augment-auto

Release 0.1.0

Aug 21, 2020

Contents

A python image augmentation library based on opencv and numpy. It can be used for augmenting images in both image classification and object detection tasks. Many different techniques of augmentation are supported, which can be clustered into three major types - geometric transformations, photometric transformations and kernel-based transformations. Library has support for images with bounding boxes as well.

CHAPTER 1

Installation

- **Install using pip**:

```
pip install augment-auto
```

- **Install by building from scratch**:

```
git clone https://github.com/keshavoct98/image-augmentation
cd image-augmentation
python setup.py install
```

CHAPTER 2

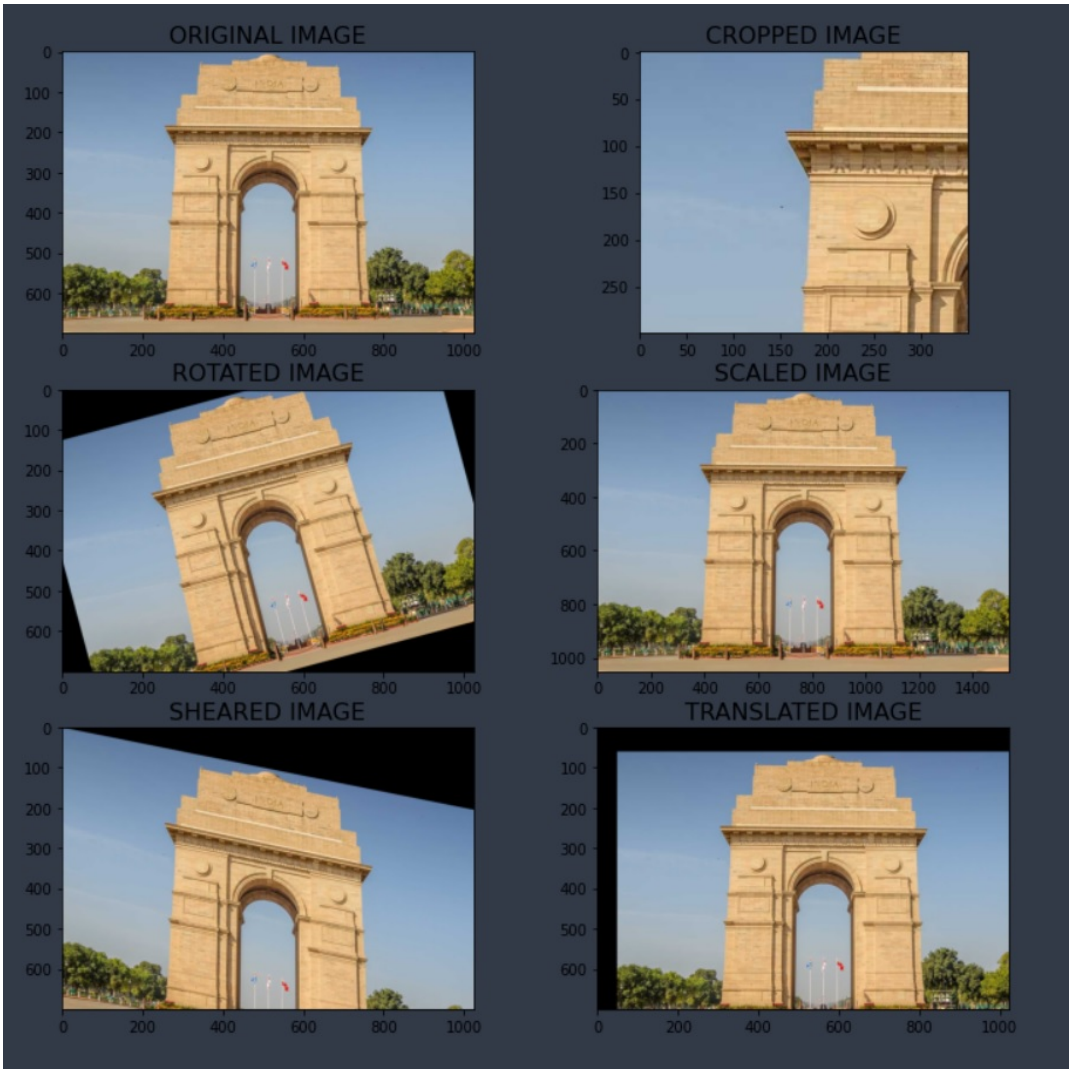
Features

1. **Geometric Features** - Image augmentation with geometric transformation of images.
    - **crop(img, point1, point2, box = None)** Returns cropped image. Image is cropped using point1(x1, y1) and point2(x2, y2).
        1. **img =** ***numpy.ndarray*** Image to be cropped.
        2. **point1 =** ***tuple of int*** initial crop coordinates in the format - (x1, y1).
        3. **point2 =** ***tuple of int*** final crop coordinates in the format - (x2, y2).
        4. **box =** ***list*****, default = None** Coordinates of bounding box in the format - (x1,y1,x2,y2). If bounding box coordinates are passed, new coordinates are calculated and returned along with output image.
    - **rotate(img, angle, keep_resolution = True, box = None)** Returns image rotated at the given angle.
        1. **img =** ***numpy.ndarray*** Image to be rotated.
        2. **angle =** ***integer or float*** value of angle at which image is to be rotated.
        3. **keep_resolution =** ***bool*****, default = True** If True, resolution of image remains same after rotation, else resolution is changed.
        4. **box =** ***list*** Coordinates of bounding box in the format - (x1,y1,x2,y2). If bounding box coordinates are passed, new coordinates are calculated and returned along with output image.
    - **scale(img, fx, fy, keep_resolution = False, box = None)** Returns scaled image.
        1. **img =** ***numpy.ndarray*** Image to be scaled.
        2. **fx =** ***integer or float*** scaling value for x-axis.
        3. **fy =** ***integer or float*** scaling value for y-axis.
        4. **keep_resolution =** ***bool*****, default = False** If True, resolution of image remains same after scaling, extra region is cropped out.

5. **box =** ***list*** Coordinates of bounding box in the format - (x1,y1,x2,y2). If bounding box coordinates are passed, new coordinates are calculated and returned along with output image.

- **shear(img, shear_val, axis = 0, box = None)** Returns sheared image along given axis.
  1. **img =** ***numpy.ndarray*** Image to be sheared.
  2. **shear_val =** ***integer or float*** shearing magnitude for given axis.
