

论文笔记：AutoAugment

原创

Zerg_Wang

于 2020-03-08 01:45:26 发布

1989

★ 收藏 21

编辑

版权

分类专栏：

Machine Learning

 文章标签：


机器学习

深度学习

augment

autoaugment

数据增强

 Machine Learning

专栏收录该内容

0 订阅

13 篇文章

第二次Paper Reading，决定读一下自动 **数据增强** 领域的开山鼻祖——AutoAugment: Learning Augmentation Policies from Data。本篇解读仅为个人理解，若有错漏，恳请指正。

作者使用到的16种增强操作（图像变换方式）

Operation Name	Description	Range of magnitudes
ShearX(Y)	Shear the image along the horizontal (vertical) axis with rate <i>magnitude</i> .	[-0.3,0.3]
TranslateX(Y)	Translate the image in the horizontal (vertical) direction by <i>magnitude</i> number of pixels.	[-150,150]
Rotate	Rotate the image <i>magnitude</i> degrees.	[-30,30]
AutoContrast	Maximize the the image contrast, by making the darkest pixel black and lightest pixel white.	
Invert	Invert the pixels of the image.	
Equalize	Equalize the image histogram.	
Solarize	Invert all pixels above a threshold value of <i>magnitude</i> .	[0,256]
Posterize	Reduce the number of bits for each pixel to <i>magnitude</i> bits.	[4,8]
Contrast	Control the contrast of the image. A <i>magnitude</i> =0 gives a gray image, whereas <i>magnitude</i> =1 gives the original image.	[0.1,1.9]
Color	Adjust the color balance of the image, in a manner similar to the controls on a colour TV set. A <i>magnitude</i> =0 gives a black & white image, whereas <i>magnitude</i> =1 gives the original image.	[0.1,1.9]
Brightness	Adjust the brightness of the image. A <i>magnitude</i> =0 gives a black image, whereas <i>magnitude</i> =1 gives the original image.	[0.1,1.9]
Sharpness	Adjust the sharpness of the image. A <i>magnitude</i> =0 gives a blurred image, whereas <i>magnitude</i> =1 gives the original image.	[0.1,1.9]
Cutout [12, 69]	Set a random square patch of side-length <i>magnitude</i> pixels to gray.	[0,60]
Sample Pairing [24, 68]	Linearly add the image with another image (selected at random from the same mini-batch) with weight <i>magnitude</i> , without changing the label.	[0, 0.4]

图中，前面14种为Python的PIL库中自带的图像变换操作，Cutout和Sample Pairing为作者额外加入的。

其中，AutoContrast、Invert、Equalize、Solarize、Posterize可以使用PIL中的ImageOps Module来实现，Contrast、Color、Brightness、Sharpness可用ImageEnhance Module来实现。Shear、Translate可用Image Module中的transform类来实现，Rotate就用Image Module的rotate类来实现。

对于其中一些操作解释一下：

首先展示下这张经典的原图：



ShearX、ShearY

分别是对图像沿X或Y轴错切。错切的效果类似于投影。



TranslateX、TranslateY

沿X、Y轴平移图像。

Invert

反色，即像素反转，如图：



Equalize

直方图均衡化。使图像变得：灰度直方图几乎覆盖了整个灰度的取值范围，并且除了个别灰度值的个数较为突出，整个灰度值分布近似于均匀分布)



