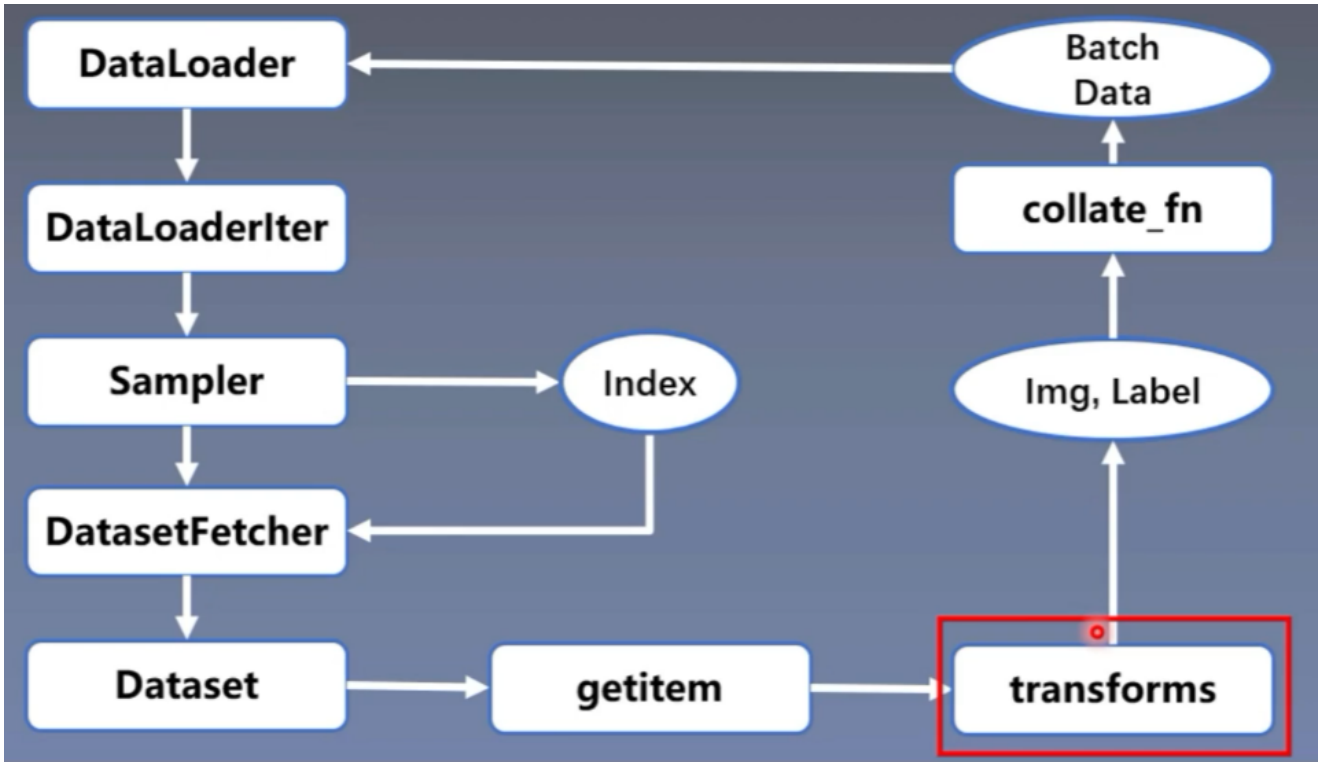
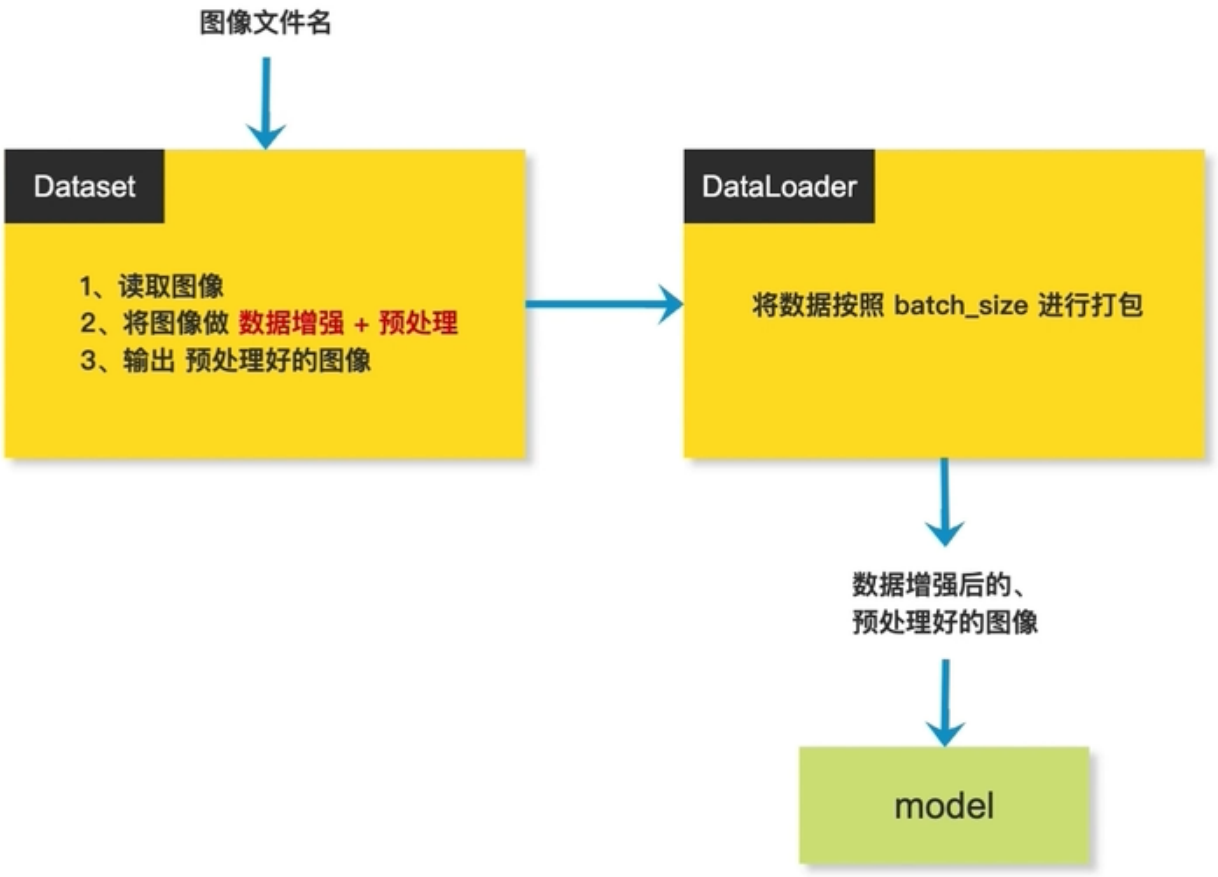
# TRANSFORM

## 一、内容

torchvision.transforms包含多种数据预处理方法：数据中心化、数据标准化、缩放、裁剪、旋转、翻转、填充、噪声添加、灰度变换、线性变换、仿射变换、亮度、饱和度及对比度变换等。






上图是transforms处理数据时所处的位置。主要在Dataset中对图像进行预处理。

## 1、将图像转换成tensor 的数据格式

## 2、将图像的 像素值范围 由 0～255 转换为 0～1

## 3、(height, width, channel) =====>>> (channel, height, width)

## 4、归一化图像

- 归一化可以优化算法的收敛速度和性能， 和 BN 层的作用差不多
- 归一化处理还可以消除不同图像之间的亮度和颜色差异，提高模型的鲁棒性。

transforms主要完成以上四个部骤。

# 二、代码

## 1、Compose

transforms.Compose是PyTorch中torchvision.transforms模块的一个类，它的主要作用是将多个图像变换操作组合在一起。这个类的构造很简单，它接受一个由多个Transform对象组成的列表作为参数，然后按照列表中的顺序依次对图像进行变换。

```
import numpy as np
from PIL import Image
from torchvision import transforms


path = "fashion_mnist_images/test/3/0000.png"
img = Image.open(path)
img = img.convert('RGB')


trans = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
     ]
)

print(str(trans))
```

```
Compose(
    ToTensor()
    Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
)
```

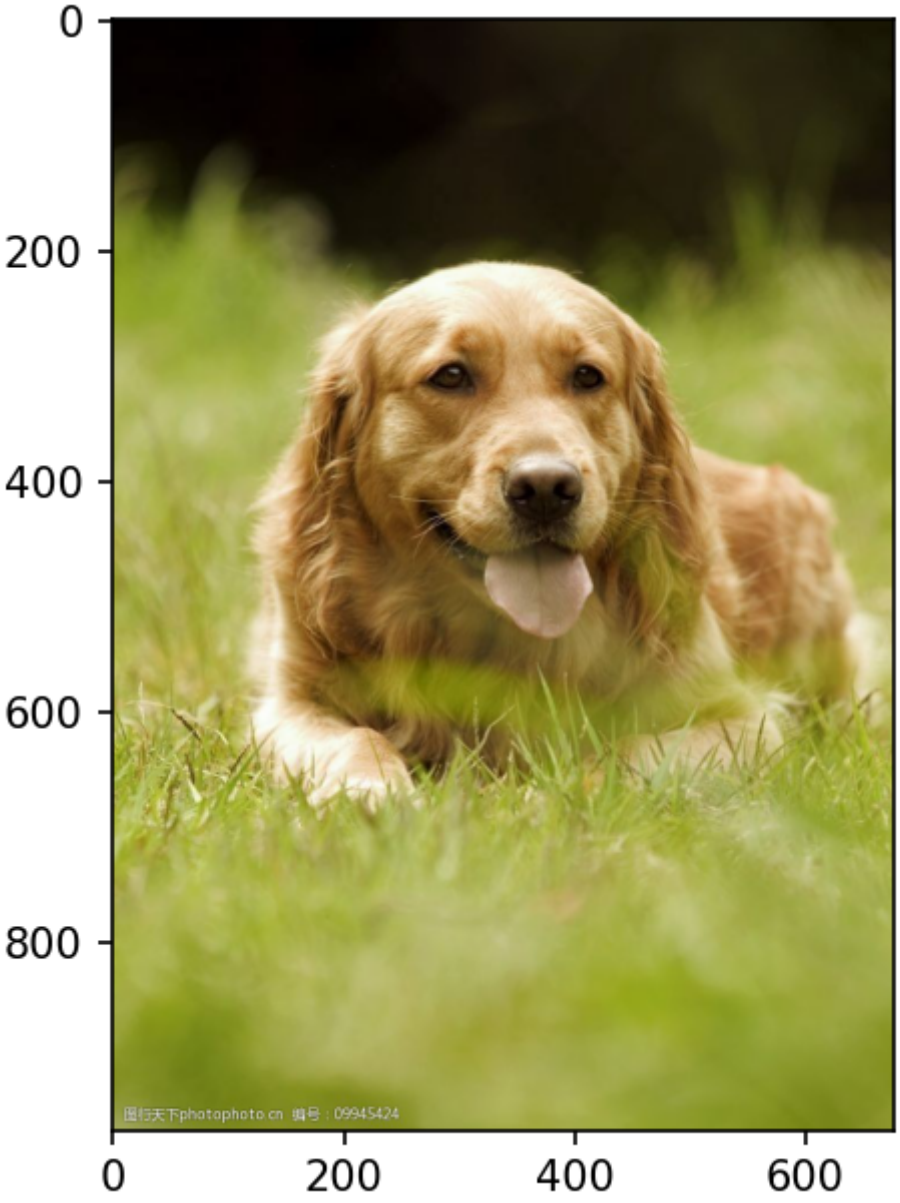
> 上图是输出的Compose结构。这些参数mean=[0.485, 0.456, 0.406]和std=[0.229, 0.224, 0.225]是ImageNet数据集的均值和标准差1。使用ImageNet的均值和标准差是一种常见的做法，因为它们是根据数百万张图像计算得出的。如果要在自己的数据集上从头开始训练，则可以计算新的均值和标准差。否则，建议使用ImageNet预训练模型及其平均值和标准差。

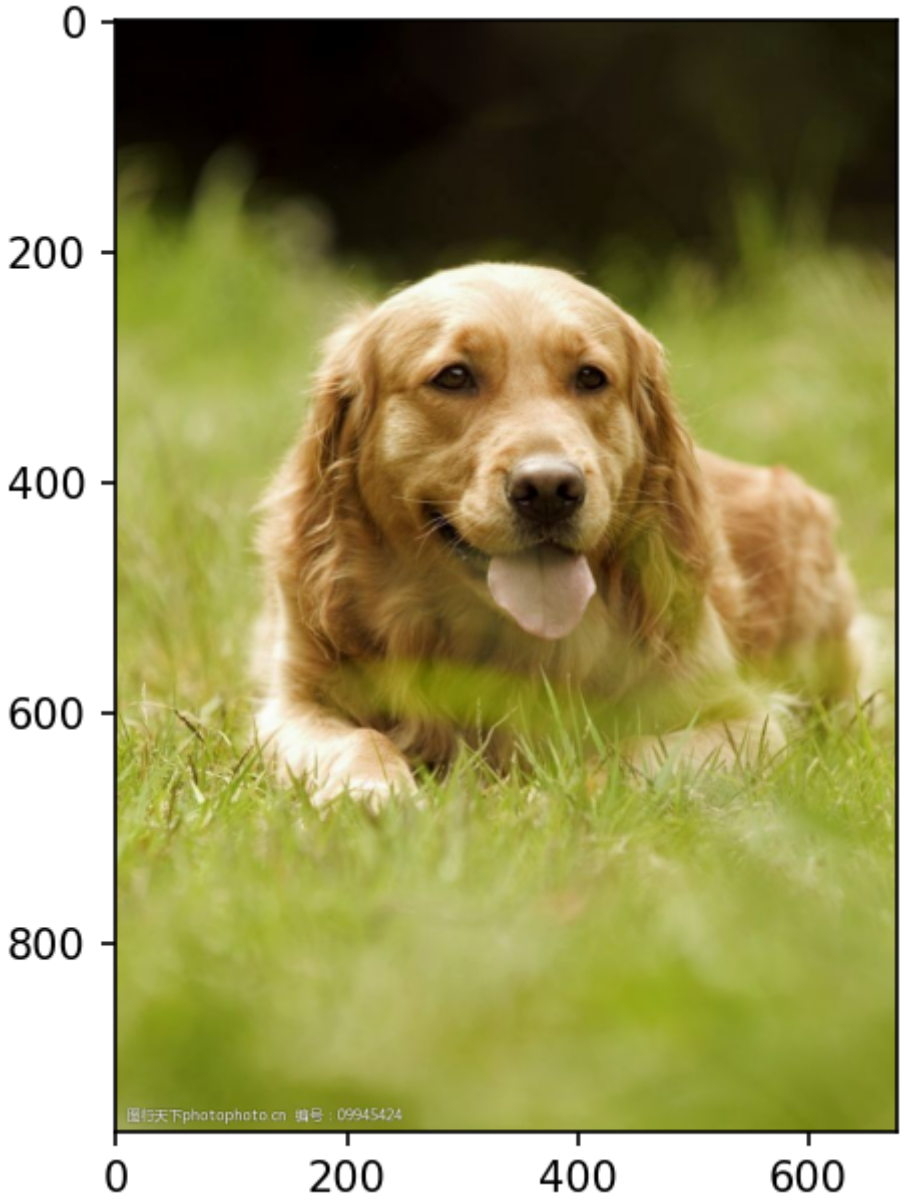
## 2、transforms_inverse

这个函数是自己实现的，主要是为了将tensor数据反向转化成PIL格式数据，方便打印，查看各种变换的效果。

```
path = "dog.jpg"
```

```
img = Image.open(path)
plt.imshow(img)
plt.show()

trans = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
     ]
)


def transforms_inverse(img, trans):
    # 反向transforms
    if "Normalize" in str(trans): # 如果使用了Normalize
        # trans.transforms返回Compose中的内容
        # trans返回Compose对象
        # filter()它接受一个函数和一个序列作为参数，然后返回一个迭代器，其中包含序列中所有使函数返回值为True的元素。
        # list()函数将返回的迭代器转换为列表
        norm_transform = list(filter(lambda x: isinstance(x, transforms.Normalize), trans.transforms))
        # 获得归一化时使用的均值和标准差
        mean = torch.tensor(norm_transform[0].mean)
        std = torch.tensor(norm_transform[0].std)
        # mul_和add_是原位操作
        # std[:, None, None]是一种用于增加张量维度的技巧
        # 形状为(3,)的一维张量转换成形状为(3, 1, 1)的三维张量，其中的3是通道数
        img.mul_(std[:, None, None]).add_(mean[:, None, None])

    img = img.transpose(0,2).transpose(0,1) # CHW => HWC
    # tensor => np
    img = np.array(img) * 255

    # 使用Image.fromarray()函数将数组转换为PIL图像
    if img.shape[2] == 3:# channel==3
        img = Image.fromarray( img.astype('uint8') ).convert('RGB')
    elif img.shape[2] == 1:    # channel==1
        # 当channel==1，必须用squeeze()函数移除多余的维度
        img = Image.fromarray(img.astype('uint8').squeeze())

    return img


img_tensor = trans(img)
img_PIL = transforms_inverse(img_tensor,trans)
plt.imshow(img_PIL)
plt.show()
```


