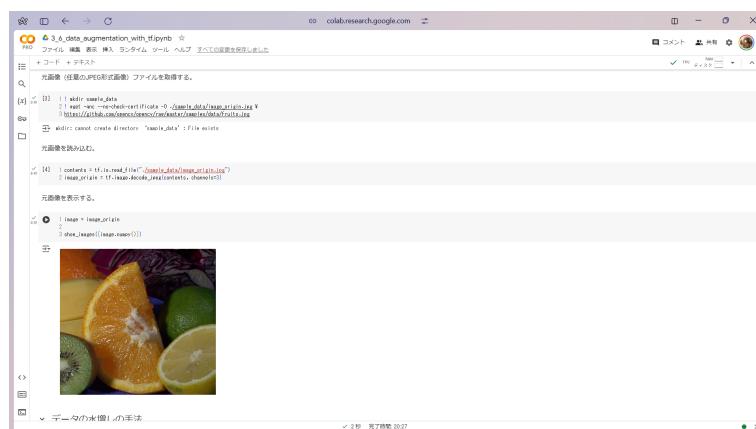


第1部

深層学習 day3 実装演習キャプチャ 2

1 3_6_data_augmentation_with_tf

```
# データの水増し
(x)
# 画像認識精度を向上のためには、あらゆるパターンを想定した画像を用意することが理想である。しかし、それは現実的に困難であるので、代替として手作による画像から解像的に別の画像を生成するというアプローチが採られるケースが多い。このアプローチのことを、データの水増し、またはデータ拡張 (data augmentation) という。具体的な手法として、画像の回転処理や色転換処理などが挙げられる。ただし、画像によっては適さない手法がある。例えば、反転処理すると「6」と「9」が、回転処理すると「8」と「0」が識別できなくなる。
TensorFlow等のライブラリをインポートする。
<cell_in>
[1]: # 一般的な処理でNumPyを使用して記述
  1 import numpy as np
  2 # テンソル操作し操作可能な型を有するライブラリ
  3 import tensorflow as tf
  4 # 離散乱数を生成するモジュール
  5 import random
  6 # 亂数生成器を生成するライブラリ
  7 import tensorflow as tf
  8 # 画像処理用のライブラリ
  9 import tensorflow_datasets as tfds
 10 # 画像処理用のライブラリ
11 # Data augmentation
12
13 # 画像を表示するshow_image関数を定義する。
<cell_in>
[2]: [1] def show_image(images):
    2     """画像の複数枚を表示する"""
    3     n = 1
    4     while n <= 2 << len(images):
    5         images[n].show()
    6
13 # 画像を表示する。
<cell_out>
[2]: ✓ 2秒 完了時間: 2027
```



```
✓ 2秒 完了時間: 2027
<cell_in>
[1]: # データの水増しの手法
<cell_in>
[2]: # Horizontal Flip
<cell_in>
[3]: # Vertical Flip
1 # 画像を表示する。
2 image = tf.image_random_flip_left_right(image)
3 show_image([image, image])
1 # 画像を表示する。
2 image = tf.image_random_flip_up_down(image)
3 show_image([image, image])
1 # 画像を表示する。
2 image = tf.image_random_flip_left_right(image)
3 show_image([image, image])
```



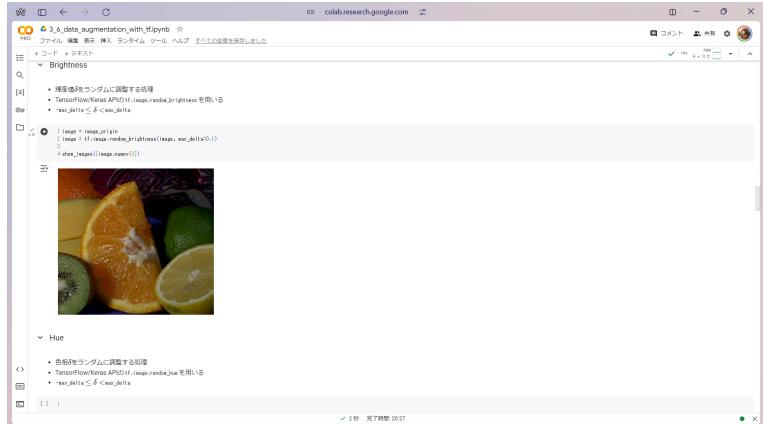
```
# [4]垂直方向(上下)反転処理
# &gt; TensorFlow/Keras APIのtf.image.random_flip_left_rightを用いる
# &gt; 引数xminで、切り出す画像のサイズを指定
[4]: image = tf.image.flip_left_right(image)
      image = tf.image_random_flip_left_right(image, seed=42)
      show_image([image.name])
```



```
# [5]あるサイズを画像からランダムに切り出す処理
# &gt; TensorFlow/Keras APIのtf.image.random_cropを用いる
# &gt; 引数xminで、切り出す画像のサイズを指定
[5]: image = tf.image_random_crop(image, size=[10, 10, 3], seed=42)
      show_image([image.name])
```



```
# [6]コントラストをランダムに調整する処理
# &gt; TensorFlow/Keras APIのtf.image.random_contrastを用いる
# &gt; 引数lowerとupperで、コントラストの係数の下限値と上限値を指定
[6]: image = tf.image_random_contrast(image, lower=0.4, upper=0.8)
      show_image([image.name])
```



