

# 随机擦除项目文档

191250102 罗雪瑾

## 论文理解

**数据增强**是一种数据扩充技术，指的是利用有限的数据创造尽可能多的利用价值。因为虽然现在各种任务的公开数据集有很多，但数据量也远远不够，而公司或者学术界去采集、制作数据的成本也很高，例如人工标注数据的任务量就很大，因此，要通过一些方法去更好的利用现有的成本，比如数据增强。

传统数据增强方式有随机翻转、旋转、裁剪、变形缩放、添加噪声、颜色扰动等。

### 痛点

深度学习模型的参数很多，模型复杂度很高，如果此时数据集数量不够导致数据集中数据的复杂度没有涵盖所有特征空间，那么模型就会学习到这些数据集的一些独有的特征，这会导致**过拟合**的问题，因此我们有必要对数据集进行增强操作以减小过拟合的风险。这就表现在有些网络在training set上表现很好，但是到了test set中表现会异乎寻常的差。为了提高网络泛化能力，数据增强是一个很重要的方向，诸如随机裁剪，翻转等已经成为非常常用的方法。

对于网络泛化能力而言，**遮挡**是一个重要的影响因素。就如行人重识别中的遮挡数据集，考验的是网络能否关注整体和局部细节，和网络对于带遮挡对象的泛化能力。带遮挡也是ReID场景中非常常见的场景，也是制约ReID技术落地一大难点之一。

### 解决方法

本论文为CNN训练提出了一种新的数据增强方法，**Random Erasing**。即在一张图片里随机选择一个区域，然后采用随机值进行覆盖，模拟遮挡场景。通过该方法能够给图片加入不同程度的遮挡，迫使网络利用局部未遮挡的数据进行识别，加大了训练难度，可以提高模型提高泛化能力，减少过拟合的风险，同时对遮挡具有一定的鲁棒性。

该方法可以被视为add noise的一种，并且与随机裁剪、随机水平翻转具有一定的互补性，综合应用他们，可以取得更好的模型表现，尤其是对噪声和遮挡具有更好的鲁棒性。该方法也很容易嵌入到现今大部分CNN模型中用于训练具有更好泛化性能模型。

### 优点

- 轻量级，不需要任何额外的参数学习或内存消耗。它可以在不改变学习策略的情况下与各种CNN模型集成。
- 现有数据扩充和正则化方法的补充方法。当两者结合时，随机擦除进一步提高了识别性能。
- 在图像分类、目标检测和人员重新识别方面不断提高最新的最先进的深度模型的性能。
- 提高神经网络对部分共包样本的鲁棒性。当我们随机添加遮挡到CIFAR-10测试数据集时，随机擦除显著优于基线模型

### 与类似工具对比（创新）

如果通过**添加遮挡的图像**来解决遮挡数据集的数据增广，花费过于昂贵，而且遮挡的等级可能会受到限制。由于选择的区域大小是随机的，所以random erasing可以生成不同遮挡级别的增强图像。

**随机擦除和随机裁剪的区别：**

**随机裁剪**是一种有效的数据增强方法，它减少了背景在CNN决策中的贡献，可以基于对象的部分存在而不是聚焦于整个对象来建立学习模型。与随机裁剪相比，随机擦除保留了物体的整体结构，只是遮挡了物体的某些部分。另外，擦除区域的像素被重新分配随机值，这可以看作是给图像添加了噪声。这两种方法在数据扩充方面是互补的。随机擦除(Random Erasing)、随机裁剪(Random cropping)以及它们的组合示例如图。



Figure 3. Examples of Random Erasing, random cropping, and the combination of them. When combining these two augmentation

