定义

项目概述

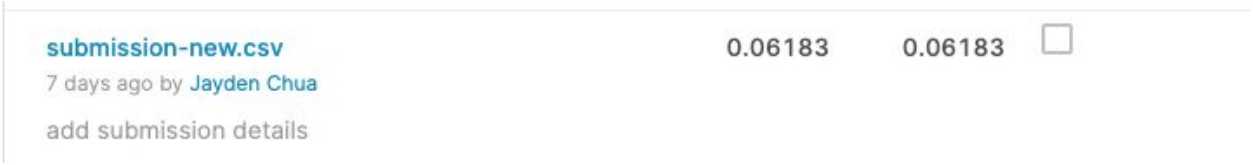
该项目于 2013 年由 Kaggle 发起。 它开始于一个有趣的竞赛：“狗与猫”。

尽管它最初是一个有趣的竞赛，但该项目也为刚接触机器学习的新手提供了良好的起点。值得一提的是，自 2013 年首次启动以来，该项目已经发生了相当大的变化。随着机器学习领域的变化，提交笔记本也随着时间而发展。 这是一个有趣的项目，因为它提供了多年来使用的不同类型技术的广泛概述。

问题陈述

该项目是有监督的单标签二进制分类问题。

所有图像都可以分类为猫或狗。 该问题的目的是训练模型，该模型的准确度应在排行榜的前 10% 之内，从而能够对图像进行分类。



通过浏览数据集，可以获得有关数据集的基本信息。 训练文件夹包含 25,000 张狗和猫的图像。 此文件夹中的每个图像都有标签作为文件名的一部分。 测试文件夹包含 12,500 张图像，并根据数字 ID 命名。

对于测试集中的每个图像，模型应该预测该图像是狗的概率（1 = 狗，0 = 猫）。

评价指标

为了确定模型的准确率，我们将使用定义的对数损失：

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

使用的符号描述如下：

- n 是测试集中的图像数量
- y_i 是图像成为狗的预测概率
- y_i 如果图像是狗，则为1；如果是猫，则为0
- $log()$ 是自然（以e为底）对数

分析

数据的探索

在 Kaggle 提供的数据集中，有 25,000 张带标签的狗和猫图像可用于训练，而有 12,500 张图像用于测试。

训练集的命名约定如下“ category.index.jpg”。 例如，猫的图像看起来像“ cat.1234.jpg”，类似地，狗的图像将被命名为“ dog.2345.jpg”。

In [21]:
```
for file in filenames[:10]:
    print(file)
```

```
train/cat.0.jpg
train/cat.1.jpg
train/cat.10.jpg
train/cat.100.jpg
train/cat.1000.jpg
train/cat.10000.jpg
train/cat.10001.jpg
train/cat.10002.jpg
train/cat.10003.jpg
train/cat.10004.jpg
```

In [22]:
```
for file in filenames[20000:20010]:
    print(file)
```

```
train/dog.5499.jpg
train/dog.55.jpg
train/dog.550.jpg
train/dog.5500.jpg
train/dog.5501.jpg
train/dog.5502.jpg
train/dog.5503.jpg
train/dog.5504.jpg
train/dog.5505.jpg
train/dog.5506.jpg
```

在训练数据集中，每个类别（狗和猫）都有 12500 张图像。

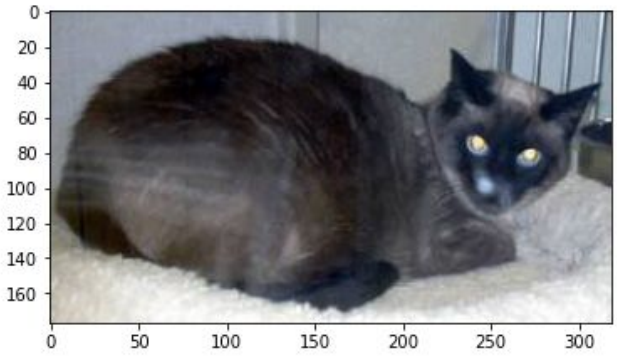
In [32]:
```
labels = np.array([(0 if 'cat' in fname else 1) for fname in filenames])
# Total number of cats
cats = np.sum(labels == 0)
dogs = np.sum(labels == 0)

print("Total number of cats: " + str(cats))
print("Total number of dogs: " + str(dogs))
```

```
Total number of cats: 12500
Total number of dogs: 12500
```