Solarize

指定一个像素值，对原图中高于该值的像素进行翻转操作，简单来说是一种部分invert操作。

例如翻转像素值128以上的像素：

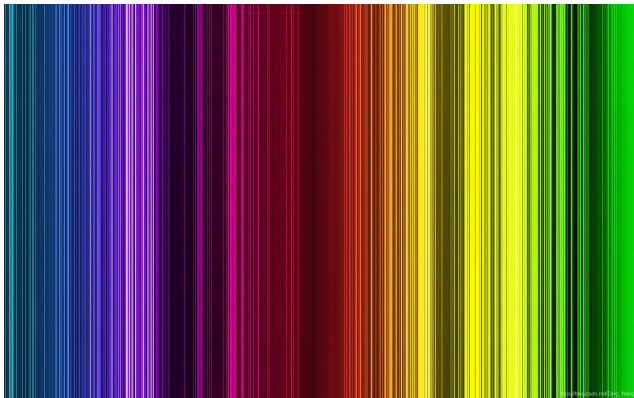


如果这个值设为0，也就是全部像素翻转，结果和invert就一样了。

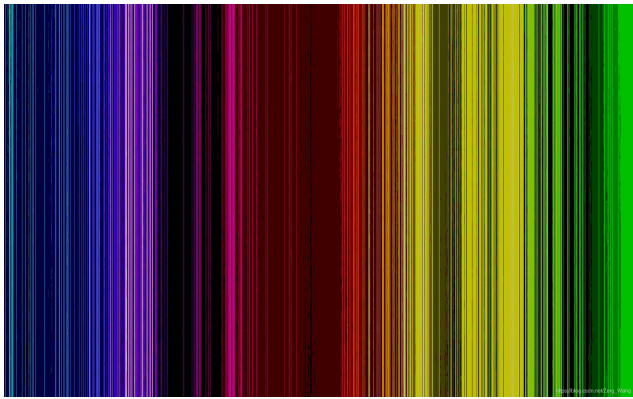
Posterize

色彩分离，色调分离，减少每个颜色通道上的位数（bit）。简单来说就是减少组成图片的色彩的种类，这里拿一张彩色图片举例：

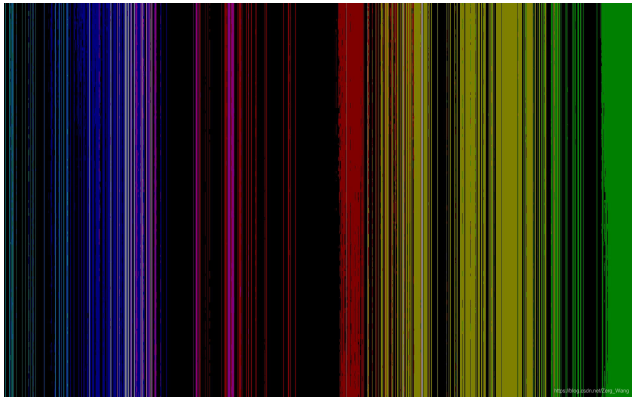
原图：



减少到仅有2位：



减少到仅有1位：



Color

调整色彩平衡，简单来说可以理解为调整图像色调和饱和度。

Python代码为：

```
ImageEnhance.Color(img).enhance(v)
```

以下分别为v=0、0.5、1、1.5处理后的图像：





Sharpness

调整锐度，锐度越高，图像中的物体边缘色彩过渡效果越差，使得物体边缘显得“尖锐”，显得图像很“干”。锐度越低，图像越柔和，但清晰度也会降低。



AutoContrast

自动对比度。设参数为 v 。首先计算输入图像的直方图，然后将其中最亮的 $v\%$ 的像素和最暗的 $v\%$ 的像素去掉，之后重新映射图像，以便保留的最暗像素变为黑色，即0，最亮的变为白色，即255。

