Blending and Pasting Images

For some computer vision systems, we'll want to be able to post our own image on top of an already existing image or video. We may also want to blend images, maybe we want to have a "highlight" effect instead of just a solid box or empty rectangle.

Let's explore what is commonly known as **Arithmetic Image Operations** with OpenCV. These are referred to as Arithmetic Operations because OpenCV is simply performing some common math with the pixels for the final effect. We'll see a bit of this in our code.

Blending Images

Blending Images is actually quite simple, let's look at a simple example.

```
import cv2
```

```
# Two images
img1 = cv2.imread('/content/dog_backpack.jpg')
img2 = cv2.imread('/content/watermark_no_copy.png')
```

```
img1.shape
```

```
(1401, 934, 3)
```

```
img2.shape
```

```
(1280, 1277, 3)
```

```
import matplotlib.pyplot as plt
%matplotlib inline
```

```
plt.imshow(img1)
```

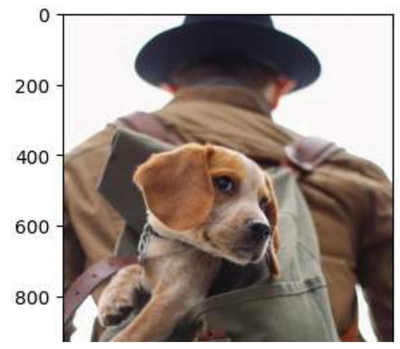
```
<matplotlib.image.AxesImage at 0x7dd4762250f0>
```

Whoops! Let's remember to fix the RGB!

```
img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)
img2 = cv2.cvtColor(img2, cv2.COLOR_BGR2RGB)
```

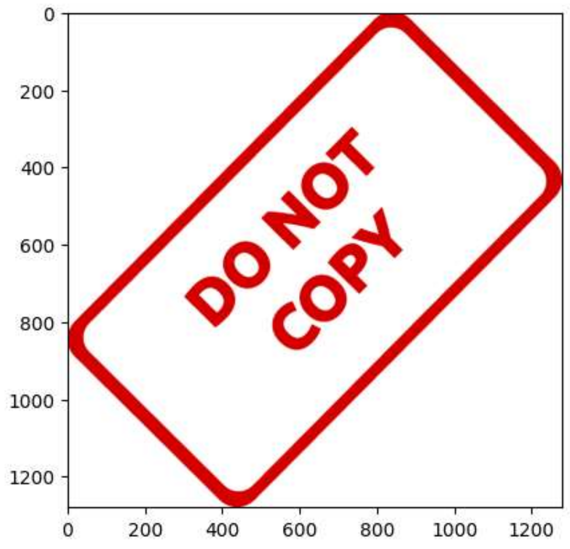
```
plt.imshow(img1)
```

```
<matplotlib.image.AxesImage at 0x7dd47614b250>
```



```
plt.imshow(img2)
```

```
<matplotlib.image.AxesImage at 0x7dd4761fcf70>
```



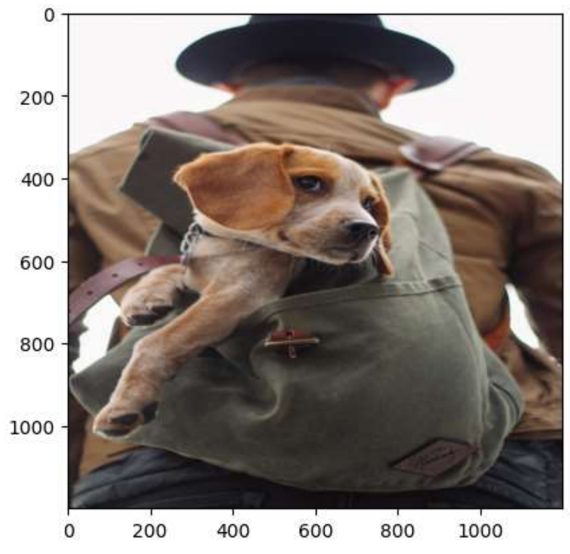
Resizing the Images

```
img1 =cv2.resize(img1,(1200,1200))
img2 =cv2.resize(img2,(1200,1200))
```

Let's practice resizing the image, since the DO NOT COPY image is actually quite large 1200 by 1200, and our puppy in backpack image is 1400 by 1000

```
plt.imshow(img1)
```

```
<matplotlib.image.AxesImage at 0x7dd4760b6fe0>
```



```
plt.imshow(img2)
```

```
<matplotlib.image.AxesImage at 0x7dd475f403a0>
```

DO NOT COPY
0
200
400
600
800
1000

Blending the Image

We will blend the values together with the formula:

$$img1 * \alpha + img2 * \beta + \gamma$$

```
img1.shape
```

```
(1200, 1200, 3)
```

