tensorflow接口研读image_op

3.1 tf.image.decode_gif(contents, name=None) {#decode_gif}

将GIF编码图片的第一帧解码成uint8类型的tensor。

参数:

- `contents`: ‘string’类型，GIF格式图片。
- `name`: 操作的名称。

3.2 tf.image.decode_jpeg(contents, channels=None, ratio=None, fancy_upscaling=None, try_recover_truncated=None, acceptable_fraction=None, dct_method=None, name=None)

将JPEG编码图片解码成uint8类型的tensor。

参数 `channels`表示解码图片期望的颜色通道数。 可配置的值为：

- 0: 使用JPEG编码图片本身的通道数。
- 1: 输出灰度图片
- 3: 输出RGB格式图片

如果需要，可以使用参数`ratio`对图片进行缩略处理。

参数:

- `contents`: `string`类型的JPEG编码图片。
- `channels`: `int`类型，默认为`0`。
- `ratio`: `int`类型，默认为`1`，下采样比率。

3.3 tf.image.encode_jpeg(image, format=None, quality=None, progressive=None, optimize_size=None, chroma_downsampling=None, density_unit=None, x_density=None, y_density=None, xmp_metadata=None, name=None)

进行JPEG图片格式编码。

`image` 必须是一个 3维[height, width, channels]格式、uint8数据类型的tensor。

`format` 可以用来重写颜色通道：

- `''`: 使用默认值，图片通道数和`image`通道数一致。
- `grayscale`: 输出JPEG编码的灰度图片。
- `rgb`: 输出JPEG编码的rgb图片。

参数:

- `image`: 3维 `[height, width, channels]`的tensor，uint8数据类型。
- `format`: 默认为''.

3.4 tf.image.decode_png(contents, channels=None, dtype=None, name=None)

将PNG编码的图片解码成uint8或uint16的tensor。
参数 `channels`表示解码图片期望的颜色通道数。 可配置的值为：

- 0: 使用JPEG编码图片本身的通道数。
- 1: 输出灰度图片
- 3: 输出RGB格式图片
- 4: 输出RGBA格式图片

参数:

- `contents`: PNG编码图片。
- `channels`: `int`类型，默认为`0`.

示例:

```
image_decoded=tf.image.decode_png(tf.read_file('example.png'),channels=3)
```

3.5 tf.image.encode_png(image, compression=None, name=None)

进行PNG图片格式编码。
`image` 为3维`[height, width, channels]`的uint8或uint16类型的tensor。
其中， `channels` 是:

- 1: 灰度格式
- 2: 灰度+alpha格式
- 3: RGB格式
- 4: RGBA格式

参数:

- `image`: 3维`[height, width, channels]`的uint8或uint16类型的tensor。

示例:

```
image_decoded=tf.image.decode_png(tf.read_file('example.png'),channels=3)
enc=tf.image.encode_png(image_decoded)
fname=tf.constant('1.png')
fwrite=tf.write_file(fname,enc)
with tf.Session() as sess:
  result=sess.run(fwrite)
```

3.6 tf.image.decode_image(contents, channels=None, name=None)

对`decode_gif`, `decode_jpeg`, and `decode_png`进行简化的函数，自动检测图片是GIG，JPEG或PNG。
注意: `decode_gif` 返回的是4维`[num_frames, height, width, 3]矩阵，而`decode_jpeg`和`decode_png`返回的是3维`[height, width, num_channels]`矩阵。

参数:

- **contents**: string类型。
- **channels**: int类型，默认为0。

3.7 tf.image.resize_images(images, size, method=0, align_corners=False)

使用方法method，将images大小调整至size，
调整图片大小可能造成图片扭曲，可以使用函数resize_image_with_crop_or_pad避免。
支持的方法method 如下：

- **ResizeMethod.BILINEAR**:双线性插值法。
- **ResizeMethod.NEAREST_NEIGHBOR**: 临近插值法。
- **ResizeMethod.BICUBIC**: 双三次插值法。
- **ResizeMethod.AREA**:面积插值法。