与cutout不同的地方有三点：

- 选择的方形大小不是固定的
- 填充区域选择的是 $[0,255]$ 的随机值，而不是0
- 方形区域限制在图像内；

## 核心算法

### 算法原理

简而言之，选择一个合适面积，一个点，然后画框，框内像素随机替代。

---

**Algorithm 1: Random Erasing Procedure**

---

**Input :** Input image  $I$ ;  
Image size  $W$  and  $H$ ;  
Area of image  $S$ ;  
Erasing probability  $p$ ;  
Erasing area ratio range  $s_l$  and  $s_h$ ;  
Erasing aspect ratio range  $r_1$  and  $r_2$ .  
**Output:** Erased image  $I^*$ .  
**Initialization:**  $p_1 \leftarrow \text{Rand}(0, 1)$ .

```
1 if  $p_1 \geq p$  then
2    $I^* \leftarrow I$ ;
3   return  $I^*$ .
4 else
5   while True do
6      $S_e \leftarrow \text{Rand}(s_l, s_h) \times S$ ;
7      $r_e \leftarrow \text{Rand}(r_1, r_2)$ ;
8      $H_e \leftarrow \sqrt{S_e \times r_e}$ ,  $W_e \leftarrow \sqrt{\frac{S_e}{r_e}}$ ;
9      $x_e \leftarrow \text{Rand}(0, W)$ ,  $y_e \leftarrow \text{Rand}(0, H)$ ;
10    if  $x_e + W_e \leq W$  and  $y_e + H_e \leq H$  then
11       $I_e \leftarrow (x_e, y_e, x_e + W_e, y_e + H_e)$ ;
12       $I(I_e) \leftarrow \text{Rand}(0, 255)$ ;
13       $I^* \leftarrow I$ ;
14      return  $I^*$ .
15    end
16  end
17 end
```

---

主要算法如上图，是图片预处理方法，输入图片  $I$ ，擦除的概率  $p$ ，擦除区域比例范围从  $s_l$  到  $s_h$ ，以及长宽比概率范围从  $r_1$  到  $r_2$ 。

首先根据概率  $p$  确定一张图片是否需要擦除：

$$p_1 = \text{Rand}(0, 1)$$

$p_1 > p$  则不对图片进行处理，反之，则需要擦除。

根据输入图片  $I$  可以得到图片的长宽  $W$  和  $H$ ，继而可以得到面积  $S$ 。根据  $\text{Rand}(s_l, s_h) * S$  得到擦除的面积  $S_e$ ，擦除面积的长宽根据下式得到：

$$r_e = \text{Rand}(r_1, r_2), H_e = \sqrt{S_e * r_e}, W_e = \sqrt{\frac{S_e}{r_e}}$$

再根据  $\text{Rand}(0, W)$  和  $\text{Rand}(0, H)$  得到在原图中擦除的左上角位置  $x_e, y_e$ 。加上长宽  $W_e, H_e$  便可以得到擦除区域的位置。考虑到  $x_e + W_e > W$  或者  $y_e + H_e > H$  的情况，一旦是该情况便重复该算法，直到满足  $x_e + W_e \leq W$  或者  $y_e + H_e \leq H$ 。

**RE的随机性包含5个方面：**

- 对于图像，随机擦除与否为随机概率  $p$
- 矩形框与图像的面积比例
- 矩形框的位置
- 矩形框的宽高比
- 填充值是随机的，其范围为[ 0 , 255 ]

## 代码实现

transforms.py

```
class RandomErasing(object):
    """ Randomly selects a rectangle region in an image and erases its pixels.
    'Random Erasing Data Augmentation' by Zhong et al.

    Args:
        probability: The probability that the Random Erasing operation will be
        performed.
        sl: Minimum proportion of erased area against input image.
        sh: Maximum proportion of erased area against input image.
        r1: Minimum aspect ratio of erased area.
        mean: Erasing value.
    """

    def __init__(self, probability=0.5, sl=0.02, sh=0.4, r1=0.3, mean=(0.4914,
0.4822, 0.4465)):
        self.probability = probability
        self.mean = mean
        self.sl = sl # 初始化可选的擦除区域面积设置的最小比例
        self.sh = sh # 初始化可选的擦除区域面积设置的最大比例
        self.r1 = r1 # 初始化最小长宽比

    def __call__(self, img):

        if random.uniform(0, 1) >= self.probability:
            return img

        for attempt in range(100):
            area = img.size()[1] * img.size()[2]

            target_area = random.uniform(self.sl, self.sh) * area # 面积范围[sl,
sh]*area=[0.02*area, 0.4*area]
            aspect_ratio = random.uniform(self.r1, 1 / self.r1) # 长宽比范围[r1,
1/r1]=[0.3, 3.333...]

            h = int(round(math.sqrt(target_area * aspect_ratio))) # 擦除区域高度
            (height)
            w = int(round(math.sqrt(target_area / aspect_ratio))) # 擦除区域宽度
            (width)

            if w < img.size()[2] and h < img.size()[1]:
                x1 = random.randint(0, img.size()[1] - h) # 随机起始点纵坐标
                (height方向)
                y1 = random.randint(0, img.size()[2] - w) # 随机起始点横坐标 (width
                方向)

                if img.size()[0] == 3: # RGB图
                    img[0, x1:x1 + h, y1:y1 + w] = self.mean[0] # 将擦除区域全部赋
                    予预设小数
                    img[1, x1:x1 + h, y1:y1 + w] = self.mean[1]
                    img[2, x1:x1 + h, y1:y1 + w] = self.mean[2]
                else: # 灰度图
                    img[0, x1:x1 + h, y1:y1 + w] = self.mean[0]
                return img
        return img
```

