AI基础：数据增强方法综述

机器学习初学者 1周前

以下文章来源于AI成长社 ，作者文瑞同学

导语

在深度学习时代，数据的规模越大、质量越高，模型就能够拥有更好的泛化能力，**数据直接决定了模型学习的上限。**然而在实际工程中，采集的数据很难覆盖全部的场景，比如图像的光照条件，同一场景拍摄的图片可能由于光线不同就会有很大的差异性，那么在训练模型的时候就需要加入光照方面的**数据增强。**另一方面，即使拥有大量的数据，也应该进行**数据增强**，这样有助于添加相关数据数据集中数据的数量，防止模型学习到不想要的模型，避免出现过拟合现象。

目前已经发布：

AI 基础：简易数学入门

AI 基础：Python开发环境设置和小技巧

AI 基础：Python 简易入门

AI 基础：Numpy 简易入门

AI 基础：Pandas 简易入门

AI 基础：Scipy(科学计算库) 简易入门

AI基础：数据可视化简易入门（matplotlib和seaborn）

AI基础：机器学习库Scikit-learn的使用

AI基础：机器学习简易入门

AI基础：机器学习的损失函数

AI基础：特征工程-类别特征

AI基础：特征工程-数字特征处理

AI基础：特征工程-文本特征处理

AI基础：词嵌入基础和Word2Vec

AI基础：图解Transformer

AI基础：一文看懂BERT

AI基础：入门人工智能必看的论文

AI基础：走进深度学习

AI基础：深度学习论文阅读路线（127篇经典论文下载）

后续持续更新

1.数据增强的方法和种类

数据增强的具体使用方法有两种，一种是事先执行所有的转换，实质是增强数据集的大小，这种方法称为线下增强。它比较适用于较小的数据集，最终将增加一定倍数的数据量，这个倍数取决于转换的图片个数，比如我需要对所有的图片进行旋转，则数据量增加一倍，本文中讨论的就是该方法。另一种是在将数据送入到机器学习模型的时候小批量（mini-batch）的转换,这种方法被称为线上增强或者飞行增强。这种方法比较适用于大数据集合，pytorch 中的 transforms 函数就是基于该方法，在训练中每次对原始图像进行扰动处理，经过多次几轮（epoch）训练之后，就等效于数据增加。

常用的数据增强有两种，有监督和无监督两种。本文只探讨有监督数据增强。有监督数据增强是基于现有的数据集，通过分析数据的完备性，采用一定的规则对现有数据进行扩充。有监督数据增强可以细分为单样本数据增强和多样本数据增强，在实际工程应用中，单样本数据增强使用更多，在 git 上有一些性能较好开源数据增强项目，他们功能较全并且处理速度也很快，开发者可以直接调用，如 imgaug 和 albumentations。在 pytorch 中，可以通过 torchvision 的 transforms 模块来实现集成了很多数据增强函数包。本文主要介绍单样本数据增强的一些常用方法。

2.裁剪

做裁剪操作主要是考虑原始图像的宽高扰动，在大多数图像分类网络中，样本在输入网络前必须要统一大小，所以通过调整图像的尺寸可以大量的扩展数据。通过裁剪有两种扩种方式，一

种是对大尺寸的图像直接按照需要送入网络的尺寸进行裁剪，比如原始图像的分辨率大小是256x256，现在网络需要输入的图像像素尺寸是224x224，这样可以直接在原始图像上进行随机裁剪224x224 像素大小的图像即可，这样一张图可以扩充32x32张图片；另外一种是将随机裁剪固定尺寸大小的图片，然后再将图像通过插值算法调整到网络需要的尺寸大小。由于数据集中通常数据大小不一，后者通常使用的较多。

使用opencv进行图像裁剪，利用随机数确定图像的裁剪范围，代码如下：

```
img_path = '../../img/ch3_img1.jpg'
img = cv2.imread(img_path)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
h, w, _ = img.shape
new_h1, new_h2 = np.random.randint(0, h-512, 2)
new_w1, new_w2 = np.random.randint(0, w-512, 2)
img_crop1 = img[new_h1:new_h1+512, new_w1:new_w1+512, :]
img_crop2 = img[new_h2:new_h2+512, new_w2:new_w2+512, :]

# 显示
plt.figure(figsize=(15, 10))
plt.subplot(1,3,1), plt.imshow(img)
plt.axis('off'); plt.title('原图')
plt.subplot(1,3,2), plt.imshow(img_crop1)
plt.axis('off'); plt.title('水平镜像')
plt.subplot(1,3,3), plt.imshow(img_crop2)
plt.axis('off'); plt.title('垂直镜像')
plt.show()
```