0
200
400
600
800
0
200
400
600
图行天下photophoto.cn 编号：09945424

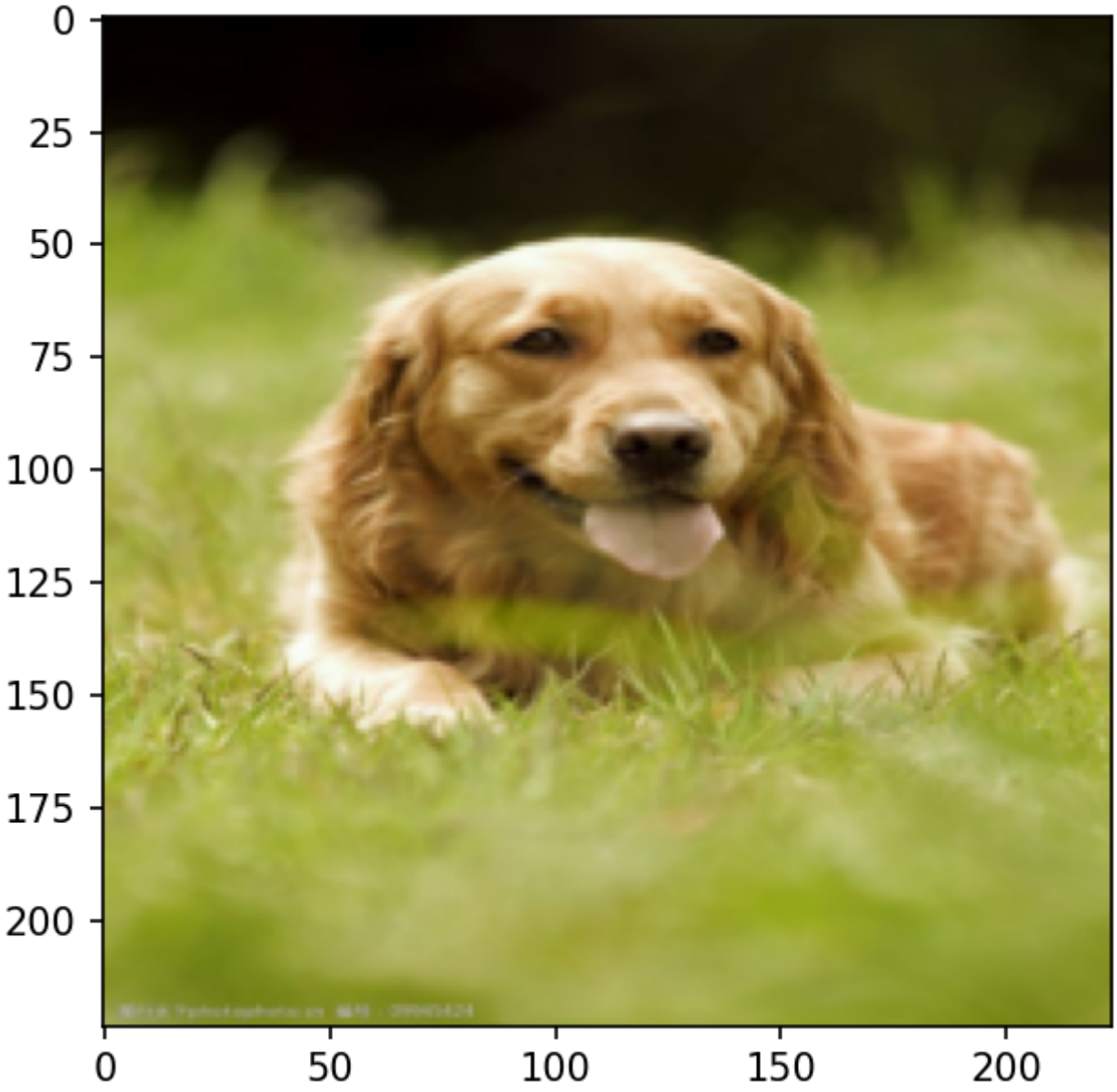



前一张是原图，后一张是反向转换后的图，与原图没有区别。

## 3、Resize

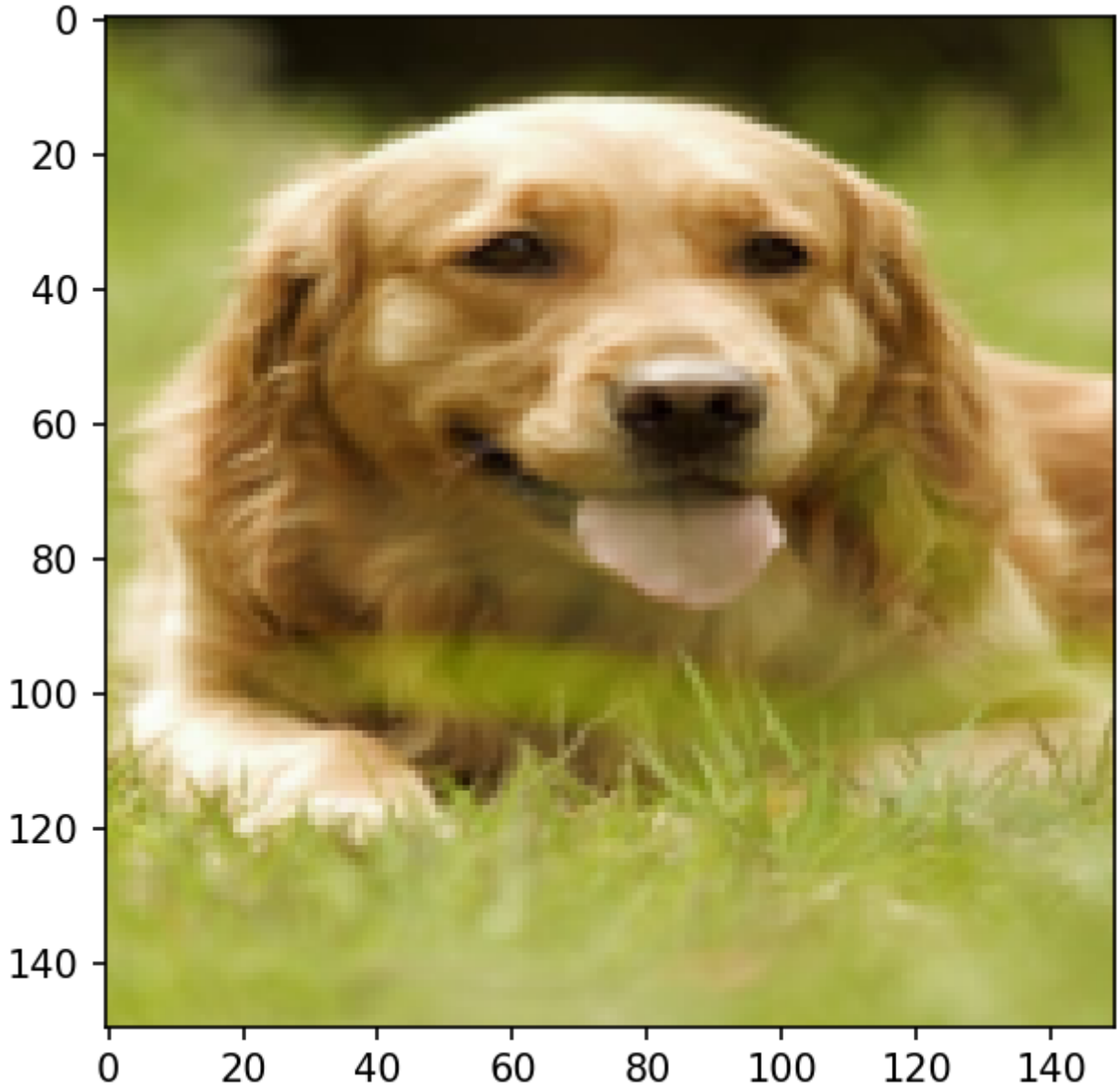
```
trans = transforms.Compose(
    [
      transforms.Resize((224,224)),# 必须是元组(224,224)
      transforms.ToTensor(),
      transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
      ]
)
```

必须是元组(224,224)

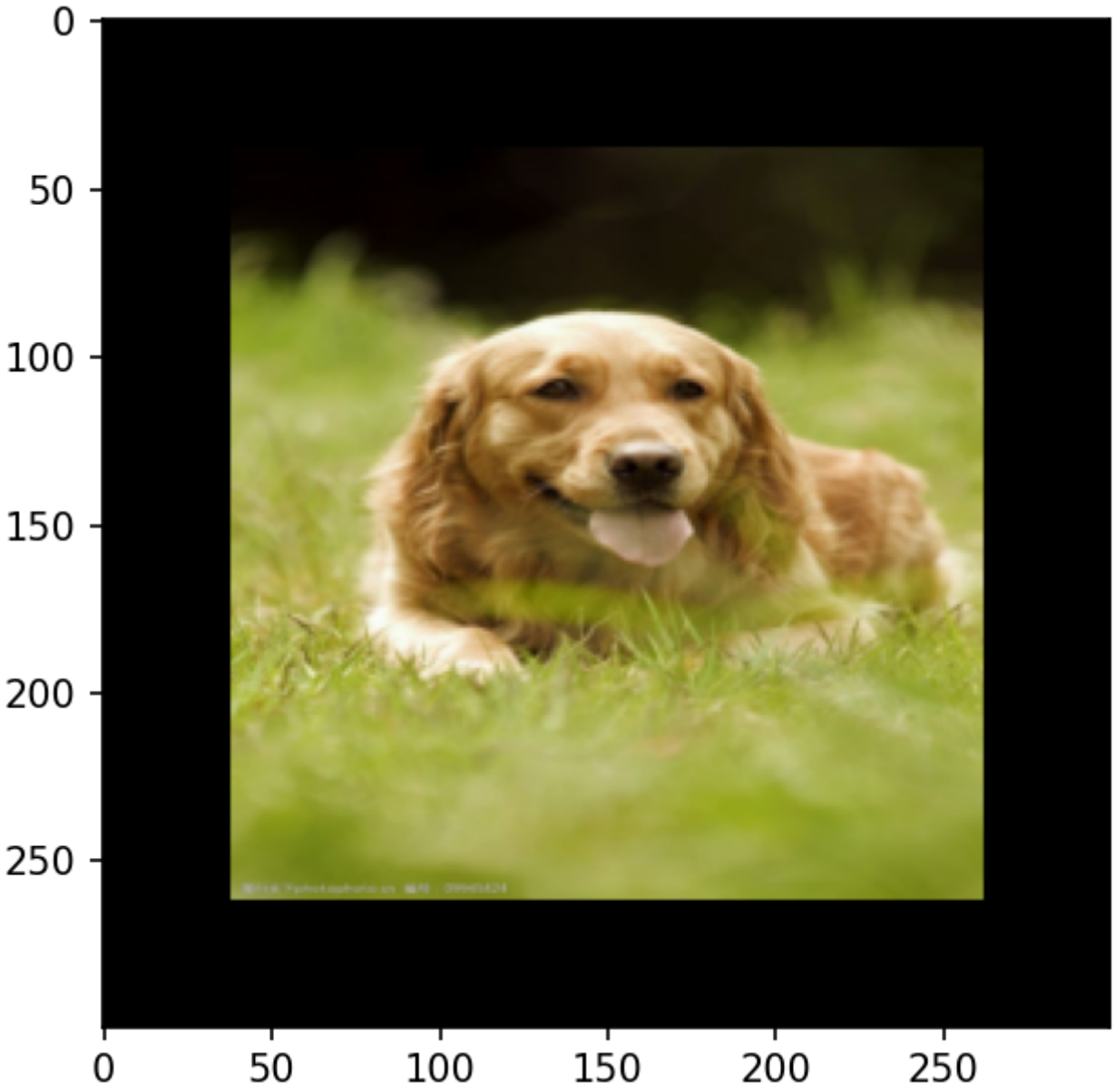



## 4、CenterCrop

```
trans = transforms.Compose(
    [
      transforms.Resize((224,224)),# 必须是元组(224,224)
      transforms.CenterCrop(150),

      transforms.ToTensor(),
      transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
      ]
)
```




```
trans = transforms.Compose(
    [
      transforms.Resize((224,224)),# 必须是元组(224,224)
      transforms.CenterCrop(300),

      transforms.ToTensor(),
      transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
      ]
)
```




当transforms.CenterCrop(300)中心的大小大于原图则会在周围补0

## 5、RandomCrop

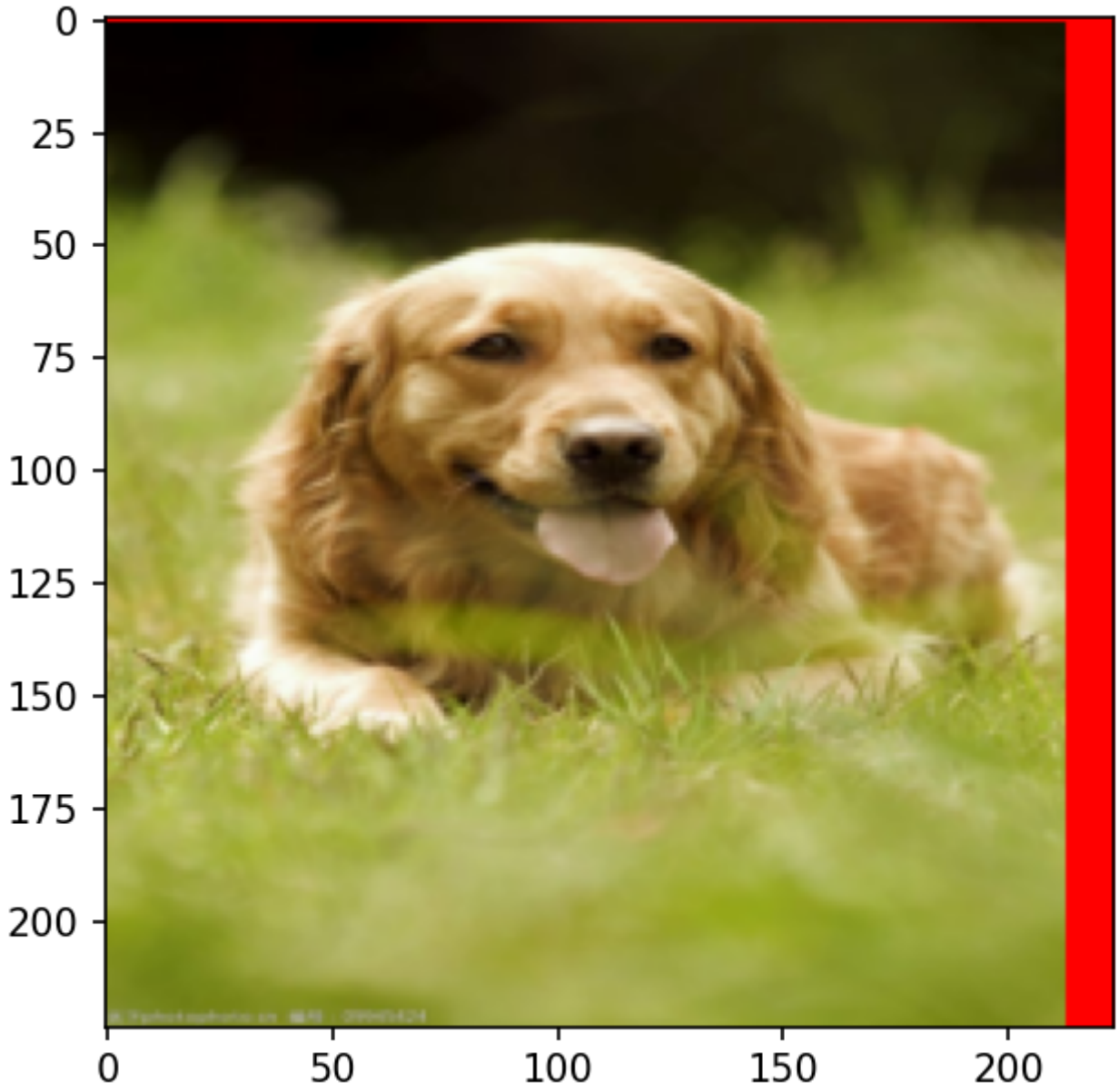
```
transforms.RandomCrop(size,
                      padding=None,
                      pad_if_needed=False,
                      fill=0,
                      padding_mode='constant')
```