数据集中的图像是具有不同图像大小的彩色图像。

```
In [34]: img = plt.imread(f'{PATH}{filenames[10]}')
         plt.imshow(img);
```




```
In [24]: img.shape
Out[24]: (374, 500, 3)
```

探索性可视化

为了更好地理解数据集，我决定通过预先训练的 resnet34 模型运行图像，并查看该模型最不正确的图像类型以及该模型最不确定的图像。此步骤可帮助我很好地了解数据集的外观。

这是模型最错误，最不确定的图像。看来，在训练数据中，有一些标注错误的数据和一些在同一图像中同时包含猫和狗的图像。

最不正确分类的图像：

最不确定的图像：

算法和技术

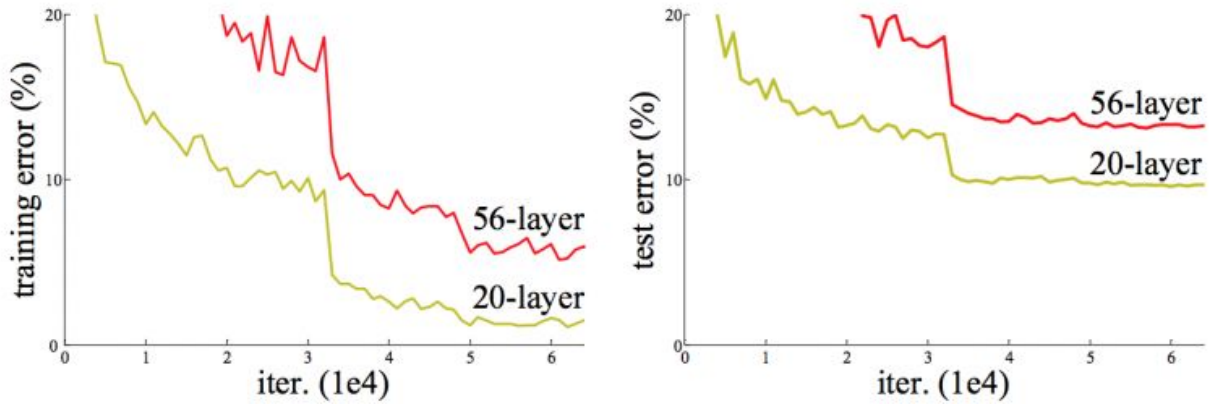
在这个项目中，我选择了FastAI（v0.7）软件包的较早版本。 选择它是为了帮助了解软件包作者的初衷以及该项目将如何发展并进一步适应未来的迭代。 对于这个项目，我想专注于几个我可以了解更多信息的领域。

建筑

对于该项目，出于效率和性能的原因，我选择了带有预训练权重的 Resnet34[2]。 在实施过程中，我们将首先冻结较早的层，直到稍后将其解冻，并在对最后一层进行了培训之后进行重新培训。

ResNets，也称为残差网络，是一种允许网络深度增加的结构，同时当研究人员在网络中添加更多层时面临的退化问题。 参见下图，摘自用于图像识别的深度残差学习[2]。