运行上述代码可到如下结果图：

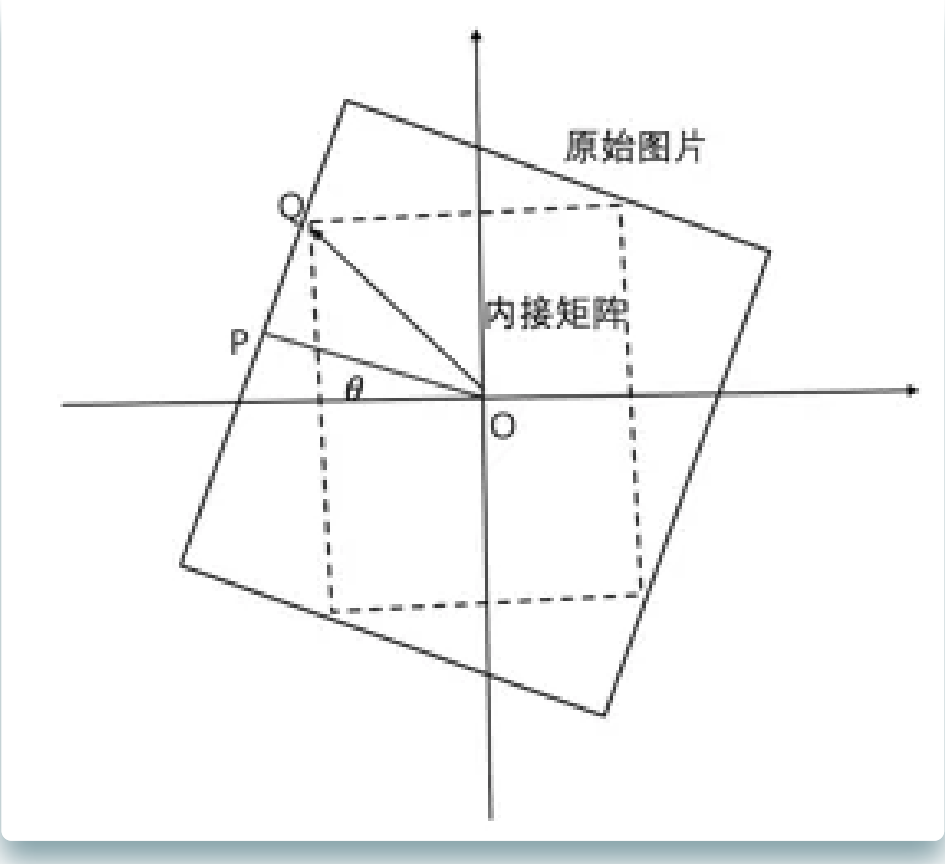
image

3.翻转和旋转

翻转和旋转都是将原始的图像像素在位置空间上做变换，图像的翻转是将原始的图像进行镜像操作，镜像操作在数据增强中会经常被使用，并且起了非常重要的作用，它主要包括水平镜像

翻转，垂直镜像翻转和原点镜像翻转，具体在使用中，需要结合数据形式选择相应翻转操作，比如数据集是汽车图像数据，训练集合测试集都是正常拍摄的图片，此时只使用水平镜像操作，如果加入垂直或者原点镜像翻转，会对原始图像产生干扰。

角度旋转操作和图像镜像相对，它主要是沿着画面的中心进行任意角度的变换，该变换是通过将原图像和仿射变换矩阵相乘实现的。为了实现图像的中心旋转，除了要知道旋转角度，还要计算平移的量才能能让仿射变换的效果等效于旋转轴的画面中心。仿射变换矩阵是一个余弦矩阵，在OpenCV中有实现的库cv2.getRotationMatrix2D可以使用，该函数的第1个参数是旋转中心，第2个参数是逆时针旋转角度，第3个参数是缩放倍数，对于只是旋转的情况参数值是1，返回的值就是做仿射变换的矩阵。然后通过cv2.warpAffine()将原图像矩阵乘以旋转矩阵得到最终的结果。
通过上述的操作旋转的图像会有存在黑边，如果想去除掉图片的黑边，需要将原始的图像做出一些牺牲，对旋转后的图像取最大内接矩阵，该矩阵的长宽比和原始图像相同，如下图所示。要计算内切矩阵的坐标Q,需要通过旋转角度 和原始图像矩阵的边长OP得到。



img

最终的计算公式如公式(1)至(3)所示

$$r = \begin{cases} \frac{h}{w} & h > w \\ \frac{w}{h} & \text{其他} \end{cases} \tag{2}$$