**从图片中随机裁剪出尺寸为size的图片**

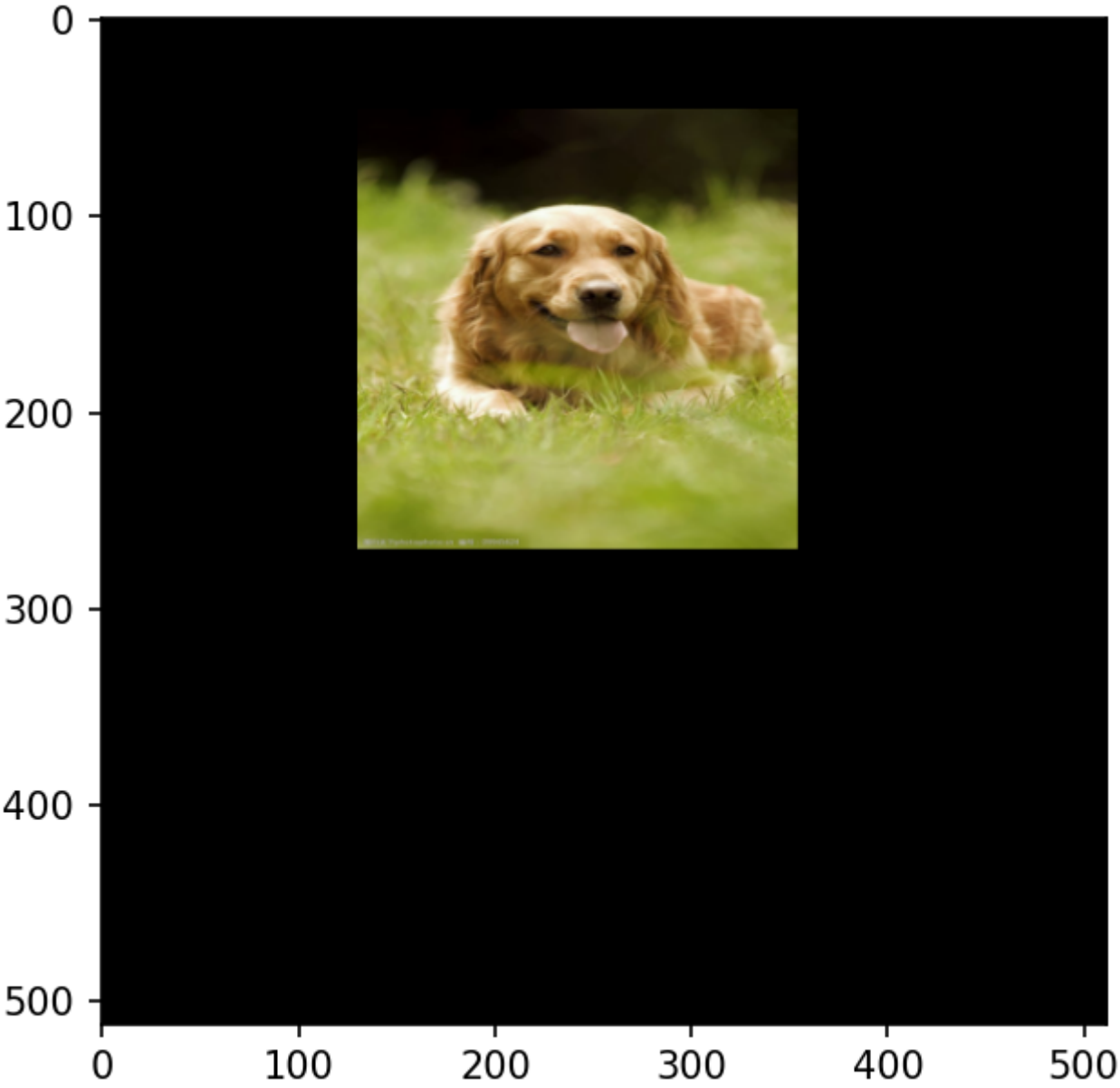
- size:所需裁剪图片尺寸

- padding :设置填充大小
    - 当为a时，上下左右均填充a个像素
    - 当为(a, b)时,上下填充b个像素，左右填充a个像素
    - 当为(a, b, c, d)时,左，上，右,下分别填充a, b, c, d
- pad_if_need:若图像小于设定size，则padding
- padding_mode :填充模式，有4种模式
    - constant :像素值由fill设定
    - edge :像素值由图像边缘像素决定
    - reflect :镜像填充,最后一个像素不镜像
    - symmetric :镜像填充,最后一个像素镜像
- fill : constant时,设置填充的像素值

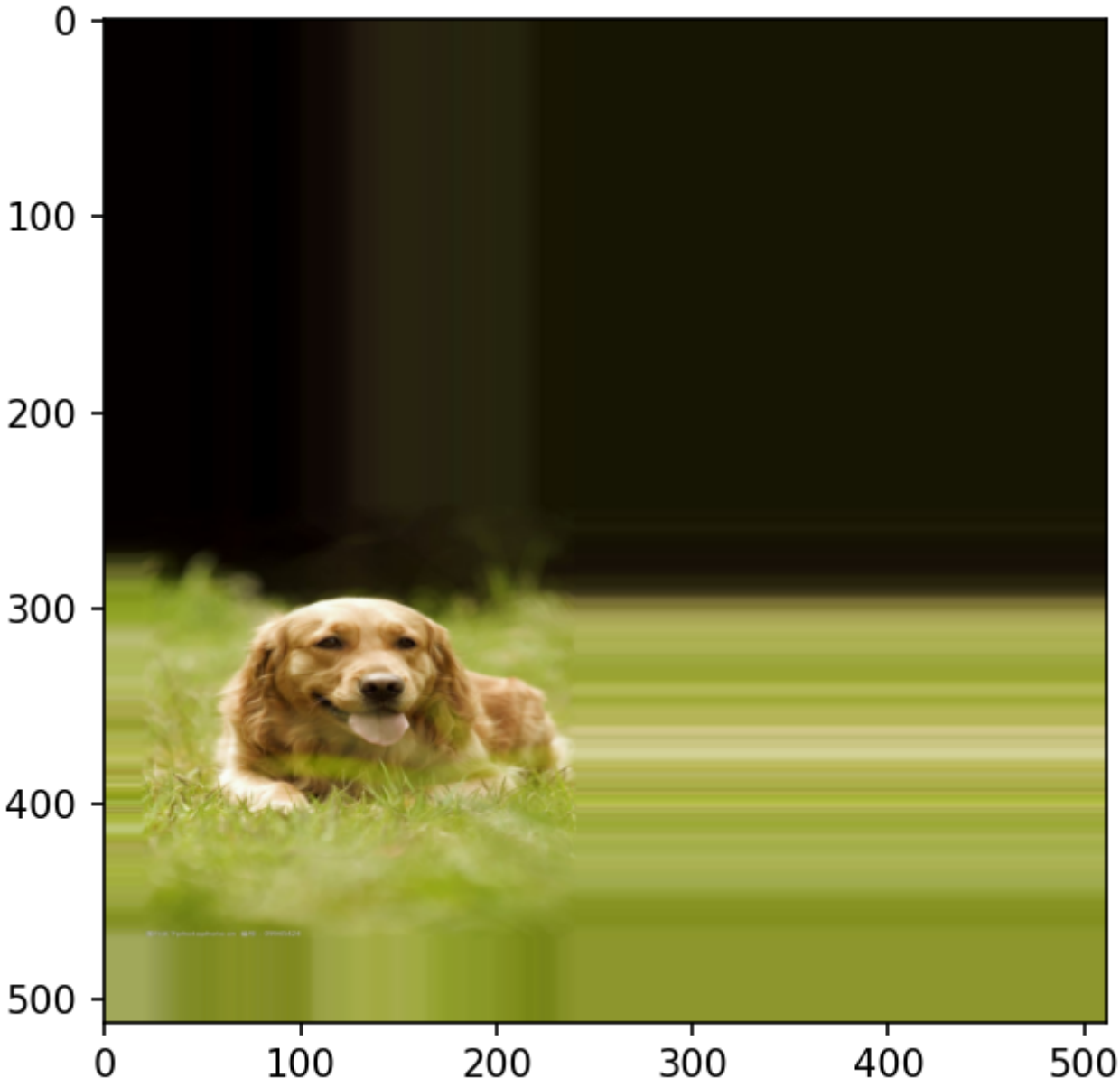
```
trans = transforms.Compose(
    [
      transforms.Resize((224,224)),# 必须是元组(224,224)
      transforms.RandomCrop(224,padding=26,fill=(255,0,0)),

      transforms.ToTensor(),
      transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
      ]
)
```



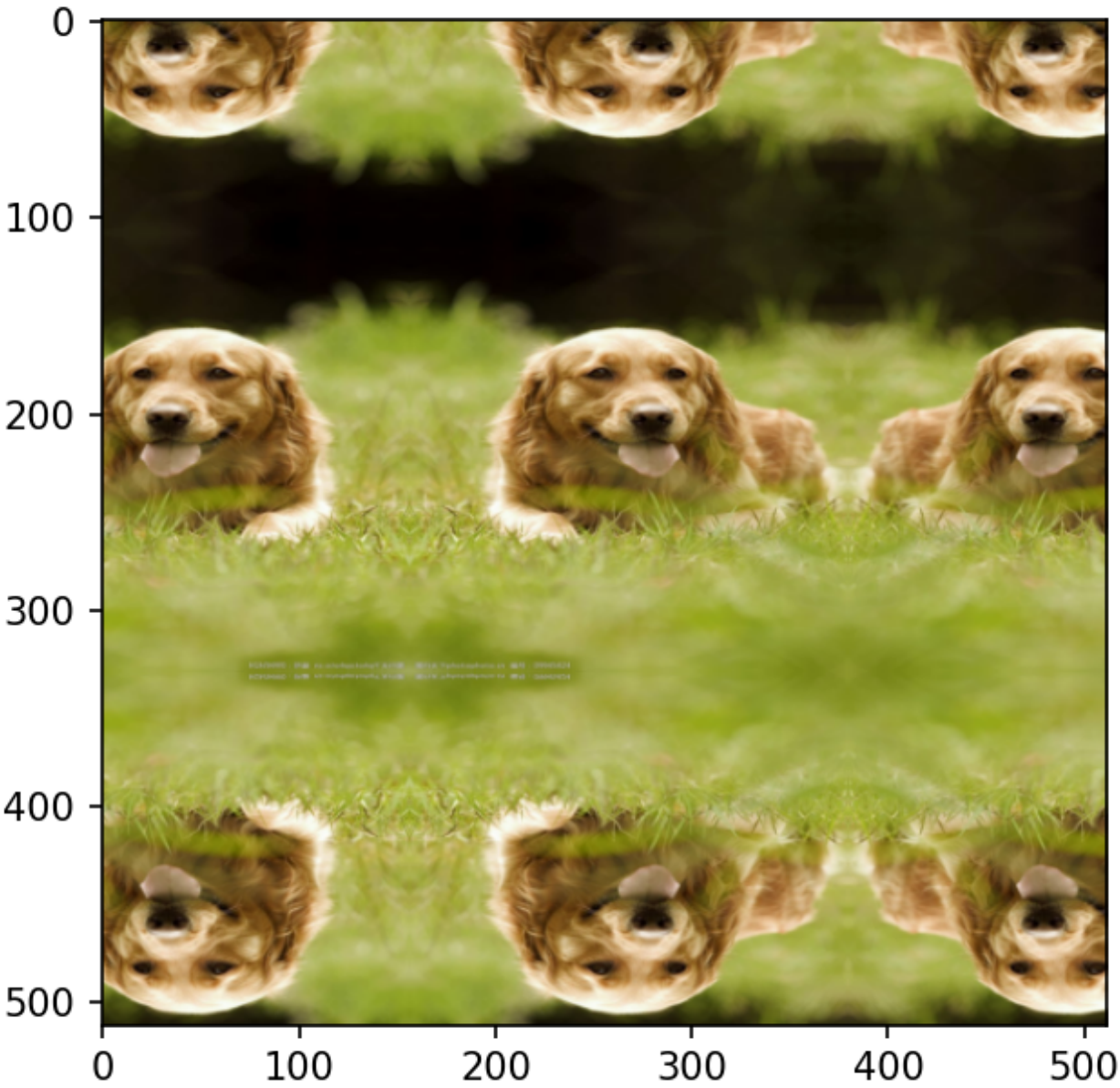

```
transforms.RandomCrop(512, pad_if_needed=True)
```




```
transforms.RandomCrop(512,pad_if_needed=True,padding_mode='edge')
```




```
transforms.RandomCrop(512, pad_if_needed=True, padding_mode='reflect')
```




