

Homework #3

Deadline: November 29, 2024 @23:59

Submissions: (1) PDF version of this file
(2) .ipynb file; template in this link

https://colab.research.google.com/drive/1OJZBrQXyv37Fy88vaSX8em6_9tqRl8Yg?usp=sharing

Download data in colab

IMPORTANT! (1) Before submitting the python file, please make sure it can be successfully compiled and correctly in its format name

(2) The scores will be 0 for all students whose source codes are very similar to each other.

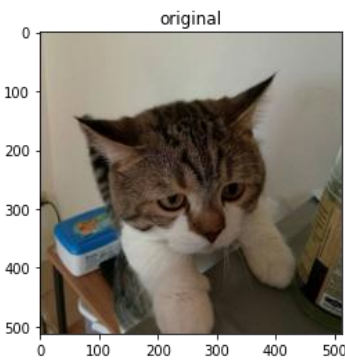

1. (10 points) Image Augmentation (no need to put in COLAB)

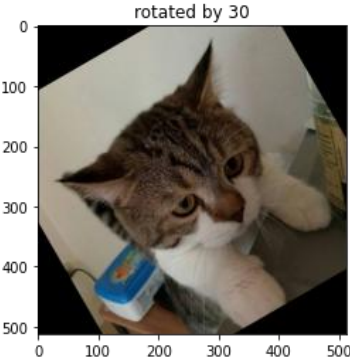
From image classification in object recognition lecture, the self-learning features are capable of learning object patterns using deep neural networks. However, you may require large number of datasets for a model to learn. Study image augmentation, which can be used to expand the size of the training dataset. Design 5 modified versions of an original image using knowledge from image processing class, so you can use them for model learning.




Implement 5 different image variations for general object classification task using the example and the template below.


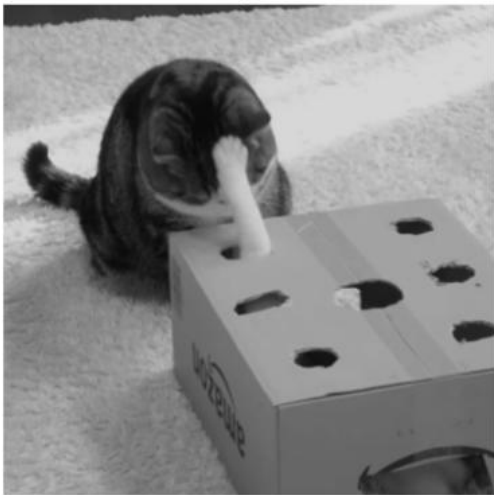
The example below is a modified version of an original image using rotation which can have variations in terms of orientation angle of an image from 0-360 degrees.

You should use a kitty image and one additional selected image and displayed in the table below. Also once call your python file, it should show the results of your 5 different modified versions of the original image.

Original image	<p>Kitty.jpg</p> 	<p>Your selected image</p> 
----------------	--	---

No.	Modification Techniques & applied images (kitty.jpg and your selected image)	Purpose and variations
1	<p>Example: Rotation</p> 	<p>Rotation can imitate the real dataset that the object can be varied in orientations.</p> <p>Variations: angles in range of [0,360)</p>
	<p>Code:</p> <pre>def rotated(image, angle): return imutils.rotate(image, angle=angle)</pre>	

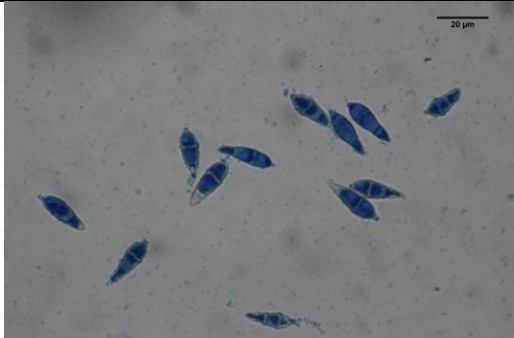
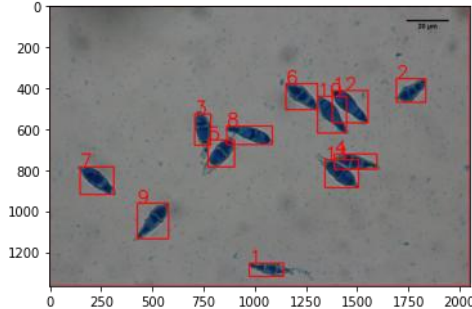
2	<p>Flipped</p> 	<p>Mimics real-world scenarios where objects appear flipped horizontally or vertically.</p>
	<p>Code:</p> <pre>def flip_image(image, flip_code): return cv2.flip(image, flipCode=flip_code)</pre>	
3	<p>Bright Adjusted</p> 	<p>Accounts for lighting changes in real-world images.</p>
	<p>Code:</p> <pre>def adjust_brightness(image, alpha, beta): return cv2.convertScaleAbs(image, alpha=alpha, beta=beta)</pre>	
4	<p>Blurred</p> 	<p>Simulates blurry conditions, helping the model learn robustness.</p>