参数:

- **images**: 4维[batch, height, width, channels] 或3维 [height, width, channels]张量。
- **size**: 只含两个值的1维张量new_height, new_width，图片的新尺寸。
- **method**: 调整大小方法，默认为 ResizeMethod.BILINEAR.
- **align_corners**: bool型，默认为false，若为true， 精确对齐输入和输出的所有4个角。

示例:

```
resize_images1=tf.image.resize_images(image_decoded,[1200,1200],method=tf.image.ResizeMethod.BILINEAR)
#输出为float类型。
```

3.8 tf.image.resize_area(images, size, align_corners=None, name=None)

通过面积插值法将 images大小调整至size，输出为float类型。
参数:

- **images**: 可以为一下类型的4维 [batch, height, width, channels]tensor， uint8, int8, int16, int32, int64, half, float32, float64.
- **size**: 只含两个值的1维张量new_height, new_width，图片的新尺寸。

3.9 tf.image.resize_bicubic(images, size, align_corners=None, name=None)

通过双三次插值法将 images大小调整至size，输出为float类型。
参数:

- **images**: 可以为一下类型的4维 [batch, height, width, channels]tensor， uint8, int8, int16, int32, int64, half, float32, float64.
- **size**: 只含两个值的1维张量new_height, new_width，图片的新尺寸。

3.10 tf.image.resize_bilinear(images, size, align_corners=None,

name=None)

通过双线性插值法将 `images`大小调整至`size`，输出为float类型。

参数:

- **`images`**: 可以为一下类型的4维 `[batch, height, width, channels]`tensor， `uint8`, `int8`, `int16`, `int32`, `int64`, `half`, `float32`, `float64`.
- **`size`**: 只含两个值的1维张量`new_height, new_width`，图片的新尺寸。

3.11 tf.image.resize_nearest_neighbor(images, size, align_corners=None, name=None)

通过临近插值法将 `images`大小调整至`size`，输出为float类型。

参数:

- **`images`**: 可以为一下类型的4维 `[batch, height, width, channels]`tensor， `uint8`, `int8`, `int16`, `int32`, `int64`, `half`, `float32`, `float64`.
- **`size`**: 只含两个值的1维张量`new_height, new_width`，图片的新尺寸。

3.12 tf.image.resize_image_with_crop_or_pad(image, target_height, target_width)

裁剪或扩充图片尺寸至目标宽度以及高度。

如果图片的`width`或`height`大于`target_width`或`target_height`，该函数会沿着对应的尺度进行中心裁剪。

如果图片的`width`或`height`小于`target_width`或`target_height`，该函数会沿着对应的尺度进行中心扩充。

参数:

- **`image`**: 3维`[height, width, channels]`tensor。
- **`target_height`**: 目标高度。
- **`target_width`**: 目标宽度。

示例:

```
resize_images5=tf.image.resize_image_with_crop_or_pad(image_decoded,300,300)
```

3.13 tf.image.central_crop(image, central_fraction)

对图片进行中心裁剪。

保留图片每个尺度的中心区域，对外部进行裁剪。如果我们配置`central_fraction`为0.5，该函数只会返回下图标识X的数据。

```
 --------
|        |
|  XXXX  |
|  XXXX  |
|        |   "X"是图片50%处于中间的部分。
 --------
```

参数:

- **`image`**: 3维 [height, width, depth]tensor。
- **`central_fraction`**: float (0, 1]，裁剪比例。

示例:

```
central=tf.image.central_crop(image_decoded,0.5)
```

3.14 tf.image.pad_to_bounding_box(image, offset_height, offset_width, target_height, target_width)

扩展 `image` 到指定的 `height` 和`width`，扩展的部分填充0。
在图片的上部填充`offset_height`行0元素和在图片的左边填充`offset_width`列0元素后，将图片沿着下部行填充和右部列填充扩展至指定的高、宽。

参数:

- **`image`**: 3维 `[height, width, channels]`tensor。
- **`offset_height`**: 上部增加0元素的行数。
- **`offset_width`**: 左部增加0元素的行数。
- **`target_height`**: 输出图片的高度。
- **`target_width`**: 输出图片的宽度。

示例：

```
pad_to=tf.image.pad_to_bounding_box(image_decoded,100,100,2000,2000)
```

3.15 tf.image.crop_to_bounding_box(image, offset_height, offset_width, target_height, target_width)

图片指定的范围裁剪，保留指定范围内的数据。
该函数从`image`中裁剪出一个长方形的区域。长方形左上角在图片中的坐标为`offset_height, offset_width`
那么它的右下角在图片中的坐标为`offset_height + target_height, offset_width + target_width`。

参数:

- **`image`**: 3维 `[height, width, channels]`tensor。
- **`offset_height`**: 左上角坐标的高度值。
- **`offset_width`**: 左上角坐标的宽度值。
- **`target_height`**: 输出图片的高度。
- **`target_width`**: 输出图片的宽度。

示例:

```
crop_to=tf.image.crop_to_bounding_box(image_decoded,100,100,400,400)
```

3.16 tf.image.extract_glimpse(input, size, offsets, centered=None, normalized=None, uniform_noise=None, name=None)

与crop_to_bounding_box功能类似。不同之处在于，input是一组4维张量 `[batch_size, height, width, channels]`，存有batch_size张图片。该函数对每张图片截取相同大小不同位置的数据。
截取后的tensor为 `[batch_size, glimpse_height,glimpse_width, channels]`。

- 若坐标不是标准化和中心指定的, 0.0 and 1.0对应最小和最大的高度或宽度。
- 若坐标是标准化和中心指定的，坐标 (-1.0, -1.0) 对应左上角，右下角对应坐标(1.0, 1.0)。
- 若坐标不是标准化，则应使用像素坐标值。

参数:

- **input**: 4维[batch_size, height, width, channels]tensor。
- **size**: 1维int32型，表示截取窗口大小，[glimpse_height,glimpse_width]。
- **offsets**: 每个截取窗口中心位置的坐标，float32型[batch_size,2]。
- **centered**: bool型，默认为True，表明offset为窗口中心位置坐标;若为False,则表示窗口左上角坐标。
- **normalized**: bool型，默认为 True，表示偏移坐标是标准化的。
- **uniform_noise**: bool型，默认为 True，表示截取是产生噪声。

示例:

```
size=tf.Variable([200,200])
offsets=tf.constant([[64,64],[159,155],[400,600]],dtype=tf.float32)
extract_glimpse=tf.image.extract_glimpse(images,size,offsets,centered=False,normalized=False,uniform_noise=False)
```

3.17 tf.image.crop_and_resize(image, boxes, box_ind, crop_size, method=None, extrapolation_value=None, name=None)

类似于crop_to_bounding_box，不同之处在于，本函数在裁剪后将所有图片重新调整为指定尺寸，指定尺寸由参数crop_size获取。

参数:

- **image**:4维[batch, image_height, image_width, depth]tensor。
- **boxes**: 2维[num_boxes, 4] float32类型tensor。第‘i’行为标准化坐标[y1, x1, y2, x2],标准化坐标值y对应图片坐标y * (image_height - 1),因此，[0,1]对应图片`[0, image_height - 1]。通常，要求y2>y1，x2>x1。
- **box_ind**: 1维[num_boxes]int32型tensor，取值范围[0, batch)`。该值指定box对应第几张图片。
- **crop_size**: 1维size = [crop_height, crop_width]``int32型tensor,所有裁剪后的图片尺寸将全部改变为该值。
- **method**: 调整图片大小的方法，默认为 bilinear，目前也只支持这种方法。

3.18 tf.image.flip_up_down(image)

将图片上下翻转。即翻转坐标为height。

参数:

- **image**: 3维[height, width, channels]tensor。

3.19 tf.image.random_flip_up_down(image, seed=None)