# 具体实现

以图像分类中fashionmnist.py为例 cifar.py同理

Fashion MNIST由60000个培训和10000个测试28x28灰度图像组成。每个图像都与来自10个类的标签相关联。

## 输入输出

载入图像数据集

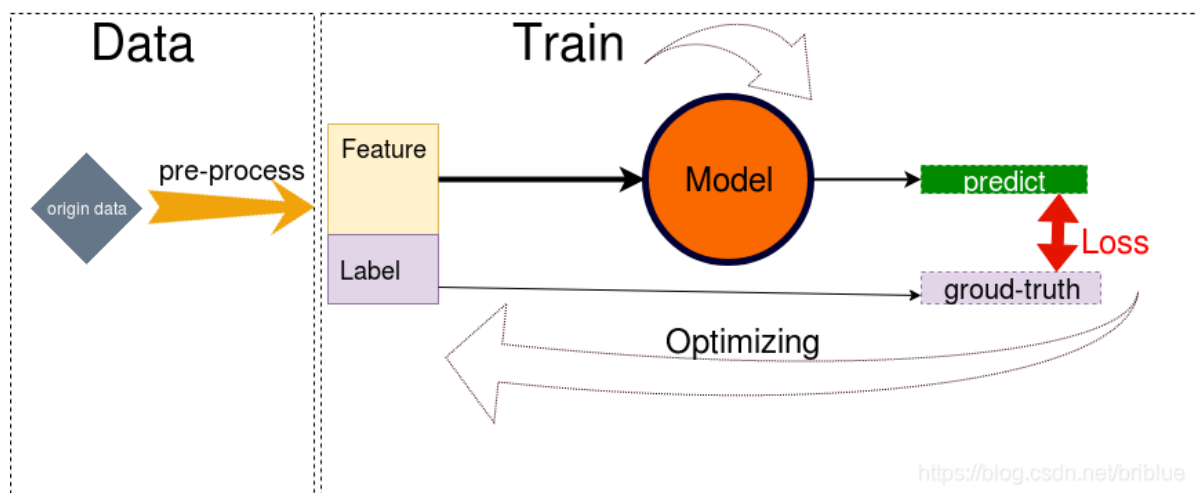
预处理-数据增强

分为训练集、测试集

迭代模型训练并测试

实时输出训练轮次+学习率

输出最佳准确率

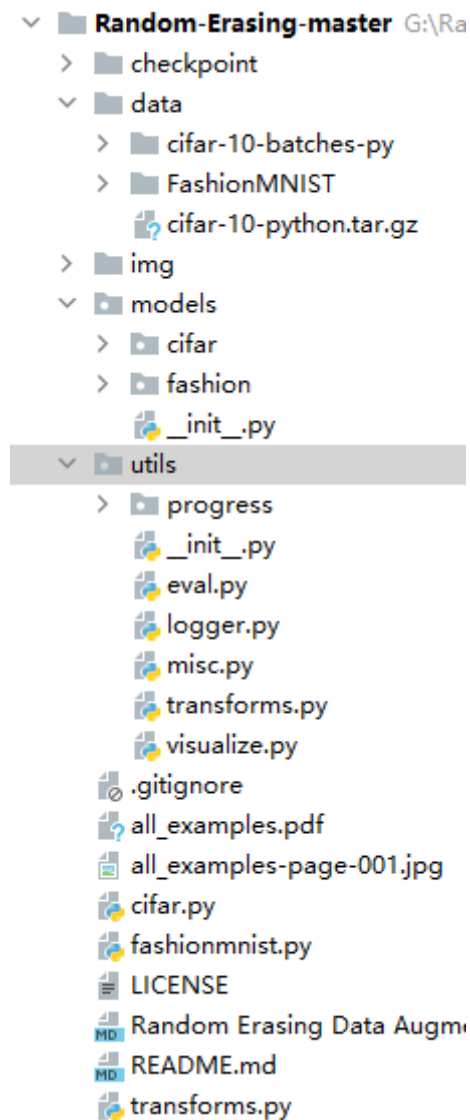


## 功能模块

本项目基于pytorch

## 项目结构

- checkpoint —设置检查点 可以resume
- data 加载cifar-10 cifar-100 FashionMNIST 数据集
- img 图像存档
- models 存放cifar和fashionmnist的ResNet和wrn模型
- utils 存放公共方法
- transforms.py为随机擦除方法（核心算法实现）
- cifar.py为对cifar-10 cifar-100数据集进行训练的文件
- fashionmnist.py为对FashionMNIST数据集进行训练的文件



## 数据流动

### 预处理

加载数据集dataset

### 设置数据增强方法

载入训练集dataloader（有随机擦除） 测试集dataloader（无随机擦除）

# Data 加载数据集

```
print('==> Preparing dataset %s' % args.dataset)
'''
```

所有模型都用数据排列进行训练:随机执行水平翻转, 并从每边 填充28个像素的图像中随机裁剪  
在模型训练过程中使用了随机裁剪、随机翻转和标签平滑规则化

```
'''
```

#数据增强

```
transform_train = transforms.Compose([
```

```
    transforms.RandomCrop(28, padding=4),#随机裁剪到28*28, padding为4
```

```
    transforms.RandomHorizontalFlip(), #随机水平翻转
```

```
    transforms.ToTensor(),#转化成Tensor
```

```
    transforms.Normalize((0.1307,), (0.3081,)),#逐channel的对图像进行标准化, 可以
```

加快模型的收敛

```
    transforms.RandomErasing(probability = args.p, sh = args.sh, r1 =
```

```
args.r1, mean = [0.4914])),
```

```
    #随机擦除
```

```

    ])#设置训练集

    transform_test = transforms.Compose([