	<p>Code:</p> <pre>def gaussian_blur(image, kernel_size): return cv2.GaussianBlur(image, (kernel_size, kernel_size), 0)</pre>	
5	<p>Scaled</p> 	<p>Introduces size variations in objects.</p>
	<p>Code:</p> <pre>def scale_image(image, scale_percent): width = int(image.shape[1] * scale_percent / 100) height = int(image.shape[0] * scale_percent / 100) return cv2.resize(image, (width, height))</pre>	
6	<p>Color Jittered</p> 	<p>Simulates changes in color conditions like lighting or shading. Helps models become invariant to color variations.</p>
	<p>Code:</p> <pre>def color_jitter(image, brightness=0.5, contrast=0.5, saturation=0.5): hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV).astype(np.float32) hsv[..., 1] *= 1 + np.random.uniform(-saturation, saturation)</pre>	

	<pre> hsv[..., 2] *= 1 + np.random.uniform(-brightness, brightness) hsv = np.clip(hsv, 0, 255) return cv2.cvtColor(hsv.astype(np.uint8), cv2.COLOR_HSV2BGR) </pre>
--	--

2) Pyricularia Oryzae, rice blast fungus can cause rice blast disease. To identify the possibility of the occurrence of rice blast disease, the density of the spores of Pyricularia Oryzae can be calculated. Plant pathologist knows that you studied image processing, so they have asked you to help them automatically count the number of spores using image processing. They have provided two image samples below for you to develop an algorithm to count them. You should provide your results in terms of `num_count` and `resulted_image` (labeled count) (you can use `cv2.rectangle(...)` and `cv2.putText(...)` functions) as the example shown below

Note: your algorithm **does not have to be 100% accurate**; you should explain your results.