## 6、RandomResizedCrop

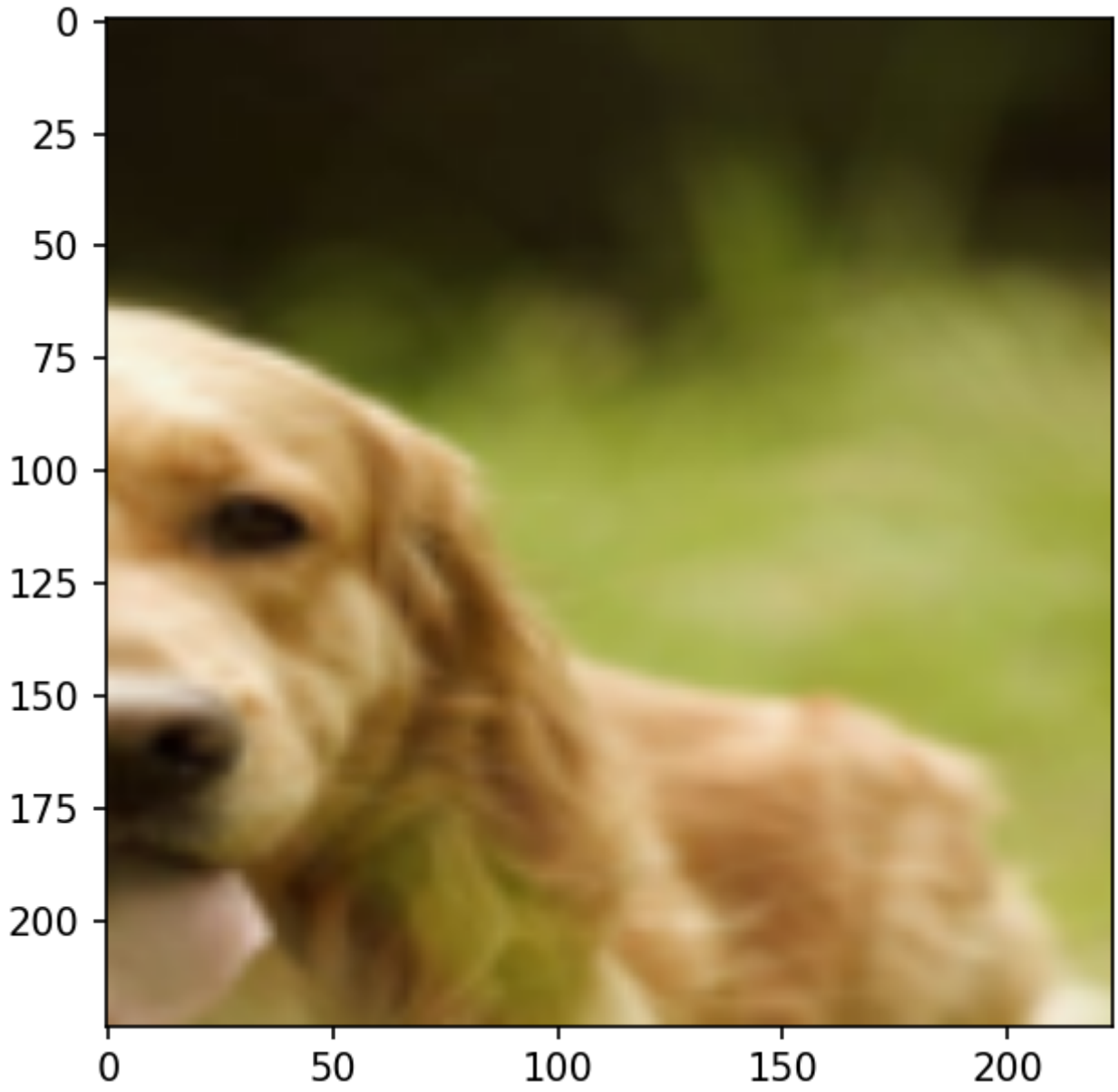
```
RandomResizedCrop(size,
                  scale=(0.08, 1.0),
                  ratio=(3/4, 4/3),
                  interpolation)
```

**随机大小、长宽比裁剪图片**

- size： 所需裁剪图片尺寸
- scale：随机裁剪面积比例，默认区间(0.08, 1)
- ratio： 随机长宽比，默认区间(3/4, 4/3)
- interpolation： 插值方法
    - PIL.Image.NEAREST
    - PIL.Image.BILINEAR

- PIL.Image.BICUBIC

```
trans = transforms.Compose(
    [
      transforms.Resize((224,224)),# 必须是元组(224,224)
      transforms.RandomResizedCrop(224,scale=(0.2,0.2)),

      transforms.ToTensor(),
      transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
      ]
)
```




随机裁剪20%的面积

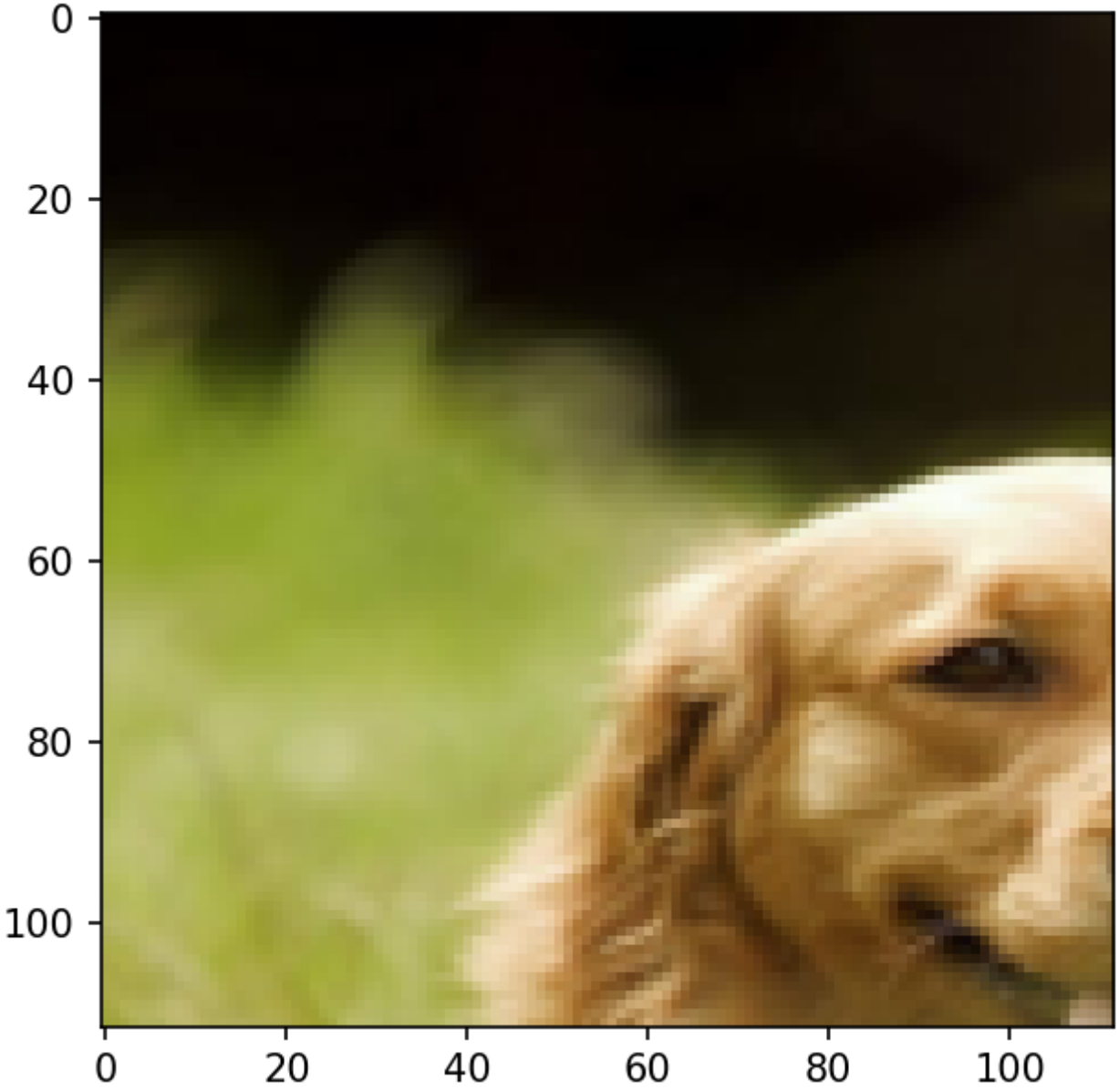
## 7、FiveCrop & TenCrop

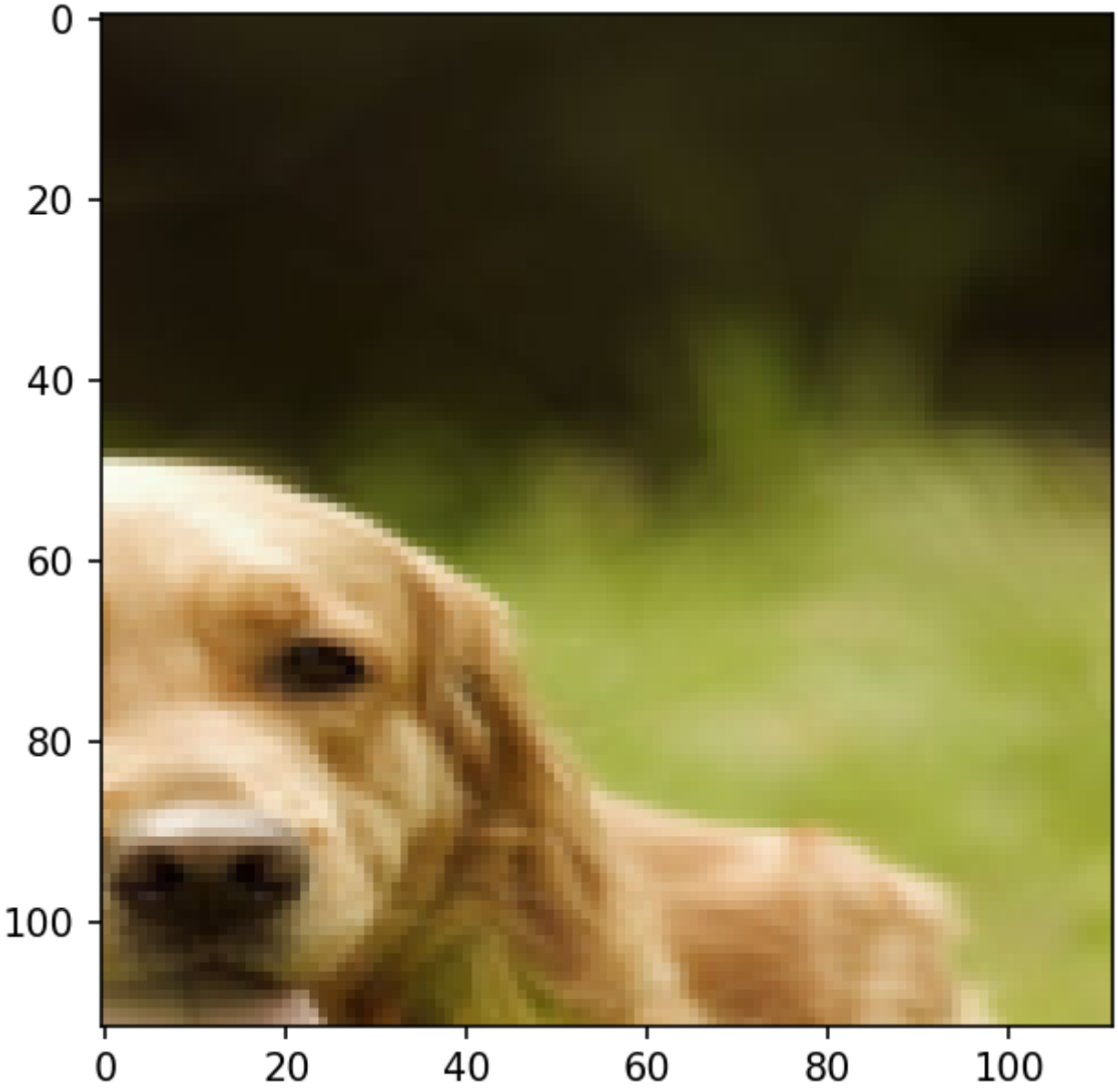
```
transforms.FiveCrop(size)

transforms.TenCrop(size,
                   vertical_flip=False)
```

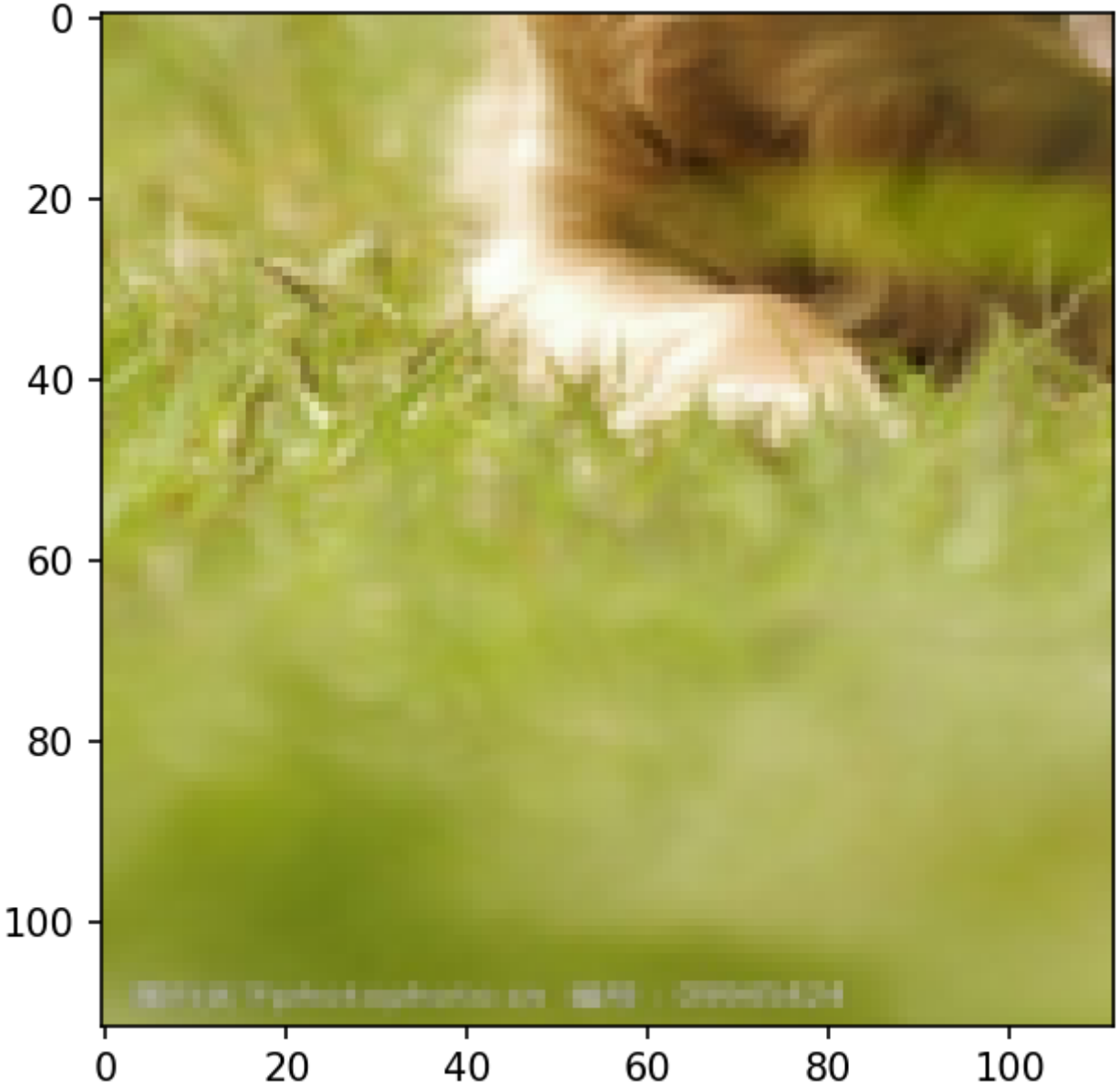
**在图像的上下左右以及中心裁剪出尺 寸为size的5张图片，TenCrop对这5张图片 进行水平或者垂直镜像获得10张图片**

