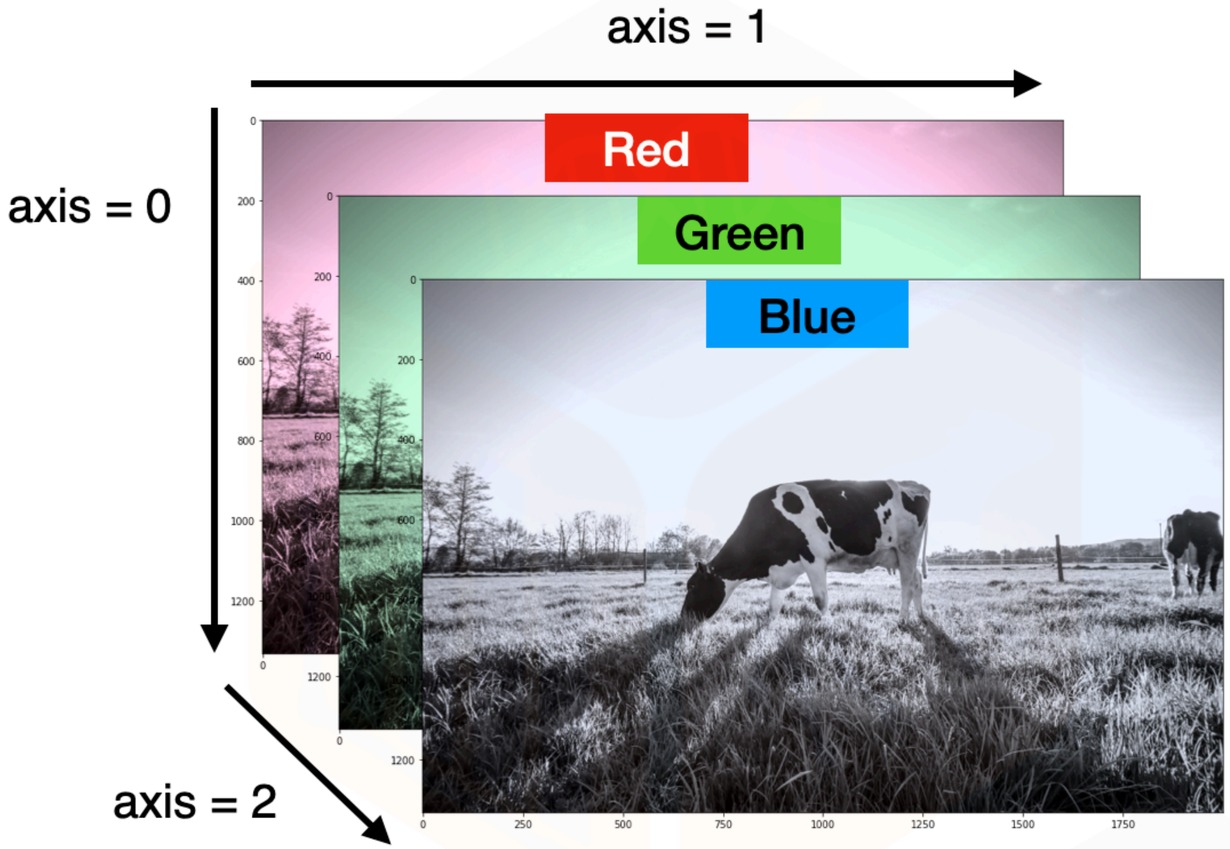


Original Image

Label: Cow

Augmented Images

Label: Cow

Label: Cow

Label: Cow

Lecture 11-1: Computer Vision

Dr. James Chen, Animal Data Scientist, School of Animal Sciences

Overview - Computer Vision (CV)

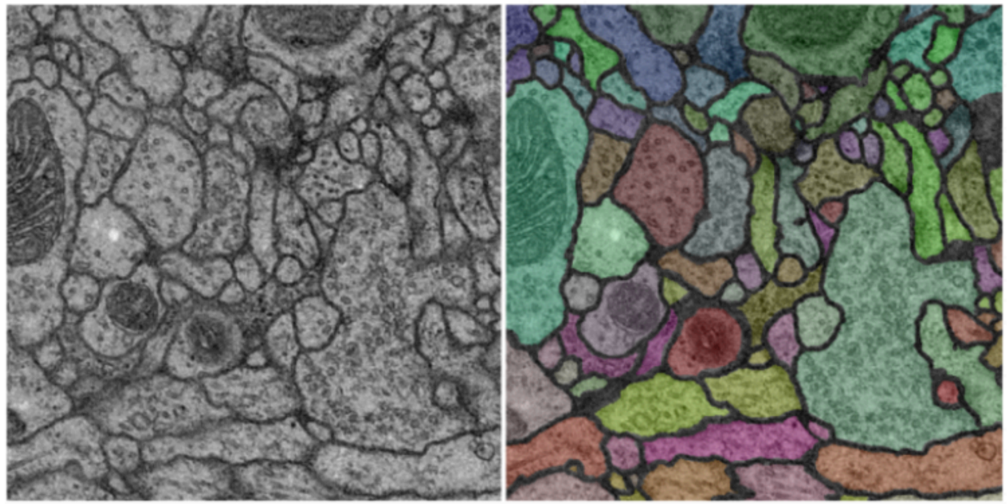
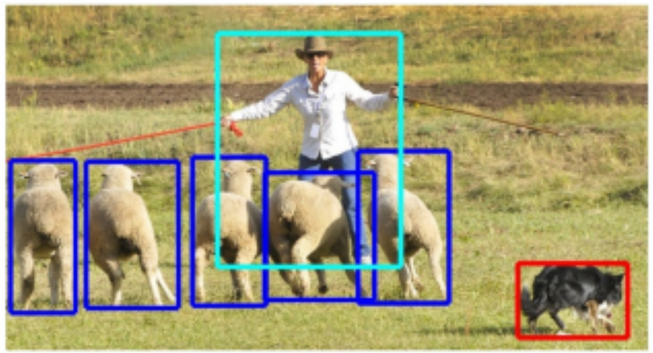
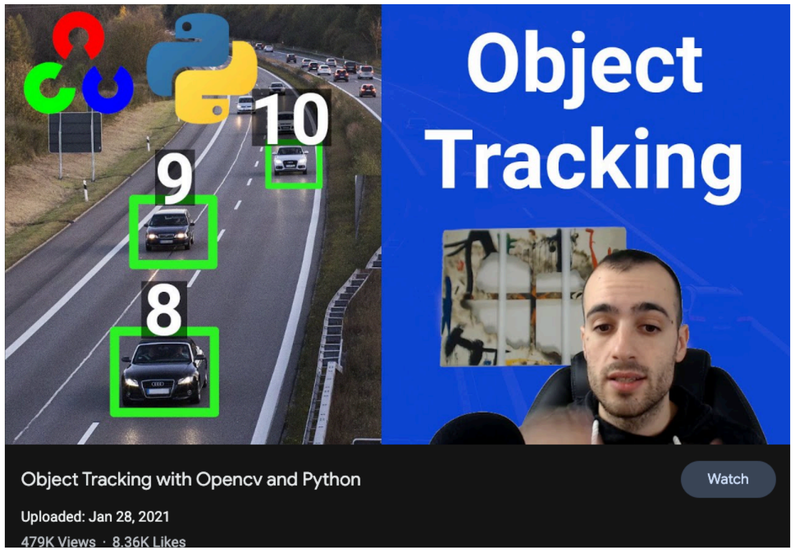


Figure 10: Ciresan - Neuronal membrane segmentation



(b) Object localization



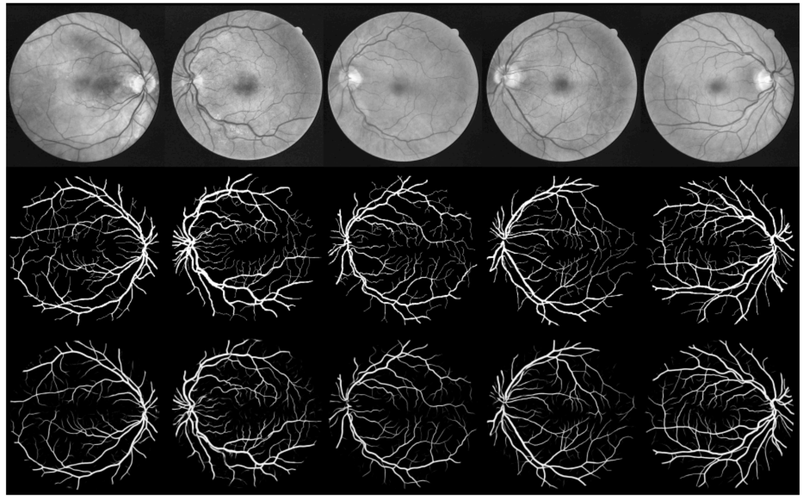


Figure 15: Ronneberger et al., U-Net: Biomedical Image Segmentation

(d) This work

https://on-demand.gputechconf.com/gtcdc/2017/presentation/dc7217-abel-brown-deep-learning-object-detection-and-segmentation.pdf