因此， v 不能超过50。注意， $v=0$ 处理出的图像与原图不同，因为通过了重新的映射，色彩分布会有变化。作者采用的操作即 $v=0$ 的。

随着 v 值提高，图像色彩种类越少：（以下分别为 $v=0$ ，20，40的图像）



Cutout

随机选择图像中的一块区域，屏蔽掉这块区域的图像内容，如图：



Sample Pairing

在训练时从同一批次的图像中选择一张，将其内容嫁接到原图上，但原图标签不变。

数据增强形式

几个定义

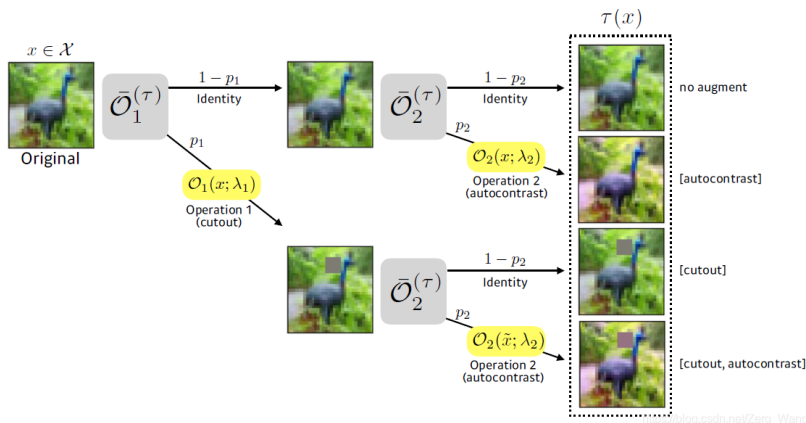
增强策略：多个子策略组成一个完整的增强策略。作者的实验中规定了每个完整策略由5个子策略构成。

子策略：由2个上文提到的增强操作组成，其中，每个操作由2个参数组成：probability和magnitude，分别是使用该操作的可能性以及该操作的使用幅度（也就是上文提到的参数 v ）。不是所有的操作都需要magnitude，比如AutoContrast操作作者就规定其参数为0，但形式上这些操作仍会保留magnitude参数。此外，对于几何变换类的操作（Shear、Translate、Rotate），虽然有magnitude参数规定了操作幅度，但方向是随机的。例如旋转45度，在增强时可能会逆时针旋转45度，也可能顺时针旋转45度。

为了适配作者使用的离散搜索算法，probability和magnitude可能的取值也是离散的。其中probability有0、0.1、0.2……1共11种可能的取值，magnitude有0、1、2……9共10种可能的取值。若算上操作的种类，一个子策略则有6个参数。因此，一个子策略有 $(16 \times 10 \times 11)^2$ 种不同的形式。

增强流程

和传统数据增强方法不同，AutoAugment针对单个数据集，通过算法找到多个增强策略（在作者的实验中，对于每个数据集，作者找到5个增强策略，共25个子策略），然后在使用数据集训练网络的过程中，针对mini batch中的单张图像，会从这一系列子策略中随机抽取一个子策略作用在该图像上。也就是说，不仅每个epoch喂进网络的数据可能都经过了不同的增强操作。甚至在同一epoch中，同一mini batch中的不同图像也可能经过不同的增强操作。然而，因为有probability的存在，即使使用了相同的子策略来增强，结果也会不同。这里借用Fast AutoAugment论文中的一幅图：



假设该子策略为[(Cutout, p_1 , v_1), (AutoContrast, p_2 , v_2)]，则有 $(1-p_1)(1-p_2)$ 的概率该图像不会被更改，也有 p_1p_2 的概率该图会经过两次增强操作（两次增强操作的顺序不可忽略）。同理可知只做一次增强操作的概率为 $(1-p_1)p_2$ 或 $(1-p_2)p_1$ 。

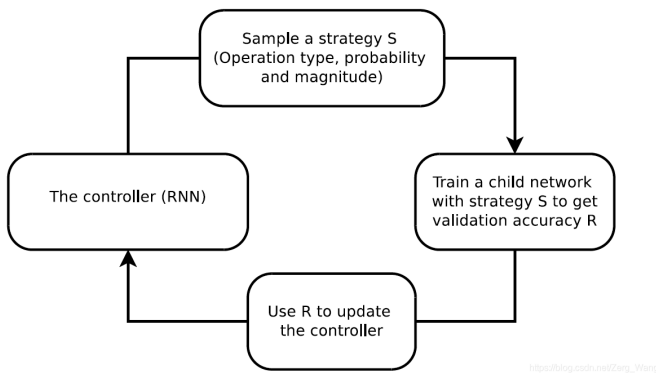
策略搜索算法细节

搜索算法的主体为增强学习，由两部分构成：控制器和训练算法。

控制器由RNN（循环神经网络（Recurrent Neural Network）来充当。更详细来说，作者采用的是单层的LSTM（长短期记忆网络，Long ShortTerm Memory Networks）。每一轮的搜索，该网络会通过Softmax预测一个增强策略。因为一个增强策略有30个值（5个子策略，每个子策略2个操作，每个操作包括操作本身、幅度、概率共3个参数，所以 $5 \times 2 \times 3 = 30$ ），所以网络设置了30个Softmax的预测值。

