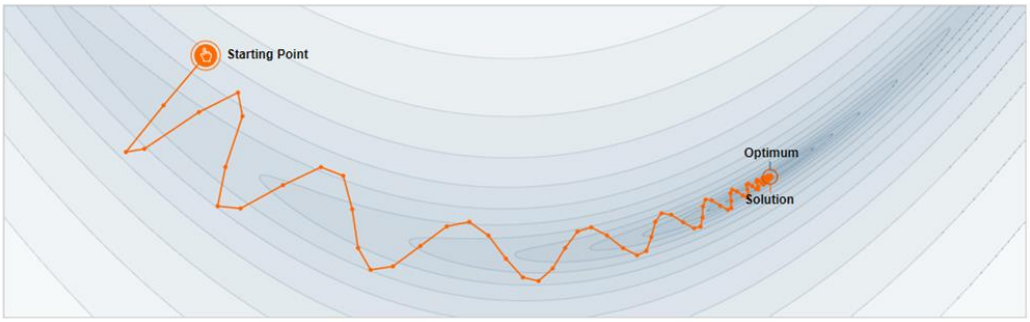
解决方法：

引入 Momentum（动量）：

$$v_{t+1}=\mu*v_t+g_{t+1}$$

$$p_{t+1}=p_t-lr*v_{t+1}$$

p：parameters，　g：gradient，　v：velocity，　μ：momentum



Momentum 梯度下降法，计算了梯度的指数加权平均数，当前权值的改变会受到上一次权值改变的影响。引入 Momentum 一方面可以使 SGD 加速，另一方面可以逃过局部最优解。

```
optimizer=torch.optim.SGD([{'params':model.classifier.parameters()},
                           {'params':model.features.norm5.parameters(),'lr':init_lr*0.1},
                           {'params':model.features.denseblock4.parameters(),'lr':init_lr*0.1}],
                           lr=init_lr,momentum=0.91)
```

调整学习率：

由于 SGD 的学习率 lr 是固定不变的，在训练后期不利于模型找到最优解，我们需要用学习率调度函数 lr_scheduler.ReduceLROnPlateau 来调整 lr。ReduceLROnPlateau 将读取一个参考指标，如果在 patience 期间没有看到该指标改善，则降低学习率。

```
    scheduler=torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer,'min',patience=2)
```

我选择根据每个 epoch 的 validation　loss 判断是否降低学习率，若 2 个 patience 后 validation　loss 还没有降低，则衰减学习率到当前的 10%（默认衰减率 *factor=0.1*）

```
    scheduler.step(epoch_dict['loss']['valid'][-1])
```

Ⅲ. 训练模型

1、直接用带杂质的训练集训练第一个模型

1.1　第一个模型使用 Adam 优化器

```
      optimizer=torch.optim.Adam(model.classifier.parameters())
```

1.2　创建训练集 DataLoader

```
train_batch=50
train_loader=torch.utils.data.DataLoader(train_sets,batch_size=train_batch,shuffle=True,num_workers=4)
```

参数 batch_size:

一般来说 batch_size 越大，其确定的下降方向越准，引起训练震荡越小，同时内存利用率越高，大矩阵乘法的并行效率越高。大的 batch_size 所需的迭代次数减少，但是对参数的修正也显得更加缓慢，batch_size 增大到一定程度后，梯度方向基本不再变化，过大的 batch_size 还可能会超出内存容量。