Overview - CV in Animal Sciences

frontiers in Veterinary Science

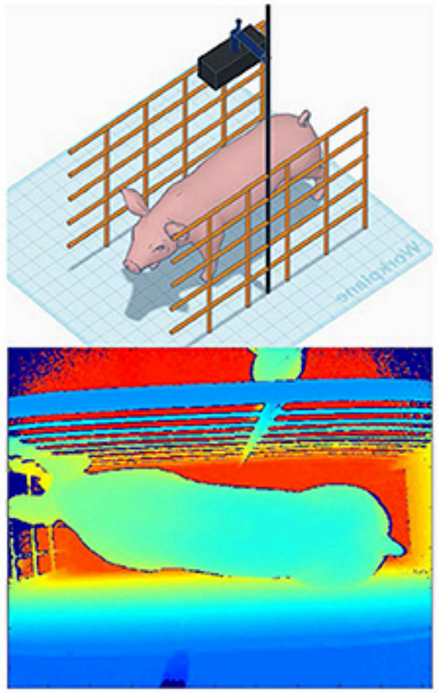
REVIEW
published: 21 October 2020
doi: 10.3389/fvets.2020.551269

Image Analysis and Computer Vision Applications in Animal Sciences: An Overview

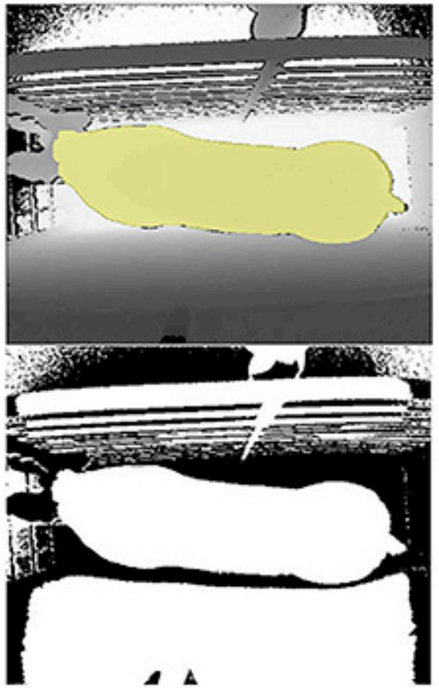
Arthur Francisco Araújo Fernandes [1], João Ricardo Rebouças Dórea [1] and Guilherme Jordão de Magalhães Rosa [1,2]*

[1] Department of Animal and Dairy Sciences, University of Wisconsin-Madison, Madison, WI, United States, [2] Department of Biostatistics and Medical Informatics, University of Wisconsin-Madison, Madison, WI, United States

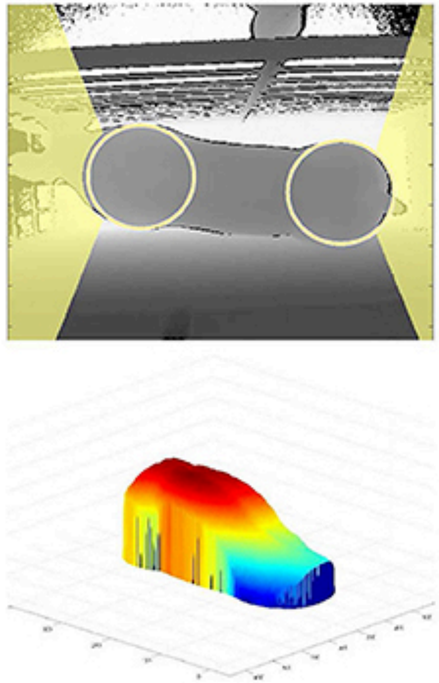
A Image acquisition



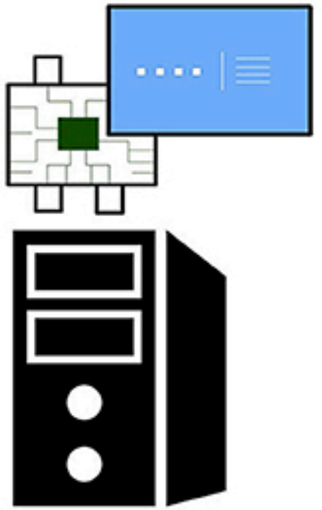
B Image Processing
- Thresholding
- Binarization



C Image Analysis
- Image Segmentation
- Feature extraction



D Data Analysis
- Data normalization
- Model fitting
- Validation and tuning
- Predicition



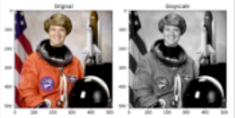
Depth camera

https://www.frontiersin.org/articles/10.3389/fvets.2020.551269/full

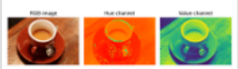
Overview - Python Community

scikit-image
image processing in python

Manipulating exposure and color channels

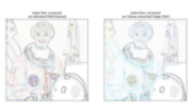


RGB to grayscale

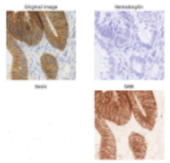


RGB to HSV

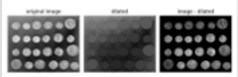
Histogram matching



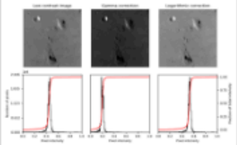
Adapting gray-scale filters to RGB images



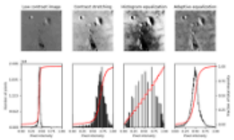
Separate colors in immunohistochemical staining



Filtering regional maxima



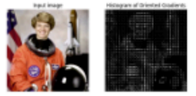
Gamma and log contrast adjustment



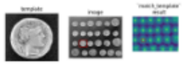
Histogram Equalization

Detection of features and objects

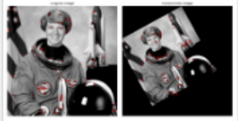
Dense DAISY feature description



Histogram of Oriented Gradients

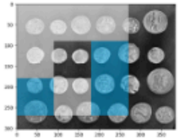


Template Matching



CENSURE feature detector

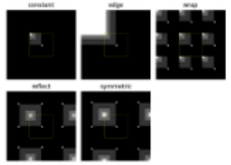
Corner detection



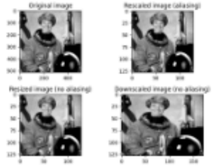
Multi-Block Local Binary Pattern for texture classification