接下来，生成的增强策略会应用到子模型的训练中。子模型在训练时，对于每个mini batch中的每一张图像都会从5个子策略中随机选择一个应用（流程和本文前面所述一致）。子模型训练完后会在验证集上进行评估，该子模型的精确度会作为“奖励信号”反馈给RNN控制器。最后，控制器会通过该信号，并在近端策略优化算法(Proximal Policy Optimization algorithm, PPO)下进行更新。

搜索过程中控制器会生成大概15000个增强策略（即进行15000轮左右的迭代），最后会选出最好的5个，组成一个有25个子策略的结果，作为实验数据集“算法”认为最好的增强操作。



近端策略优化算法（Proximal Policy Optimization algorithm, PPO）

控制器是如何在PPO算法下更新的？本文不会详细展开（毕竟我也不会……），这里根据AI研习社的教程（<https://www.jianshu.com/p/57c59d580d40>），简单讲讲其流程：

首先讲讲策略梯度算法，其简要思想为：在第*t*步时，策略 π_t 接受状态 s_t ，输出动作概率分布，在动作概率分布中采样动作，执行动作 a_t ，得到回报，跳到下一个状态。在这样的步骤下，我们可以使用策略 π_t 收集一批样本，然后使用梯度下降算法学习这些样本。策略损失函数为：（这里一般采用对数概率）

$$L^{PG}(\theta) = E_t[\log \pi_{\theta}(a_t | s_t) * A_t]$$

Policy Loss Expected log probability of taking that action at that state

Advantage if $A > 0$, this action is better than the other action possible at that state

通过梯度下降算法来更新策略，并实现回报一步步提高。对于差的action，在概率分布中其对应概率会下降（之后选中其的可能性更小了，例如在AutoAugment中，较差的子模型，即accuracy低的，之后选中的可能性变小，对应的那组子策略选中的可能性也就变小），较好的action则其概率会上升。

但问题在于，策略梯度算法不好把握步长。面对看似不错或看似较差的action，可能导致该action在动作概率分布中的概率剧烈波动，可能导致之后反复采取一个action，陷入局部最优中。对此，PPO引入“裁剪的替代目标函数”（Clipped surrogate objective function），用当前策略下的行动概率 $\pi(a|s)$ 和上一个策略的行动概率（ $\pi_{old}(a|s)$ ）的比例来代替对数概率。该比例大于1，说明当前策略下的行动比原先策略的更有可能发生，若小于1，则当前策略下行动发生的概率低于原先。

新的损失函数为：

$$L^{CPI}(\theta) = \hat{\mathbb{E}}_t \left[\frac{\pi_{\theta}(a_t | s_t)}{\pi_{\theta_{old}}(a_t | s_t)} \hat{A}_t \right] = \hat{\mathbb{E}}_t \left[r_t(\theta) \hat{A}_t \right]$$

但如果当前策略的行動的可能性和之前策略的可能性差距太大的话，其比例也可能过大，进而导致过于剧烈的梯度更新，为此PPO进一步加了限制：

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(\frac{\pi_{\theta}(a_t | s_t)}{\pi_{\theta_{old}}(a_t | s_t)} \hat{A}_t, \text{clip} \left(\frac{\pi_{\theta}(a_t | s_t)}{\pi_{\theta_{old}}(a_t | s_t)}, 1 - \epsilon, 1 + \epsilon \right) \hat{A}_t \right) \right]$$

L CPI Modifies the surrogate objective by clipping the prob ratio.
--> Which removes the incentive for moving r_t outside of the interval $[1 - \epsilon, 1 + \epsilon]$

clip裁剪函数规定了这个比例要在 $[1 - \epsilon, 1 + \epsilon]$ 之间，若小于该区间取 $1 - \epsilon$ ，大于取 $1 + \epsilon$ 。 ϵ 为超参数。