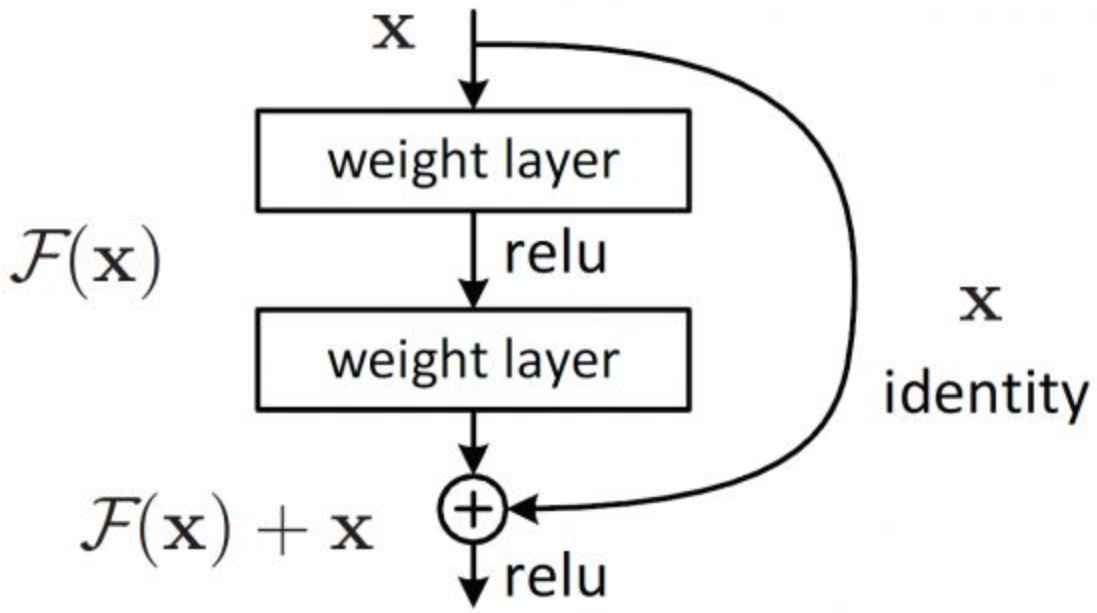
我们可以看到，在“普通”网络上增加了更多的层后，训练和测试的错误率都更高。



图片来自原始论文[2]

在具有20层和56层“普通”网络的CIFAR-10上，训练错误（左）和测试错误（右）。

ResNets的主要区别在于，它们明确允许这些层适合残差映射。 在各层之间添加了快捷连接，以便可以实现 $F(x) + x$ 的公式化（为了更好地可视化），我们可以看到如何通过下图实现此目的。