太小的 batch_size 类似于单样本更新，收敛相对困难，可能会得到局部最优解。

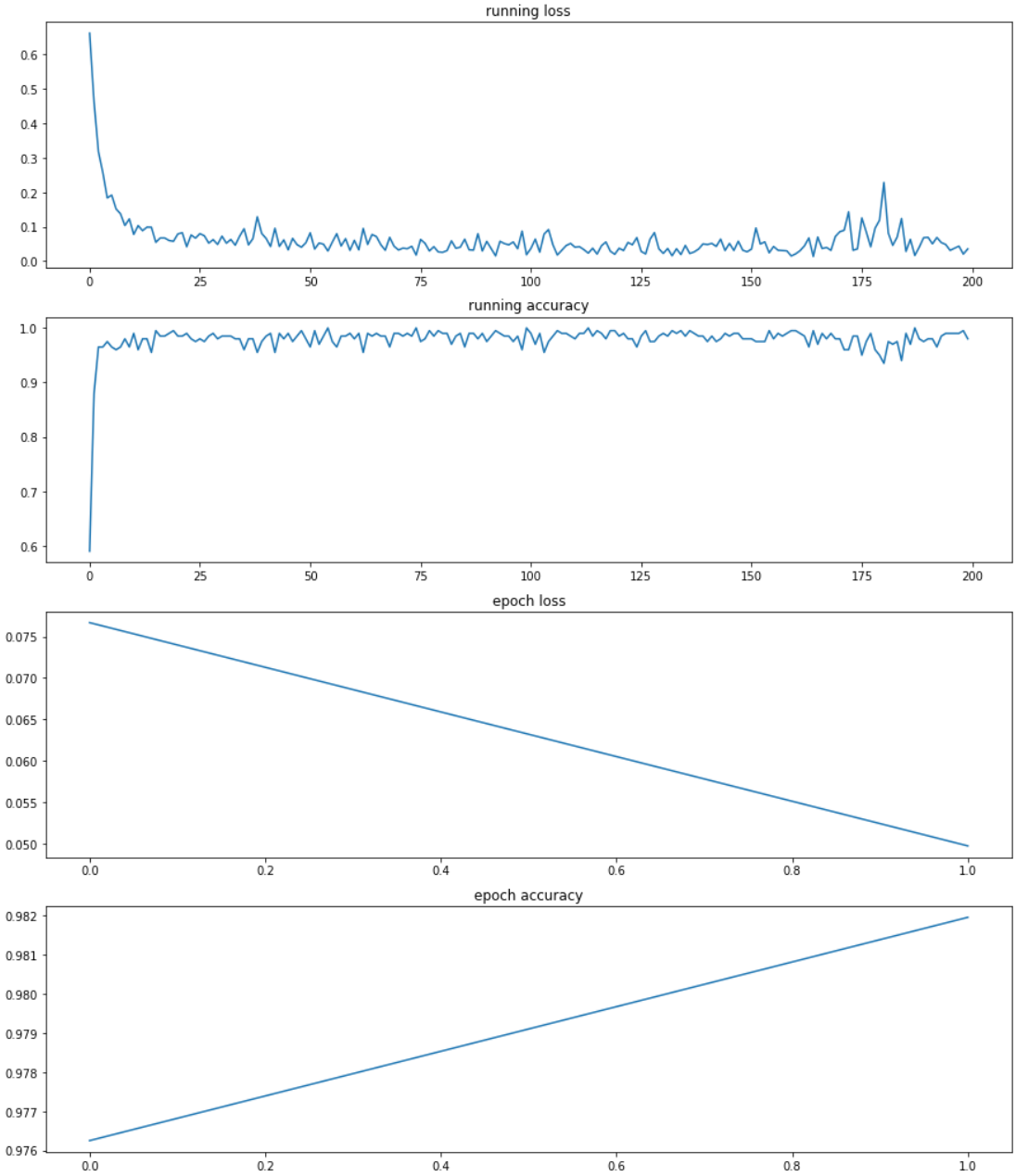
所以我们在选择 batch_size 时需要权衡。

参数 Shuffle:

原始训练集前半部分是猫，后半部分是狗，为了防止模型在学习的过程中“偏科“，使用 shuffle =True 打乱原本的样本顺序，均衡数据分布。

1.3　第一个模型训练过程中的评价指标

下图是训练集 loss 和 accuracy。使用了 Adam 算法训练，loss 收敛很快，当然，模型准确率也不是最佳，但是帮我们筛出异常图片已经足够了。

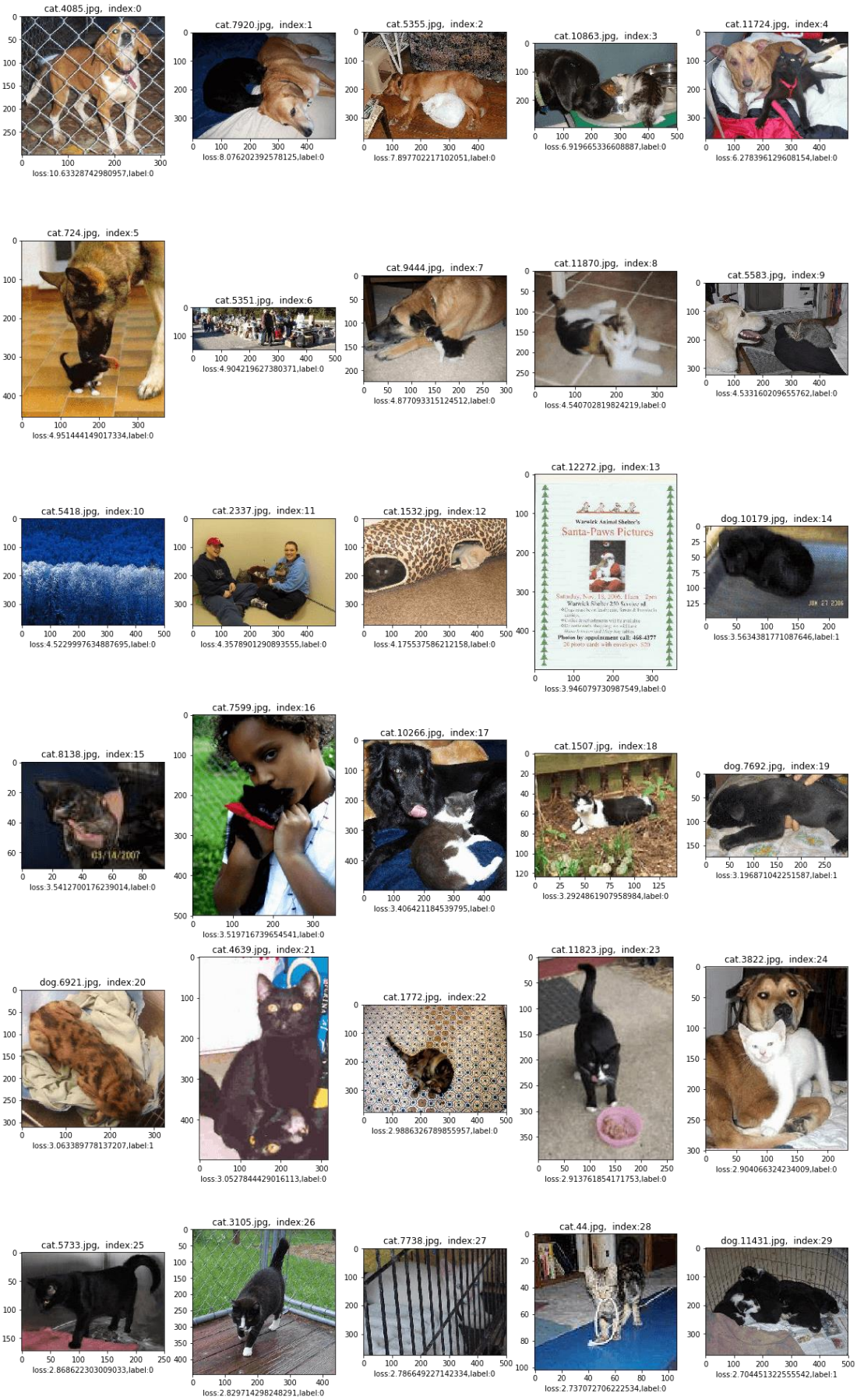


1.4　用训练过的模型预测一次训练集样本，筛出 loss 大于阈值的样本。

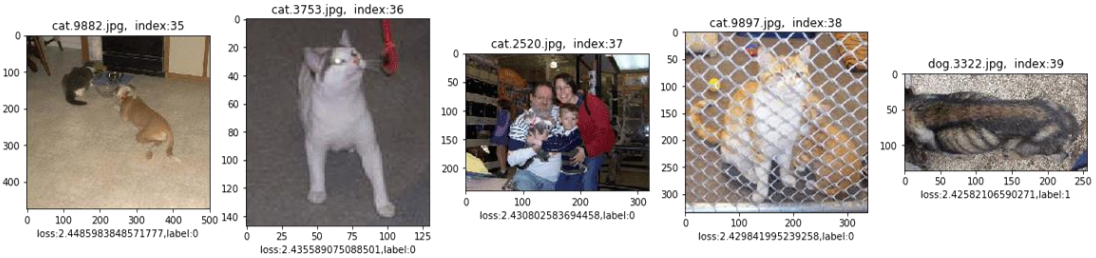
因为训练集绝大部分都是正常图片，异常值只占很小一部分，经过两轮 epoch 训练，模型准确率已达 98%，学到的特征偏向于占绝对优势的正常图片，而此时 loss 仍然大于 1.2040 的（预测正确的概率不超过 30%，由负对数似然函数逆推），属于比较难学的样本，其中不乏一些异常图片，标签错误、非猫非狗、像素模糊、颜色深暗、背景复杂、角度刁钻，或者一张图同时存在猫和狗。