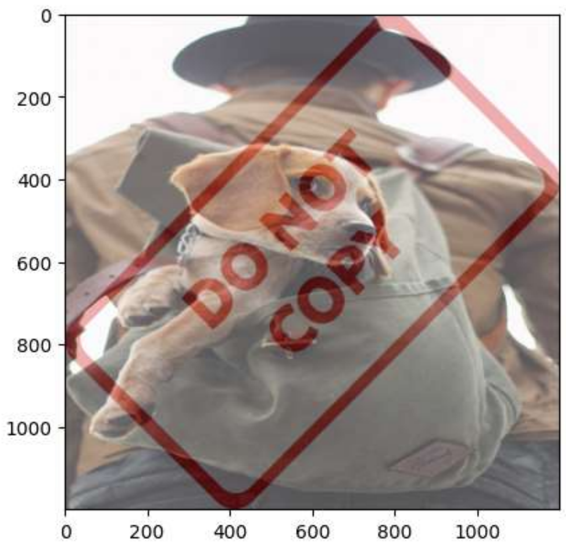
```
img2.shape
```

```
(1200, 1200, 3)
```

```
blended = cv2.addWeighted(src1=img1,alpha=0.7,src2=img2,beta=0.3,gamma=0)
```

```
plt.imshow(blended)
```

```
<matplotlib.image.AxesImage at 0x7dd475fb02e0>
```



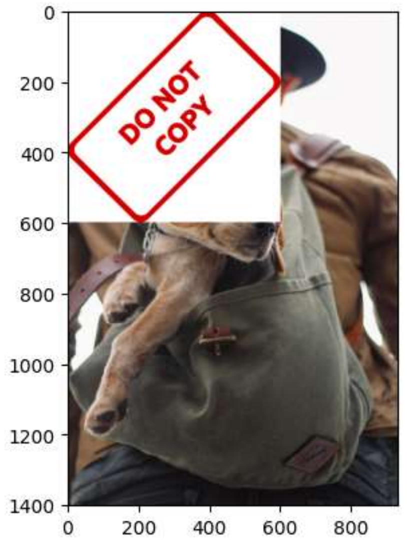
Overlaying Images of Different Sizes

We can use this quick trick to quickly overlap different sized images, by simply reassigning the larger image's values to match the smaller image.

```
# Load two images
img1 = cv2.imread('/content/dog_backpack.jpg')
img2 = cv2.imread('/content/watermark_no_copy.png')
img2 =cv2.resize(img2,(600,600))
```

```
img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)
img2 = cv2.cvtColor(img2, cv2.COLOR_BGR2RGB)

large_img = img1
small_img = img2


x_offset=0
y_offset=0


large_img[y_offset:y_offset+small_img.shape[0], x_offset:x_offset+small_img.shape[1]] = small_img


plt.imshow(large_img)
```

```
<matplotlib.image.AxesImage at 0x7dd475e20730>
```



Blending Images of Different Sizes

Imports

```
import numpy as np
import cv2
```

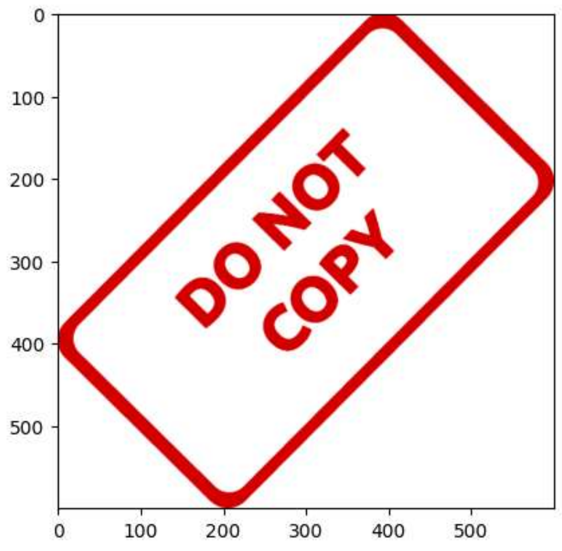
Importing the images again and resizing

```
# Load two images
img1 = cv2.imread('/content/dog_backpack.jpg')
img2 = cv2.imread('/content/watermark_no_copy.png')
img2 =cv2.resize(img2,(600,600))

img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)
img2 = cv2.cvtColor(img2, cv2.COLOR_BGR2RGB)


plt.imshow(img1)
```

```
<matplotlib.image.AxesImage at 0x7dd475f0e0b0>
```

```
plt.imshow(img2)
```

```
<matplotlib.image.AxesImage at 0x7dd475ddbc70>
```