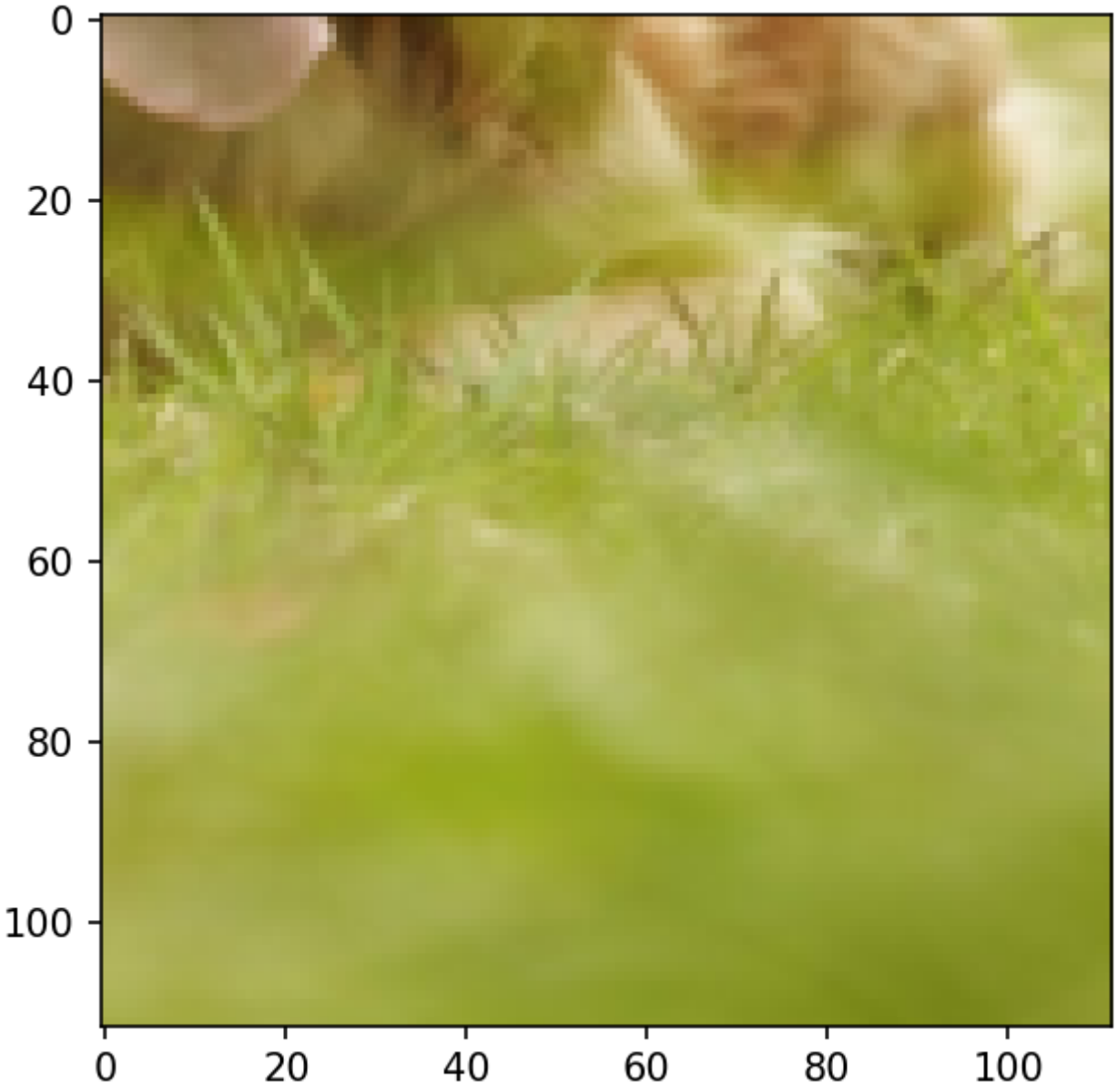
- size :所需裁剪图片尺寸
- vertical_flip :是否垂直翻转

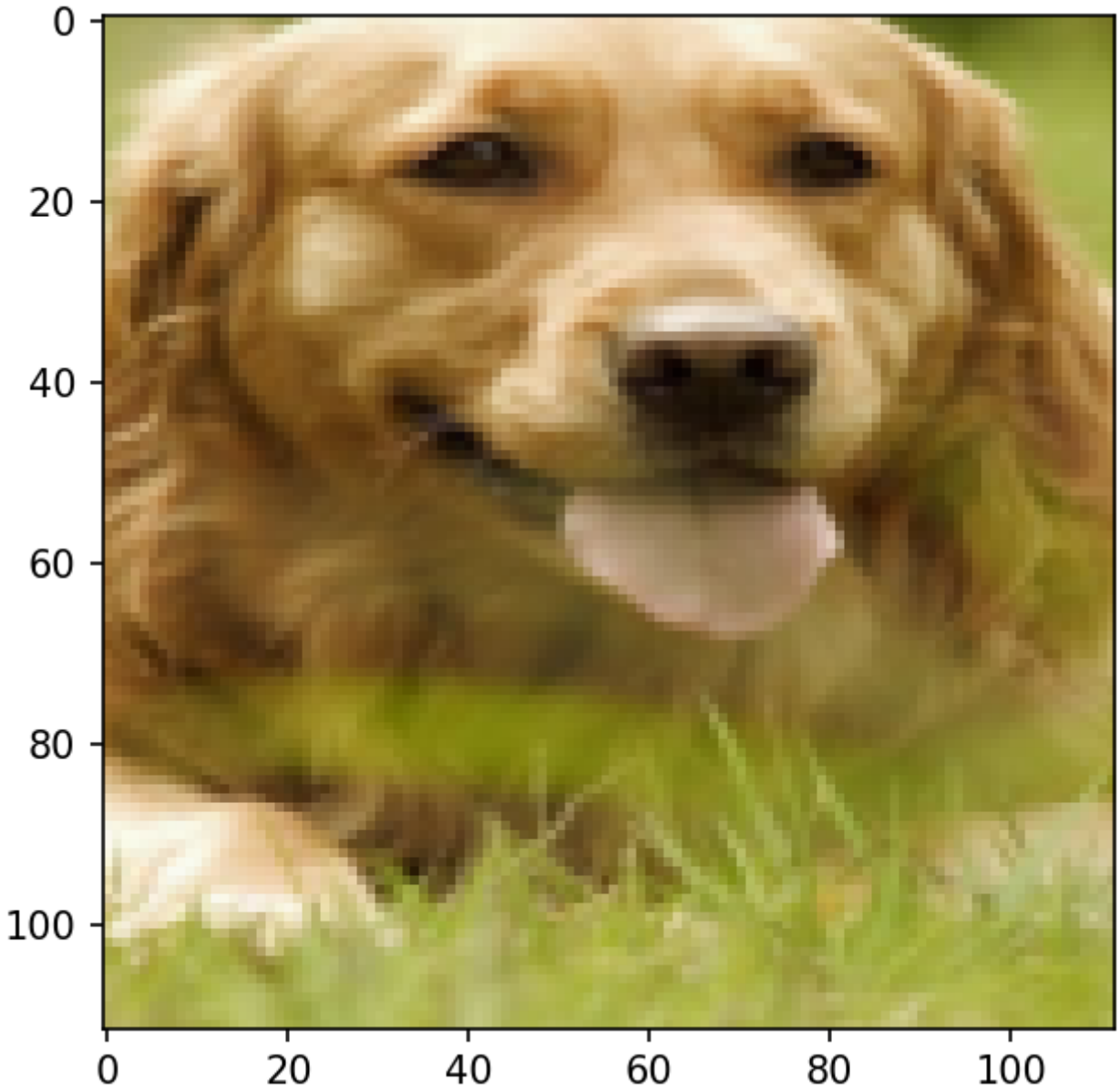
```
trans = transforms.Compose(
    [
      transforms.Resize((224,224)),# 必须是元组(224,224)
      transforms.FiveCrop(112), # 将获得5张图，是一个tuple元组，要拆开
      transforms.Lambda( lambda crops: torch.stack( [ (transforms.ToTensor()(each)) for
each in crops ] ) ),

      # transforms.ToTensor(),
      # transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
      ]
)
img_tensor = trans(img)

Ncrops, C, H, M = img_tensor.shape
for i in range(Ncrops):
     # transforms.FiveCrop(112),  # 将获得5张图,遍历每张图
     img_PIL = transforms_inverse(img_tensor[i],trans)

     plt.imshow(img_PIL)
     plt.show()
```