Geometrical transformations and registration

Swirl



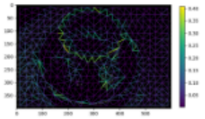
Interpolation: Edge Modes



Rescale, resize, and downscale

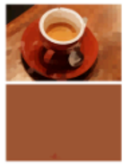
Build image pyramids

Segmentation of objects

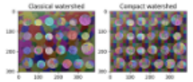


Region Boundary based RAGs

RAG Thresholding



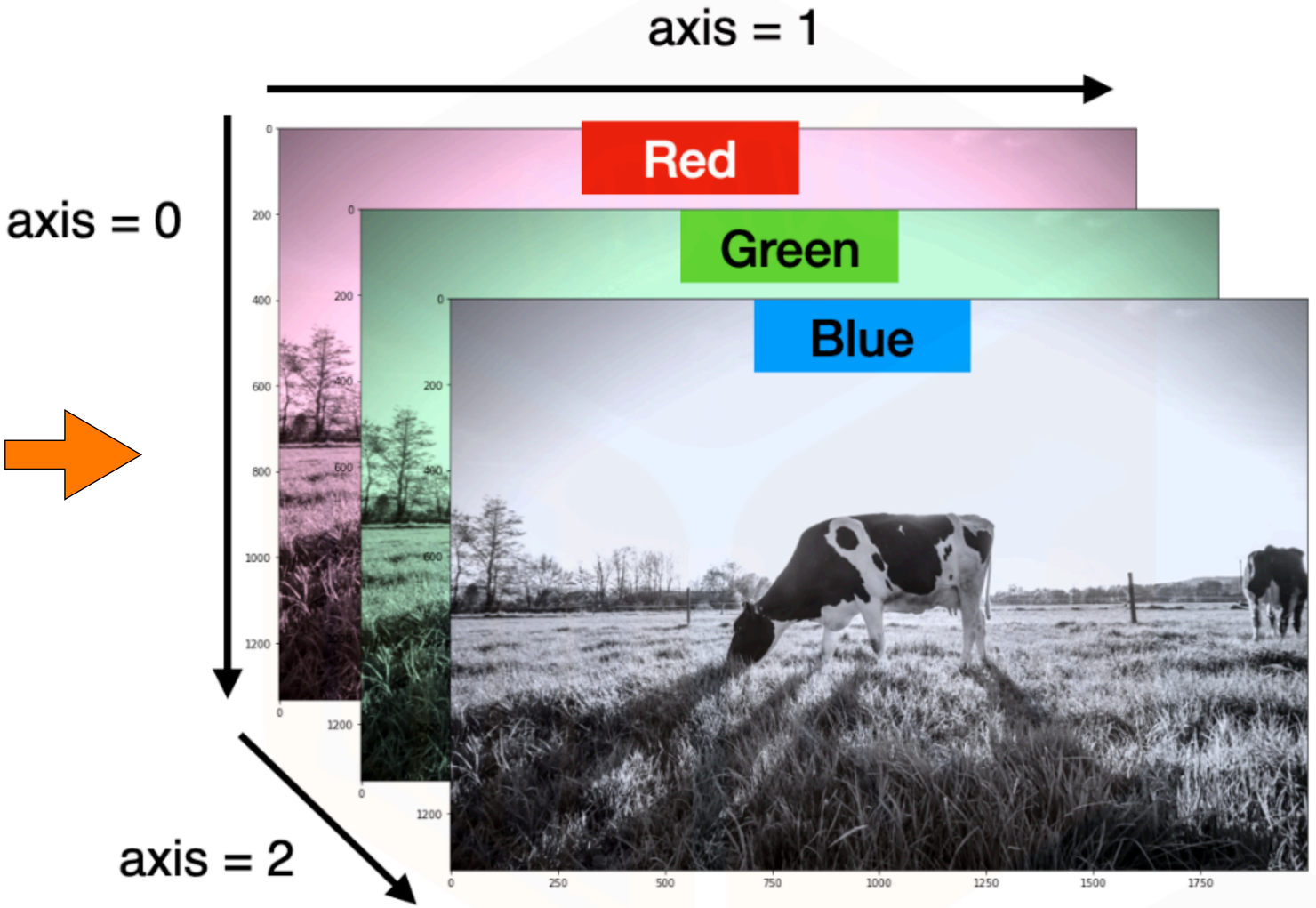
Normalized Cut



Find Regular Segments Using Compact Watershed

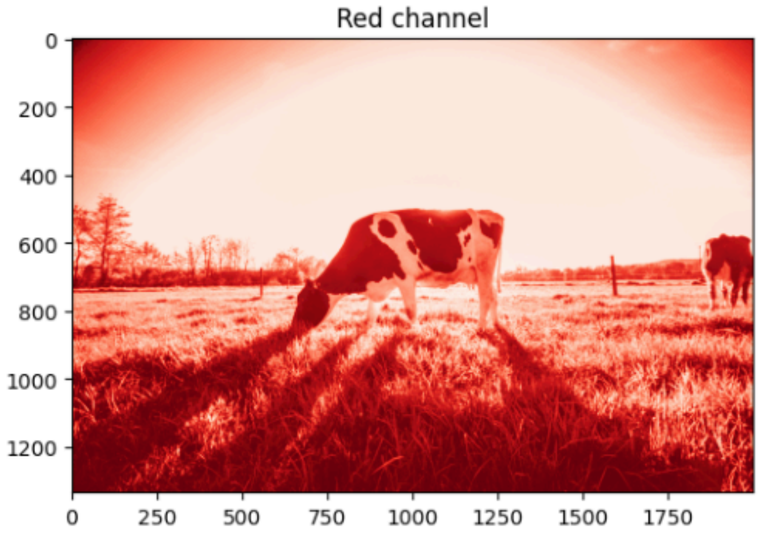
https://scikit-image.org/docs/stable/auto_examples/

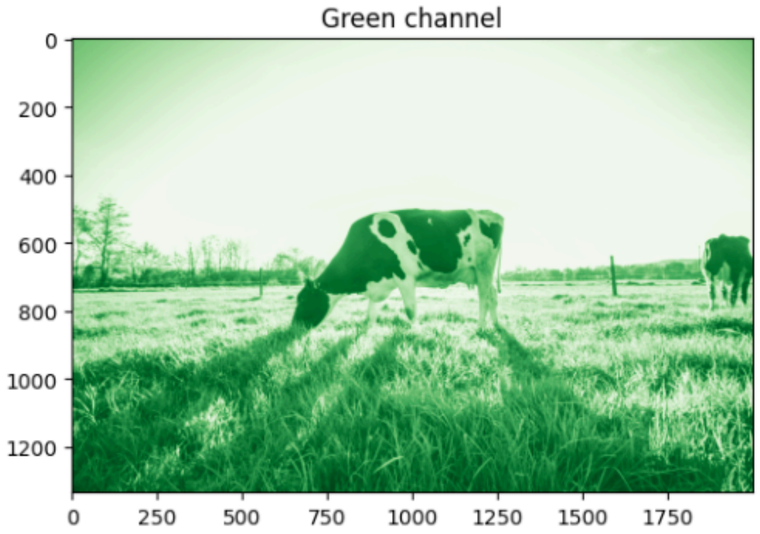
An Image Array Is Typically a Three-Dimensional Array

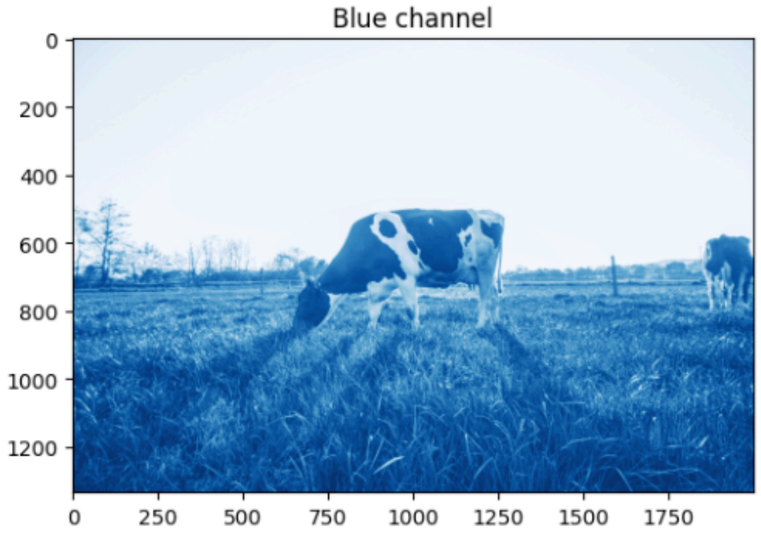


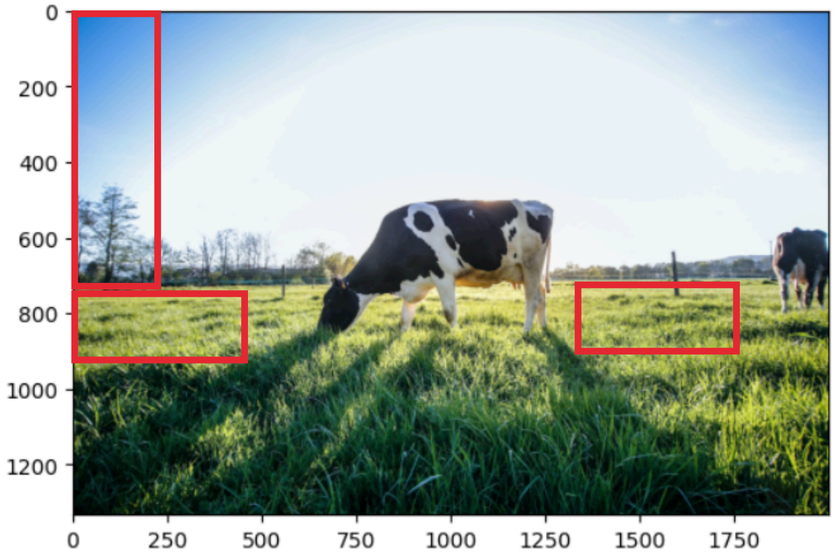
An Image Array Is Typically a Three-Dimensional Array

The brighter the color, the higher the value of the corresponding channel

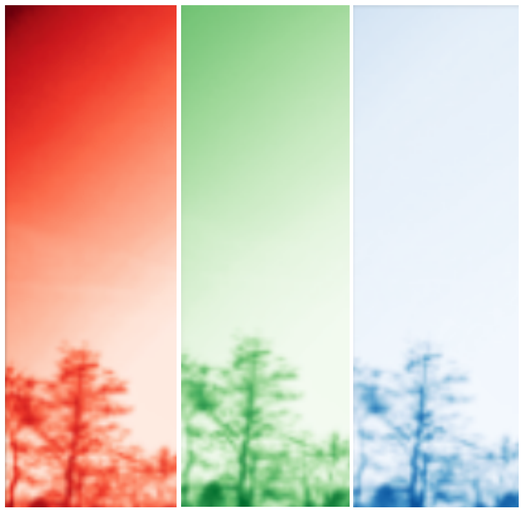




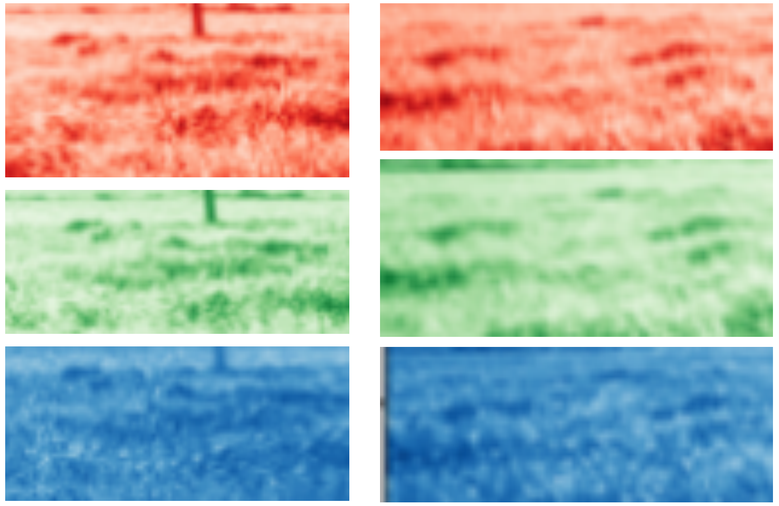




Blue sky



Green pastures



Python Image Library (PIL)