Create a Region of Interest (ROI)

```
img1.shape
```

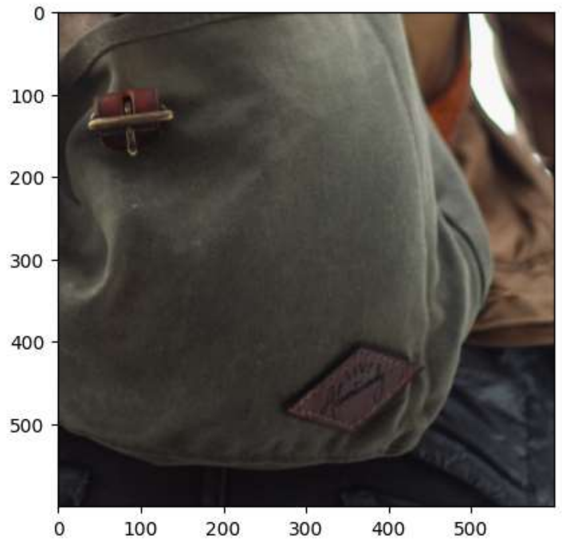
```
(1401, 934, 3)
```

```
x_offset=934-600
y_offset=1401-600
```

```
# Creating an ROI of the same size of the foreground image (smaller image that will go on top)
rows,cols,channels = img2.shape
# roi = img1[0:rows, 0:cols ] # TOP LEFT CORNER
roi = img1[y_offset:1401,x_offset:943] # BOTTOM RIGHT CORNER
```

```
plt.imshow(roi)
```

```
<matplotlib.image.AxesImage at 0x7dd475cd5cc0>
```



```
roi.shape
```

```
(600, 600, 3)
```

Creating a Mask

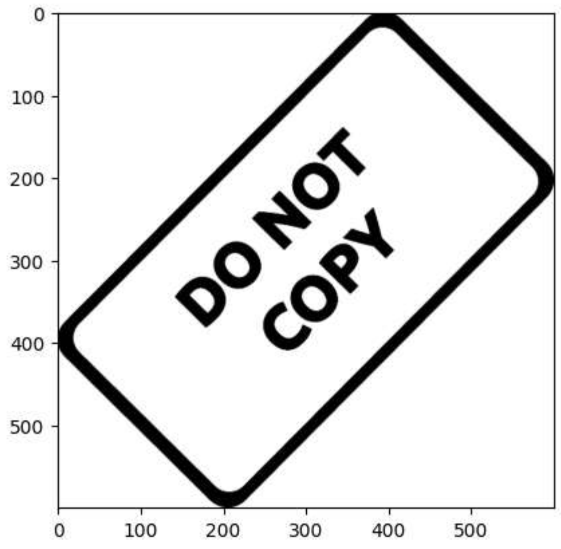
```
# Now create a mask of logo and create its inverse mask also
img2gray = cv2.cvtColor(img2,cv2.COLOR_BGR2GRAY)
```

```
img2gray.shape
```

```
(600, 600)
```

```
plt.imshow(img2gray,cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7dd475b47340>
```



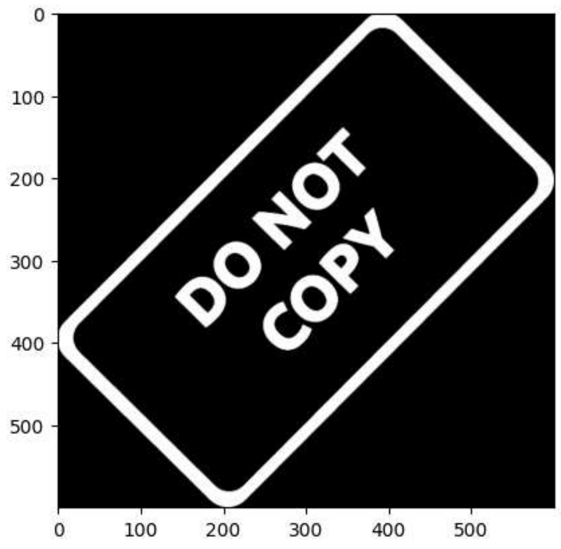
```
mask_inv = cv2.bitwise_not(img2gray)
```

```
mask_inv.shape
```

```
(600, 600)
```

```
plt.imshow(mask_inv,cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7dd475bbf820>
```



Convert Mask to have 3 channels

```
white_background = np.full(img2.shape, 255, dtype=np.uint8)
```