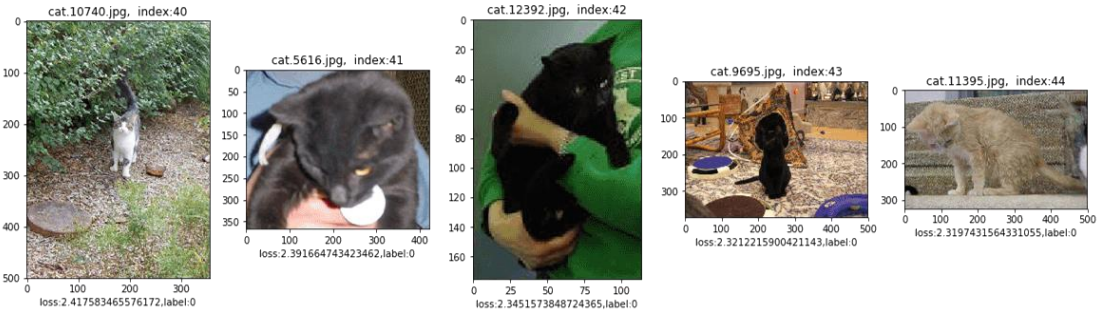
这种方法不能作为筛选异常值的标准方法，但是可以辅助人工初步筛选一次异常值。

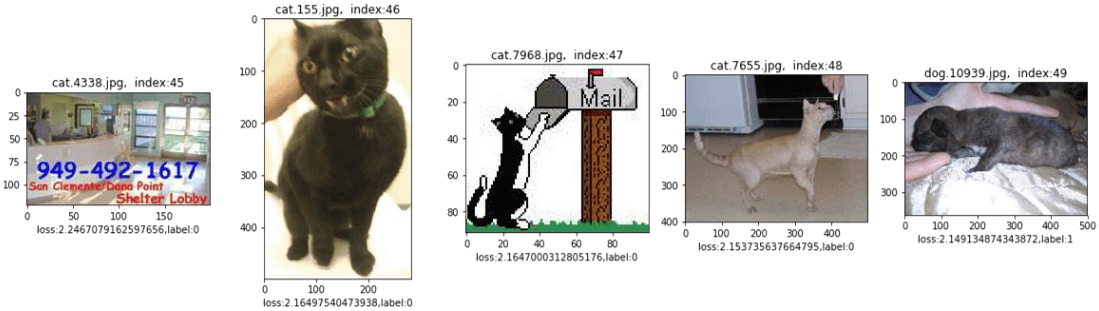
按 loss 大于阈值筛出来的图片

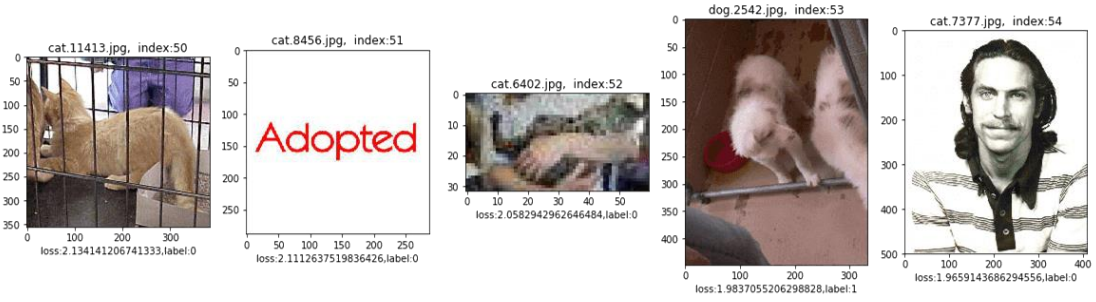


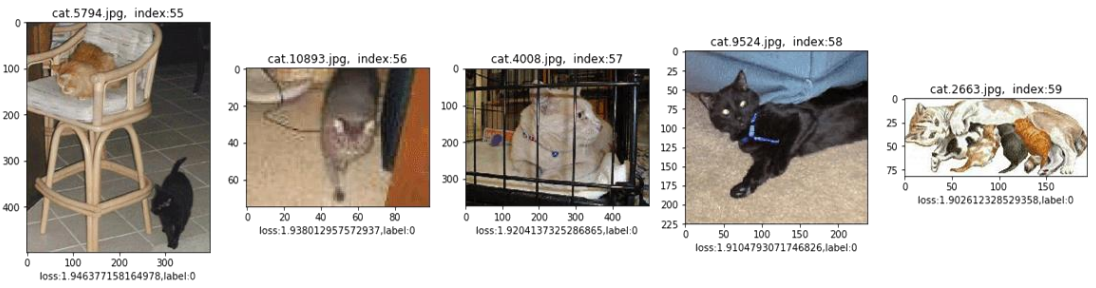

cat.1011.jpg, index:30
loss:2.700676202774048,label:0
cat.2457.jpg, index:31
loss:2.6302695274353027,label:0
dog.10155.jpg, index:32
loss:2.629530906677246,label:1
cat.7734.jpg, index:33
loss:2.5629324913024902,label:0
cat.6655.jpg, index:34
loss:2.495168685913086,label:0