尽管本文详细介绍了更多体系结构，但我想重点介绍关键点。 您可以在下面的输出层上看到应用了哪些卷积。

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

图片出自原始论文[2] - ResNet34的输出大小和卷积内核

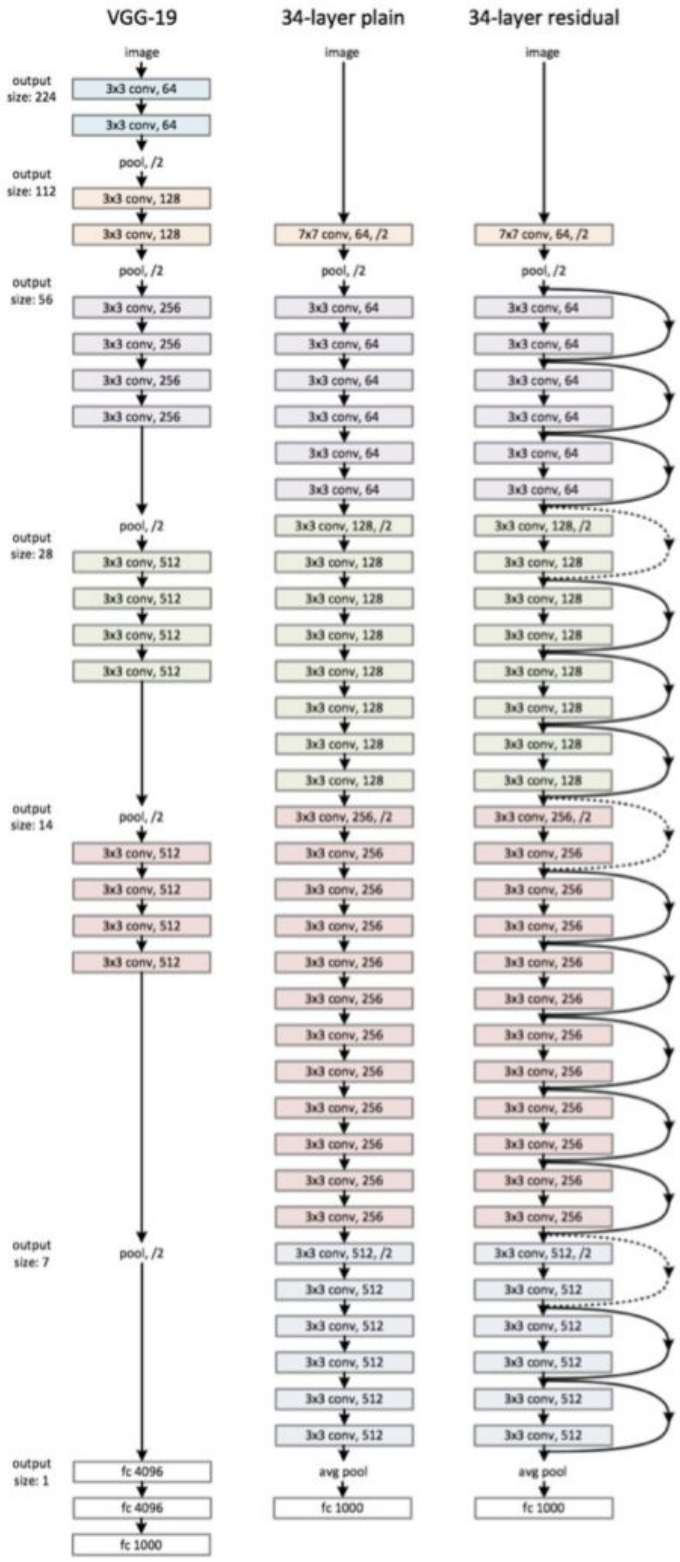
最后，如果您从下图看到快捷方式连接的另一点注意事项，则有两种可能的情况。

从右边的架构图中可以看到带有实线的一个。 这是输出尺寸相同时。

另一种情况是带有虚线，这些虚线在输出尺寸更改时发生，在这种情况下，本文考虑了2个选项。

首先，将执行身份映射，并添加额外的零以增加维度。 这不会引入其他参数。

第二，投影快捷方式用于匹配尺寸（按1x1卷积完成）。

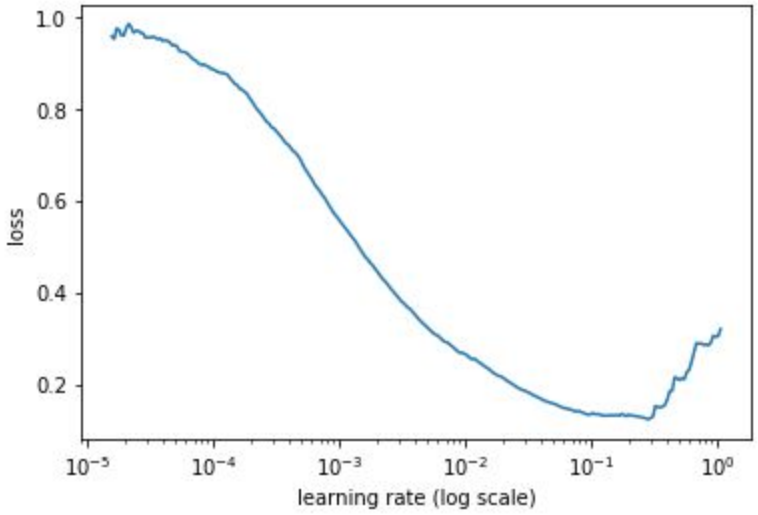


原始论文中的ResNet34 [2]

学习率选择

第一项有助于调整学习速度的技术。 由于学习速率决定了权重的更新速率，因此学习速率可能是至关重要但难以设置的超参数。 使用 FastAI 库中`learn.lr_find()`方法，该方法使用 2015 年论文 Cyclical Learning Rates for Training Neural Networks [1] 中开发的技术。 本文介绍了如何通过从很小的值开始增加学习率直到损失停止减少来实现这一点。