        transforms.ToTensor(),#转化成Tensor
        transforms.Normalize((0.1307,), (0.3081,)),
    ])#设置测试集

    if args.dataset == 'fashionmnist':
        dataloader = datasets.FashionMNIST
        num_classes = 10

        #利用dataset形成dataloader以便于我们后续的训练、验证与测试。
        trainset = dataloader(root='./data', train=True, download=True,
            transform=transform_train)
        trainloader = data.DataLoader(trainset, batch_size=args.train_batch,
            shuffle=True, num_workers=args.workers)
        #shuffle: 是否将数据集打乱 num_works: 并行数量
        testset = dataloader(root='./data', train=False, download=False,
            transform=transform_test)
        testloader = data.DataLoader(testset, batch_size=args.test_batch,
            shuffle=False, num_workers=args.workers)

```

## 选择训练模型

CNN架构:CIFAR-10、CIFAR-100和Fashion MNIST采用了四种架构:

ResNet、预激活ResNet、ResNeXt和宽剩余网络WRN

使用ResNeXt-29-8×64和WRN-28-10(详见resnet.py和wrn.py)

```

# Model
print("==> creating model '{}'.format(args.arch))
if args.arch.startswith('wrn'):
    model = models.__dict__[args.arch](
        num_classes=num_classes,
        depth=args.depth,
        widen_factor=args.widen_factor,
        dropRate=args.drop,
    ) #跑wrn
elif args.arch.endswith('resnet'):
    model = models.__dict__[args.arch](
        num_classes=num_classes,
        depth=args.depth,
    )#跑resnet