  3. **axis =** ***{0,1}*****, default = 0** 0 for shear along x-axis, 1 for shear along y-axis.
  4. **box =** ***list*** Coordinates of bounding box in the format - (x1,y1,x2,y2). If bounding box coordinates are passed, new coordinates are calculated and returned along with output image.
- **translate(img, tx, ty, box = None)** Returns translated image.
  1. **img =** ***numpy.ndarray*** Image to be translated.
  2. **tx =** ***integer or float*** translation magnitude along x-axis.
  3. **ty =** ***integer or float*** translation magnitude along y-axis.
  4. **box =** ***list*** Coordinates of bounding box in the format - (x1,y1,x2,y2). If bounding box coordinates are passed, new coordinates are calculated and returned along with output image.

```
# Geometric Transformations

img = cv2.imread('images/3.jpg')

img_new = crop(img, point1 = (100, 100), point2 = (450, 400))

img_new = rotate(img, angle = 15, keep_resolution = True)

img_new = scale(img, fx = 1.5, fy = 1.5, keep_resolution = False)

img_new = shear(img, shear_val = 0.2, axis = 1)

img_new = translate(img, tx = 50, ty = 60)
```




```
# Geometric Transformations with bounding box

img = cv2.imread('images/0.jpeg')
bbox = [581, 274, 699, 321]

img_new, bbox_new = crop(img, point1 = (100, 100), point2 = (650, 400),␣
↪box = bbox)

img_new, bbox_new = rotate(img, angle = 15, keep_resolution = True, box =␣
↪bbox)

img_new, bbox_new = scale(img, fx = 1.5, fy = 1.3, keep_resolution =␣
↪False, box = bbox)

img_new, bbox_new = shear(img, shear_val = 0.2, axis = 0, box = bbox)

img_new, bbox_new = translate(img, tx = 50, ty = 160, box = bbox)
```

2. **Photometric Features** - Image augmentation with photometric transformation of images.
    - **brightness_contrast(img, alpha = 1.5, beta = 0)** Returns image with new pixel intensities.

      *img_new = img * alpha + beta*

      1. **img =** ***numpy.ndarray*** Image whose brightness and contrast has to be modified.
      2. **alpha =** ***integer or float, non-negative*****, default = 1.5** All pixel values of the passed image are multiplied by value of alpha.
      3. **beta =** ***integer or float*****, default = 0** Vaue of beta is added to all pixel values of the passed image after multiplication of pixel values with value of alpha.
    - **colorSpace(img, colorspace = 'hsv')** Returns image converted to the new colorspace. Three types of colorspace are supported - HSV, YCrCb, LAB.
      1. **img =** ***numpy.ndarray*** Image whose colorspace has to be converted.
      2. **colorspace =** ***{'hsv', 'ycrcb', 'lab'}*****, default = 'hsv'** Colorspace to which image is to be converted.

- **addNoise(img, noise_type = 'gaussian', mean = 0, var = 0.05, sp_ratio = 0.5, noise_amount = 0.02)**
  Returns image with added noise. Three different types of noise are supported - GAUSSIAN, Salt n Pepper, Poisson.