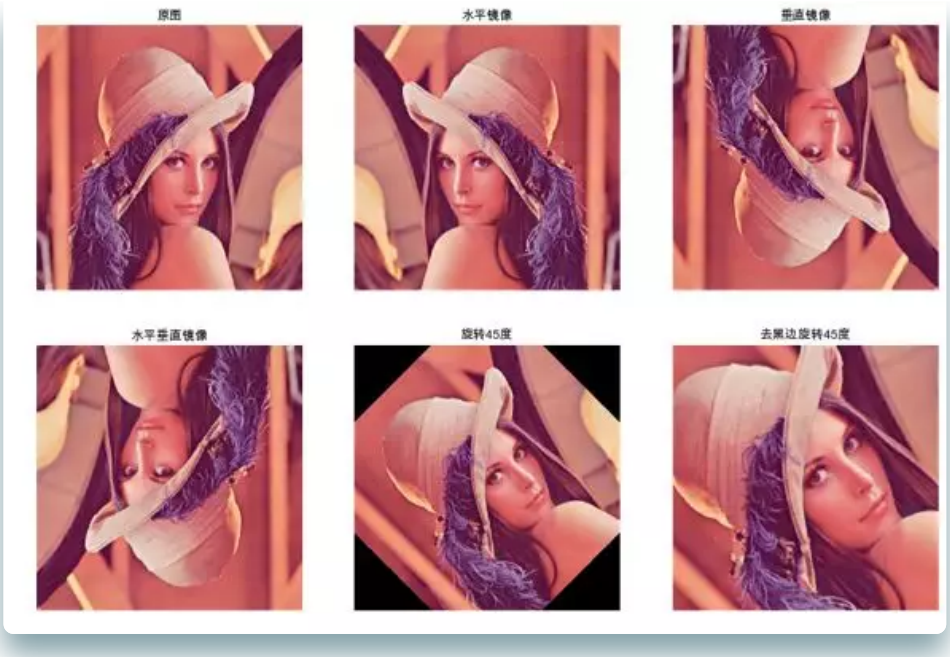
$$w' = kw, h' = kh. \tag{3}$$

利用opencv实现的代码如下：

```
# 去除黑边的操作
crop_image = lambda img, x0, y0, w, h: img[y0:y0+h, x0:x0+w]  # 定义裁切函数，后续裁切黑边使用

def rotate_image(img, angle, crop):
    """
    angle: 旋转的角度
    crop: 是否需要进行裁剪，布尔向量
    """
    w, h = img.shape[:2]
    # 旋转角度的周期是360°
    angle %= 360
    # 计算仿射变换矩阵
    M_rotation = cv2.getRotationMatrix2D((w / 2, h / 2), angle, 1)
    # 得到旋转后的图像
    img_rotated = cv2.warpAffine(img, M_rotation, (w, h))

    # 如果需要去除黑边
    if crop:
        # 裁剪角度的等效周期是180°
        angle_crop = angle % 180
        if angle > 90:
            angle_crop = 180 - angle_crop
        # 转化角度为弧度
        theta = angle_crop * np.pi / 180
        # 计算高宽比
        hw_ratio = float(h) / float(w)
        # 计算裁剪边长系数的分子项
        tan_theta = np.tan(theta)
        numerator = np.cos(theta) + np.sin(theta) * np.tan(theta)

        # 计算分母中和高宽比相关的项
        r = hw_ratio if h > w else 1 / hw_ratio
        # 计算分母项
        denominator = r * tan_theta + 1
        # 最终的边长系数
        crop_mult = numerator / denominator

        # 得到裁剪区域
        w_crop = int(crop_mult * w)
        h_crop = int(crop_mult * h)
        x0 = int((w - w_crop) / 2)
        y0 = int((h - h_crop) / 2)
```

```
        img_rotated = crop_image(img_rotated, x0, y0, w_crop, h_crop)
    return img_rotated
#水平镜像
h_flip = cv2.flip(img,1)
#垂直镜像
v_flip = cv2.flip(img,0)
#水平垂直镜像
hv_flip = cv2.flip(img,-1)
#90度旋转
rows, cols, _ = img.shape
M = cv2.getRotationMatrix2D((cols/2, rows/2), 45, 1)
rotation_45 = cv2.warpAffine(img, M, (cols, rows))
#45度旋转
M = cv2.getRotationMatrix2D((cols/2, rows/2), 135, 2)
rotation_135 = cv2.warpAffine(img, M,(cols, rows))
#去黑边旋转45度
image_rotated = rotate_image(img, 45, True)

#显示
plt.figure(figsize=(15, 10))
plt.subplot(2,3,1), plt.imshow(img)
plt.axis('off'); plt.title('原图')
plt.subplot(2,3,2), plt.imshow(h_flip)
plt.axis('off'); plt.title('水平镜像')
plt.subplot(2,3,3), plt.imshow(v_flip)
plt.axis('off'); plt.title('垂直镜像')
plt.subplot(2,3,4), plt.imshow(hv_flip)
plt.axis('off'); plt.title('水平垂直镜像')
plt.subplot(2,3,5), plt.imshow(rotation_45)
plt.axis('off'); plt.title('旋转45度')
plt.subplot(2,3,6), plt.imshow(image_rotated)
plt.axis('off'); plt.title('去黑边旋转45度')
plt.show()
```