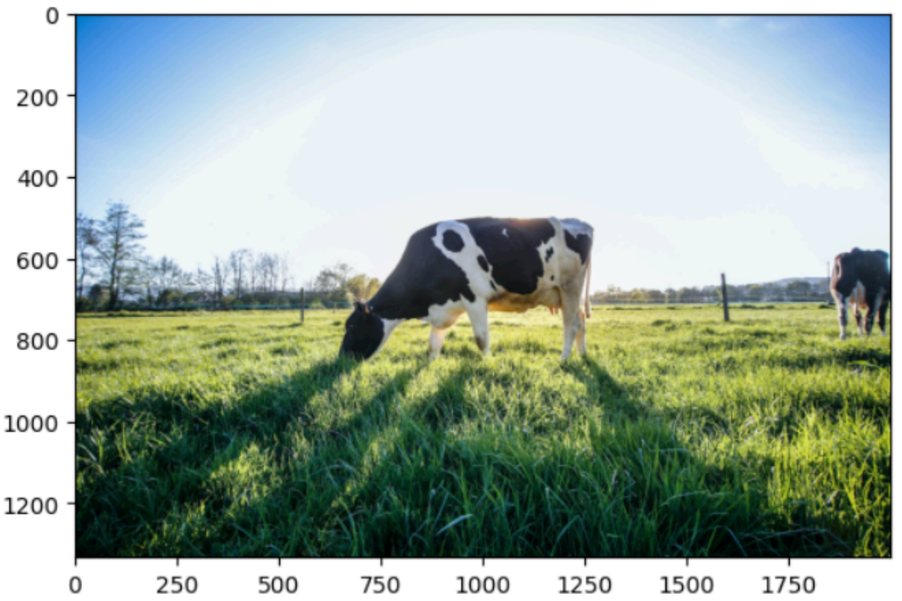
File loading

We can use the knowledge of matrix operations we learned earlier to conduct basic image processing.
Let's load an image and convert it to a matrix using the library `PIL` (Python Imaging Library).

```
# Option 1: Load the data (with statement)
with PIL.Image.open('cow.jpg') as image_file:
    image = np.array(image_file)

# Option 2: Equivalent to the above
image_file = PIL.Image.open('cow.jpg')
image = np.array(image_file)
image_file.close()
```

cow.jpg



Inspect the Image Array


```
# check the dimensions of the image
print("Shape of the image: ", image.shape)

# check the data type of the image
print("Data type of the image: ", image.dtype)

# check the minimum and maximum values of the image
print("Minimum value of the image: ", image.min())
print("Maximum value of the image: ", image.max())
```

```
Shape of the image:  (1333, 2000, 3)
Data type of the image:  uint8
Minimum value of the image:  0
Maximum value of the image:  255
```

Use ".imshow" method to show an image

```
plt.imshow(image)
```

```
<matplotlib.image.AxesImage at 0×17ff53190>
```

Inspect the Image Array

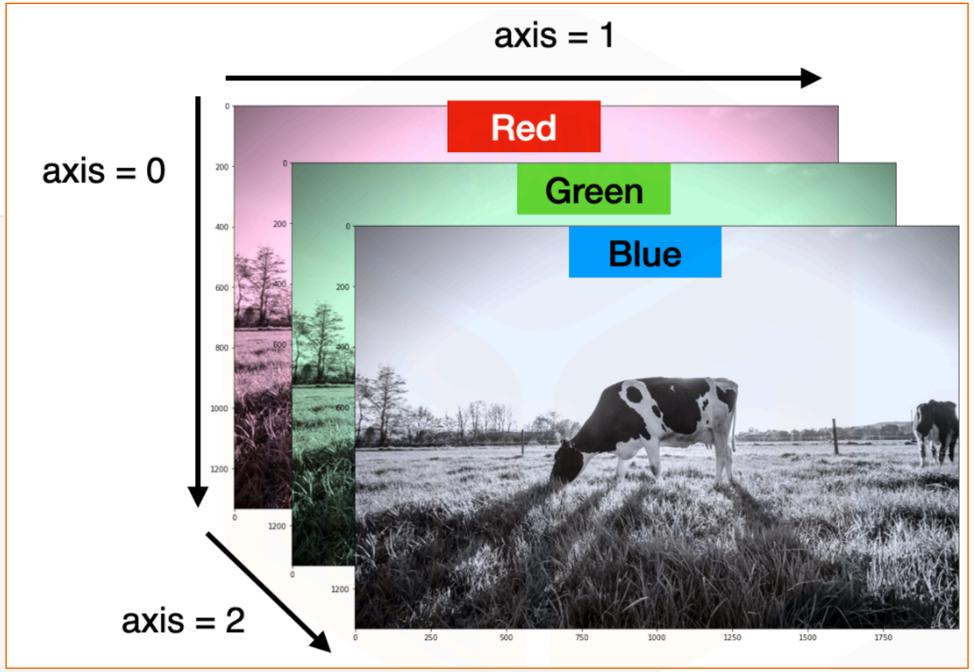
Check each color channel (axis = 2)

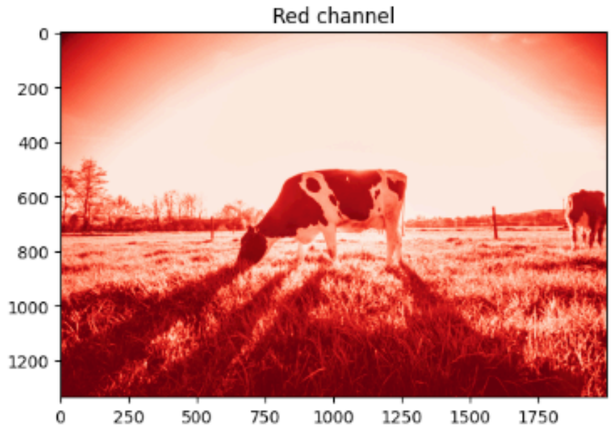
`image[:, :, 0]` Print out the first channel

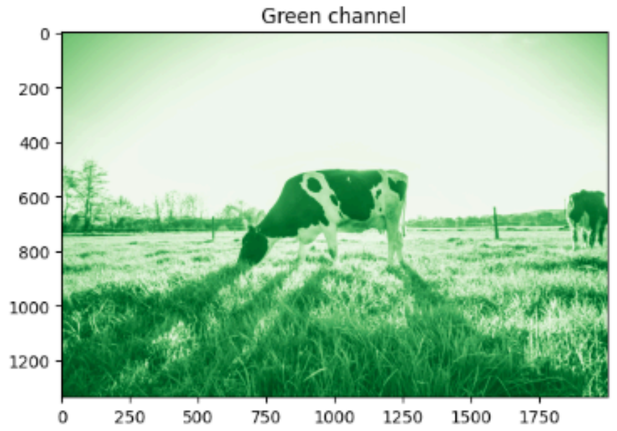
```
fig, ax = plt.subplots(1, 3, figsize=(20, 15))
ax[0].imshow(image[:, :, 0], cmap='Reds_r')
ax[0].set_title('Red channel')
ax[1].imshow(image[:, :, 1], cmap='Greens_r')
ax[1].set_title('Green channel')
ax[2].imshow(image[:, :, 2], cmap='Blues_r')
ax[2].set_title('Blue channel')
```

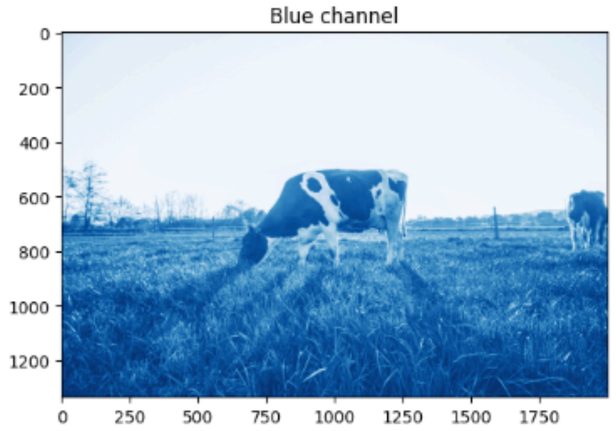
✓ 0.6s

```
Text(0.5, 1.0, 'Blue channel')
```









Inspect the Image Array

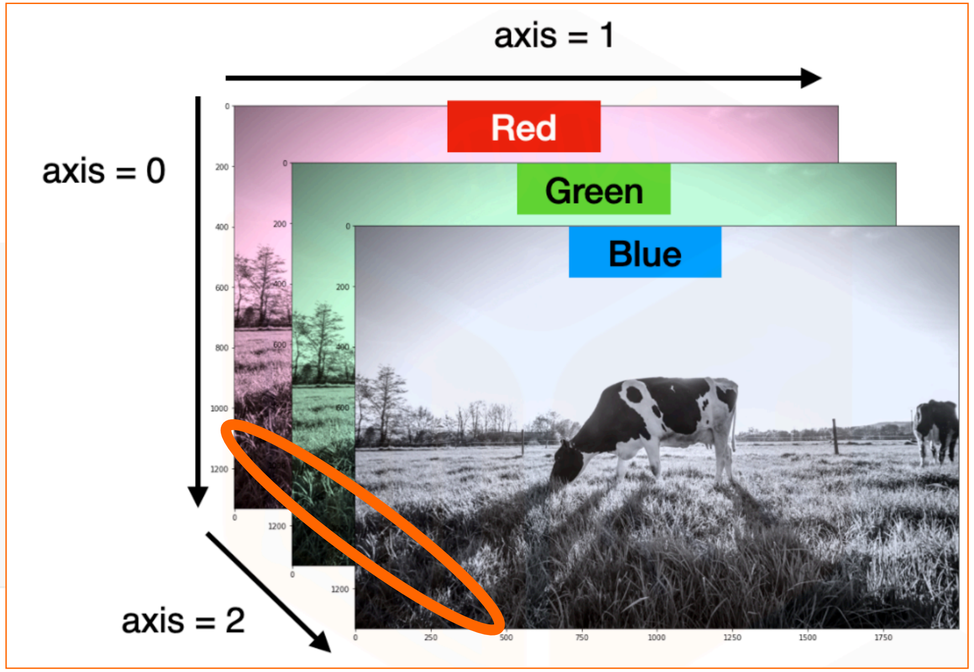
Compute a pixel-wise variation

`image.std(axis=2)` channel axis

```
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(17, 10))
axes[0].imshow(image)
axes[0].set_title("Original image")
# std image (the brighter the higher)
axes[1].imshow(image.std(axis=2), cmap="gray")
axes[1].set_title("Standard deviation of each pixel")
```