```

设置了args.resume可以重新恢复加载模型参数

## 制定训练策略

学习率从0.1开始,在第150和225纪元后除以10,在第300个纪元前停止训练

```

# Train and valid
for epoch in range(start_epoch, args.epochs):#逐步迭代数据
    adjust_learning_rate(optimizer, epoch)

    print('\nEpoch: [%d | %d] LR: %f' % (epoch + 1, args.epochs,
        state['lr']))

```

```

# 每次跑一次epoch都保存一下模型
train_loss, train_acc = train(trainloader, model, criterion, optimizer,
epoch, use_cuda)
test_loss, test_acc = test(testloader, model, criterion, epoch,
use_cuda)

# append logger file
logger.append([state['lr'], train_loss, test_loss, train_acc, test_acc])

# save model 比对每轮 记录最佳准确率
is_best = test_acc > best_acc
best_acc = max(test_acc, best_acc)
#保存网络训练状态。
save_checkpoint({
    'epoch': epoch + 1,
    'state_dict': model.state_dict(),
    'acc': test_acc,
    'best_acc': best_acc,
    'optimizer' : optimizer.state_dict(),
}, is_best, checkpoint=args.checkpoint)

```

## 训练与测试

节约篇幅 贴出重要代码

```

#定义训练函数
def train(trainloader, model, criterion, optimizer, epoch, use_cuda):
    # compute output计算输出
    outputs = model(inputs)#计算损失函数
    loss = criterion(outputs, targets)

    # measure accuracy and record loss
    prec1, prec5 = accuracy(outputs.data, targets.data, topk=(1, 5))
    losses.update(loss.item(), inputs.size(0))
    top1.update(prec1.item(), inputs.size(0))
    top5.update(prec5.item(), inputs.size(0))

    # compute gradient and do SGD step
    optimizer.zero_grad()#通过输入得到预测的输出
    loss.backward()#反向传播
    optimizer.step()

    # measure elapsed time
    batch_time.update(time.time() - end)
    end = time.time()
    # plot progress
    return (losses.avg, top1.avg)

#定义测试函数
def test(testloader, model, criterion, epoch, use_cuda):
    global best_acc
    # switch to evaluate mode
    model.eval()

    # measure data loading time
    # compute output
    outputs = model(inputs)
    loss = criterion(outputs, targets)

```



```

# measure accuracy and record loss
prec1, prec5 = accuracy(outputs.data, targets.data, topk=(1, 5))
losses.update(loss.item(), inputs.size(0))
top1.update(prec1.item(), inputs.size(0))
top5.update(prec5.item(), inputs.size(0))

# measure elapsed time
batch_time.update(time.time() - end)
end = time.time()

# plot progress
return (losses.avg, top1.avg)

```

## 输出训练结果

```

print('Best acc:')#输出最佳准确率
print(best_acc)

```

## 附：目标检测

当算法对象是分类任务时，由于不知道对象具体位置，所以random erasing选择区域是在整个图像上；当算法对象是目标检测任务时，由于目标位置可知，所以有三种方案：（1）IRE, Image-aware Random Erasing，不区分目标框和背景区域，在整个图像来做擦除处理；（2）ORE, Object-aware Random Erasing，区分目标框和背景区域，来做擦除处理，只在目标框内做擦除处理；（3）I+ORE，区分目标框和背景区域，分别在目标框和整个图像来做擦除处理。