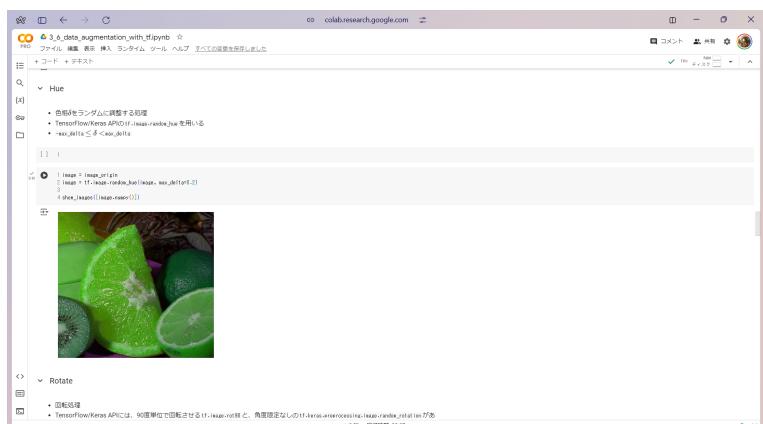
```

[4]: 
 1 image = image_orig[0]
 2 image = tf.image.random_brightness(image, max_delta=0.1)
 3 image = tf.image.resize([image], [image_size])
 4 show_image([image.numpy()])

```

Brightness

- 画像をランダムに調整する処理
- TensorFlow/Keras APIのtf.image.random_brightness()を用いる
- max_delta ≤ δ < max_delta



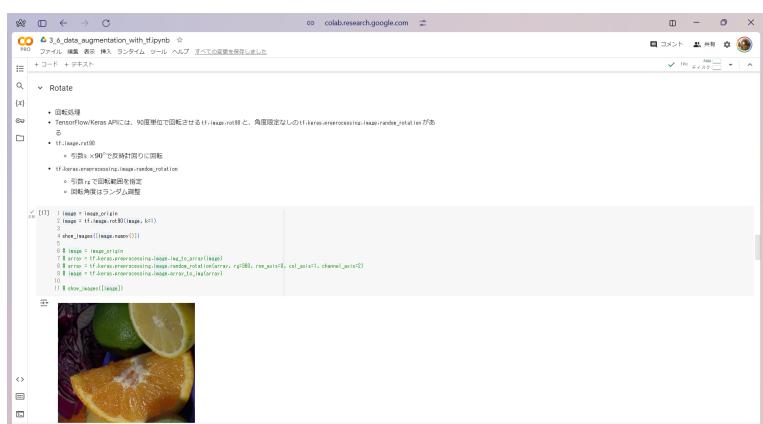
```

[4]: 
 1 image = image_orig[0]
 2 image = tf.image.random_hue(image, max_delta=0.1)
 3 image = tf.image.resize([image], [image_size])
 4 show_image([image.numpy()])

```

Hue

- 画像をランダムに調整する処理
- TensorFlow/Keras APIのtf.image.random_hue()を用いる
- max_delta ≤ δ < max_delta