上述代码通过opencv自带的flip函数实现了翻转操作，该函数的第二个参数是控制翻转的方向。通过内切矩阵计算公式可得无黑边剪切结果。上述代码的结果图如下图所示。通过结果可以看出，旋转

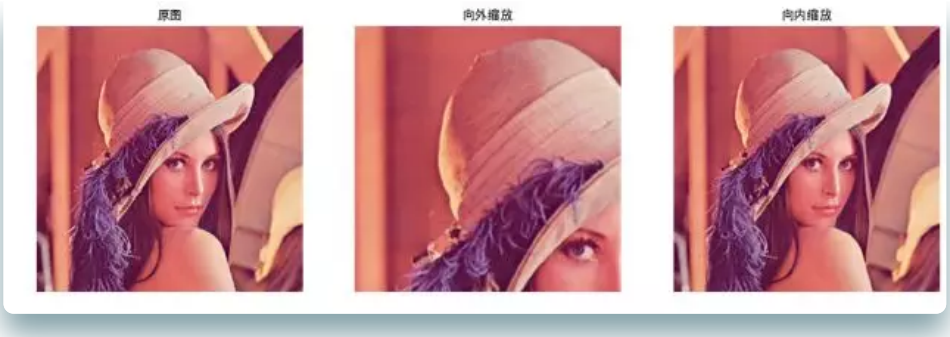


img

4.缩放

图像可以向外或向内缩放。向外缩放时，最终图像尺寸将大于原始图像尺寸，为了保持原始图像的大小，通常需要结合裁剪，从缩放后的图像中裁剪出和原始图像大小一样的图像。另一种方法是向内缩放，它会缩小图像大小，缩小到预设的大小。缩放也会带了一些问题，如缩放后的图像尺寸和原始图像尺寸的长宽比差异较大，会出现图像失帧的现象，如果在实验中对最终的结果有一定的影响，需要做等比例缩放，对不足的地方进行边缘填充。以下是缩放的代码和结果图像。

```
img_2 = cv2.resize(img, (int(h * 1.5), int(w * 1.5)))
img_2 = img_2[int((h - 512) / 2) : int((h + 512) / 2), int((w - 512) / 2) : int((w + 512) /2), :]
img_3 = cv2.resize(img, (512, 512))

## 显示
plt.figure(figsize=(15, 10))
plt.subplot(1,3,1), plt.imshow(img)
plt.axis('off'); plt.title('原图')
plt.subplot(1,3,2), plt.imshow(img_2)
plt.axis('off'); plt.title('向外缩放')
plt.subplot(1,3,3), plt.imshow(img_3)
plt.axis('off'); plt.title('向内缩放')
plt.show()
```