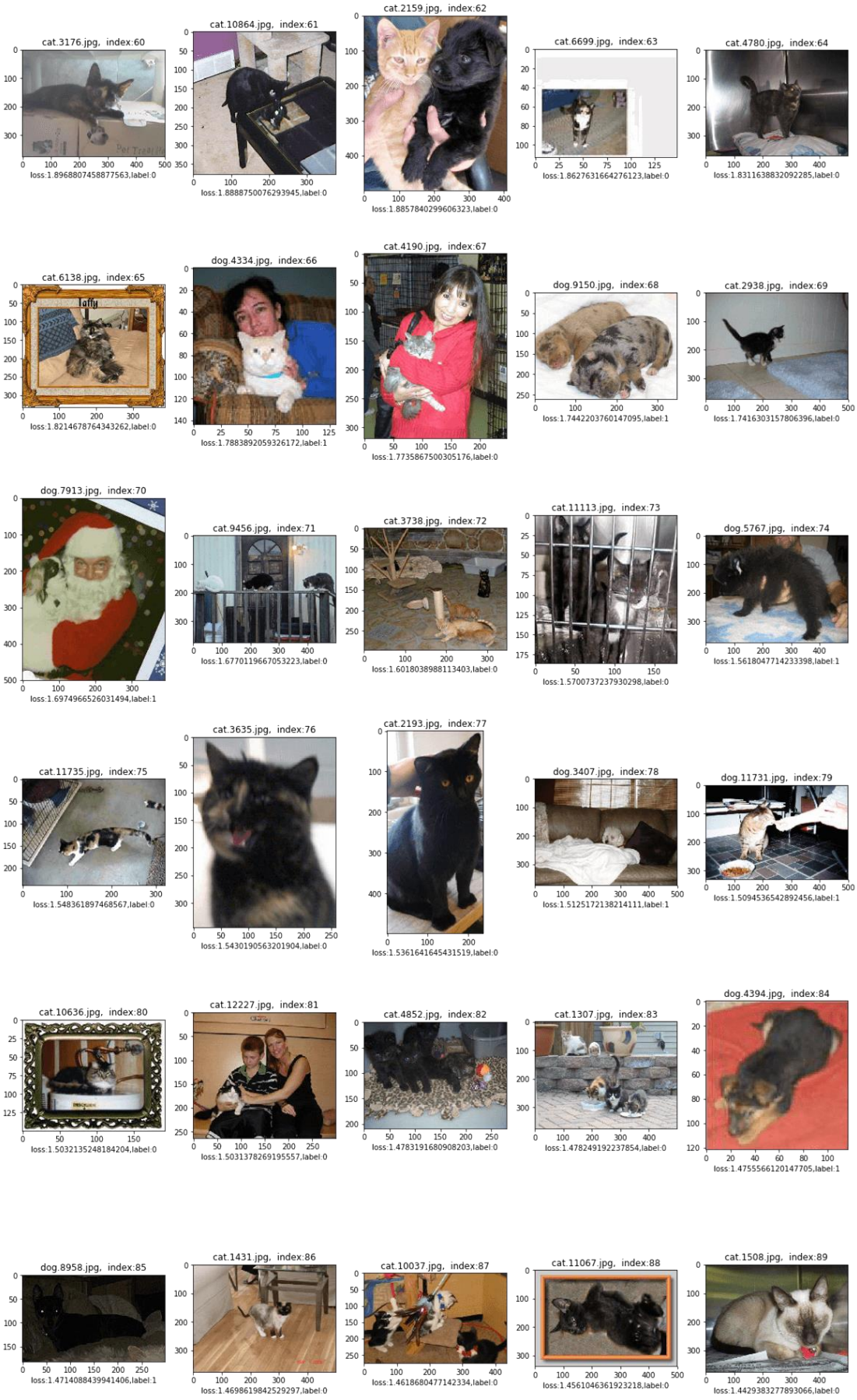

cat.9882.jpg, index:35
loss:2.4485983848571777,label:0
cat.3753.jpg, index:36
loss:2.435589075088501,label:0
cat.2520.jpg, index:37
loss:2.430802583694458,label:0
cat.9897.jpg, index:38
loss:2.429841995239258,label:0
dog.3322.jpg, index:39
loss:2.42582106590271,label:1