0
20
40
60
80
100
0
20
40
60
80
100


0
20
40
60
80
100
0
20
40
60
80
100

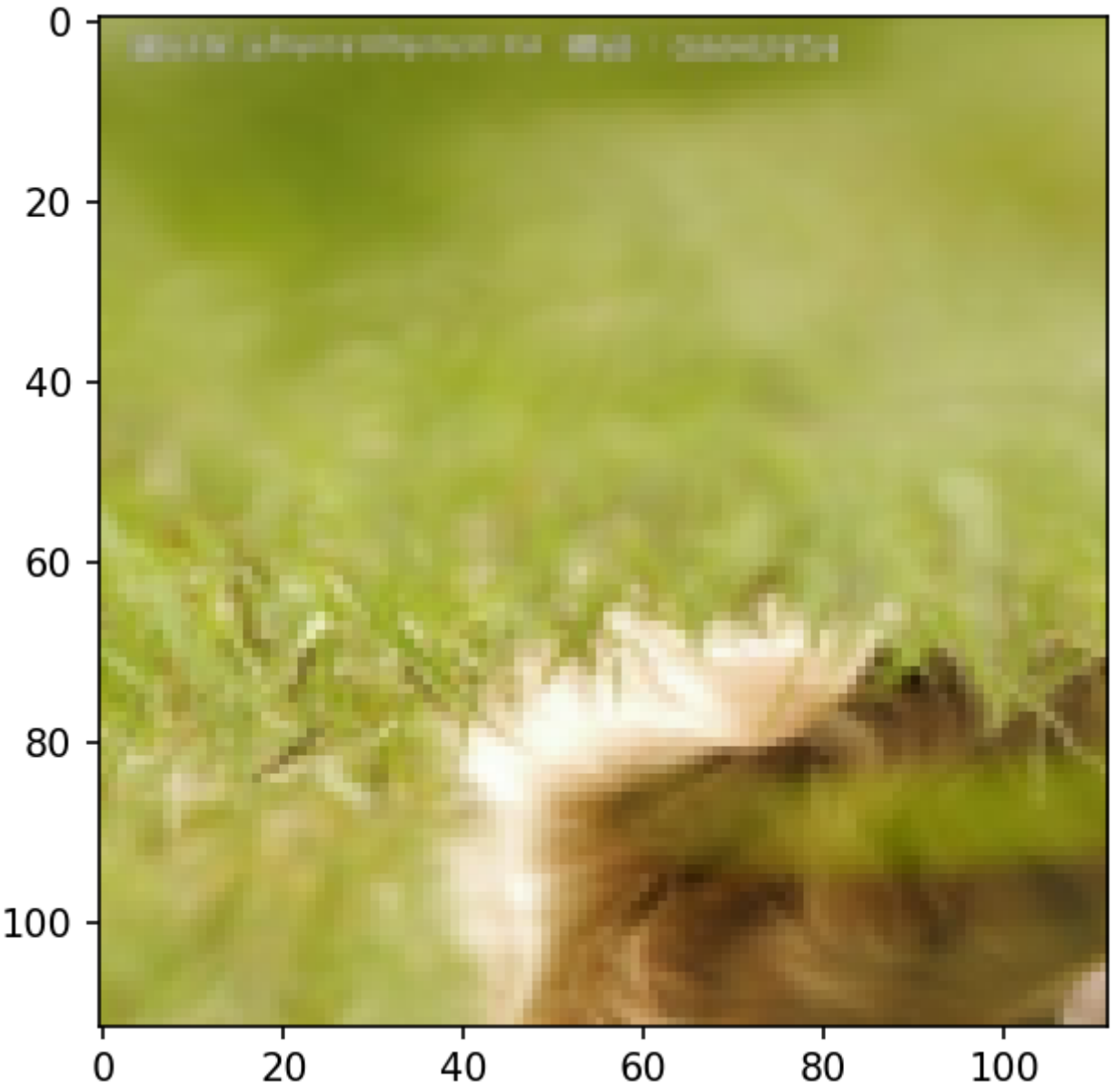

0
20
40
60
80
100
0
20
40
60
80
100

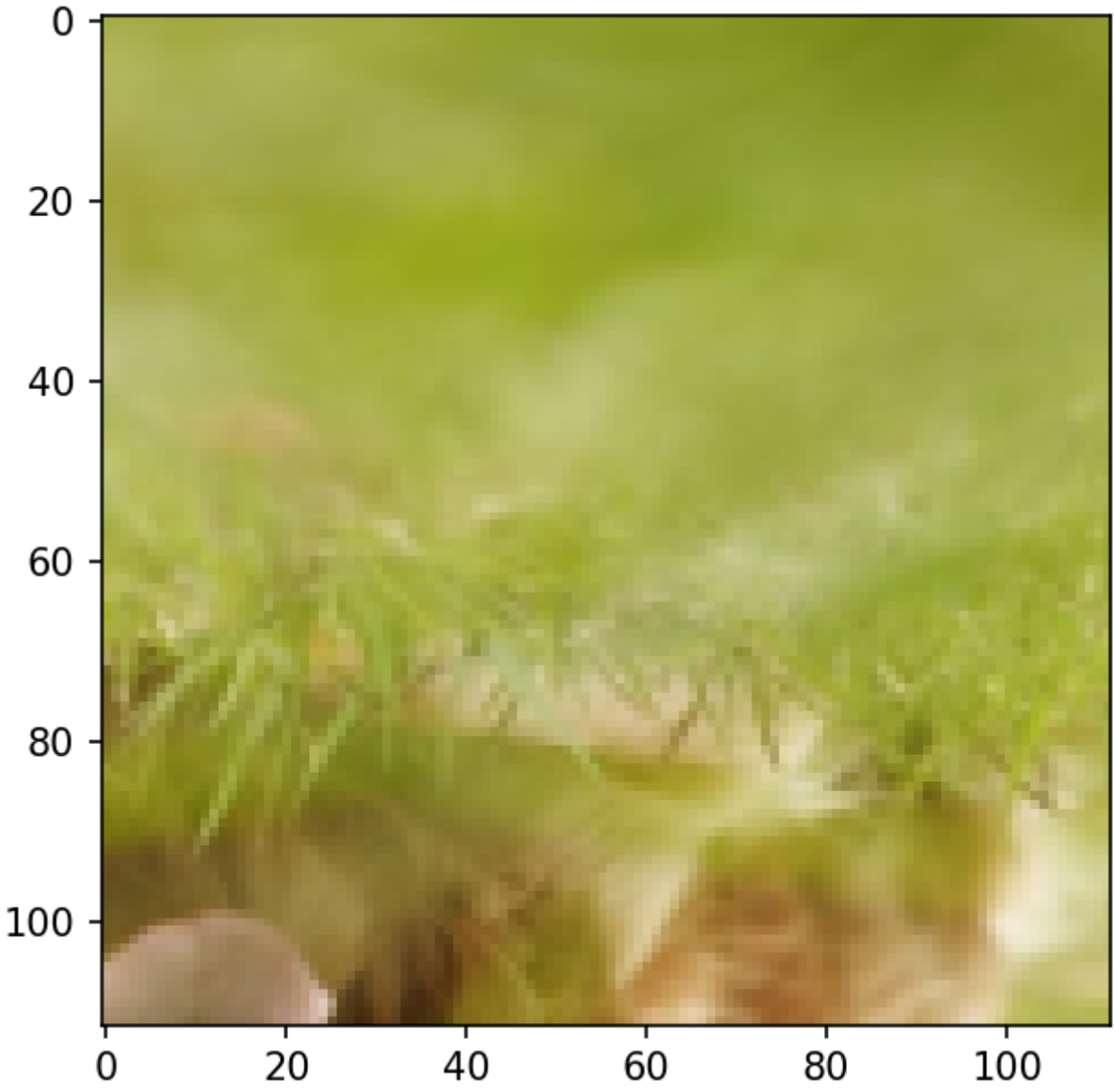

0
20
40
60
80
100
0
20
40
60
80
100

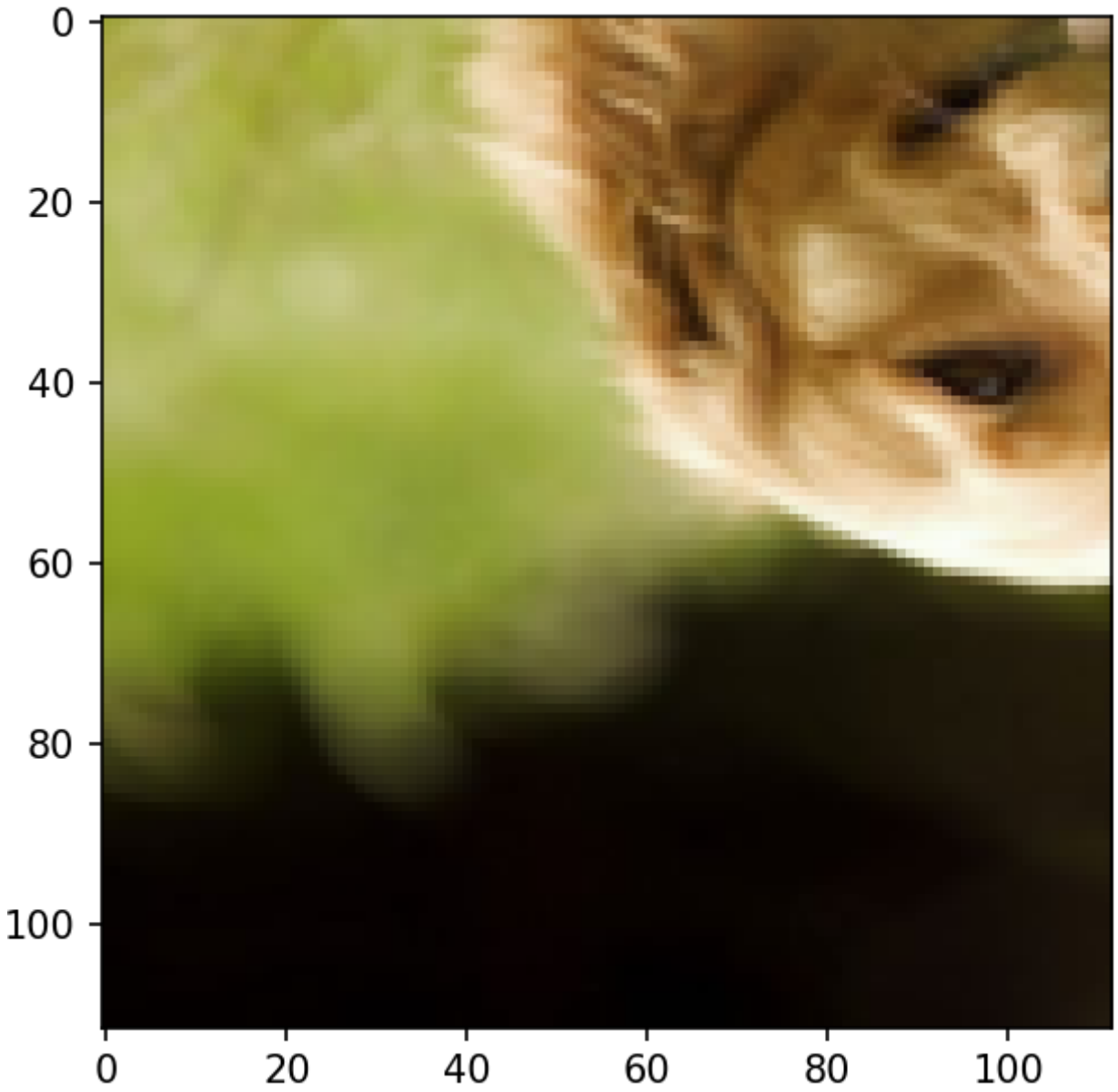


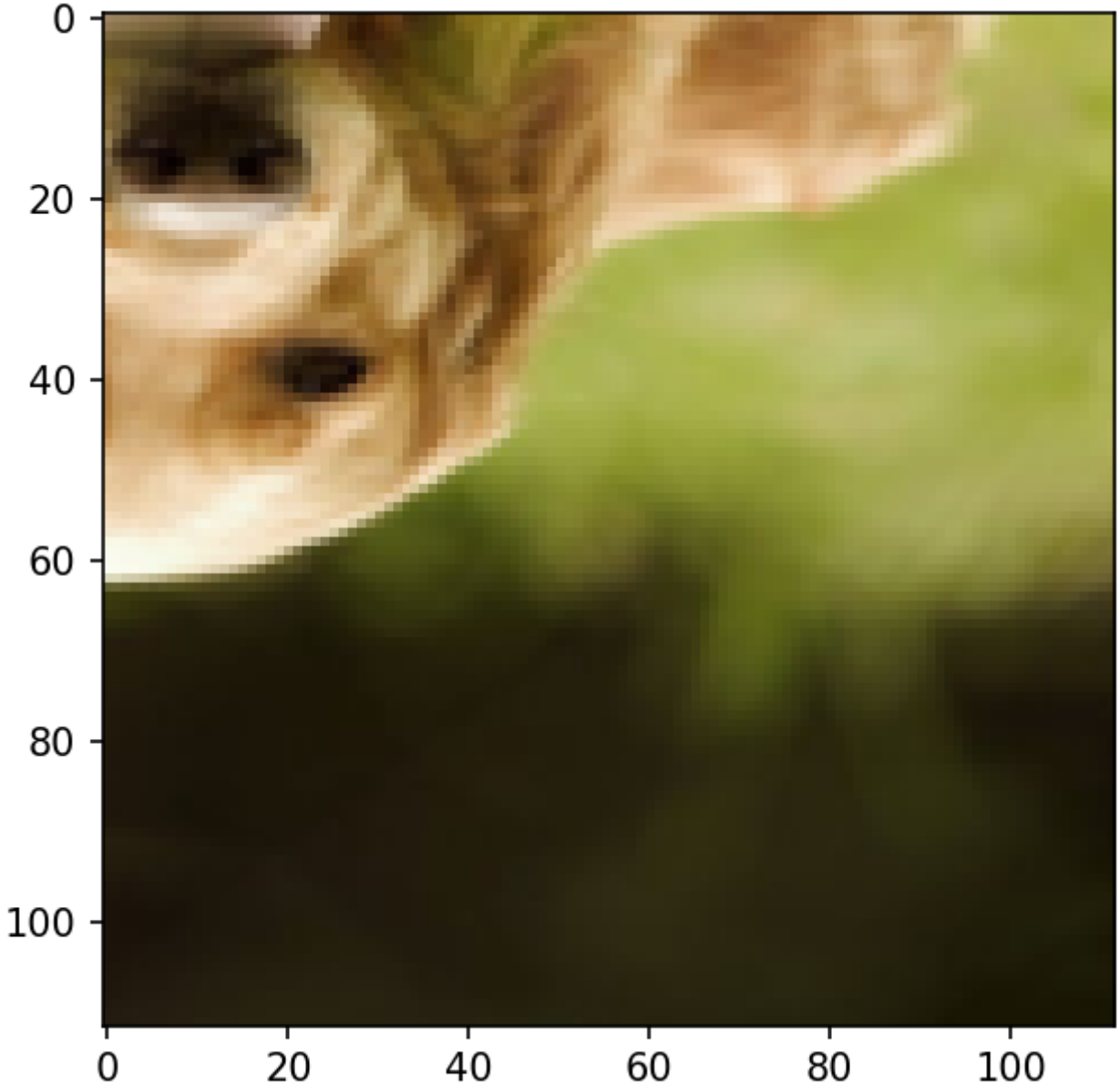

获得上左，上右，下左，下右，中，5张图。

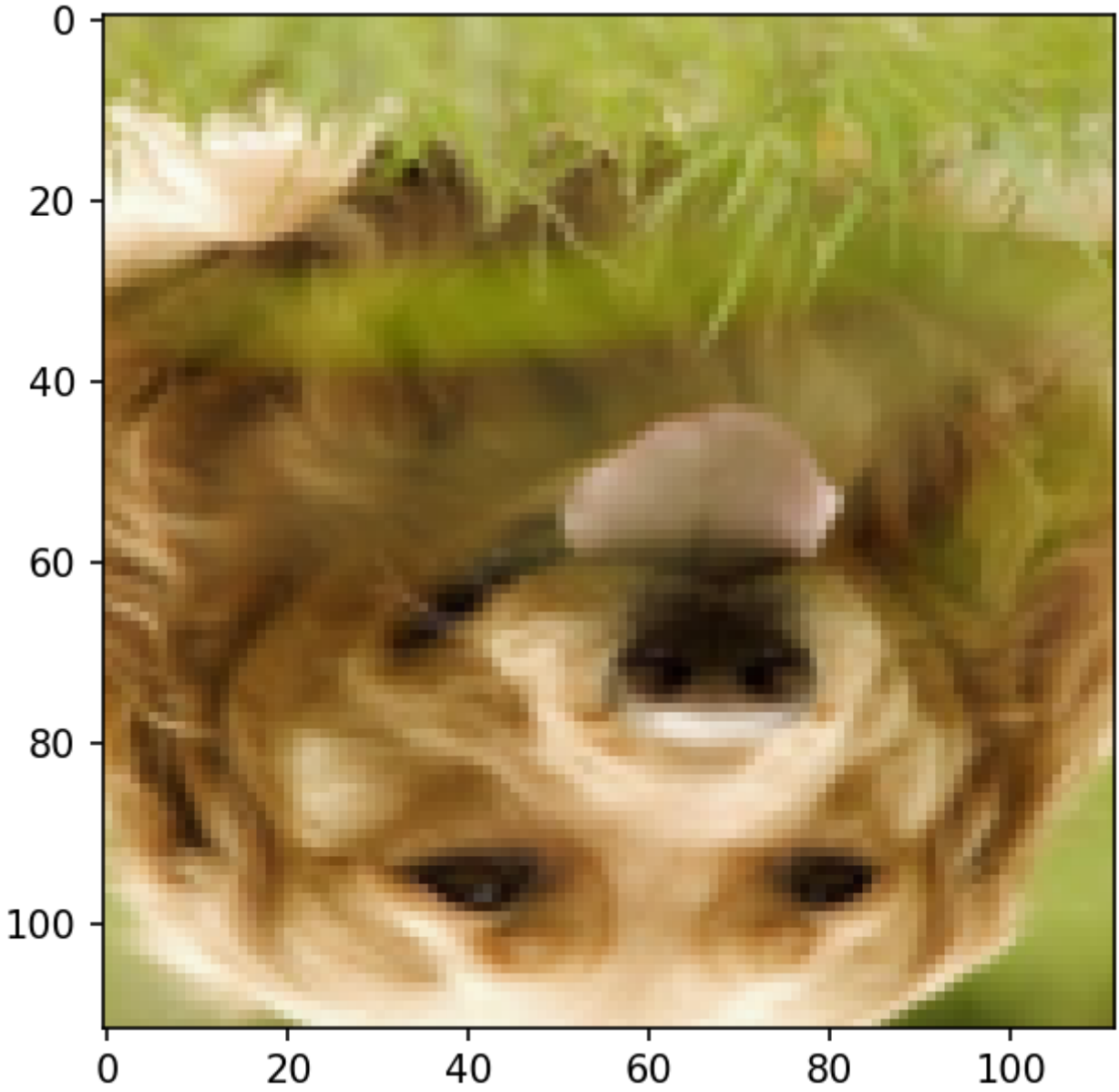
```
trans = transforms.Compose(
    [
      transforms.Resize((224,224)),# 必须是元组(224,224)
      transforms.TenCrop(112, vertical_flip=True), # 将获得5张图，是一个tuple元组，要拆开
      transforms.Lambda( lambda crops: torch.stack( [ (transforms.ToTensor()(each)) for each in crops ] ) ),
      ]
)
```


0
20
40
60
80
100
0
20
40
60
80
100


0
20
40
60
80
100
0
20
40
60
80
100


0
20
40
60
80
100
0
20
40
60
80
100


0
20
40
60
80
100
0
20
40
60
80
100




再增加5张翻转的图片。

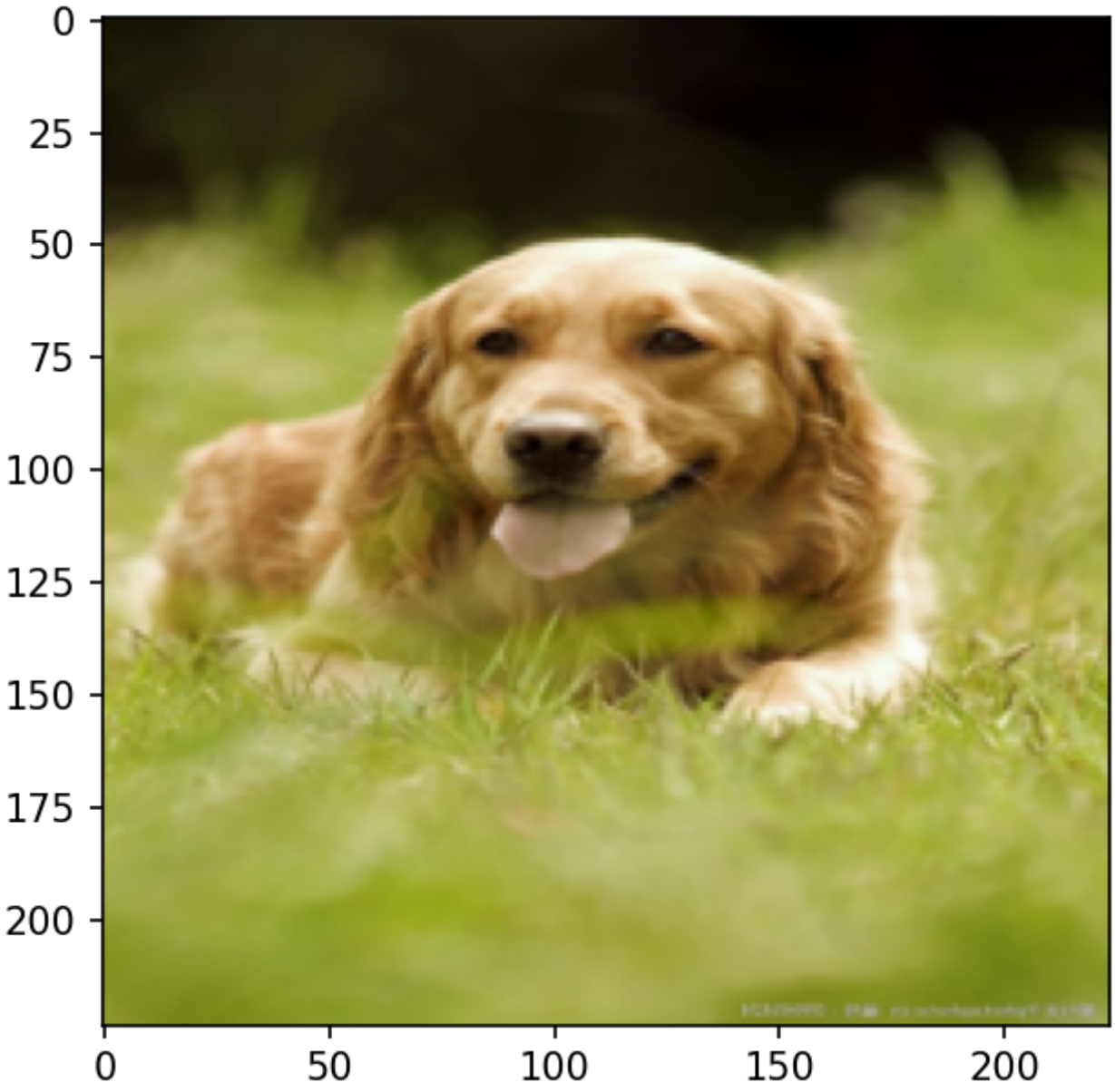
## 8、RandomVerticalFlip & RandomHorizontalFlip

```
RandomHorizontalFlip(p=0.5)

RandomVerticalFlip(p=0.5)
```