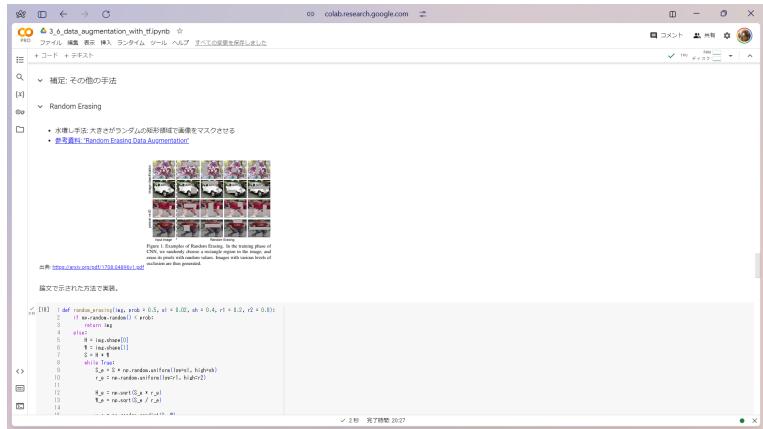
```

[4]: 
 1 image = image_orig[0]
 2 image = tf.image.rotate(image, 90)
 3 image = tf.image.resize([image], [image_size])
 4 show_image([image.numpy()])

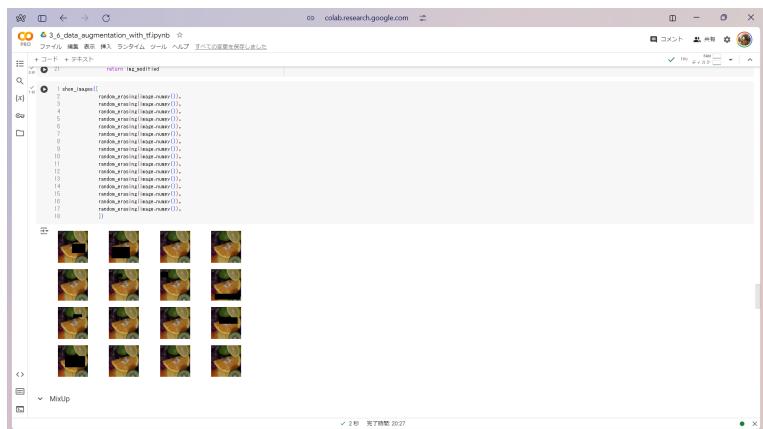
```

Rotate

- 回転処理
- TensorFlow/Keras APIには、90度単位で回転させるtf.keras.preprocessing.image.rotate()がある
- tf.image.rotate()
 - 引数x: 90°で回転する度数
 - tf.keras.preprocessing.image.rotate()
 - 引数x: 回転度数を定義
 - 引数y: 回転度数を定義
 - 回転角度はランダム調整



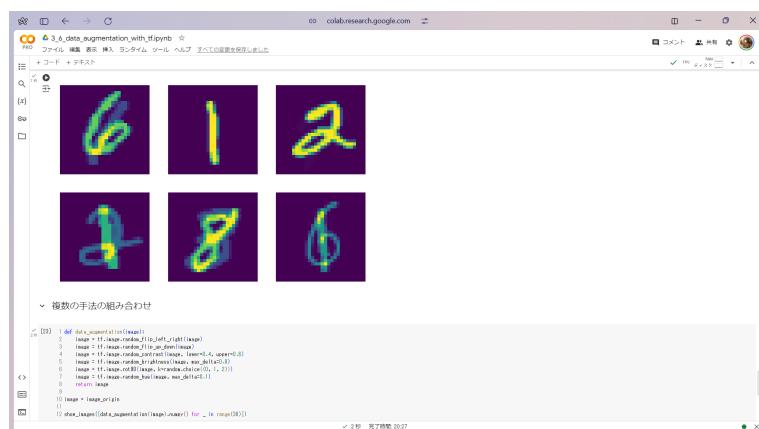
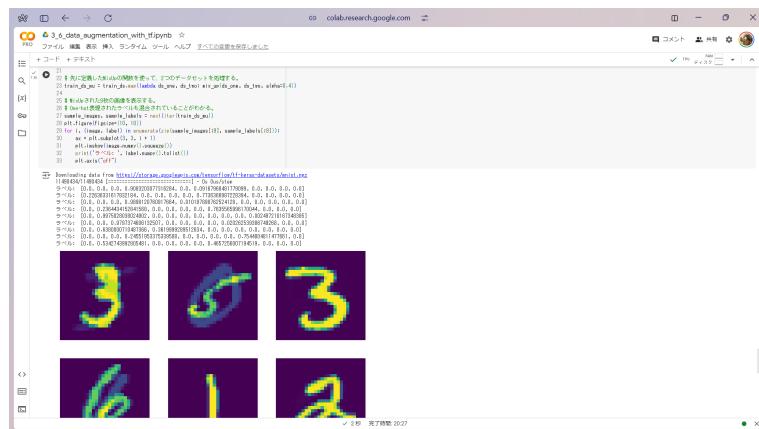
```
# [markdown] I'd like to introduce "Random Erasing" to the training process of CNN, so randomly choose a rectangle region in the image, and then set it to black. Images with random erasing scheme are shown below.  
出典: https://arxiv.org/pdf/1704.08197.pdf  
論文で示された方法で実装。  
[1]:  
def random_erasing(img, r1=0.5, s1=1.02, s2=1.4, r2=1.2, r3=0.8):  
    if random.random() < r1:  
        return img  
    else:  
        h, w = img.shape[0], img.shape[1]  
        x1 = int(w * random.uniform(r2, r3))  
        y1 = int(h * random.uniform(s1, s2))  
        x2 = min(x1 + int(w * random.uniform(r1, r2)), w)  
        y2 = min(y1 + int(h * random.uniform(r1, r2)), h)  
        x3 = max(x1 - int(w * random.uniform(r3, r4)), 0)  
        y3 = max(y1 - int(h * random.uniform(r3, r4)), 0)  
        x4 = min(x3 + int(w * random.uniform(r1, r2)), w)  
        y4 = min(y3 + int(h * random.uniform(r1, r2)), h)  
        r = random.randint(0, 255)  
        g = random.randint(0, 255)  
        b = random.randint(0, 255)  
        img[y1:y2, x1:x2] = [r, g, b]  
        img[y3:y4, x3:x4] = [r, g, b]  
    return img
```