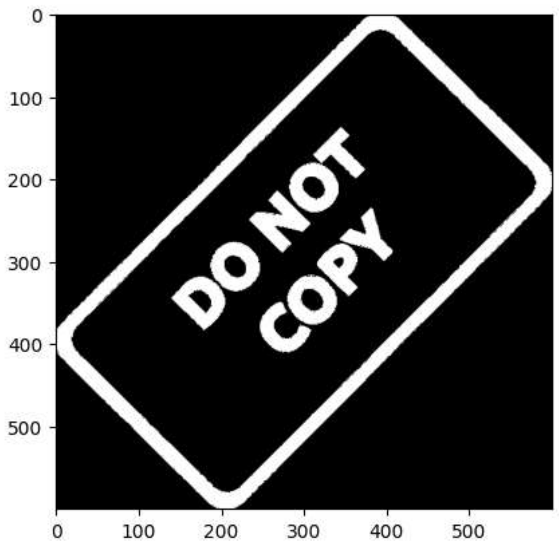
```
bk = cv2.bitwise_or(white_background, white_background, mask=mask_inv)
```

```
bk.shape
```

```
(600, 600, 3)
```

```
plt.imshow(bk)
```

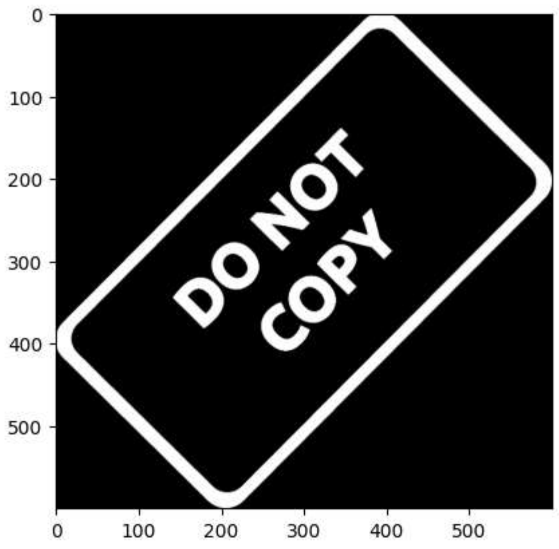
```
<matplotlib.image.AxesImage at 0x7dd475a48b80>
```



Grab Original FG image and place on top of Mask

```
plt.imshow(mask_inv,cmap='gray')
```

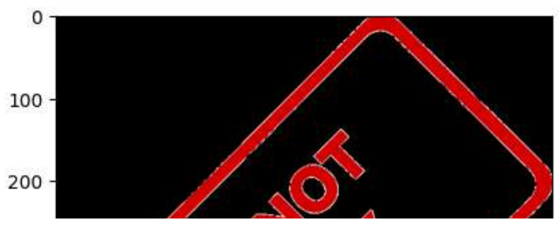
```
<matplotlib.image.AxesImage at 0x7dd475ac10f0>
```



```
fg = cv2.bitwise_or(img2, img2, mask=mask_inv)
```

```
plt.imshow(fg)
```

```
<matplotlib.image.AxesImage at 0x7dd47592a740>
```



```
fg.shape
```

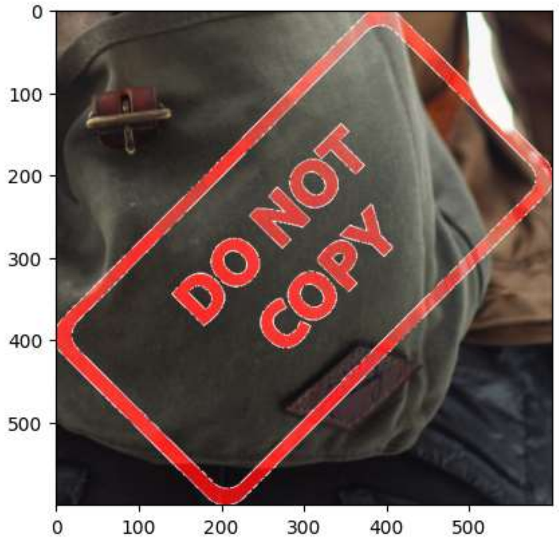
```
(600, 600, 3)
```

Get ROI and blend in the mask with the ROI

```
final_roi = cv2.bitwise_or(roi,fg)
```

```
plt.imshow(final_roi)
```

```
<matplotlib.image.AxesImage at 0x7dd475999330>
```



Now add in the rest of the image

```
large_img = img1
small_img = final_roi
```

```
large_img[y_offset:y_offset+small_img.shape[0], x_offset:x_offset+small_img.shape[1]] = small_img
```

```
plt.imshow(large_img)
```

```
<matplotlib.image.AxesImage at 0x7dd475824790>
```

Great Work!

Check out these documentation examples and links for more help for these kinds of tasks (which can be really tricky!)

1. https://stackoverflow.com/questions/10469235/opencv-apply-mask-to-a-color-image/38493075
2. https://stackoverflow.com/questions/14063070/overlay-a-smaller-image-on-a-larger-image-python-opencv
3. https://docs.opencv.org/3.4/d0/d86/tutorial_py_image_arithmetics.html

800
1000
1200
1400
0
200
400
600
800
DO NOT COPY

✓ 0s completed at 10:47 AM