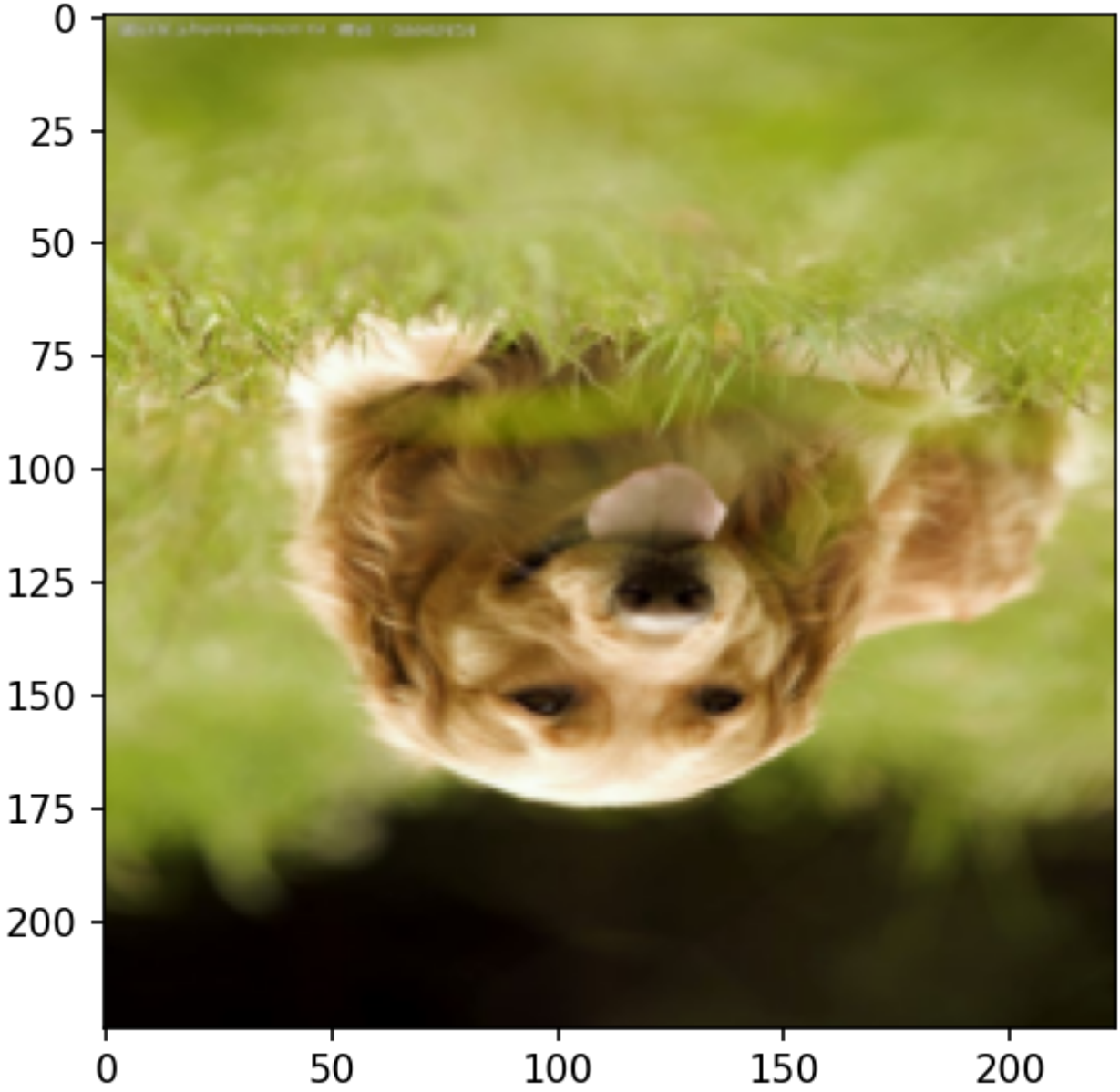
**依概率水平（左右）或垂直（上下）翻转图片**

- P:翻转概率

```
trans = transforms.Compose(
    [
      transforms.Resize((224,224)),# 必须是元组(224,224)
      transforms.RandomHorizontalFlip(p=1),

      transforms.ToTensor(),
      transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
      ]
)
```




```
trans = transforms.Compose(
    [
      transforms.Resize((224,224)),# 必须是元组(224,224)
      transforms.RandomVerticalFlip(p=1),

      transforms.ToTensor(),
      transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
      ]
)
```




## 9、RandomRotation

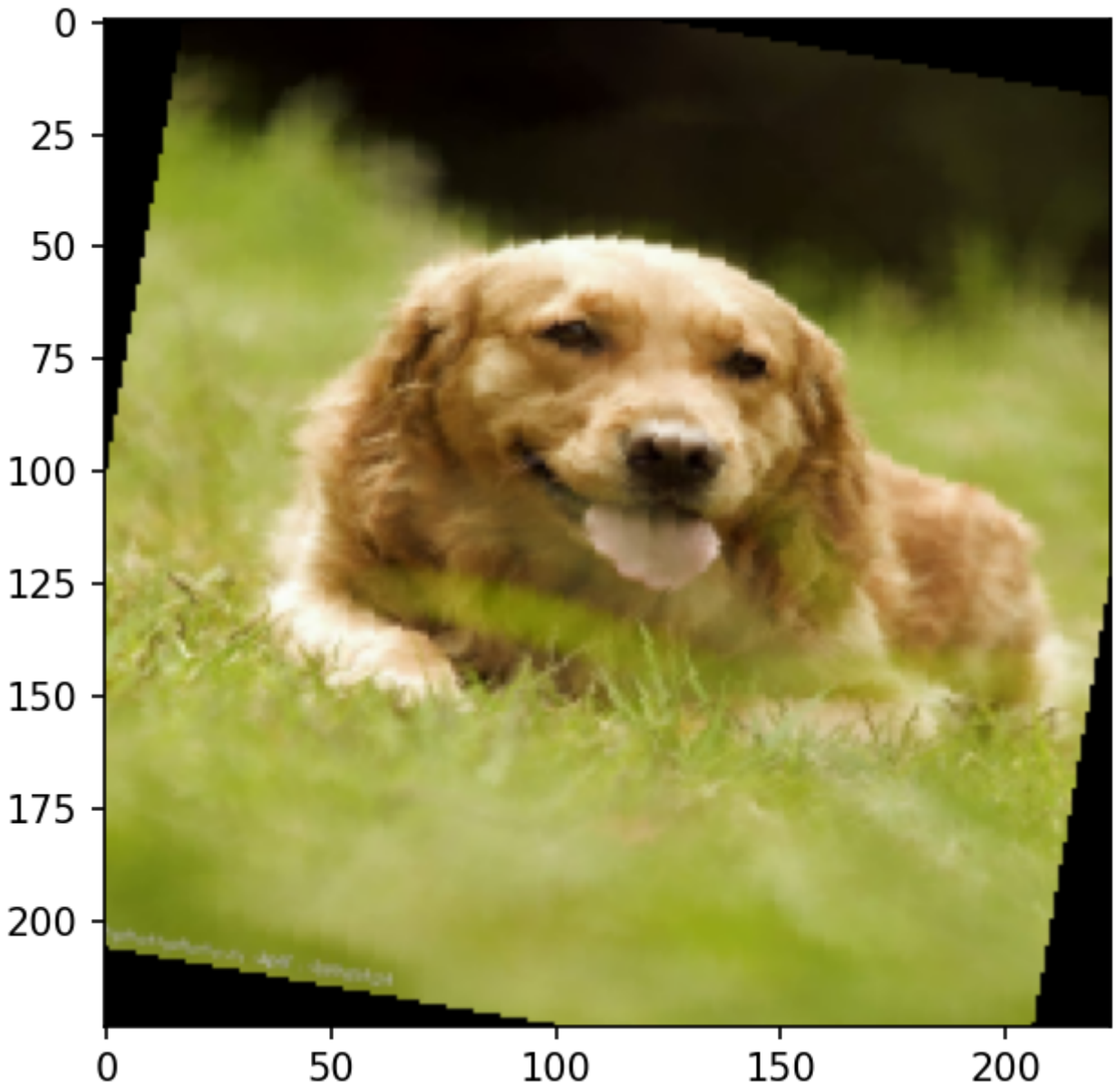
```
RandomRotation(degrees,
               resample=False,
               expand=False,
               center=None)
```

**随机旋转图片**

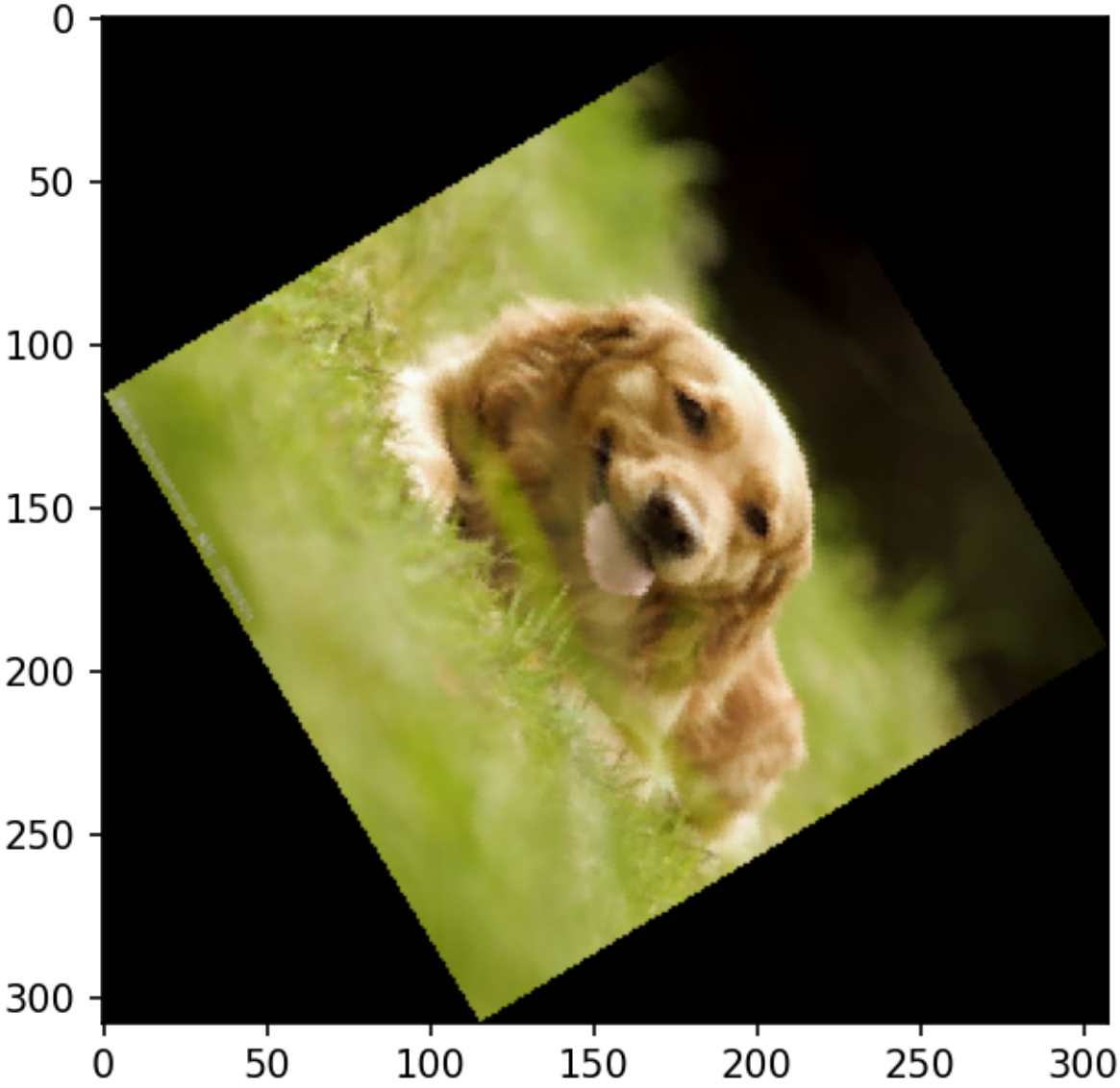
- degrees： 旋转角度
  - 当为a时，在(-a, a)之间选择旋转角度
  - 当为(a, b)时，在(a, b)之间选择旋转角度
- resample :重采样方法
- expand :是否扩大图片,以保持原图信息

- center :旋转点设置,默认中心旋转

```
trans = transforms.Compose(
    [
      transforms.Resize((224,224)),# 必须是元组(224,224)
      transforms.RandomRotation(90),

      transforms.ToTensor(),
      transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
      ]
)
```




可以看到当旋转后，四个角被转到外面，如果要保留角部信息，则transforms.RandomRotation(90, expand=True)




## 10、Pad

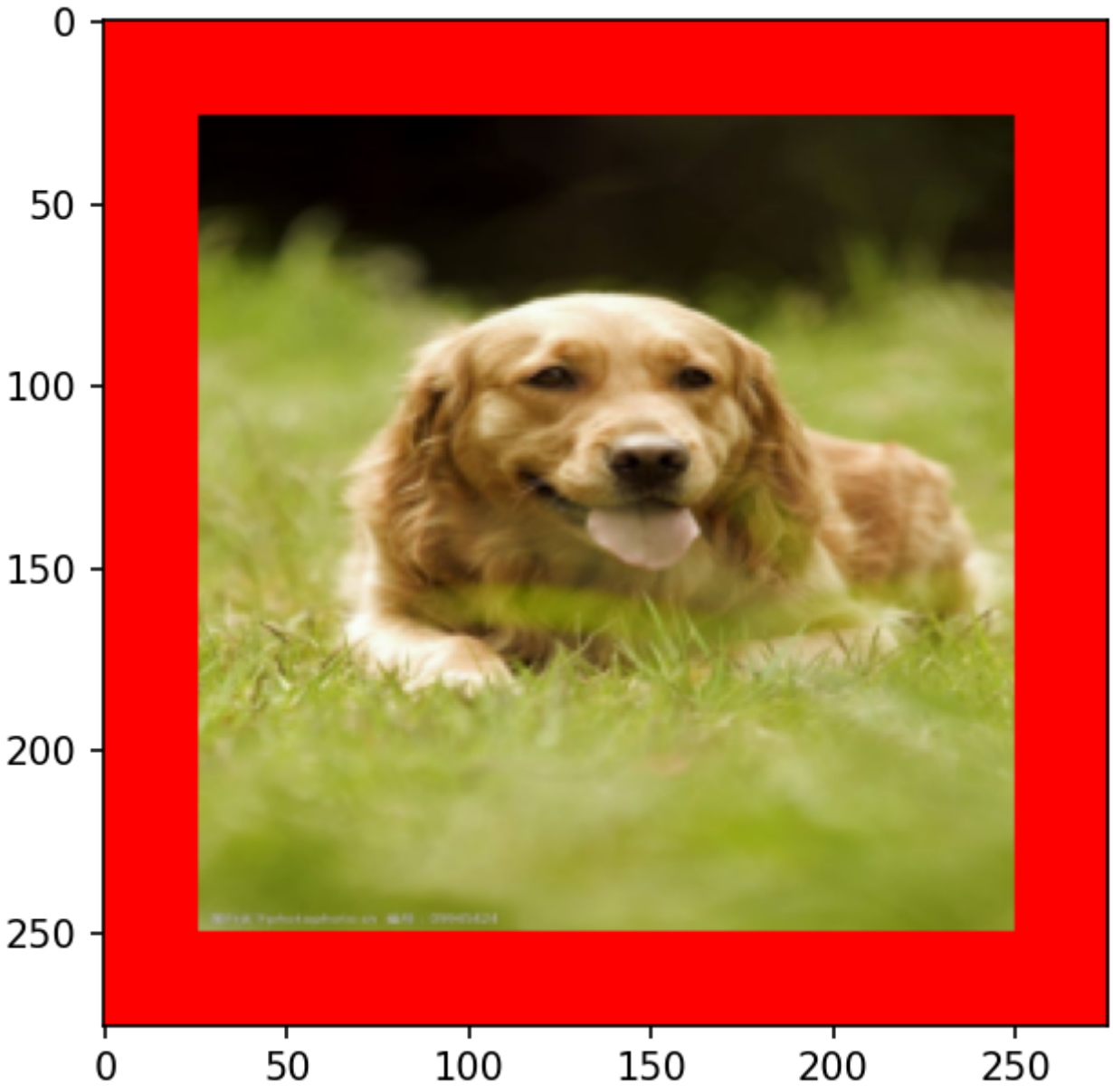
```
transforms.Pad(padding,
               fill=0,
               padding_mode='constant')
```