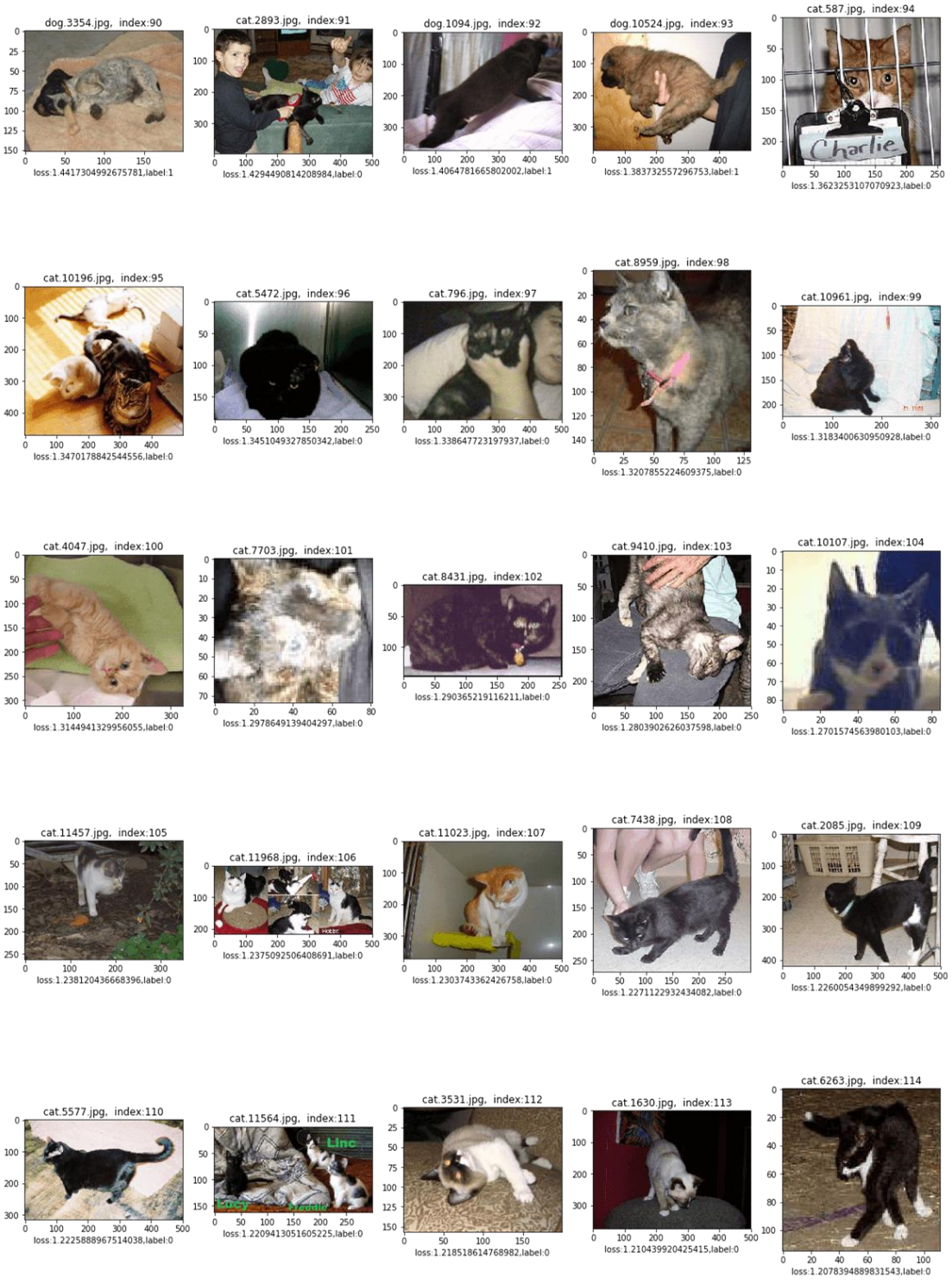

cat.10740.jpg, index:40
loss:2.417583465576172,label:0
cat.5616.jpg, index:41
loss:2.391664743423462,label:0
cat.12392.jpg, index:42
loss:2.3451573848724365,label:0
cat.9695.jpg, index:43
loss:2.3212215900421143,label:0
cat.11395.jpg, index:44
loss:2.3197431564331055,label:0


cat.4338.jpg, index:45
949-492-1617
San Clemente/Dana Point
Shelter Lobby
loss:2.2467079162597656,label:0
cat.155.jpg, index:46
loss:2.16497540473938,label:0
cat.7968.jpg, index:47
Mail
loss:2.1647000312805176,label:0
cat.7655.jpg, index:48
loss:2.153735637664795,label:0
dog.10939.jpg, index:49
loss:2.149134874343872,label:1


cat.11413.jpg, index:50
loss:2.134141206741333,label:0
cat.8456.jpg, index:51
Adopted
loss:2.1112637519836426,label:0
cat.6402.jpg, index:52
loss:2.0582942962646484,label:0
dog.2542.jpg, index:53
loss:1.9837055206298828,label:1
cat.7377.jpg, index:54
loss:1.9659143686294556,label:0


cat.5794.jpg, index:55
loss:1.946377158164978,label:0
cat.10893.jpg, index:56
loss:1.938012957572937,label:0
cat.4008.jpg, index:57
loss:1.9204137325286865,label:0
cat.9524.jpg, index:58
loss:1.9104793071746826,label:0
cat.2663.jpg, index:59
loss:1.902612328529358,label:0


cat.3176.jpg, index:60
loss:1.8968807458877563,label:0
cat.10864.jpg, index:61
loss:1.8888750076293945,label:0
cat.2159.jpg, index:62
loss:1.8857840299606323,label:0
cat.6699.jpg, index:63
loss:1.8627631664276123,label:0
cat.4780.jpg, index:64
loss:1.8311638832092285,label:0
cat.6138.jpg, index:65
loss:1.8214678764343262,label:0
dog.4334.jpg, index:66
loss:1.7883892059326172,label:1
cat.4190.jpg, index:67
loss:1.7735867500305176,label:0
dog.9150.jpg, index:68
loss:1.7442203760147095,label:1
cat.2938.jpg, index:69
loss:1.7416303157806396,label:0
dog.7913.jpg, index:70
loss:1.6974966526031494,label:1
cat.9456.jpg, index:71
loss:1.6770119667053223,label:0
cat.3738.jpg, index:72
loss:1.6018038988113403,label:0
cat.11113.jpg, index:73
loss:1.5700737237930298,label:0
dog.5767.jpg, index:74
loss:1.5618047714233398,label:1
cat.11735.jpg, index:75
loss:1.548361897468567,label:0
cat.3635.jpg, index:76
loss:1.5430190563201904,label:0
cat.2193.jpg, index:77
loss:1.5361641645431519,label:0
dog.3407.jpg, index:78
loss:1.5125172138214111,label:1
dog.11731.jpg, index:79
loss:1.5094536542892456,label:1
cat.10636.jpg, index:80
loss:1.5032135248184204,label:0
cat.12227.jpg, index:81
loss:1.5031378269195557,label:0
cat.4852.jpg, index:82
loss:1.4783191680908203,label:0
cat.1307.jpg, index:83
loss:1.4782491922378854,label:0
dog.4394.jpg, index:84
loss:1.4755566120147705,label:1
dog.8958.jpg, index:85
loss:1.4714088439941406,label:1
cat.1431.jpg, index:86
loss:1.4698619842529297,label:0
cat.10037.jpg, index:87
loss:1.4618680477142334,label:0
cat.11067.jpg, index:88
loss:1.4561046361923218,label:0
cat.1508.jpg, index:89
loss:1.4429383277893066,label:0



1.5 人工对筛选出来的图片再检查一遍

①修改标签错误的样本。比如第一张，实际为 dog 被标记为 cat，应该将其移回正常训练集。

②黑猫黑狗很多，这些图片相对难以辨认，但如果图片是清晰且正常的，应该移回正常训练集，因为我们无法避免识别黑猫黑狗。

③如果连人工都难以辨认的应该予以剔除。

经过调整，训练集最终剩下 19906 个样本。

2、使用清洗过的训练集训练第二个模型

2.1　再次修改 model.classifier，使用更复杂的全连接网络，因为我希望网络学习到更多样化的特征。

```
model.classifier=nn.Sequential(nn.Dropout(0.2),
                               nn.Linear(in_features,int(in_features/2)),
                               nn.ReLU(),
                               nn.Dropout(0.2),
                               nn.Linear(int(in_features/2),int(in_features/4)),
                               nn.ReLU(),
                               nn.Dropout(0.2),
                               nn.Linear(int(in_features/4),int(in_features/8)),
                               nn.ReLU(),
                               nn.Dropout(0.2),
                               nn.Linear(int(in_features/8),2))
model=model.to(device)
```