三种方案如下图所示。

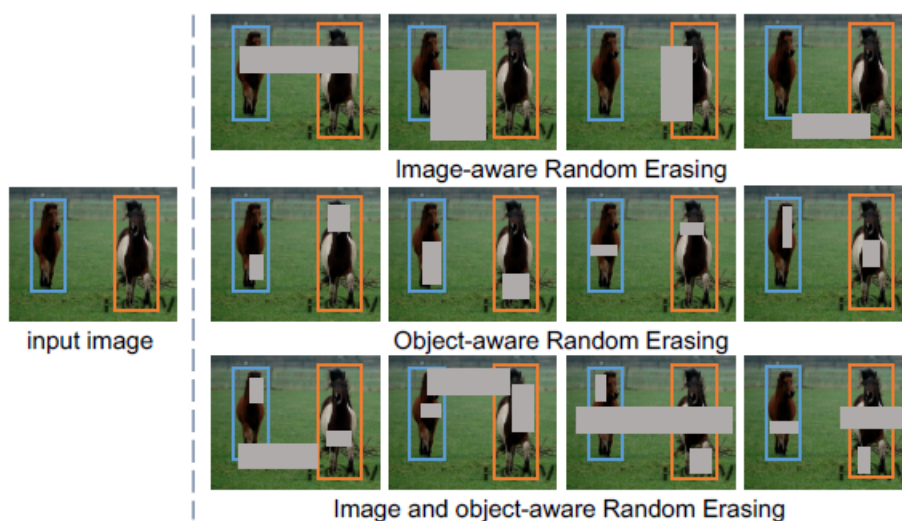
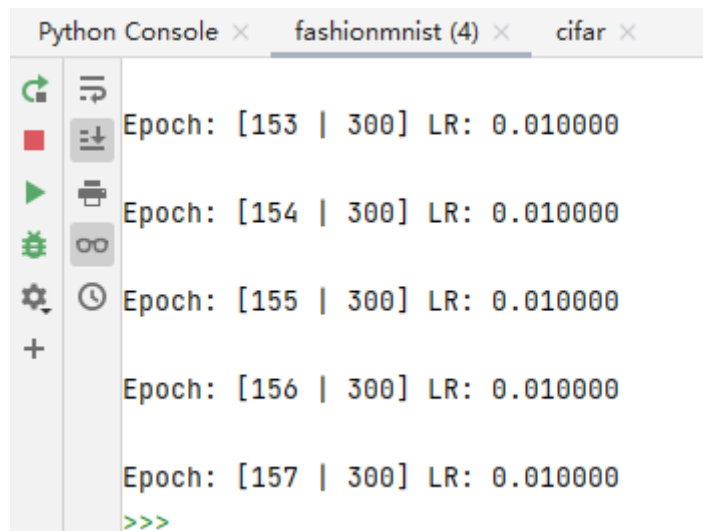


Figure 2. Examples of Random Erasing for object detection with Image-aware Random Erasing (**IRE**), Object-aware Random Erasing (**ORE**) and Image and object-aware Random Erasing (**I+ORE**).

## 工具验证

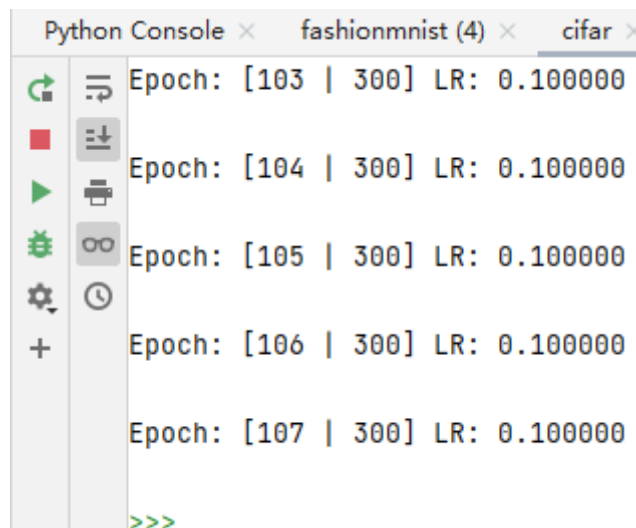
## 工具运行

- fashionmnist.py



```
Python Console × fashionmnist (4) × cifar ×
Epoch: [153 | 300] LR: 0.010000
Epoch: [154 | 300] LR: 0.010000
Epoch: [155 | 300] LR: 0.010000
Epoch: [156 | 300] LR: 0.010000
Epoch: [157 | 300] LR: 0.010000
>>>
```

- cifar.py



```
Python Console × fashionmnist (4) × cifar ×
Epoch: [103 | 300] LR: 0.100000
Epoch: [104 | 300] LR: 0.100000
Epoch: [105 | 300] LR: 0.100000
Epoch: [106 | 300] LR: 0.100000
Epoch: [107 | 300] LR: 0.100000
>>>
```