**对图片边缘进行填充**

- padding :设置填充大小
  - 当为a时，上下左右均填充a个像素
  - 当为(a, b)时,上下填充b个像素，左右填充a个像素
  - 当为(a, b, c, d)时,左，上，右,下分别填充a, b, c, d
- pad_if_need:若图像小于设定size，则padding
- padding_mode :填充模式，有4种模式
  - constant :像素值由fill设定

    - edge :像素值由图像边缘像素决定
    - reflect :镜像填充,最后一个像素不镜像
    - symmetric :镜像填充,最后一个像素镜像
- fill : constant时,设置填充的像素值

```
trans = transforms.Compose(
    [
      transforms.Resize((224,224)),# 必须是元组(224,224)
      transforms.Pad(padding=26,fill=(255,0,0)),

      transforms.ToTensor(),
      transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
      ]
)
```



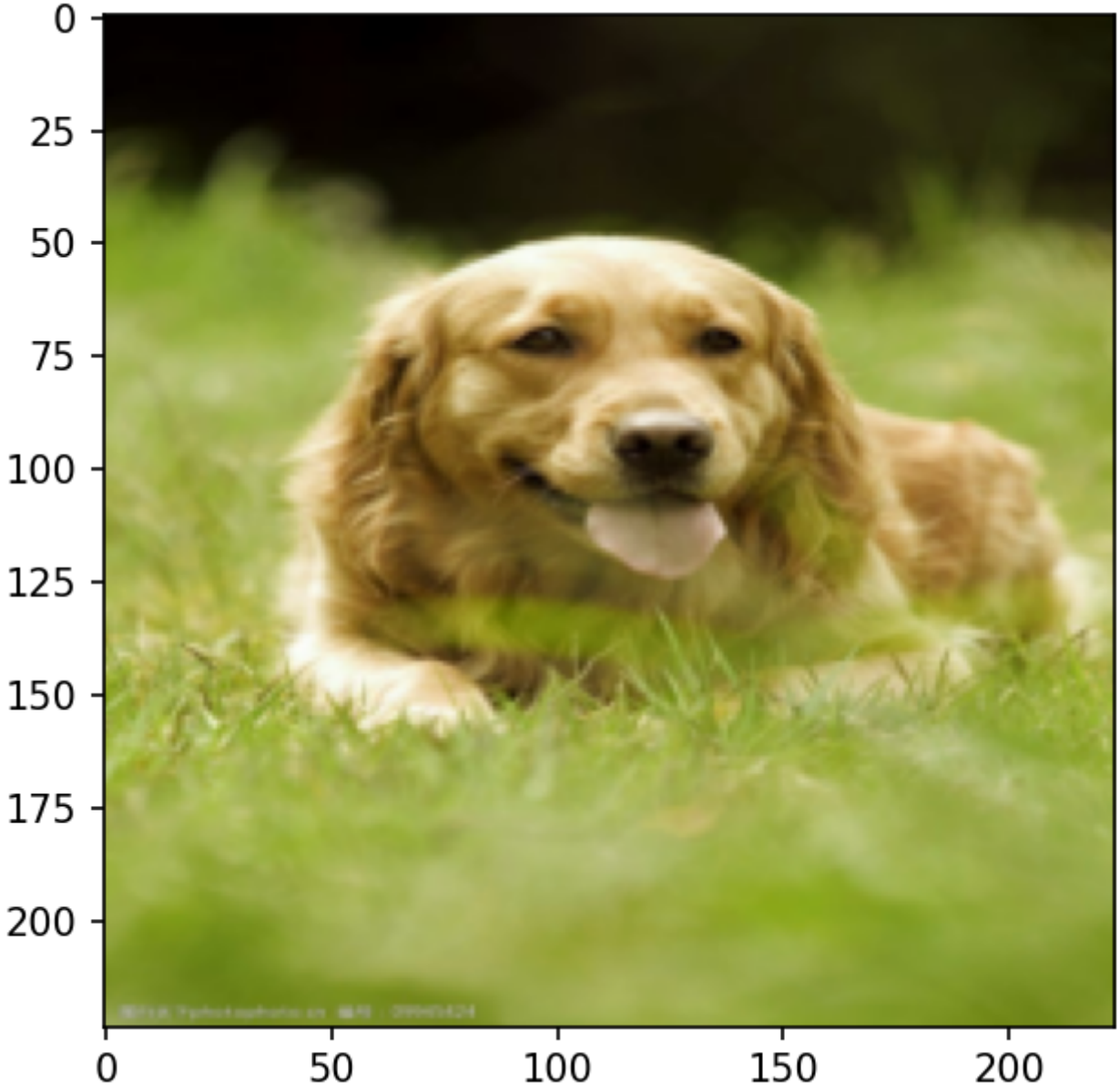
## 11、ColorJitter

```
transforms.ColorJitter(brightness=0,
                       contrast=0,
                       saturation=0,
                       hue=0)
```

**调整亮度、对比度、饱和度和色相**

- brightness:亮度调整因子
  - 当为a时,从［max(0, 1-a), 1+a］中随机选择
  - 当为(a, b)时,从［a, b］中随机选择
- contrast:对比度参数，同brightness
- saturation:饱和度参数，同brightness
- hue：色相参数，
  - 当为a时，从［-a, a］中选择参数，注：0 <= a <= 0.5
  - 当为(a, b)时，从［a, b］中选择参数，注：-0.5 <= a <= b <= 0.5

```
trans = transforms.Compose(
    [
      transforms.Resize((224,224)),# 必须是元组(224,224)
      transforms.ColorJitter(hue=0.5),

      transforms.ToTensor(),
      transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
      ]
)
```



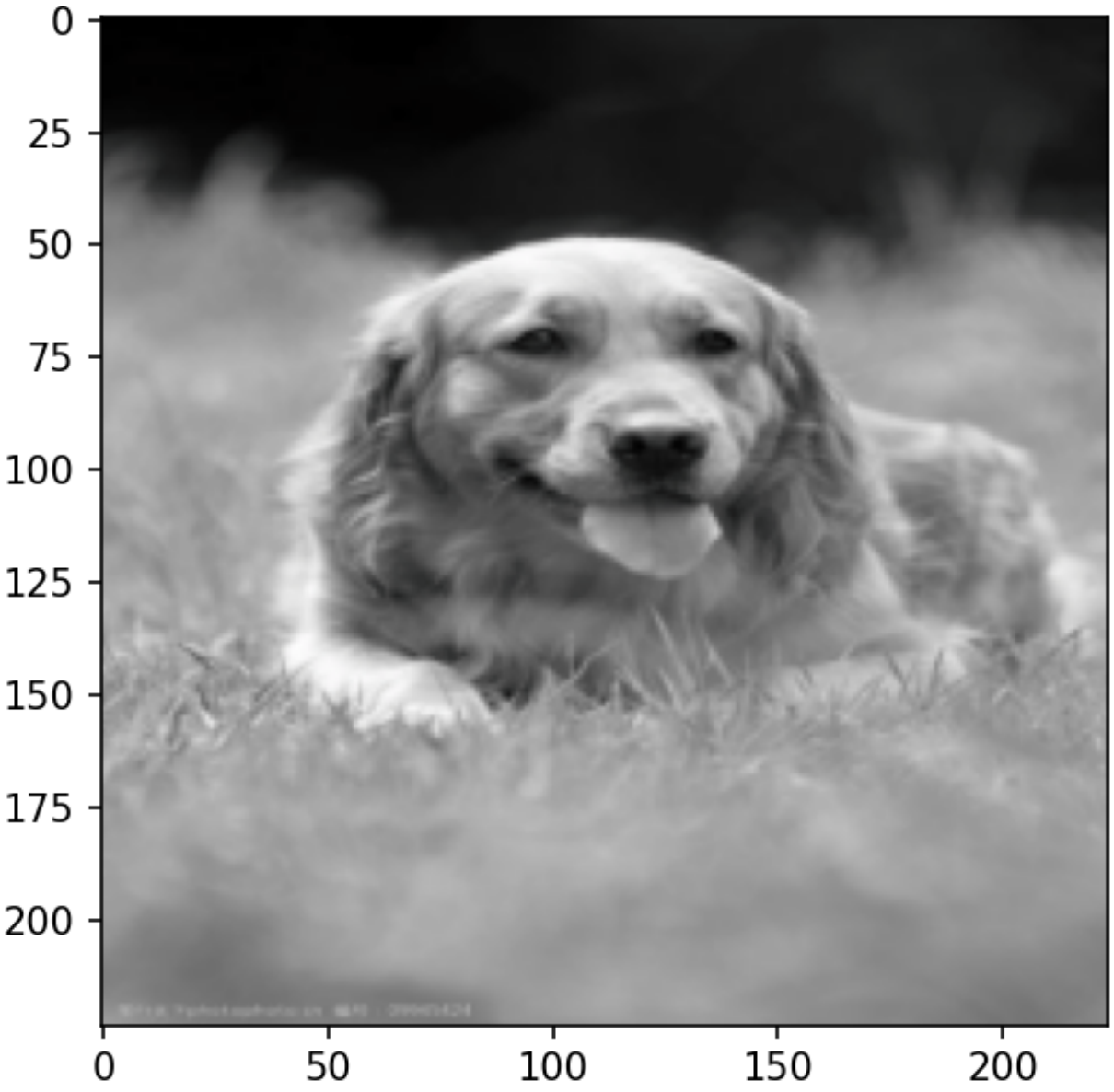

## 12、Grayscale & RandomGrayscale

```
RandomGrayscale(num_output_channels,
                p=0.1)

Grayscale(num_output_channels)
```

**依概率将图片转换为灰度图**

- num_ouput_channels：输出通道数 只能设1或3
- P:概率值,图像被转换为灰度图的概率

```
trans = transforms.Compose(
    [
      transforms.Resize((224,224)),# 必须是元组(224,224)
      transforms.Grayscale(num_output_channels=3),

      transforms.ToTensor(),
      transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
      ]
)
```



## 13、RandomErasing

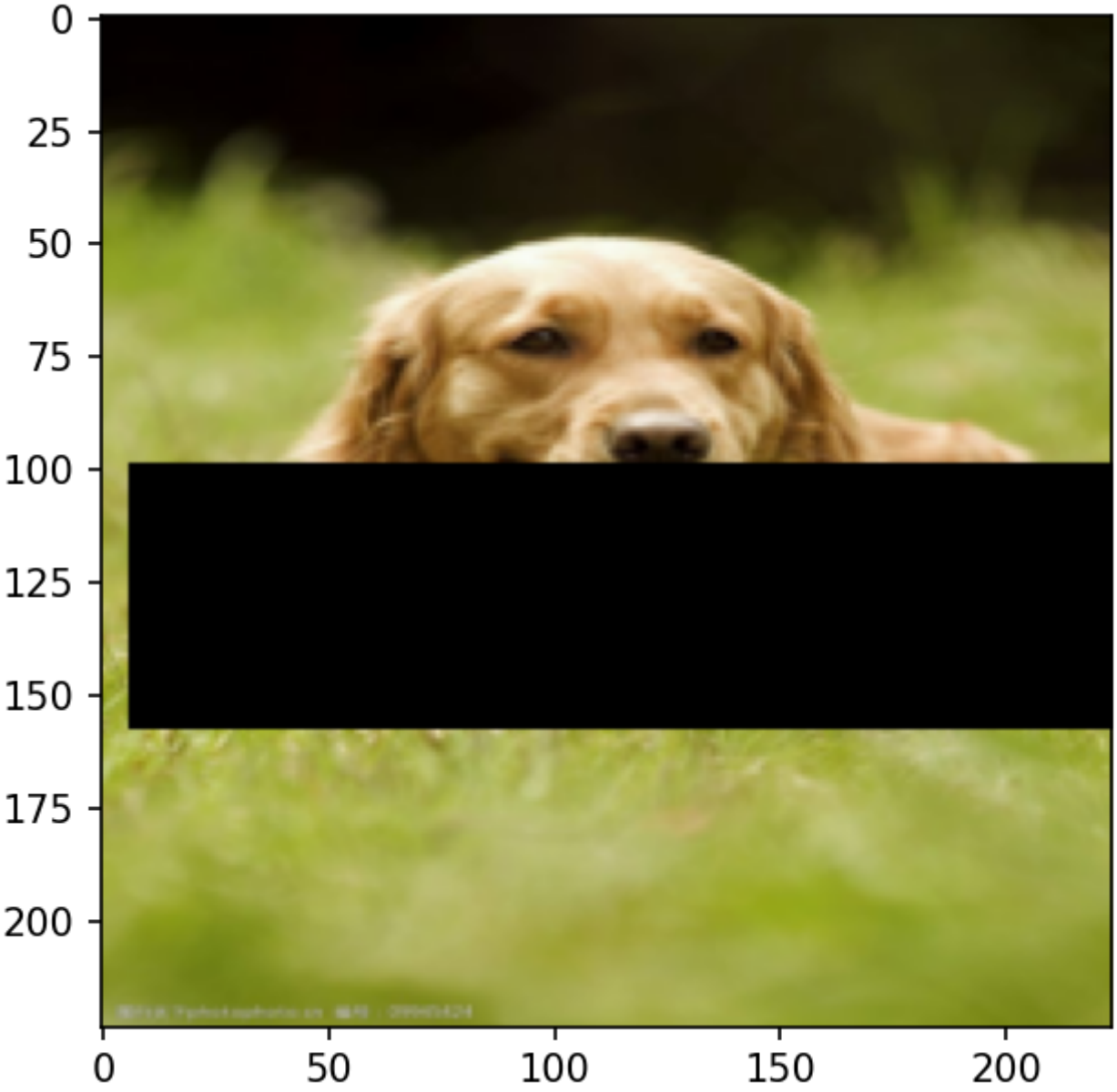
```
RandomErasing(p=0.5,
              scale=(0.02, 0.33),
              ratio=(0.3, 3.3),
              value=0,
              inplace=False)
```

**对图像进行随机遮挡**

- p:概率值,执行该操作的概率
- scale:遮挡区域的面积
- ratio:遮挡区域长宽比
- value:设置遮挡区域的像素值,(R, G, B) or (Gray)

```
trans = transforms.Compose(
    [
      transforms.Resize((224,224)),# 必须是元组(224,224)

      transforms.ToTensor(),
      transforms.RandomErasing(p=1,scale=(0.2,0.3),ratio=(0.2,0.3)),

      # transforms.ToTensor(),
      transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
      ]
)
```

> 注意，RandomErasing必须要在tensor下进行，所以之前要加transforms.ToTensor()




## 14、自定义转换

```
class Noise_transform():
      def __init__(self, snr, p):
            # 信噪比
            self.snr = snr
            # 中入噪声的概率
            self.p = p

      def __call__(self, img):
            if np.random.uniform(0,1) < self.p:
                  noise_p = self.p
                  signal_p = 1 - self.p
                  img = np.array(img).copy()
                  H, W, C = img.shape
                  # 按signal_p, noise_p/2, noise_p/2的概率生成0,1,2矩阵
                  mask = np.random.choice((0,1,2), size=(H,W,1), p=(signal_p, noise_p/2,
noise_p/2))
```

```
            mask = np.repeat(mask, C, axis=2)
            img[mask==1] = 0
            img[mask==2] = 255
            return Image.fromarray(img.astype('uint8')).convert('RGB')

        else:
            return img
```

这里实现了一个类，对图像加入噪声，通过初始化参数snr, p（信噪比，中入噪声的概率）来初始化该类。