随机将图片进行上下翻转，翻转概率为50%。

参数:

- **image**: 3维[height, width, channels]tensor。
- **seed**: 随机种子，用法在constant_ops中有介绍。

3.20 tf.image.flip_left_right(image)

将图片左右翻转。即翻转坐标为width。

参数:

- `image`: 3维`[height, width, channels]`tensor。

示例：

```
flip_left_right=tf.image.flip_left_right(image_decoded)
```

3.21 tf.image.random_flip_left_right(image, seed=None)

随机将图片进行左右翻转，翻转概率为50%。

参数:

- `image`: 3维`[height, width, channels]`tensor。
- `seed`: 随机种子，用法在constant_ops中有介绍。

3.22 tf.image.transpose_image(image)

对图片第一第二维度进行转置操作。

参数:

- `image`:3维`[height, width, channels]`tensor。完成为3维`[width,height, channels]`tensor。

示例：

```
transpose_image=tf.image.transpose_image(image_decoded)
```

3.23 tf.image.rot90(image, k=1, name=None)

将图片逆时针旋转，步长为90度。

参数:

- `image`:3维`[height, width, channels]`tensor。
- `k`:旋转倍率，默认为1。图片旋转角度为`k*90`。

示例：

```
rot180=tf.image.rot90(image_decoded,k=2)
```

3.24 tf.image.rgb_to_grayscale(images, name=None)

将一个或多个图片由RGB模式转化为灰度模式。

参数:

- `images`: RGB tensor，最末维度必须为3，对应RGB三个通道。

3.25 tf.image.grayscale_to_rgb(images, name=None)

将一个或多个图片由灰度模式转化为RGB模式。

参数:

- `images`: 灰度 tensor，最末维度必须为1。

3.26 tf.image.hsv_to_rgb(images, name=None)

将一个或多个图片由HSV模式转化为RGB模式。

参数:

- `images`: 最后一个维度必须为3。

3.27 tf.image.rgb_to_hsv(images, name=None)

将一个或多个图片由RGB模式转化为HSV模式。

参数:

- `images`: 最后一个维度必须为3。

3.28 `tf.image.convert_image_dtype(image, dtype, saturate=False, name=None)

转化图片数据类型至`dtype`。并将数据归一为 [0,1)。

参数:

- `image`: 图片
- `dtype`: 将要转化成的数据类型。
- `saturate`: 如果 `True`,在转化前裁剪越限的值

示例:

```
image_decoded=tf.image.decode_png(tf.read_file('871002.png'),channels=1)
max1=tf.reduce_max(image_decoded)#max1=215
convert=tf.image.convert_image_dtype(image_decoded,tf.float32)
max=tf.reduce_max(convert)#max=0.843137(215/255)
```

3.29 tf.image.adjust_brightness(image, delta)

调整RGB或灰度图的明暗度。

在运算前，image和delta都先转换为float类型，运算完成后再返回初始类型。delta取值范围为[0,1)。

参数:

- `image`: tensor.
- `delta`: image_out=image*delta。

示例:

```
adjust_brightness=tf.image.adjust_brightness(image_decoded,0.4)
```

3.30 tf.image.random_brightness(image, max_delta, seed=None)

随机调整RGB或灰度图的明暗度。随机值范围 `[-max_delta, max_delta)`。

参数:

- `image`:image.
- `max_delta`: float, 必须为非负。
- `seed`: 种子。

3.31 tf.image.adjust_contrast(images, contrast_factor)

调整RGB或灰度图的对比度。

在运算前，image和delta都先转换为float类型，运算完成后再返回初始类型。

对图片而言，每个通道的对比度调节是独立的。该函数先计算该通道像素平均值mean，而后对每个值进行运算

`(x - mean) * contrast_factor + mean`.