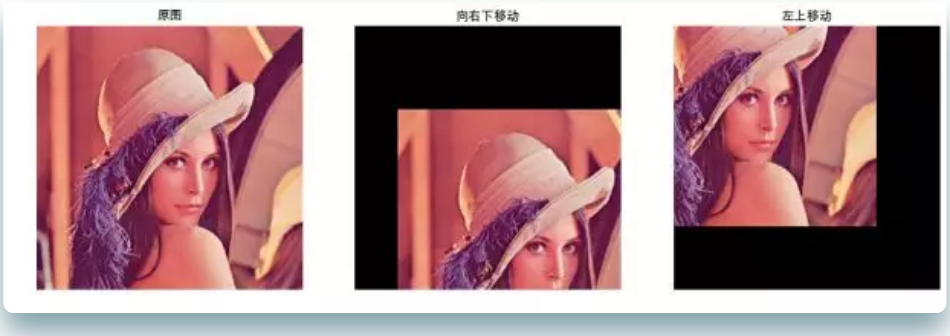


img

5.移位

移位只涉及沿X或Y方向（或两者）移动图像，如果图像的背景是单色被背景或者是纯的黑色背景，使用该方法可以很有效的增强数据数量，可以通过cv2.warpAffine实现该代码：

```
mat_shift = np.float32([[1,0,100], [0,1,200]])
img_1 = cv2.warpAffine(img, mat_shift, (h, w))
mat_shift = np.float32([[1, 0, -150], [0, 1, -150]])
img_2 = cv2.warpAffine(img, mat_shift, (h, w))

## 显示
plt.figure(figsize=(15, 10))
plt.subplot(1,3,1), plt.imshow(img)
plt.axis('off'); plt.title('原图')
plt.subplot(1,3,2), plt.imshow(img_1)
plt.axis('off'); plt.title('向右下移动')
plt.subplot(1,3,3), plt.imshow(img_2)
plt.axis('off'); plt.title('左上移动')
plt.show()
```



img

6.高斯噪声

当神经网络试图学习可能无用的高频特征（即图像中大量出现的模式）时，通常会发生过度拟合。具有零均值的高斯噪声基本上在所有频率中具有数据点，从而有效地扭曲高频特征。这也意味着较低频率的组件也会失真，但你的神经网络可以学会超越它，添加适量的噪音可以增强学习能力。

基于噪声的数据增强就是在原来的图片的基础上，随机叠加一些噪声，最常见的做法就是高斯噪声。更复杂一点的就是在面积大小可选定、位置随机的矩形区域上丢弃像素产生黑色矩形块，从而产生一些彩色噪声，以Coarse Dropout方法为代表，甚至还可以对图片上随机选取一块区域并擦除图像信息。以下是代码和图像：

```
img_s1 = gasuss_noise(img, 0, 0.005)
img_s2 = gasuss_noise(img, 0, 0.05)
plt.figure(figsize=(15, 10))
plt.subplot(1,3,1), plt.imshow(img)
plt.axis('off'); plt.title('原图')
plt.subplot(1,3,2), plt.imshow(img_s1)
plt.axis('off'); plt.title('方差为0.005')
plt.subplot(1,3,3), plt.imshow(img_s2)
plt.axis('off'); plt.title('方差为0.05')
plt.show()
```