  1. **img =** ***numpy.ndarray*** Image to which noise has to be added.
  2. **noise_type =** ***{'gaussian', 'salt_pepper', 'poisson'}*****, default = 'gaussian** Type of noise to add.
  3. **mean =** ***int or float, (required only with noise_type = 'gaussian').*****, default = 0**
     Gaussian noise is generated using mean value.
  4. **var =** ***int or float, non-negative, (required only with noise_type = 'gaussian').*****, default = 0.05**
     Gaussian noise is generated from the standard deviation calculated value of variance provided.
  5. **sp_ratio =** ***int or float, range :- 0 <= sp_ratio <= 1, (required only with noise_type = 'salt_pepper').*****, default =**
     Percentage of salt noise and pepper noise. if value passed is equal to 1, only salt noise is present. Similarly if value is 0, only pepper noise is present.
  6. **noise_amount =** ***int or float, non-negative, (required only with noise_type = 'salt_pepper' or 'poisson').*****, defau**
     magnitude of salt n pepper/poisson noise is calculated using noise_amount.

```
# Photometric Transformations

img = cv2.imread('images/1.jpg')
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

img_new = brightness_contrast(img, alpha = 1.3, beta = 20)
img_new = brightness_contrast(img, alpha = 0.7, beta = -10)

img_new = colorSpace(img, colorspace = 'hsv')
img_new = colorSpace(img, colorspace = 'ycrcb')
img_new = colorSpace(img, colorspace = 'lab')

img_new = addNoise(img, 'gaussian', mean = 0, var = 0.08)
img_new = addNoise(img, 'salt_pepper', sp_ratio = 0.5, noise_amount = 0.1)
img_new = addNoise(img, 'poisson', noise_amount = 0.5)
```

3. **Kernel-based features**

   - **blur(img, blur_type = 'avg', ksize = (5, 5), median_ksize = 5, gaussian_sigma = 0)** Returns blurred image. Three different types of blurring are supported - GAUSSIAN, Salt n Pepper, Poisson.

     1. **img =** ***numpy.ndarray*** Image to be blurred.
     2. **blur_type =** ***{'avg', 'gaussian', 'median'}*****, default = 'avg'** Type of blurring to perform.
     3. **ksize =** ***tuple of odd positive integers, (required only with blur_type = 'avg' or 'gaussian').*****, default = (5, 5)** kernel size used for average or gaussian blurring.
     4. **median_ksize =** ***int, odd positive integer, (required only with blur_type = 'median').*****, default = 5** kernel size used for median blurring.
     5. **gaussian_sigma =** ***int or float, (required only with blur_type = 'gaussian').*****, default = 0** Standard deviation used to calculate gaussian kernel.

   - **randomErase(img, size, box = None)** A random rectangular region is erased and replaced by mean value of image pixels. Returns modified image.

     1. **img =** ***numpy.ndarray*** Image to be modified.
     2. **size =** ***tuple of int*** Size of rectangular region to erase.
     3. **box =** ***list*** Coordinates of bounding box in the format - (x1,y1,x2,y2). If bounding box coordinates are passed, rectangular region is erased from the bounding box region.

- **randomCropAdd(img, size, box = None)** A random rectangular region is erased and added to another region of image. Returns modified image.
    1. **img =** ***numpy.ndarray*** Image to be modified.
    2. **size =** ***tuple of int*** Size of rectangular region to erase and add.
    3. **box =** ***list*** Coordinates of bounding box in the format - (x1,y1,x2,y2). If bounding box coordinates are passed, rectangular region is cropped from and added to the bounding box region.
- **sharpen(img)** Returns sharpened image.
    1. **img =** ***numpy.ndarray*** Image to be sharpened.

```
# Kernel-based Transformations

img = cv2.imread('images/0.jpeg')
bbox = [581, 274, 699, 321]

img_new = randomErase(img, size = (100, 100))

img_new = randomCropAdd(img, size = (100, 100))

img_new = sharpen(img)

img_new = randomErase(img, size = (60, 40), box = bbox)

img_new = randomCropAdd(img, size = (60, 40), box = bbox)

img_new = blur(img, 'avg', ksize = (9,9))
img_new = blur(img, 'gaussian', ksize = (9,9), gaussian_sigma = 0)
img_new = blur(img, 'median', median_ksize = 11)
```


ORIGINAL IMAGE
RANDOM ERASING
RANDOM ERASE AND ADD
SHARPENED IMAGE
RANDOM ERASE INSIDE BBOX
RANDOM ERASE AND ADD INSIDE BBOX
AVERAGE BLUR
GAUSSIAN BLUR
MEDIAN BLUR

CHAPTER 3

LICENSE