参数：

- `images`: Images，至少为 3维。
- `contrast_factor`: 调整因子。

示例：

```
adjust_contrast=tf.image.adjust_contrast(image_decoded,0.4)
```

3.32 tf.image.random_contrast(image, lower, upper, seed=None)

随机调整RGB或灰度图的对比度。对比于`adjust_contrast`， `contrast_factor`从`[lower,upper]`中随机取值。

参数：

- `image`: Images，至少为 3维。
- `lower`: float. 随机调整因子的最低值。
- `upper`: float. 随机调整因子的最高值。
- `seed`:种子。

3.33 tf.image.adjust_hue(image, delta, name=None)

调节RGB图像的色彩。

该函数将图片先转换为float类型，之而转换为HSV模式，对HSV模式中的hue通道进行运算，完成后再转回RGB模式，乃至原始数据类型。

参数：

- `image`: RGB格式图片，最末尺度必须为3。
- `delta`: float. 添加到hue通道的值，必须在[-1,1]之间。

示例：

```
adjust_hue=tf.image.adjust_hue(image_decoded,delta=0.4)
```

3.34 tf.image.random_hue(image, max_delta, seed=None)

随机调节RGB图像的色彩。随机delta值，范围为`[-max_delta, max_delta]`.

参数：

- `image`: RGB格式图片，最末尺度必须为3。
- `max_delta`: float. 最大delta。
- `seed`: 种子。

3.35 tf.image.adjust_gamma(image, gamma=1, gain=1)

进行灰度矫正。Out = In*gamma。

若gamma>1，图片将变暗;若gamma<1，图片将变亮;

参数：

- image : A Tensor.
- gamma : A scalar. 非负实数。
- gain : A scalar. The constant multiplier.

示例：

```
adjust_gamma=tf.image.adjust_gamma(tf.cast(image_decoded,dtype=tf.float32),0.8)
```

3.36 tf.image.adjust_saturation(image, saturation_factor, name=None)

调节RGB图像的饱和度。

该函数将图片先转换为float类型，之而转换为HSV模式，对HSV模式中的saturation通道进行运算，完成后再转回RGB模式，乃至原始数据类型。

参数：

- `image`: RGB格式图片，最末尺度必须为3。
- `saturation_factor`: float. 饱和因子。

示例：

```
adjust_saturation=tf.image.adjust_saturation(image_decoded,0.4)
```

3.37 tf.image.random_saturation(image, lower, upper, seed=None)

随机调节RGB图像的饱和度。`saturation_factor`随机在`[lower,upper]`中取值。

参数：

- `image`: RGB image or images. Size of the last dimension must be 3.
- `lower`: float. 饱和因子最小值。
- `upper`: float. 饱和因子最大值。
- `seed`: 种子。

3.38 tf.image.per_image_standardization(image)

将图片标准化，`out=(x - mean) / adjusted_stddev`。

其中，`mean`是图片所有值的平均值。

`adjusted_stddev = max(stddev, 1.0/sqrt(image.NumElements()))`.

`stddev` 是图片所有值的标准方差。

参数：

- `image`: 3维 `[height, width, channels]`tensor。

3.39 tf.image.draw_bounding_boxes(images, boxes, name=None)

给一批图片绘制方框。

每张图片绘制相同数量的方框，方框在图片中的坐标定义为 `[y_min, x_min, y_max, x_max]`，取值都在`[0.0, 1.0]`之间。

例如，一幅为100 x 200像素的图像，绘制的方框坐标为`[0.1, 0.2, 0.5, 0.9]`, 那么它的左下角及右上角的像素点位置为 `(10, 40)` to `(50, 180)`。

参数：

- `images`: 4维 `[batch, height, width, depth]`tensor。
- `boxes`: 3维 `[batch, num_bounding_boxes, 4]` 包含需绘制的方框的信息。

示例：

```
#images中batch为3。
boxes=tf.Variable([[[0.1,0.2,0.5,0.9]],[[0.3,0.4,0.6,0.5]],[[0.5,0.6,0.7,0.8]]])
```

```
draw_bounding_boxes=tf.image.draw_bounding_boxes(images,boxes)
```