电脑配置太低，模型训练不完就已经超时了所以以下对比实验摘自原论文

## 对比实验

任务	数据集
Image Classfication	CIFAR-10 CIFAR-100 Fashion-MNIST

### 图像分类

#### 数据集上的实验

当 $p = 0.5$ ,  $s_l = 0.02$ ,  $s_h = 0.4$ ,  $r_1 = 1$ ,  $r_2 = 0.3$ ,各网络在CIFAR-10、CIFAR-100、Fashion-MNIST上的实验如下图所示：可以看出，random erasing都是有益于性能提升的。

Table 1: Test errors (%) with different architectures on CIFAR-10, CIFAR-100 and Fashion-MNIST. **RE**: Random Erasing.

Model	CIFAR-10		CIFAR-100		Fashion-MNIST	
	Baseline	RE	Baseline	RE	Baseline	RE
ResNet-20	$7.21 \pm 0.17$	$6.73 \pm 0.09$	$30.84 \pm 0.19$	$29.97 \pm 0.11$	$4.39 \pm 0.08$	$4.02 \pm 0.07$
ResNet-32	$6.41 \pm 0.06$	$5.66 \pm 0.10$	$28.50 \pm 0.37$	$27.18 \pm 0.32$	$4.16 \pm 0.13$	$3.80 \pm 0.05$
ResNet-44	$5.53 \pm 0.08$	$5.13 \pm 0.09$	$25.27 \pm 0.21$	$24.29 \pm 0.16$	$4.41 \pm 0.09$	$4.01 \pm 0.14$
ResNet-56	$5.31 \pm 0.07$	$4.89 \pm 0.07$	$24.82 \pm 0.27$	$23.69 \pm 0.33$	$4.39 \pm 0.10$	$4.13 \pm 0.42$
ResNet-110	$5.10 \pm 0.07$	$4.61 \pm 0.06$	$23.73 \pm 0.37$	$22.10 \pm 0.41$	$4.40 \pm 0.10$	$4.01 \pm 0.13$
ResNet-18-PreAct	$5.17 \pm 0.18$	$4.31 \pm 0.07$	$24.50 \pm 0.29$	$24.03 \pm 0.19$	$4.31 \pm 0.06$	$3.90 \pm 0.06$
WRN-28-10	$3.80 \pm 0.07$	<b><math>3.08 \pm 0.05</math></b>	<b><math>18.49 \pm 0.11</math></b>	<b><math>17.73 \pm 0.15</math></b>	<b><math>4.01 \pm 0.10</math></b>	<b><math>3.65 \pm 0.03</math></b>
ResNeXt-8-64	<b><math>3.54 \pm 0.04</math></b>	$3.24 \pm 0.03$	$19.27 \pm 0.30$	$18.84 \pm 0.18$	$4.02 \pm 0.05$	$3.79 \pm 0.06$

## 与dropout和随机添加噪声的实验比较

- 随机添加噪声的方案为用随机值改变像素值，改变一定比例的像素。对比结果如下图所示。当 $\lambda = 0.4$ 时，改变的像素数量与random erasing相当。

Table 4: Comparing Random Erasing with dropout and random noise on CIFAR-10 with using ResNet18 (pre-act).

Method	Test error (%)	Method	Test error (%)
Baseline	$5.17 \pm 0.18$	Baseline	$5.17 \pm 0.18$
Ours	<b><math>4.31 \pm 0.07</math></b>	Ours	<b><math>4.31 \pm 0.07</math></b>
Dropout	Test error (%)	Noise	Test error (%)
$\lambda_1 = 0.001$	$5.37 \pm 0.12$	$\lambda_2 = 0.01$	$5.38 \pm 0.07$
$\lambda_1 = 0.005$	$5.48 \pm 0.15$	$\lambda_2 = 0.05$	$5.79 \pm 0.14$
$\lambda_1 = 0.01$	$5.89 \pm 0.14$	$\lambda_2 = 0.1$	$6.13 \pm 0.12$
$\lambda_1 = 0.05$	$6.23 \pm 0.11$	$\lambda_2 = 0.2$	$6.25 \pm 0.09$
$\lambda_1 = 0.1$	$6.38 \pm 0.18$	$\lambda_2 = 0.4$	$6.52 \pm 0.12$

## 和其他数据增强方法的比较

- flip+crop比单独使用的好；三种方法结合起来效果更好

Table 5: Test errors (%) with different data augmentation methods on CIFAR-10 based on ResNet18 (pre-act). **RF**: Random flipping, **RC**: Random cropping, **RE**: Random Erasing.