下面是损失与学习率的关系图。 在这里，我们可以看到，随着将学习率的值增加到 `1e-2`，损失仍在减少。 但是根据经验，似乎最好的性能提升是在 `1e-3` 左右，这就是我们使用的价值。



数据扩充

为了防止过度拟合，数据增强可以证明是一个很好的工具。 为了实现这一点，我们将使用 `tfms_from_model()`来帮助我们提供一组增强图像。 我们选择的变换列表是随机切口，旋转，照明，翻转和缩放，以帮助提供更多样化但可能的增强图像集。 下面是增强图像外观的一些示例。

基准模型

在 Kaggle 中，已经进行了许多尝试以许多有趣的技术来改善对数丢失。 在这里，我们将集中精力使用 resnet34 并一起针对最后几层进行优化，并使用上面列出的技术。 下一节将详细解释实现细节。

目标是尝试成为排行榜的前10%。 当前，排行榜上有1314个条目，因此我们力求比当前日志损失为0.06127的第131个条目更好。

方法

数据预处理

该项目中使用的数据预处理步骤包括规范化，删除和规范化。

正常化

为了规范化数据，我们确保所用图像的大小都相同。 在此项目中，我们将图像标准化为尺寸为 224 x 224 像素的正方形。

另一个关键的规范化是确保从训练集中删除同时包含猫和狗的图片。 这样可以确保我们实际上正在培训正确的问题。 在这种情况下，我们对该问题作为二元分类问题的最初假设成立，并且我们现在将删除将导致我们接受多标签分类问题训练的图像。

清除

为了确保在训练过程中提供的数据尽可能准确，我们可以实际绘制出图像以查看模型正在理解的内容。 模型可以很好地理解什么样的图像。 在此可视化步骤中，我们发现有许多贴错标签的图像。 这可能会导致模型的准确性降低。

正则化

为了实现这一点，我们已经提到了如何使用数据增强将噪声添加到数据中。 数据预处理中的这一步骤对于帮助减少过度拟合至关重要。

执行过程

我已决定使用针对此问题的预训练模型作为起点。 如前所述，我将使用resnet34模型。该模型已在ImageNet上进行了训练。 当我们创建一个预训练模型时，我将使用默认的预训练模型功能。
我将首先以2个时期训练模型，并进行快速可视化以查看模型正在处理的图像类型。 我对分类最错，不确定性最高的那些特别感兴趣。 然后从可视化中删除训练数据中错误分类的图像。

接下来，我找到最佳学习率并将其用于模型。 在这种情况下，我发现最佳学习率是 `1e-3` 。

确定学习率之后，我现在将在培训期间继续使用数据增强技术。 我决定对训练过程中随机应用的图像使用一些变换，此步骤的目的是减少训练过程中的过度拟合。

我将使用的另一种技术是学习速率退火的一种变体。 这种技术称为随机梯度下降，具有重启功能。 本质上，随着训练的进行，该技术逐渐降低了学习率。 一旦确定模型不再改进，我对模型感到满意。 我将继续下一步。

完善

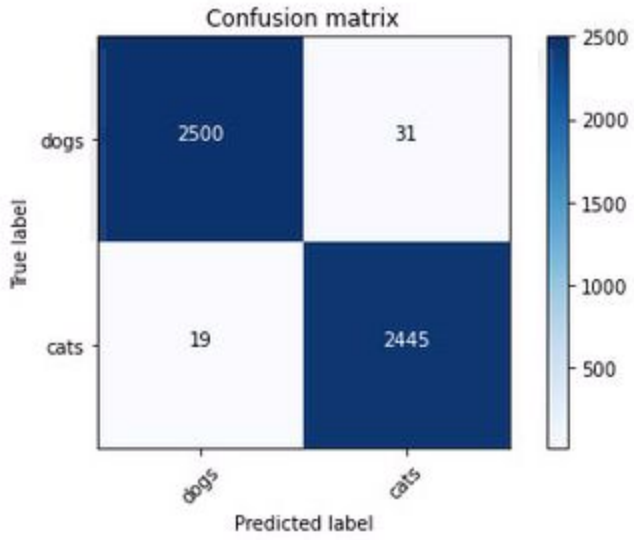
在下一步中，我将开始解冻模型的较早层并重新训练所有层。 但是，我将在各层应用差异学习率。 由于早期的层确定了更多的基本结构，因此我决定使用较小的学习率，并在早期，中间和最后层之间以 10 的比例增加它们。 在这里，我分别使用了 `1e-4`、`1e-3` 和 `1e-2`。