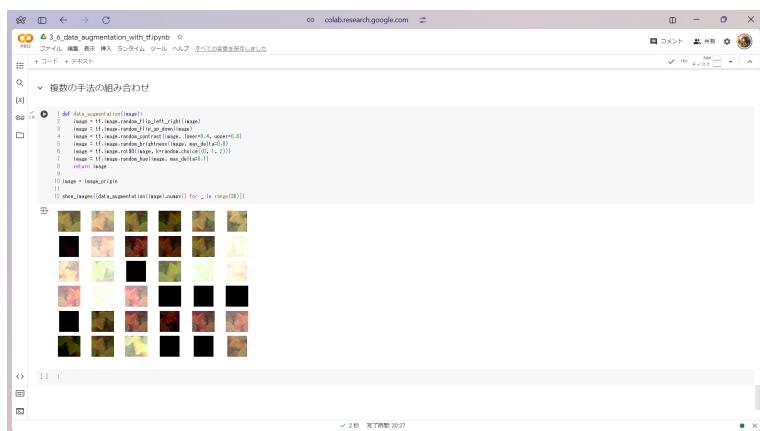


```
[1]:  
def mixup_data(images, labels, alpha=1.0):  
    if alpha == 1:  
        return images, labels  
    L = len(images)  
    index = np.random.choice(L, size=L)  
    mixed_images, mixed_labels = np.zeros_like(images), np.zeros_like(labels)  
    for i in range(L):  
        image, label = images[i], labels[i]  
        alpha_i = np.random.beta(alpha, alpha)  
        mixed_image = alpha_i * image + (1 - alpha_i) * images[index[i]]  
        mixed_label = alpha_i * label + (1 - alpha_i) * labels[index[i]]  
        mixed_images[i] = mixed_image  
        mixed_labels[i] = mixed_label  
    return mixed_images, mixed_labels
```



```
[1]:  
def mixup_data(images, labels, alpha=1.0):  
    if alpha == 1:  
        return images, labels  
    L = len(images)  
    index = np.random.choice(L, size=L)  
    mixed_images, mixed_labels = np.zeros_like(images), np.zeros_like(labels)  
    for i in range(L):  
        image, label = images[i], labels[i]  
        alpha_i = np.random.beta(alpha, alpha)  
        mixed_image = alpha_i * image + (1 - alpha_i) * images[index[i]]  
        mixed_label = alpha_i * label + (1 - alpha_i) * labels[index[i]]  
        mixed_images[i] = mixed_image  
        mixed_labels[i] = mixed_label  
    return mixed_images, mixed_labels
```





The screenshot shows a Google Colab notebook titled "3_6_data_augmentation_with_tf.ipynb". The code cell contains Python code for data augmentation using TensorFlow's `tf.image` module. The output of the code is a 4x8 grid of 32x32 images, each showing a different image of a leaf or flower with various transformations applied, such as rotations and color shifts.

```
#(1) #if data augmentation is used
#(2) image = tf.image.random_flip_left_right(image)
#(3) image = tf.image.random_flip_up_down(image)
#(4) image = tf.image.random_brightness(image, max_delta=32.0)
#(5) image = tf.image.random_contrast(image, lower=0.2, upper=1.8)
#(6) image = tf.image.random_hue(image, max_delta=0.2)
#(7) image = tf.image.random_saturation(image, lower=0.2, upper=1.8)
#(8) image = tf.image.random_center_crop(image, [128, 128])
#(9) print(image)
#(10) image = image_to_gif()
#(11) data_images = [data_to_numpy(image) for _ in range(50)]
```