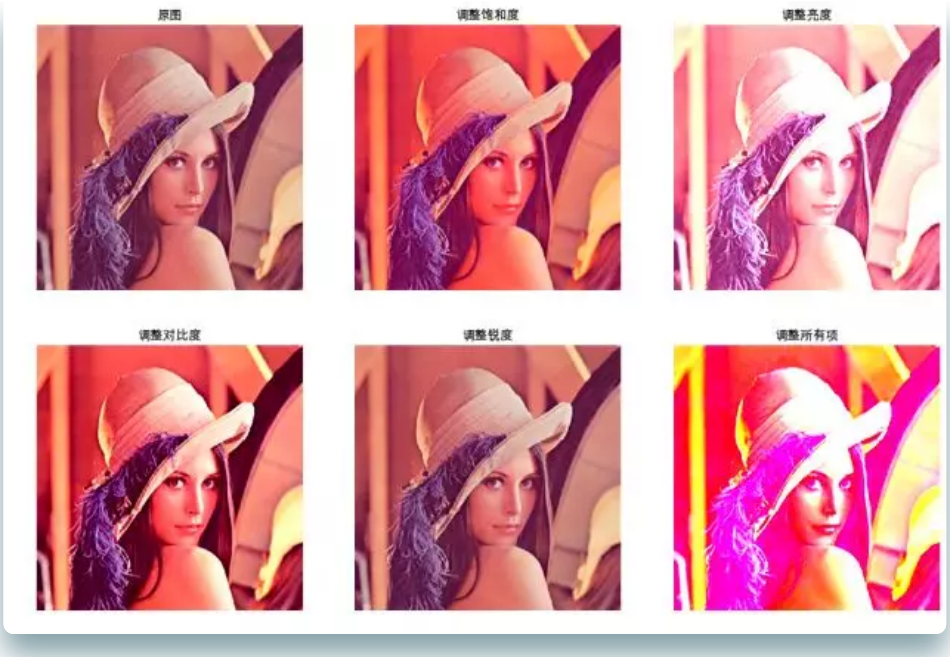
img

7.色彩抖动

上面提到的图像中有一个比较大的难点是背景干扰，在实际工程中为了消除图像在不同背景中存在的差异性，通常会做一些色彩抖动操作，扩充数据集合。色彩抖动主要是在图像的颜色方面做增强，主要调整的是图像的亮度，饱和度和对比度。工程中不是任何数据集都适用，通常如果不同背景的图像较多，加入色彩抖动操作会有很好的提升。

```
def randomColor(image, saturation=0, brightness=0, contrast=0, sharpness=0):
    if random.random() < saturation:
        random_factor = np.random.randint(0, 31) / 10.  # 随机因子
```

```
        image = ImageEnhance.Color(image).enhance(random_factor)  # 调整图像的饱和度
    if random.random() < brightness:
        random_factor = np.random.randint(10, 21) / 10.  # 随机因子
        image = ImageEnhance.Brightness(image).enhance(random_factor)  # 调整图像的亮度
    if random.random() < contrast:
        random_factor = np.random.randint(10, 21) / 10.  # 随机因1子
        image = ImageEnhance.Contrast(image).enhance(random_factor)  # 调整图像对比度
    if random.random() < sharpness:
        random_factor = np.random.randint(0, 31) / 10.  # 随机因子
        ImageEnhance.Sharpness(image).enhance(random_factor)  # 调整图像锐度
    return image


cj_img = Image.fromarray(img)
sa_img = np.asarray(randomColor(cj_img, saturation=1))
br_img = np.asarray(randomColor(cj_img, brightness=1))
co_img = np.asarray(randomColor(cj_img, contrast=1))
sh_img = np.asarray(randomColor(cj_img, sharpness=1))
rc_img = np.asarray(randomColor(cj_img, saturation=1, \
                                brightness=1, contrast=1, sharpness=1))
plt.figure(figsize=(15, 10))
plt.subplot(2,3,1), plt.imshow(img)
plt.axis('off'); plt.title('原图')
plt.subplot(2,3,2), plt.imshow(sa_img)
plt.axis('off'); plt.title('调整饱和度')
plt.subplot(2,3,3), plt.imshow(br_img)
plt.axis('off'); plt.title('调整亮度')
plt.subplot(2,3,4), plt.imshow(co_img)
plt.axis('off'); plt.title('调整对比度')
plt.subplot(2,3,5), plt.imshow(sh_img)
plt.axis('off'); plt.title('调整锐度')
plt.subplot(2,3,6), plt.imshow(rc_img)
plt.axis('off'); plt.title('调整所有项')
plt.show()
```



img

参考文献

- https://www.cnblogs.com/fydeblog/p/10734733.html
- https://blog.csdn.net/zhu_hongji/article/details/80984341
- http://docs.opencv.org/master/d4/d13/tutorial_py_filtering.html
- 叶韵.深度学习与计算机视觉：算法原理、框架应用与代码实现[M].北京：机械工业出版社，2018:194-195.
- Zhang H, Cisse M, Dauphin Y N, et al. mixup: Beyond Empirical Risk Minimization[J]. 2017.
- https://medium.com/nanonets/how-to-use-deep-learning-when-you-have-limited-data-part-2-data-augmentation-c26971dc8ced
- https://medium.com/nanonets/nanonets-how-to-use-deep-learning-when-you-have-limited-data-f68c0b512cab

备注：公众号菜单包含了整理了一本**AI小抄**，非常适合在通勤路上用学习。

往期精彩回顾

- 那些年做的学术公益-你不是一个人在战斗
- 适合初学者入门人工智能的路线及资料下载
- 机器学习在线手册
- 深度学习在线手册
- AI基础下载（第一部分）

备注：加入本站微信群或者qq群，请回复“加群”

加入知识星球（4500+用户，ID：92416895），请回复“知识星球”

喜欢文章，点个在看