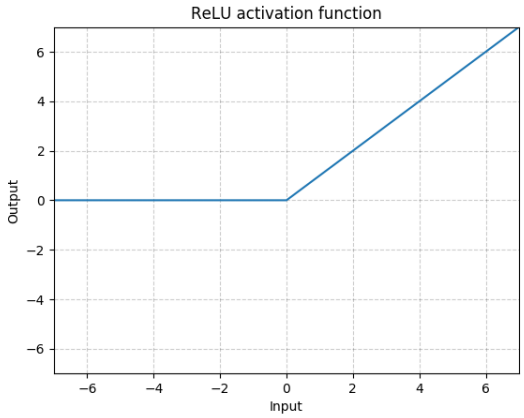
注意：由于修改了 model，需要将 model 重新移动到 GPU。

函数解释：

nn.Dropout(p=0.2)：在训练期间，以概率 p 随机置零输入张量的一些元素，这是一种可以防止神经元过拟合的正则化技术。

nn.ReLU()：Rectified Linear Unit，激活函数。在每个线性层后面添加 Relu 激活函数，可以更有效率地梯度下降以及反向传播，避免了梯度爆炸和梯度消失问题，同时使神经网络整体计算成本下降。

$$\mathrm{ReLU}(x)=\max(0,x)$$



2.2　打开 model 后面几层的参数训练。

对于深度学习模型来说，前面的特征层泛化能力较强，越往后面的层，越专注于原始数据(这里指 ImageNet 数据)。所以我们打开模型后面几层参数训练，来学习我们现在的数据。

```
for i,param in enumerate(model.parameters()):
    if i>335:
        param.requires_grad=True
```

关于判断条件“i > 335”的解释：

遍历模型参数可以知道修改后模型一共有 490 个参数，其中 classifier 有 8 个参数，features.norm5 有 2 个参数，features.denseblock4 有 144 个参数，减去后面这几层的参数量，可以推断在 denseblock4 之前共有 336 个参数。

2.3 优化器

为了得到最优解，正式训练使用随机梯度下降算法。

其中 classifier 起步学习率 0.01，features.norm5 和 features.denseblock4 起步学习率 0.001。

另外使用 ReduceLROnPlateau 根据每个 epoch 的平均 validation loss 控制学习率衰减。

```
init_lr=0.01  #起步 learning rate


optimizer=torch.optim.SGD([{'params':model.classifier.parameters()},
                           {'params':model.features.norm5.parameters(),'lr':init_lr*0.1},
                           {'params':model.features.denseblock4.parameters(),'lr':init_lr*0.1}],
                           lr=init_lr,momentum=0.91)


#根据每个 epoch 的 validation loss 判断是否衰减 learning rate
scheduler=torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer,'min',patience=2)
```

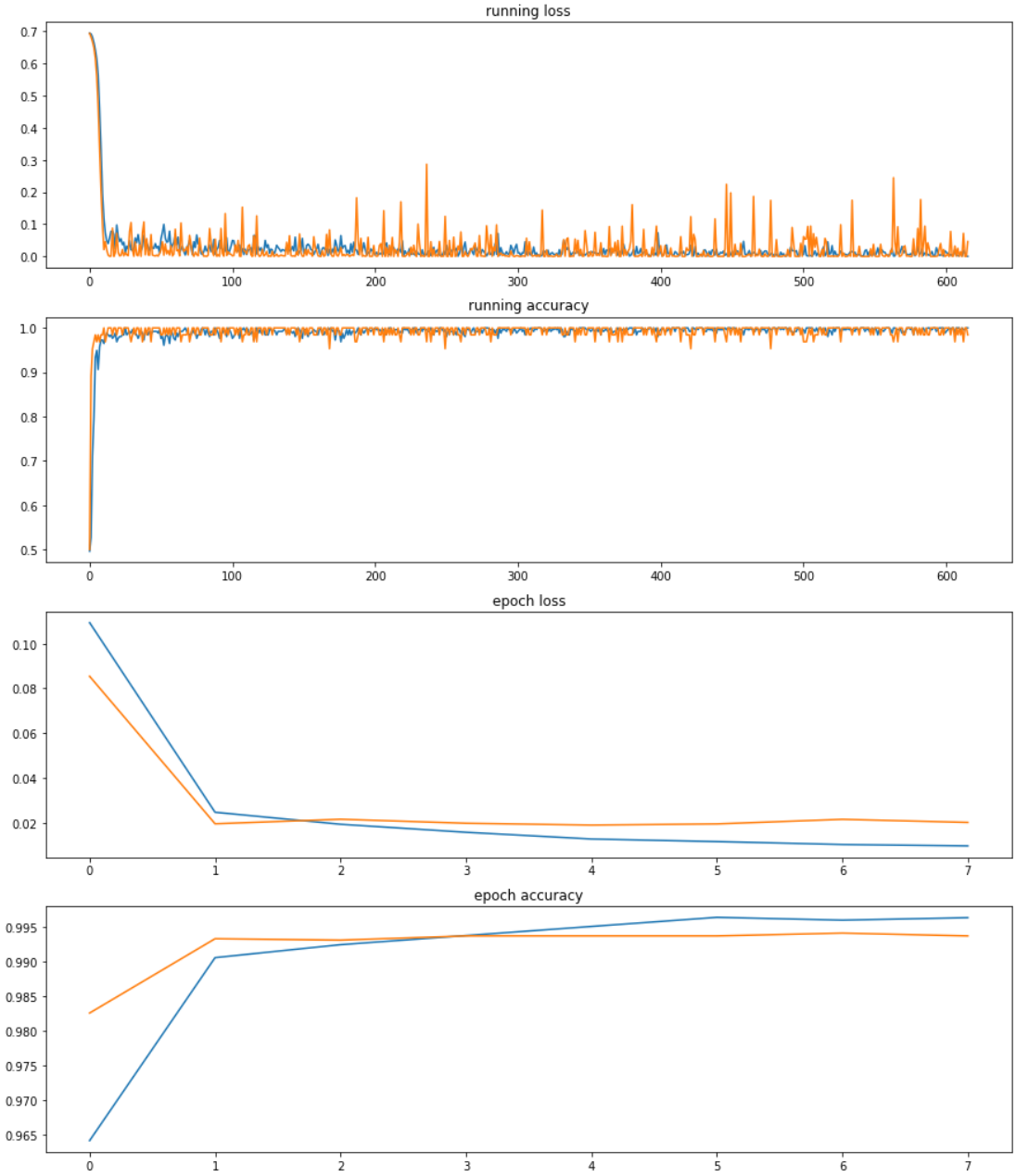
2.4 DataLoader

```
train_batch,valid_batch=64,64
train_loader=torch.utils.data.DataLoader(normal_sets,batch_size=train_batch,shuffle=True,num_workers=4,drop_last=True)
valid_loader=torch.utils.data.DataLoader(valid_sets,batch_size=valid_batch,shuffle=True,num_workers=4,drop_last=True)
```