✓ 0.6s

```
Text(0.5, 1.0, 'Standard deviation of each pixel')
```



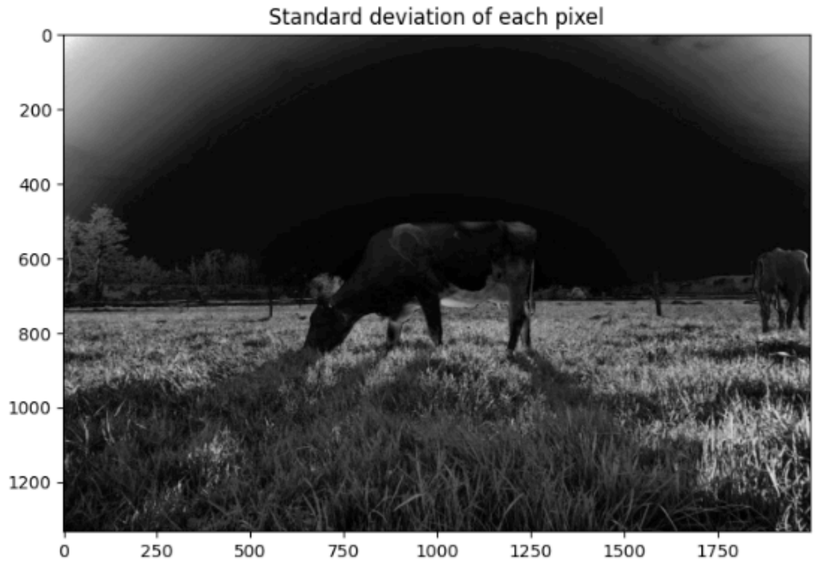


Image Cropping

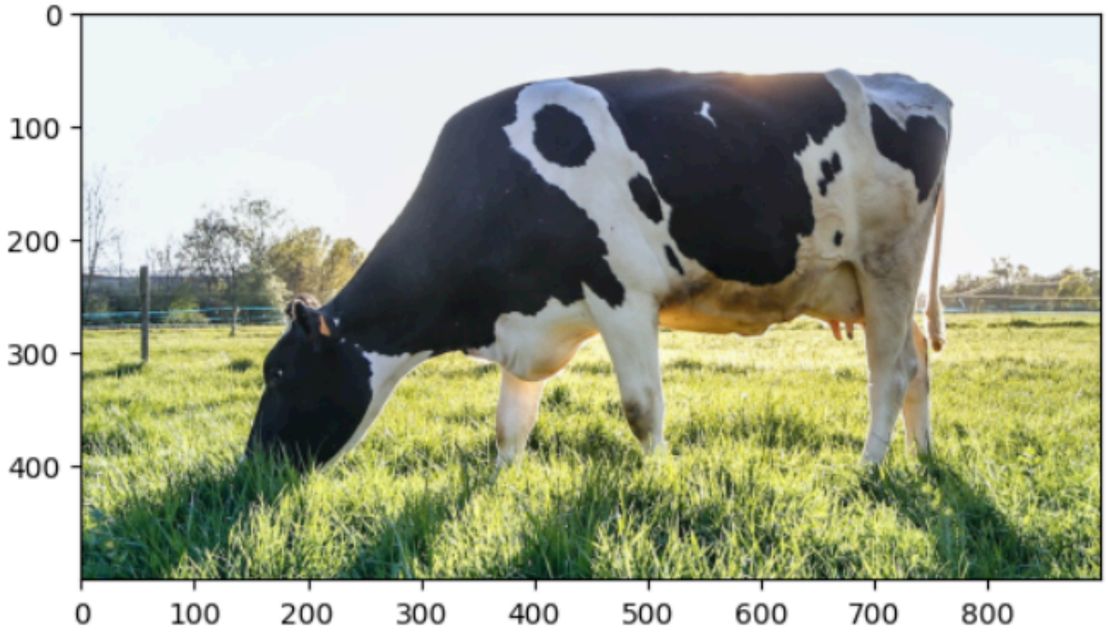
We can apply any matrix (NumPy) operation to an image matrix

We can crop the image by slicing the array.

```
image_crop = image[450:950, 500:1400]
plt.imshow(image_crop)
```

✓ 0.1s

```
<matplotlib.image.AxesImage at 0x292e77d60>
```



Original image

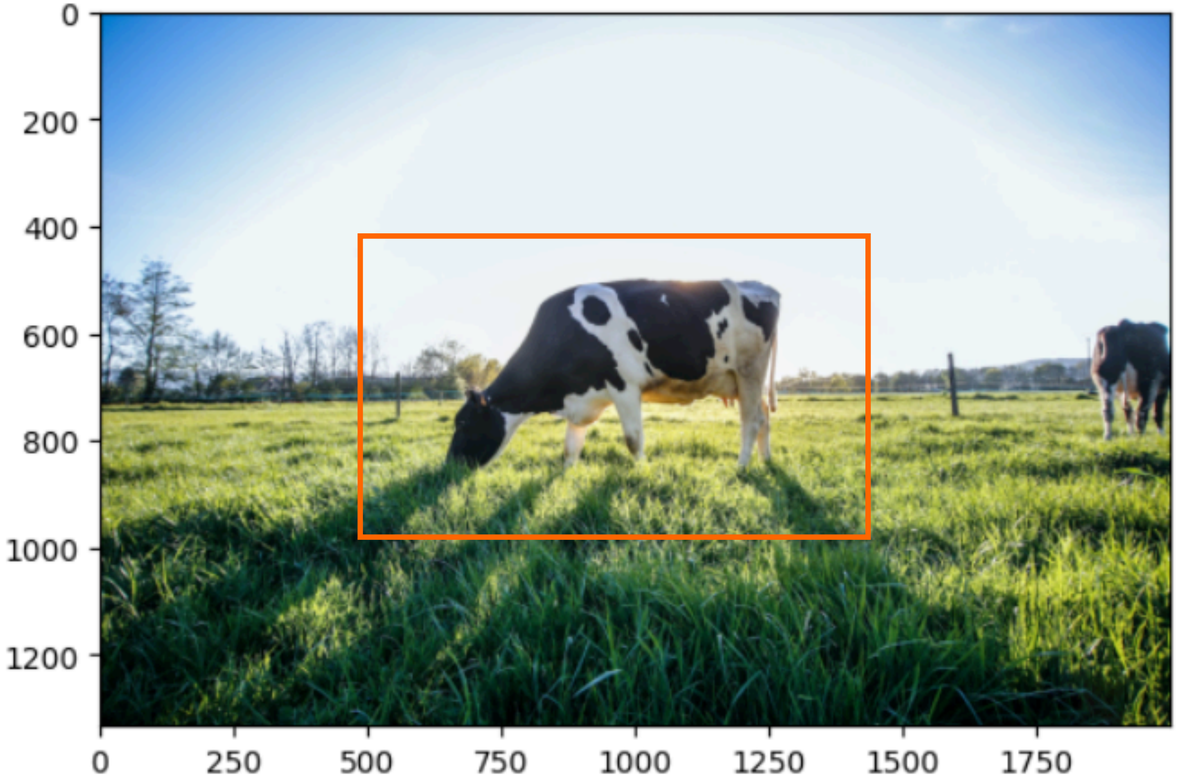
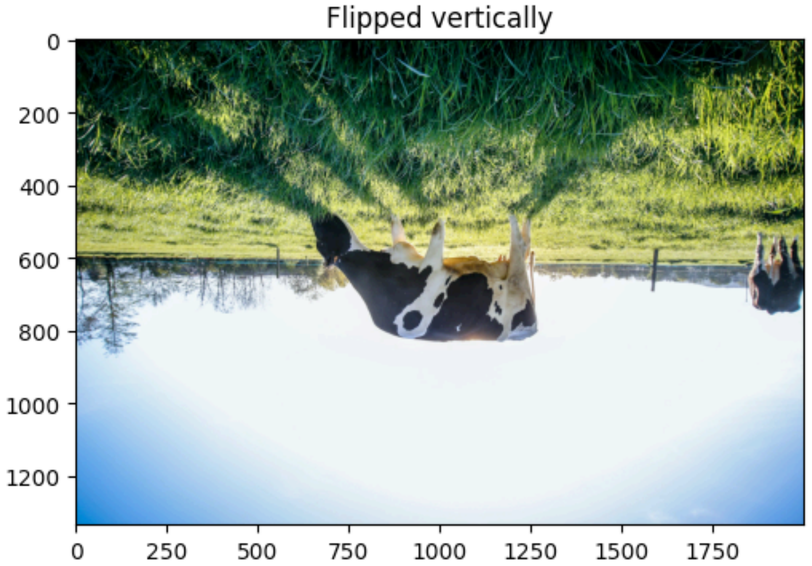