最后，最后一种技术是在推理过程中使用数据增强。 这种方法可以通过 `learn.TTA()` 方法轻松实现，该方法代表测试时间的增加。 此方法实质上将原始图像与该图像的许多随机增强版本一起使用，并从所有这些图像中获取平均预测。 我希望减少推理过程中的错误数量。

结果

模型的评价与验证

为了可视化我们基于验证集的模型的性能，我们可以使用混淆矩阵来查看有多少模型正确。



在下面的时间段0-62进行训练时，我们可以看到模型的训练损失，验证损失和准确性。训练是在整个网络上以不同的学习率进行的。 如果我们以排行榜得分为基准，那么在50-62时期之间，我们将再也看不到明显的收益，并且我们的模型已经处于相当不错的水平。

```
In [43]: lr=np.array([1e-4,1e-3,1e-2])

In [44]: learn.fit(lr, 6, cycle_len=1, cycle_mult=2)
```

epoch	trn_loss	val_loss	accuracy
0	0.190831	0.049065	0.982983
1	0.16624	0.03888	0.986987
2	0.141265	0.036554	0.988989
3	0.138532	0.031759	0.987588
4	0.117481	0.029631	0.988589
5	0.120785	0.029291	0.98979
50	0.056935	0.020208	0.992793
51	0.056746	0.020956	0.992793
52	0.062577	0.020214	0.993794
53	0.059009	0.01987	0.993994
54	0.060876	0.020246	0.992993
55	0.05788	0.022229	0.992392
56	0.061917	0.02012	0.993193
57	0.058934	0.020321	0.993193
58	0.056817	0.019923	0.993393
59	0.055754	0.020312	0.993393
60	0.056288	0.022561	0.992593
61	0.055399	0.019994	0.992993
62	0.053699	0.020484	0.993193

以下是该模型分类的最不正确的猫和狗预测。从显示的图像中，不难理解为什么某些分类未正确完成。

最不正确的猫

最不正确的狗

合理性分析

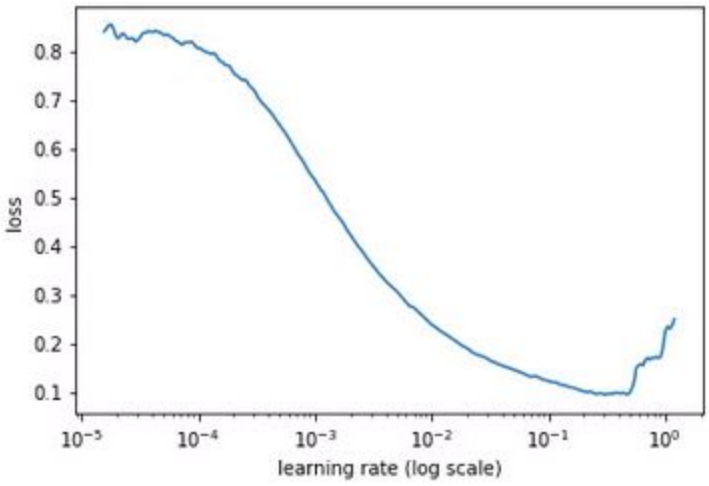
我相信这个项目中引入的方法确实很有趣，并且设法提高了准确性，而不必在现有的基础上增加更多的层或引入更复杂的体系结构，从而可以通过一些非常小但功能强大的调整来进行改进。