第二个模型训练，训练集、验证集 batch_size 均选择 64，仍然乱序加载图片 shuffle=True

2.5 第二个模型训练过程中的评价指标

Blue color：training， Orange color：validation



在第 8 个 epoch 完成时，训练集 loss：0.009714， accuracy：99.63%；验证集 loss：0.020199， accuracy：99.37%

上图可以看出使用 SGD 训练模型，loss 收敛也很快，可能是由于我使用了较大的学习率(0.01)起步，同时 validation loss 波动也很大，第一个 epoch 还没训练完准确率就达到了 95%以上，这得益于预训练权重和 DesNet 的优秀性能。

但是想要得到更小的 loss 和更高的 accuracy，需要降低学习率继续逼近最优解。

由于有 ReduceLROnPlateau 帮忙控制学习率，我一口气训练了 8 个 epoch，每个 epoch 都保存了模型权重，用于备选或回溯。从指标走势可以看出，到第 8 个 epoch 时模型其实已经过拟合了。

Ⅳ. 预测

1、准备测试集数据

1.1 读取测试集文件名

保存成（File_Path, ID）的格式，测试集共 12500 个样本。

```
test_sets=[(os.path.join(test_dir,fn),int(fn[:-4])) for fn in test_images]
```

1.2 DataLoader

```
test_loader=torch.utils.data.DataLoader(test_sets,batch_size=100,num_workers=4)
```

由于只进行 forward 不需要计算梯度，batch_size 的大小也不会对结果产生影响，所以根据内存容量选择尽量大的 batch_size 来提高并行效率。

2、预测结果

2.1 用训练好的模型对测试集进行预测

用 softmax 函数计算预测结果，取第二列（索引[:,1]）数据得到图片是狗的概率。

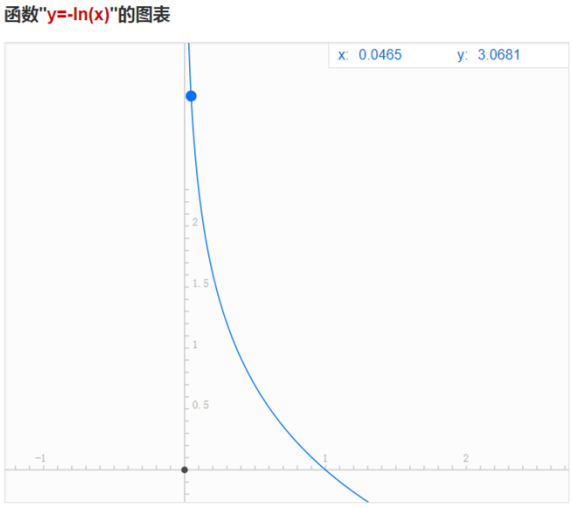
```
dog_probs=torch.nn.functional.softmax(outputs,dim=1).data[:,1]
```

我保存了 8 个模型权重，由于不确定哪一个是最优解，我使用每个权重分别对测试集进行了预测。

2.2 将结果 clip 到[0.005,0.995]

```
df.label.clip(0.005,0.995,inplace=True)
```

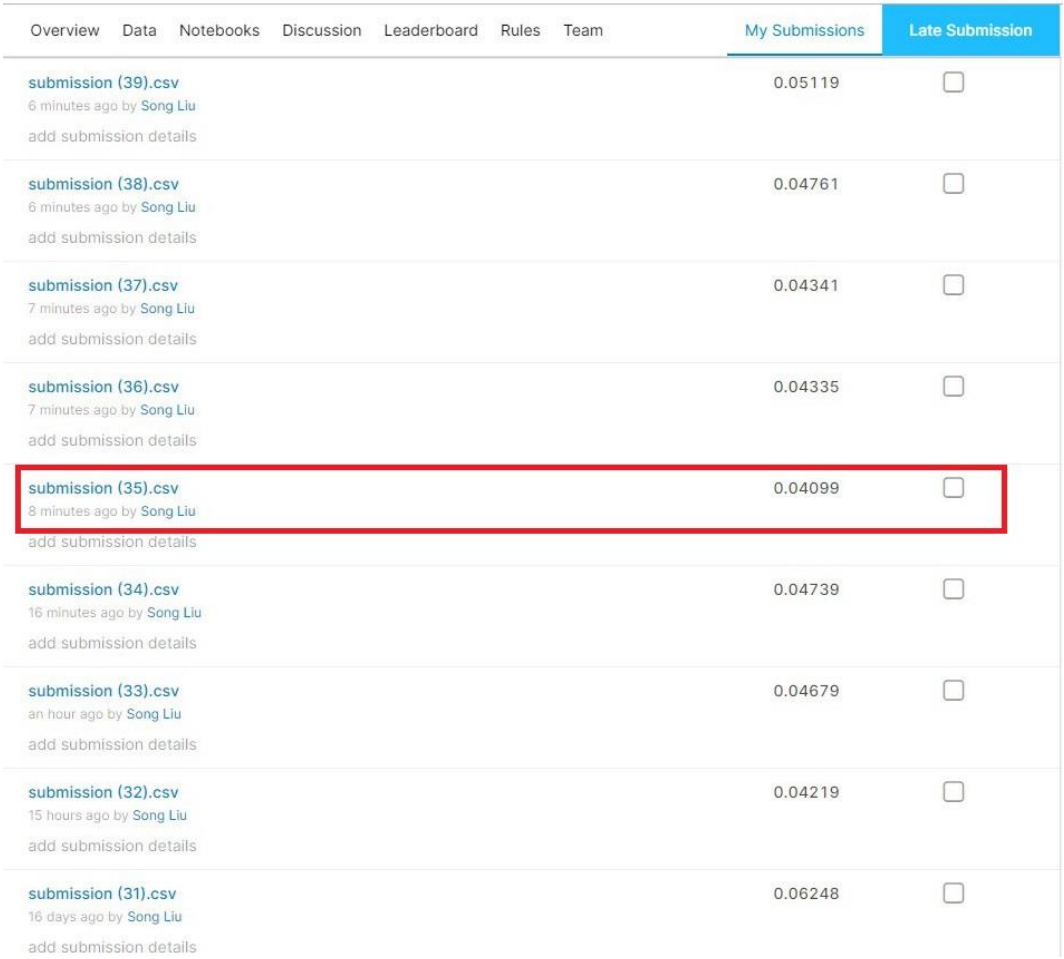
Log Loss 实际上是负对数函数，我们可以从 y = -ln(x)图形观察，这是一个单调递减函数，在(0, 0.05)区间的变化率远大于在(0.995,1]区间的变化率，将结果 clip 到[0.005,0.995]后，对于预测正确的样本 loss 不会升高很多，但是对于预测错误的样本 loss 却可以降低很多，压缩变量 x 的区间，可以带来 y 区间平均值降低的效果。（Log Loss 计算的是所有样本的平均 loss）