Image Flipping

Reversing the order of the sequence (matrix)

row col channel

```
image[::-1, :, :]
```



row col channel

```
image[:, ::-1, :]
```

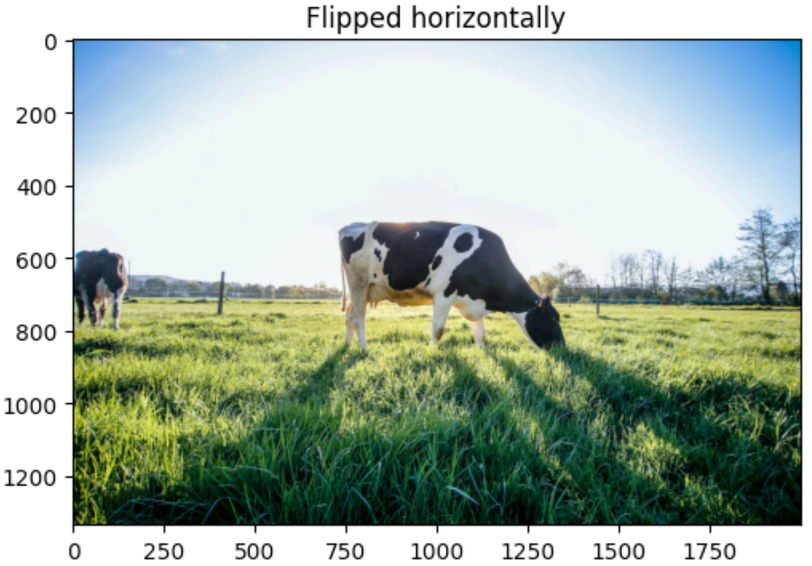


Image Resizing

Resizing an image can be understood as mapping the original image to a new array with different size.

Original image

Enlarged by a scale factor of 2

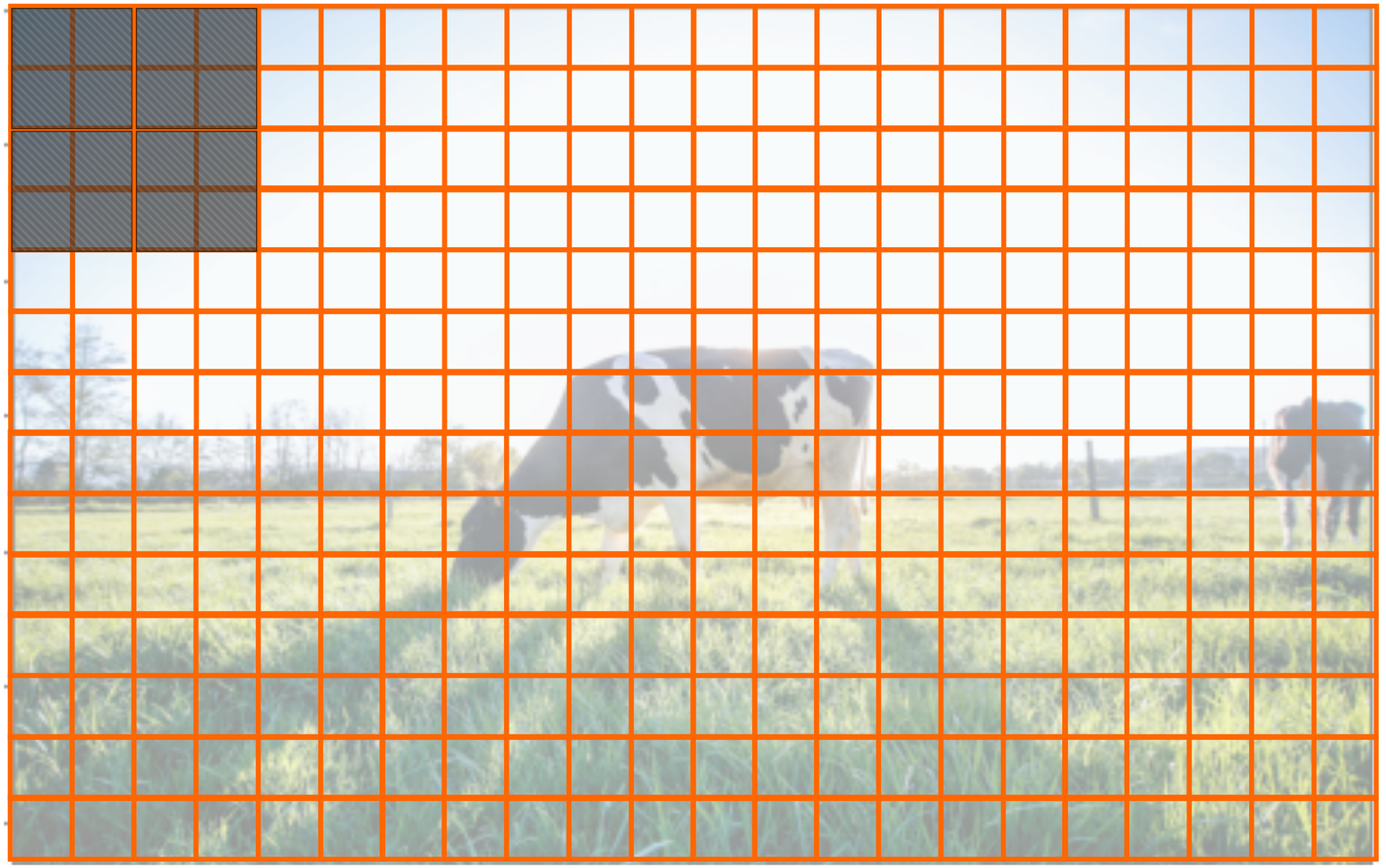
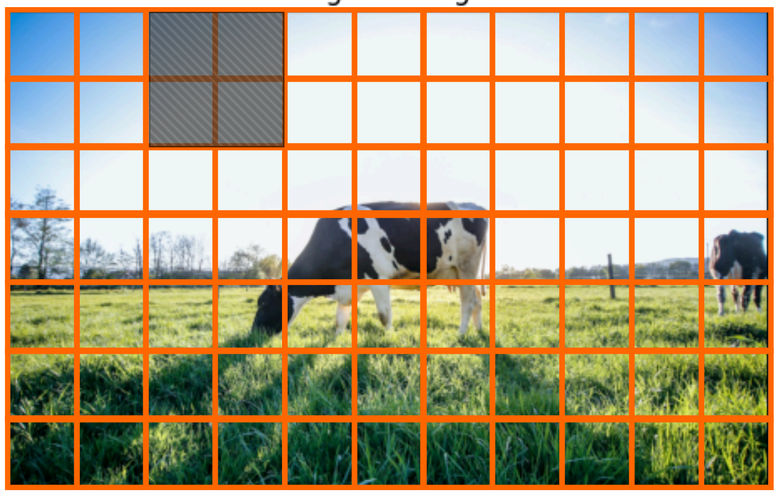


Image Resizing

Resizing an image can be understood as mapping the original image to a new array with different size.