效果分析

CIFAR10、CIFAR100

策略搜索细节

作者在实验中发现，AutoAugment对数据集中的样本数量不敏感。作者称，在固定的搜索时间下，让子模型使用更少样本，训练更多回合，效果要比训练回合少但样本数量大的情况好。由此，作者从CIFAR10的50000张图像中抽出4000张，作为“Reduced CIFAR10”作为实验数据集搜索增强策略。搜索使用的子模型为Wide-ResNet-40-2，这是因为每个子模型可以从头训练，从而能计算整个控制器的梯度更新。超参数为：epoch为120，learning rate为0.01，weight decay为0.0001。学习率退火为一个周期的cosine。

效果分析

在Reduced CIFAR10上搜索到的策略，大多为色彩变换，例如Equalize，AutoContrast，Color和Brightness。

利用在Reduced CIFAR10上搜索到的策略，作者在CIFAR10、Reduced CIFAR10以及CIFAR100数据集上进行了训练，结果如图：

Dataset	Model	Baseline	Cutout [12]	AutoAugment
CIFAR-10	Wide-ResNet-28-10 [67]	3.9	3.1	2.6±0.1
	Shake-Shake (26 2x32d) [17]	3.6	3.0	2.5±0.1
	Shake-Shake (26 2x96d) [17]	2.9	2.6	2.0±0.1
	Shake-Shake (26 2x112d) [17]	2.8	2.6	1.9±0.1
	AmoebaNet-B (6,128) [48]	3.0	2.1	1.8±0.1
	PyramidNet+ShakeDrop [65]	2.7	2.3	1.5 ± 0.1
Reduced CIFAR-10	Wide-ResNet-28-10 [67]	18.8	16.5	14.1±0.3
	Shake-Shake (26 2x96d) [17]	17.1	13.4	10.0 ± 0.2
CIFAR-100	Wide-ResNet-28-10 [67]	18.8	18.4	17.1±0.3
	Shake-Shake (26 2x96d) [17]	17.1	16.0	14.3±0.2
	PyramidNet+ShakeDrop [65]	14.0	12.2	10.7 ± 0.2
SVHN	Wide-ResNet-28-10 [67]	1.5	1.3	1.1
	Shake-Shake (26 2x96d) [17]	1.4	1.2	1.0
Reduced SVHN	Wide-ResNet-28-10 [67]	13.2	32.5	8.2
	Shake-Shake (26 2x96d) [17]	12.3	24.2	5.9

为了和Baseline对比，作者在使用搜索到的增强策略的实验时，具体的增强操作为：先使用Baseline的增强操作，之后应用AutoAugment搜索到的操作，最后应用Cutout。其中，最后使用的Cutout的区域大小为默认的16像素值。但需要注意的是，AutoAugment搜索的操作中，可能也包含Cutout，这个Cutout的区域大小就取决于搜索到的magnitude。（然而实际结果是Cutout没有出现在AutoAugment搜索到的最好的25个子策略中.....）

顺便提一下，作者还做了一个比较有意思的实验：将搜索得来的策略中的probability和magnitude的数值随机化，将这个更改过的策略再应用到训练集中，发现效果也很好（原策略2.6%，更改后3.0%）。说明AutoAugment搜索出的增强策略有较强的适用性。然后，作者进一步地，不仅将probability和magnitude的数值随机化，操作种类也随机化了，平均结果为3.1%。说明即使是在搜索空间中完全随机地取一个策略出来，效果都比baseline要好，这一方面说明了AutoAugment的增强方式有其优越性，另一方面也说明了AutoAugment搜索算法并非“鸡肋”，在进一步提高正确率上是十分必要的。

SVHN

策略搜索细节

在SVHN上作者同样使用了简化的数据集。作者使用的“Reduced SVHN”含有1000张从core training set中随机抽取的图像。搜索过程中的模型和超参数则与在Reduced CIFAR10上的一致。

效果分析

与从Reduced CIFAR10上搜索到的策略不同，在Reduced SVHN上，最常出现的操作为Invert、Equalize、Shear和Rotate。这也基本符合人们的认知。数字识别的重点在于它们的形状，而不是它们的颜色。此外，几何变换类的操作中，Shear是最多的。这同样说明识别数字时是多角度的，非直视状态下，数字产生“投影”式的变换，与Shear操作契合。