2.3　将结果按要求格式保存为 cvs，并提交至 kaggle 评分。

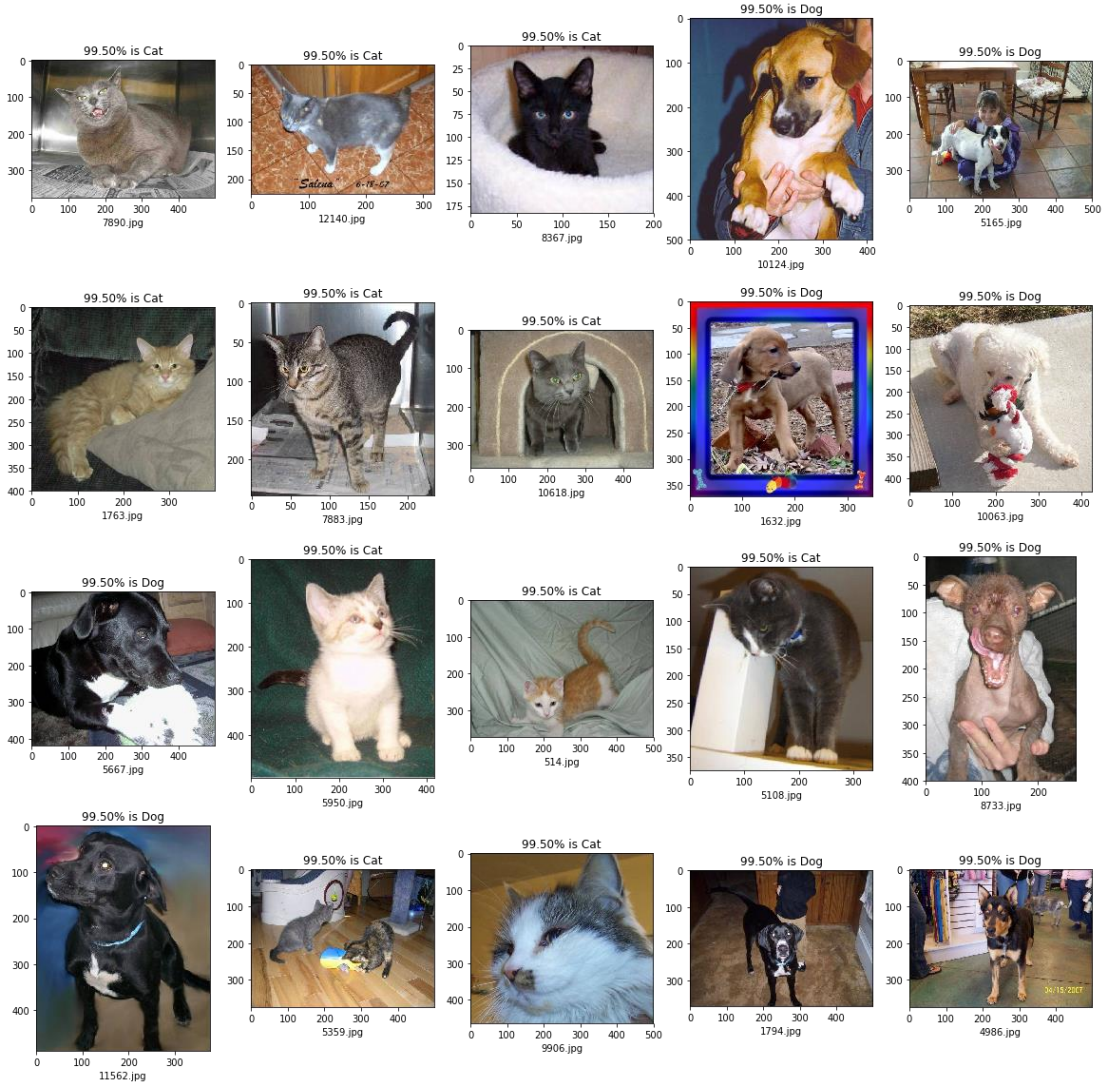
https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition/submit

使用每个保存的模型权重对测试集进行预测，8 个模型中的第 5 个取得了最好成绩 0.04099，达成目标。



3、可视化预测结果

使用第 5 个模型预测的结果可视化测试集，这里只随机抽样了 20 个，即使一些不常见的品种也可以准确预测。



Ⅴ. 对项目的思考

- 训练数据中存在五花八门的异常值，刚开始我根本没有在意，一直改模型改参数，殊不知在某些任务中清洗数据可能比构建模型更为重要。
- 超参数 learning rate 和 batch_size 是会对结果产生影响的，如何选择合适的超参数，需要综合数据分布及特点、不同的模型和优化器等因素考虑。
- 之前因为担心计算资源消耗过大，始终不肯增加全连接层深度，结果评分一直在 0.07 左右徘徊，在思考数据集多样化的特点后，增加网络深度，最终取得了较好的成绩。事实上神经网络的复杂度就像人的脑容量，太简单的神经网络记不住多样化的特征，而太复杂的神经网络又会学到过多细节，导致过拟合，在实际应用中，还得考虑计算成本和计算速度的问题。在“脑容量”和模型表现之间寻找平衡点是机器学习工程师必需要考虑的问题。
- 多读论文，多查资料，理解和学习最先进的技术，事半功倍。

参考文献：

1. Gao Huang/Cornell University, Zhuang Liu/Tsinghua University, Laurens van der Maaten/Facebook AI Research, 《Densely Connected Convolutional Networks》

2. ADAM 算法：Diederik P. Kingma, Jimmy Lei Ba,《A METHOD FOR STOCHASTIC OPTIMIZATION》

3. momentum：Ilya Sutskever, James Martens, George Dahl, Geoffrey Hinton,

《On the importance of initialization and momentum in deep learning》

4. Dropout：G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever and R. R. Salakhutdinov,《Improving neural networks by preventing co-adaptation of feature detectors》

5. 博客：https://bigquant.com/community/t/topic/127342

6. 知乎：https://zhuanlan.zhihu.com/p/34435247