Original image



Enlarged by a scale factor of 2

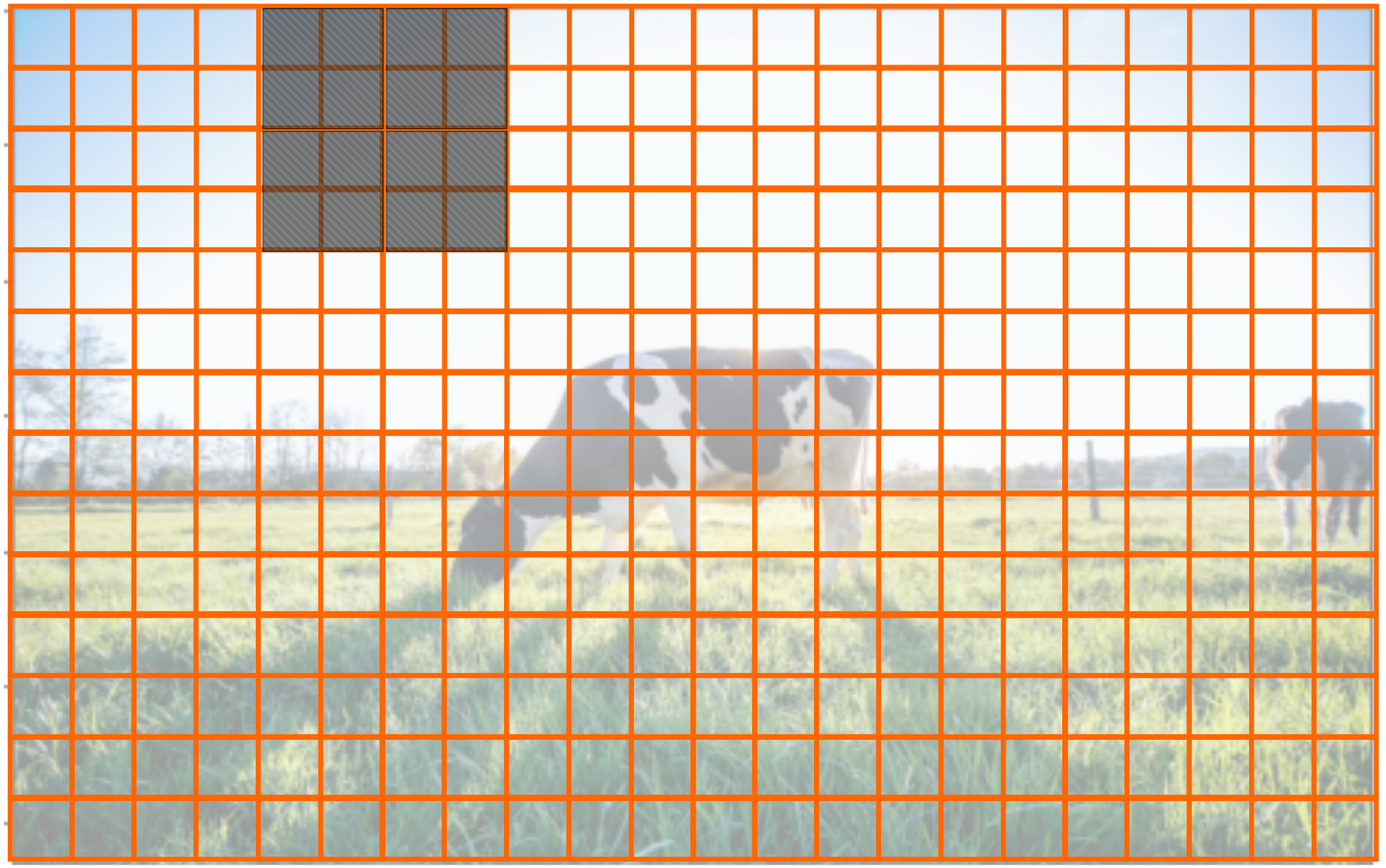


Image Resizing

Resizing an image can be understood as mapping the original image to a new array with different size.

Original image

```python
# resize an image in a larger canvas
H, W, C = image.shape
RESIZE_RATE = 2
nH, nW = H * RESIZE_RATE, W * RESIZE_RATE

# create a new image
img_new = np.zeros((nH, nW, C), dtype=np.uint8)
for h in range(nH):
    for w in range(nW):
        img_new[h, w, :] = image[int(h / RESIZE_RATE),
                                 int(w / RESIZE_RATE),
                                 :]
```

(2.5, 1.5) — — — — — — — — (5, 3)

(3 , 2)

2x

Rounding to the nearest integers

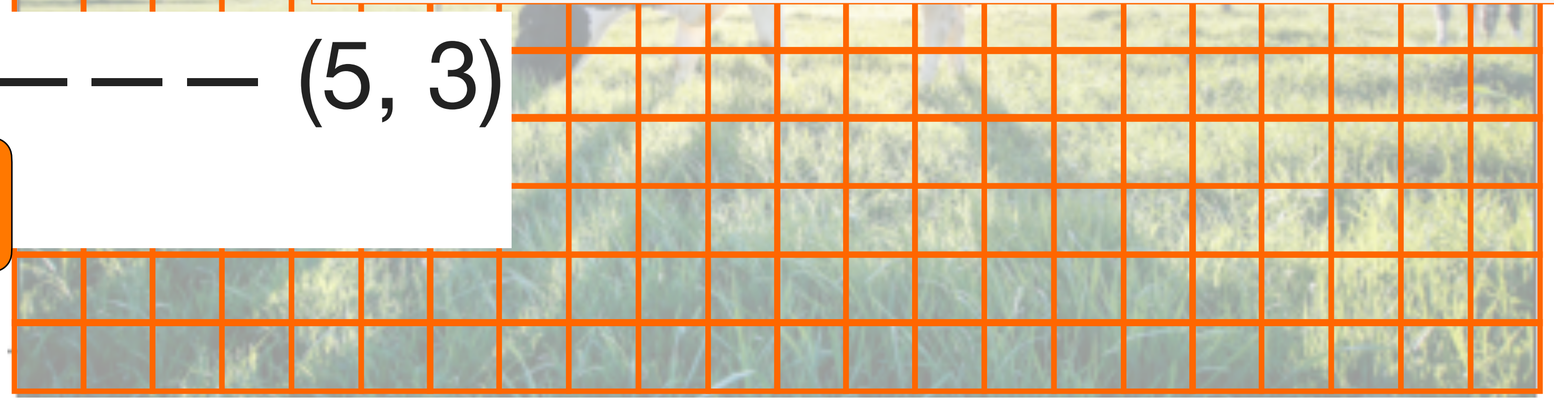


Image Resizing - Function


```
def resize_img(image, resize_rate):
    """
    """
    H, W, C = image.shape
    nH, nW = int(H * resize_rate), int(W * resize_rate)
    img_new = np.zeros((nH, nW, C), dtype=np.uint8)
    for h in range(nH):
        for w in range(nW):
            img_new[h, w, :] = image[int(h / resize_rate),
                                     int(w / resize_rate),
                                     :]
    print("original image shape: ", image.shape)
    print("new image shape: ", img_new.shape)
    return img_new
```

Create the output

Debug messages

Image Resizing - Function

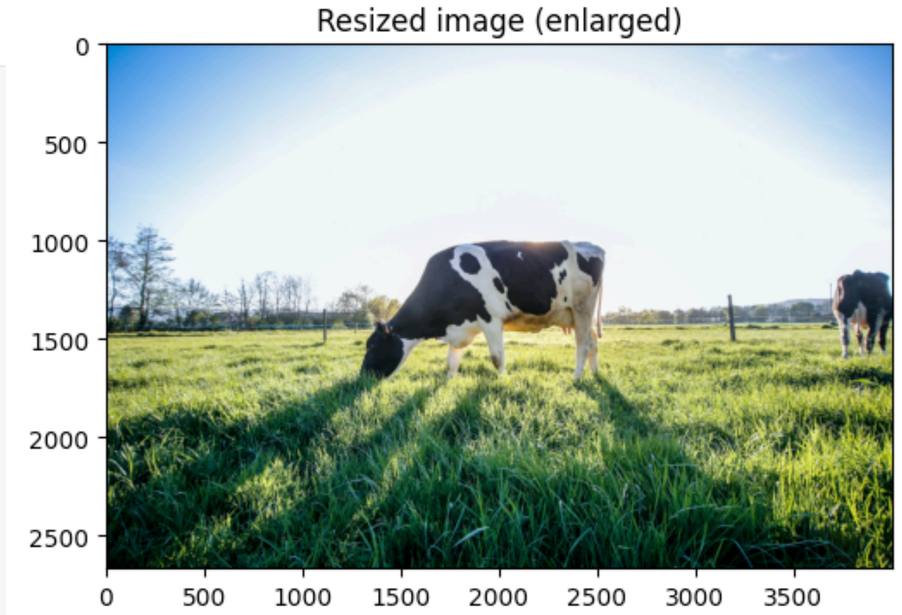

```
img_large = resize_img(image, 2)
img_small = resize_img(image, 0.1)

fig, ax = plt.subplots(1, 3, figsize=(20, 15))
ax[0].imshow(image)
ax[0].set_title('Original image')
ax[1].imshow(img_large)
ax[1].set_title('Resized image (enlarged)')
ax[2].imshow(img_small)
ax[2].set_title('Resized image (shrinked)')
```

✓ 5.8s

```
original image shape:  (1333, 2000, 3)
new image shape:  (2666, 4000, 3)
original image shape:  (1333, 2000, 3)
new image shape:  (133, 200, 3)

Text(0.5, 1.0, 'Resized image (shrinked)')
```



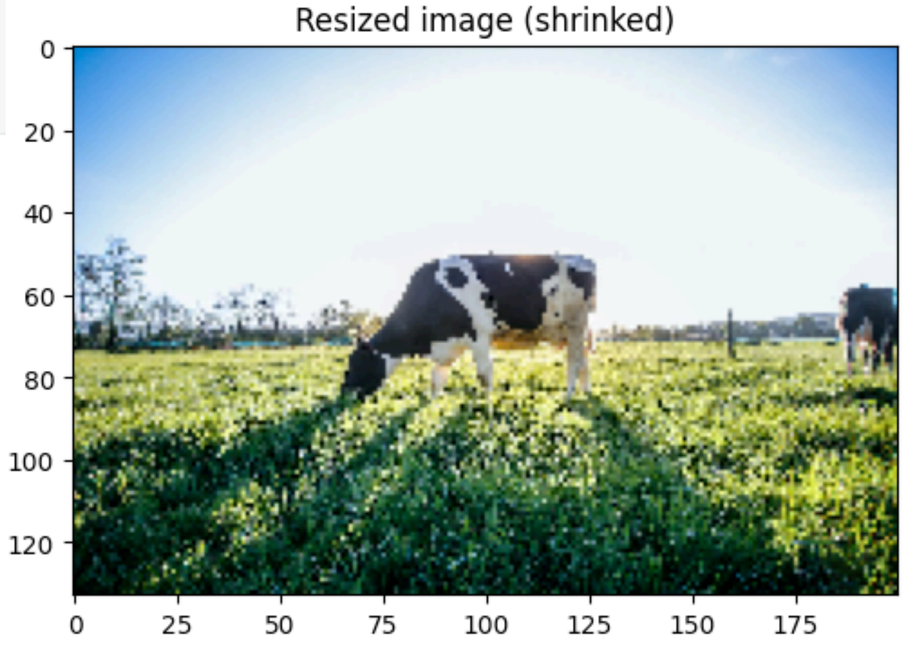


Image Histogram

Image histogram is a histogram plot to visualize the distribution of pixel values in each channel. This plot allows us to visually inspect if the image is correctly exposured, or, for example, if any channel has significant higher signals than other channels.