添加常规训练与验证损失并与新模型结果进行比较。

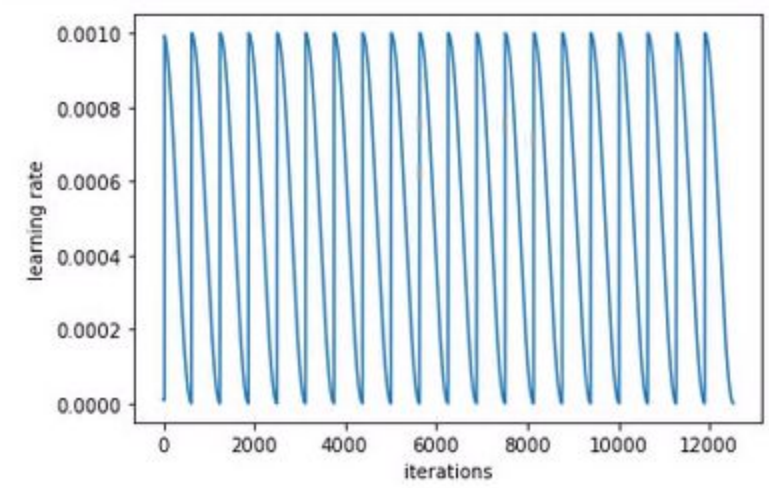
项目结论

结果可视化

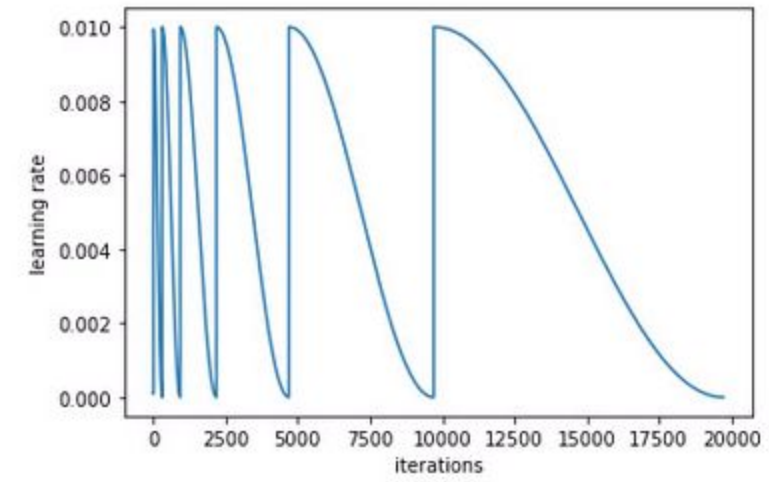
从下图可以很好地了解学习率在一定范围内的表现。 根据下图选择1e-3的学习率。



下图显示了学习率在迭代过程中如何变化。 我们在图中看到的尖峰可以帮助我们了解“重新启动”在“具有重新启动的随机梯度下降”中的样子。



接下来，了解带有 `cycle_mult` 参数的学习率差异。 下图显示了学习速率如何在最后一层的 `cycle_mult` 值为2的情况下随着迭代而变化。我们没有在固定周期上改变学习率，而是开始以2的周期乘数重新启动学习率，因此图的形状 。



对项目的思考

我们对问题做出了一些假设，这些假设使我们可以将问题假定为二进制分类问题。 我非常想知道如何根据当前的实现方式进行修改，以将其从单标签分类问题变为多标签分类。

FastAI 库能够使用 beta 版本的库引入许多有趣的技术，这也让我感到惊讶。 我对使用更高版本并重新实现此解决方案非常感兴趣。

最后，我还想探索其他体系结构和库，以了解如何也可以实现它，以及我们现在用来解决同一问题的可用的最新技术有哪些。

需要作出的改进

除了实际提高模型的准确性外，我还想探索项目的其他部分，例如推理部分。 我想探讨如何以及在何处将其部署到不同的边缘设备。

我感兴趣的一些边缘用例是网络，iOS，Android 和 HarmonyOS 等移动应用程序。

参考文献

[1] Cyclical Learning Rates for Training Neural Networks

[2] Deep Residual Learning for Image Recognition

[3] Learning Rate Schedules and Adaptive Learning Rate Methods for Deep Learning

[4] A disciplined approach to neural network hyper-parameters: Part 1 -- learning rate, batch size, momentum, and weight decay