实验中为了与Baseline作对比，作者在使用搜索到的增强策略做实验时，增强操作为：Baseline的操作+AutoAugment搜索到的增强操作。与CIFAR的略有不同，因为作者发现之后再使用Cutout会显著降低精确度。

具体实验结果也在上面的表中，这里不再赘述。

ImageNet

使用由120类共6000张图像构成的“Reduced ImageNet”数据集。搜索所用子模型为Wide-ResNet-40-2，learning rate为0.1，weight decay为10⁻⁵。

搜索到的增强策略与CIFAR类似，以色彩变换为主，但略有不同的地方在于ImageNet中Rotate操作较多。

在与baseline的对比中，作者同样使用了baseline的增强操作+Autoaugment搜索到的操作，具体实验结果见下表：

Model	Inception Pre-processing [59]	AutoAugment ours
ResNet-50	76.3 / 93.1	77.6 / 93.8
ResNet-200	78.5 / 94.2	80.0 / 95.0
AmoebaNet-B (6,190)	82.2 / 96.0	82.8 / 96.2
AmoebaNet-C (6,228)	83.1 / 96.1	83.5 / 96.5

Table 3. Validation set Top-1 / Top-5 accuracy (%) on ImageNet.

策略迁移

从Reduced ImageNet上搜索得来的增强策略，直接应用到FGVC数据集（细粒度分类数据集，即对属于同一基础类别的图像（如汽车、狗、花、鸟等）进行更加细致的子类划分的数据集）上也有亮眼的表现：

Dataset	Train Size	Classes	Baseline	AutoAugment- transfer
Oxford 102 Flowers [43]	2,040	102	6.7	4.6
Caltech-101 [15]	3,060	102	19.4	13.1
Oxford-IIIT Pets [14]	3,680	37	13.5	11.0
FGVC Aircraft [38]	6,667	100	9.1	7.3
Stanford Cars [27]	8,144	196	6.4	5.2

Table 4. Test set Top-1 error rates (%) on FGVC datasets for Inception v4 models trained from scratch with and without AutoAugment-transfer. Lower rates are better. AutoAugment-transfer results use the policy found on ImageNet. Baseline models used Inception pre-processing.

https://blog.csdn.net/Zerg_Wang

这也说明了AutoAugment的算法所找到的增强操作并没有“overfitting”相应的数据集，相反，这些操作具有更广泛的适用性。

其他

epoch与sub-policy

在搜索和应用增强策略的时候，对于每个mini batch中的每张图像，都是从多个子策略（sub-policy）中随机选择一个来应用的。因为策略的随机应用，模型则需要足够多样的增强操作来保证其泛化能力。因此，搜索时需要运行足够多的epoch，从而令控制器生成足够多的子策略进行择优与验证。

子策略数量与精度的关系

通过搜索找出的策略中，作者只选择了其中最好的5个（共25个子策略）。如果我们将这个数字扩大，将得来的更多子策略用在后面模型的训练中，精度是否会提高呢？作者做了如下实验，结论是：20个左右的子策略已经足够了。

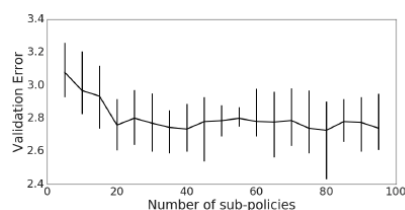


Figure 4. Validation error (averaged over 5 runs) of Wide-ResNet-28-10 trained on CIFAR-10 as a function of number of *randomly selected* sub-policies (out of a pool of 500 good sub-policies) used in training with AutoAugment. Bars represent the range of validation errors for each number.

https://blog.csdn.net/Zerg_Wang

参考资料

<https://arxiv.org/abs/1805.09501v3>

http://www.tensorinfinity.com/paper_144.html

<https://zhuanlan.zhihu.com/p/88525394>

<https://www.jianshu.com/p/57c59d580d40>