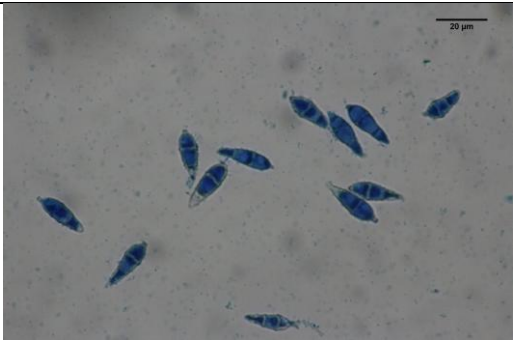
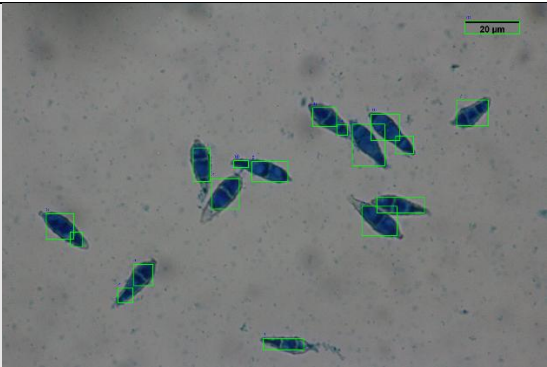
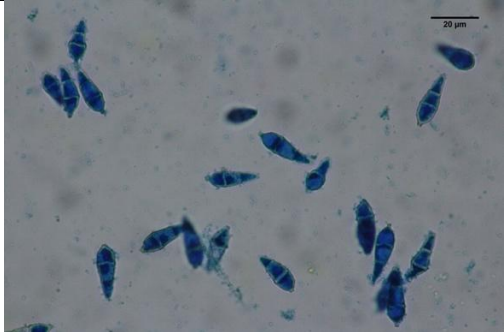
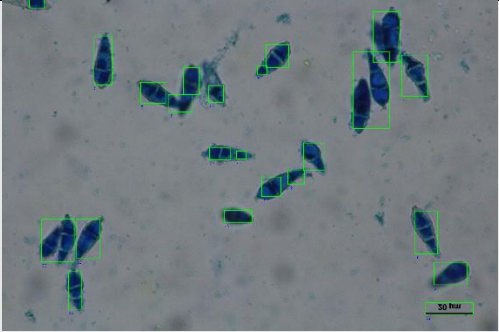
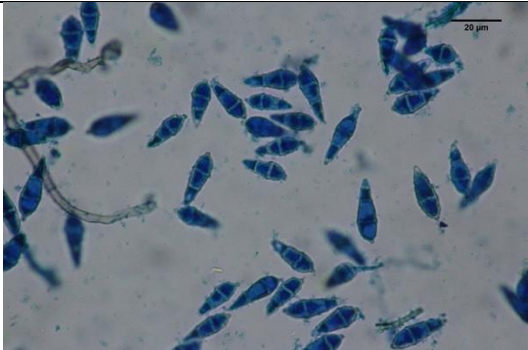
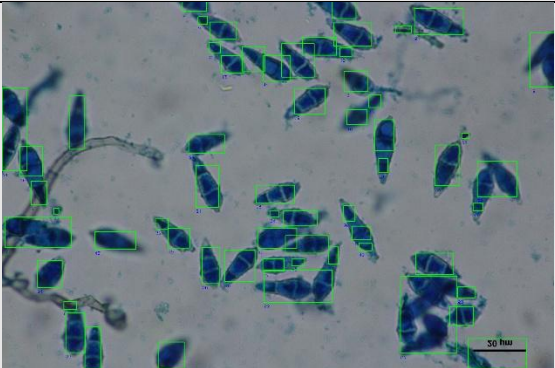
Original image	Your results / number of counted spores
 <p>pyri02.png</p>	<p>EXAMPLE</p>  <p>num_count = 12</p>

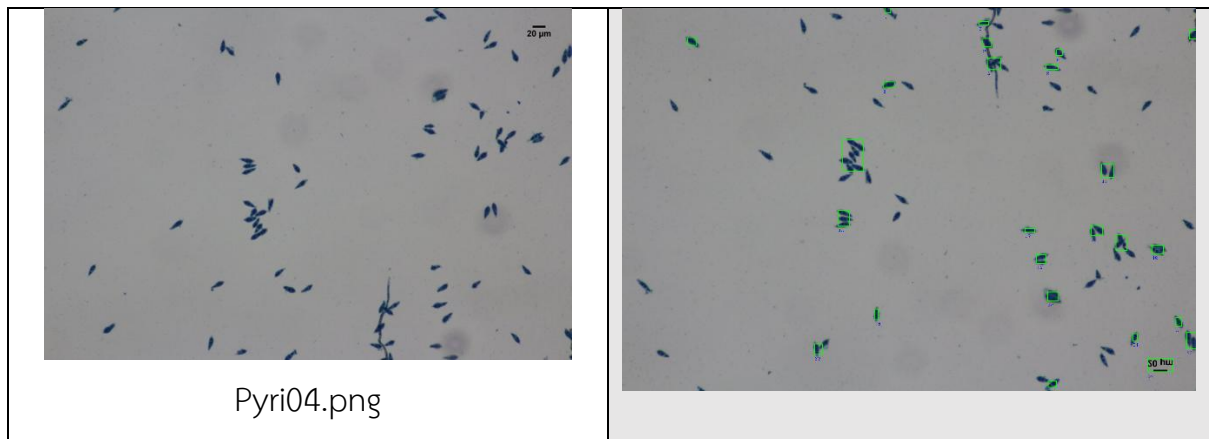
2.1) Describe steps of your algorithm

Steps	Description and purposes
1	Convert the image to grayscale and blur it to reduce noise.
2	Threshold the grayscale image to isolate spores.
3	Use morphological operations to connect nearby parts of the same spore.
4	Detect outlines of merged spore regions.

5	Filter out noise and draw bounding rectangles around valid spores.
6	Count spores and label them for visualization.

2.2) Results

Original image	Your results / number of counted spores
 <p>pyri01.png</p>	
 <p>pyri02.png</p>	
 <p>pyri03.png</p>	



2.3) Analyze the results.