Method	RF	RC	RE	Test errors (%)
Baseline				$11.31 \pm 0.18$
	✓			$8.30 \pm 0.17$
		✓		$6.33 \pm 0.15$
			✓	$10.13 \pm 0.14$
	✓	✓		$5.17 \pm 0.18$
	✓		✓	$7.19 \pm 0.10$
		✓	✓	$5.21 \pm 0.14$
	✓	✓	✓	<b><math>4.31 \pm 0.07</math></b>

## 对遮挡的鲁棒性实验

- 说明无随机擦除和有随机擦除随着遮挡区域的级别（占整个图像的面积）增大,后者drops的更慢,更低。即随机擦除可以提升CNNs对遮挡的鲁棒性。

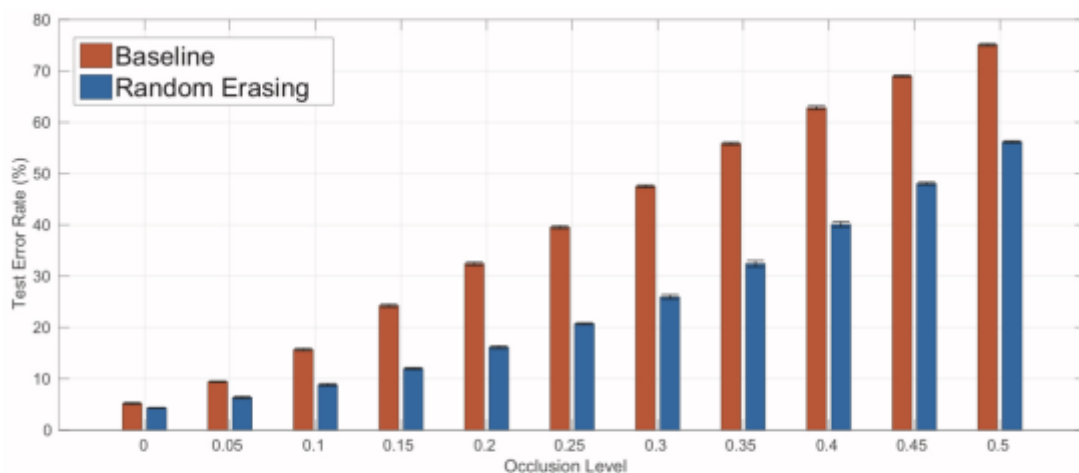


Figure 3: Test errors (%) under different levels of occlusion on CIFAR-10 based on ResNet18 (pre-act). The model trained with Random Erasing is more robust to occlusion.

## 技术难点

Random Erasing核心算法思想比较清楚，实现起来不难。

难点在于怎样将它适配进各种CNN训练模型中，设计出合理的CNN架构，以及如何设置对比检测实验并各种训练参数的选取。此外，还有如何将随机擦除和其他数据增强工具联合使用。

该工具还可通过进一步的适配用于其他 CNN 识别任务，例如图像检索、人脸识别和细粒度分类。

## References

<https://github.com/zhunzhong07/Random-Erasing>

```
@inproceedings{zhong2020random,
  title={Random Erasing Data Augmentation},
  author={Zhong, Zhun and Zheng, Liang and Kang, Guoliang and Li, Shaozi and Yang, Yi},
  booktitle={Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)},
  year={2020}
}
```

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In NIPS, 2012. 1, 2

L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus. Regularization of neural networks using dropconnect. In ICML, 2013

J. Ba and B. Frey. Adaptive dropout for training deep neural networks. In NIPS, 2013

M. D. Zeiler and R. Fergus. Stochastic pooling for regularization of deep convolutional neural networks. In ICLR, 2013

L. Xie, J. Wang, Z. Wei, M. Wang, and Q. Tian. Disturblabel: Regularizing cnn on the loss layer. In CVPR, 2016