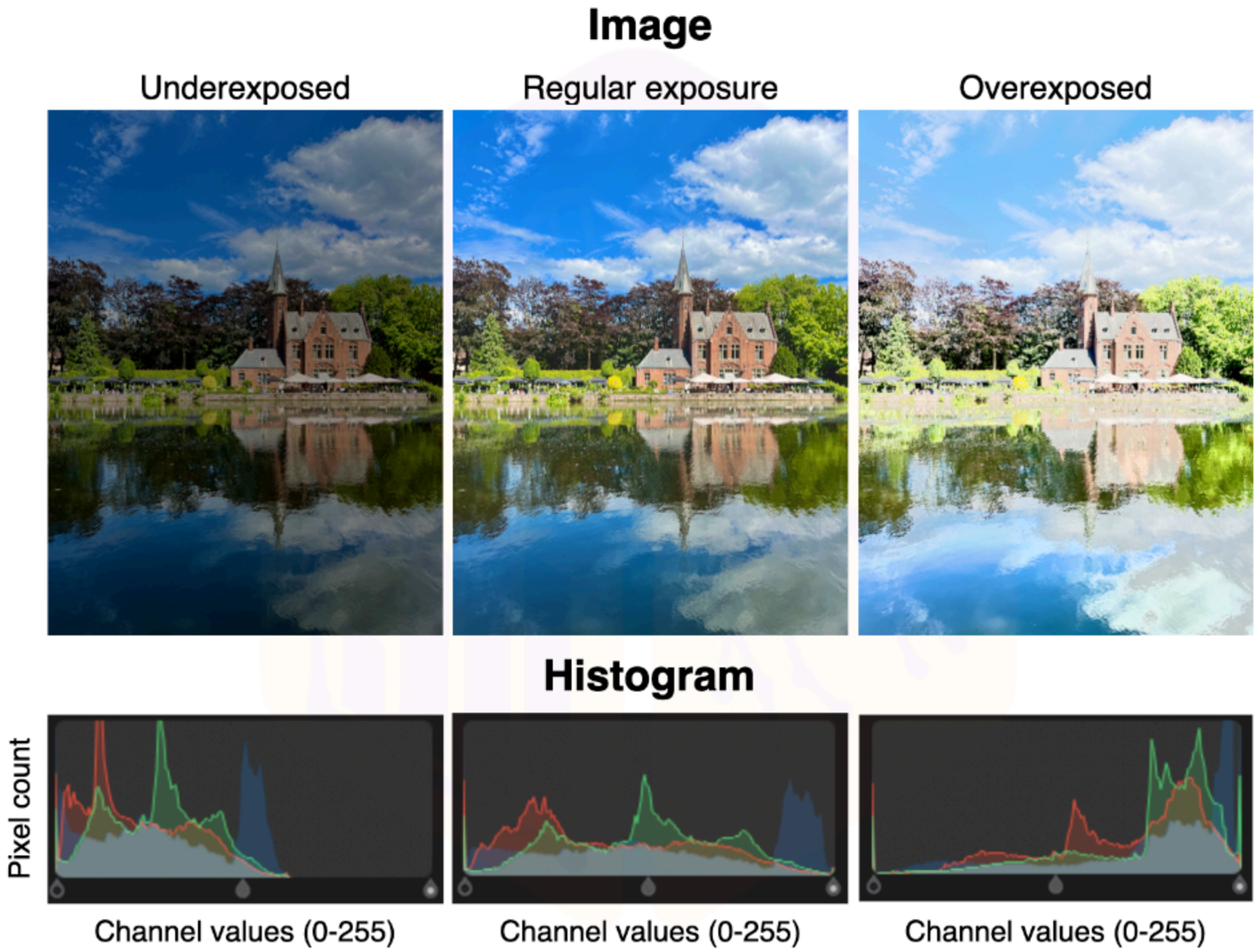
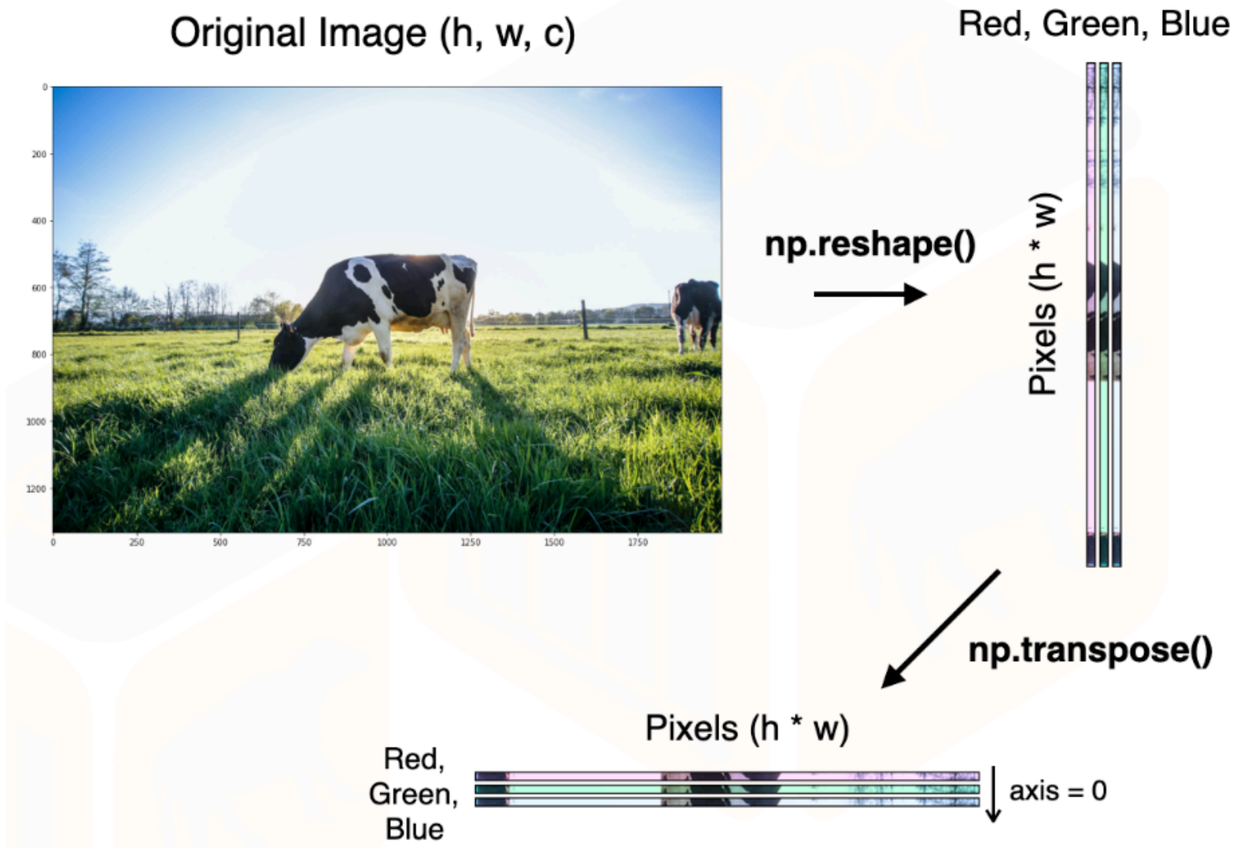


Image Histogram - Reshape the Matrix

We can change the image dimension (i.e., reshaping) from 3D to 2D for a better representation of the pixel distribution.




```
print("original dimension ", image_crop.shape)

# Use "-1" to let Python figure out the remaining dimension
# (pixels, channels)
channels = image_crop.reshape((-1, 3))
print("flattened dimension", channels.shape)

# Let the channel axis become the first axis
# (channels, pixels)
channels = channels.transpose()
print("final dimension    ", channels.shape)
```

✓ 0.0s

```
original dimension  (500, 900, 3)
flattened dimension (450000, 3)
final dimension     (3, 450000)
```

```
# since (r, g, b) are now on the first axis.
red, green, blue = channels
print("red:   ", red.shape)
print("green: ", green.shape)
print("blue:  ", blue.shape)
```

✓ 0.0s

```
red:    (450000,)
green:  (450000,)
blue:   (450000,)
```

Image Histogram - Visualization


```python
# define parameters
params = dict(bins=30, alpha=.3, edgecolor="black")
colors = ["red", "green", "blue"]
# plotting
plt.figure(figsize=(10, 5))
plt.title("Histogram of RGB channels")
for i, bar in enumerate([red, green, blue]):
    # i is the iteration number starting from 0 (and 1, 2, ...)
    plt.hist(bar, color=colors[i], label=colors[i], **params)
plt.legend(fontsize=20)
```

0.1s