Hint: in terms of how accurate is your technique, any further improvement can be done?

Strengths:

- The morphological closing operation effectively merges nearby parts of the same spore into a single contour. This reduces the issue of over-segmentation where one spore might be counted as multiple smaller parts.
- Contour filtering based on area ensures that small noise or irrelevant features are excluded.
- Bounding boxes and labels provide a clear visualization of detected spores, allowing easy verification of accuracy.

Limitations:

- Missed Spores: Some spores may still be missed if their intensity is too similar to the background or if they are very small (area < 200 pixels). The threshold and area filters need careful tuning based on the image dataset.
- Over-Merging: Larger morphological kernels might result in merging distinct spores into one contour, leading to undercounting. For instance, spores that are close to each other may be merged if the kernel size is too large.

Suggestions for Improvement

- Dynamic Thresholding: Instead of a fixed binary threshold, using adaptive thresholding (e.g., `cv2.adaptiveThreshold`) could handle uneven lighting better.

- Iterative Morphological Operations: Apply morphological closing with smaller kernels iteratively to reduce over-merging of distinct spores.
- Refinement with Watershed: For clustered spores, the watershed algorithm could help separate connected regions that belong to different spores.
- Histogram Analysis: Analyze the pixel intensity distribution to dynamically choose a threshold that separates spores from the background more effectively.
- Post-Processing: Further refine the detected contours using convex hulls or shape analysis to better capture irregular spore boundaries.

3. The "Gemini star" refers to the brightest stars within the constellation of Gemini, which is one of the twelve zodiac constellations. Gemini is best known for its two brightest stars, **Castor** and **Pollux**, which are often called the "twins" due to the mythology behind the constellation.



Fig. 3.1 Gemini stars

The **line of stars in Gemini** stretches from **Pollux** and **Castor** down to their "feet," represented by dimmer stars. The formation appears like two parallel stick figures standing side by side, connected by their heads.

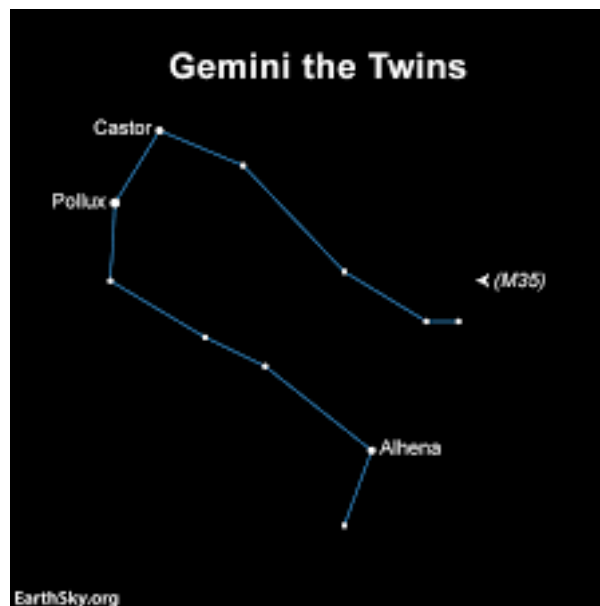




Figure 3.2 The **line of stars in Gemini**

To achieve this, use image processing techniques about erosion and dilation techniques from morphological image processing, enhance a specific star in an astronomical image. The challenge is to extract the main star (or stars) by reducing background noise and enhancing the desired features.

The final result should show only the stars of Gemini to shine bright without others as seen in Fig. 3.2, without a connection line. The stars should maintain its original size or similar.

The Original image	Your Result Image
<p>!gdown 1F4gZPU5KUboFBj7cVPHIrJyfrRcwJj92</p> 	



Explain Your Code about Erosion and Dilation below

Erosion: Reduces noise and small details in the binary image by shrinking white regions (stars).

```
eroded = cv2.erode(binary, kernel, iterations=1)
```

This kernel slides over the binary image, and at each pixel, it "shrinks" the white regions by keeping a pixel white only if all pixels under the kernel are white.

Dilation: Restores or expands the remaining white regions (stars) after erosion, ensuring the stars maintain or regain their original size.

```
dilated = cv2.dilate(eroded, kernel, iterations=2)
```

The same elliptical kernel is applied in a reverse manner. Instead of shrinking, it "grows" the white regions by setting a pixel to white if any part of the kernel overlaps with a white pixel.