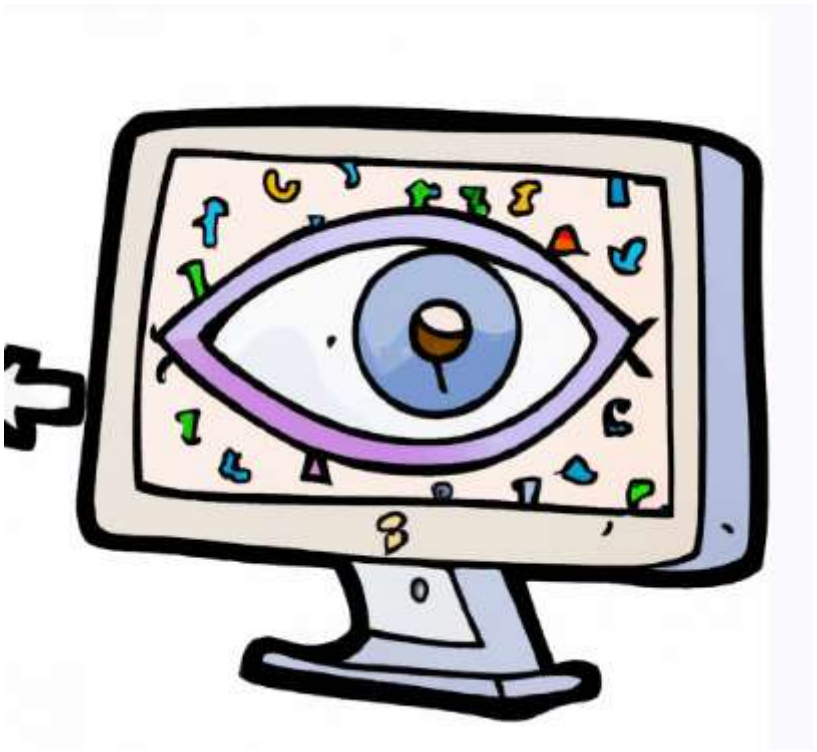
Pre-Processing of Image Data

Authors: Nick Bear Brown



Introduction

There are several real-world applications of image processing. Unfortunately, a few problems associated with image data includes complexity, inaccuracy, and inadequacy. To get the intended outcomes, the data must be preprocessed (cleaned and processed to the proper format) before creating a computer vision model.

Pre-processing is intended to improve the image data by enhancing certain crucial visual features or suppressing unintentional distortions.

Real world examples of Image processing:

• Medical Imaging: To more quickly detect irregularities, scientists in the field of medicine examine the inside organs and tissues of living things. Medical imaging image processing can help create crisp, high-quality images for scientific and medicinal research, ultimately assisting doctors in making diagnoses.

• Military and defence: Steganography is a fascinating way that image processing is used in the military. In order to communicate information back and forth without a third party noticing the message, experts can conceal a message or an image inside another image.

Why is it important?

To prepare image data for model input, some pre-processing is required. One example of this is for convolutional neural networks, where the images need to be in arrays of the same size for fully connected layers. Additionally, pre-processing can help to reduce model training time and improve model inference speed. For example, if the input images are very large, reducing their size can significantly shorten the time required for model training without compromising the model's performance. Although geometric transformations of images, such as rotation, scaling, and translation, are considered pre-processing techniques, the main goal of pre-processing is to improve the image data by reducing unintended distortions or enhancing important image features for further processing.

Steps for Image Preprocessing:

- Resizing
- Normalization
- Data Augumentation
- Greyscale
- Image Filtering

```
import tensorflow
import keras
import os
import glob
from skimage import io
import random
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
     Mounted at /content/drive
```

```
dataset_path = '/content/drive/MyDrive/Animals'
```

```
class_names = ['Cheetah', 'Jaguar', 'Leopard', 'Lion','Tiger']
```

```
animal_path = os.path.join(dataset_path, class_names[1], '*')
animal_path = glob.glob(animal_path)
```

```
image = io.imread(animal_path[4])

# plotting the original image
i, (im1) = plt.subplots(1)
i.set_figwidth(15)
im1.imshow(image)
```

```
<matplotlib.image.AxesImage at 0x7f99913b2c10>
```

1. Resizing and Scaling

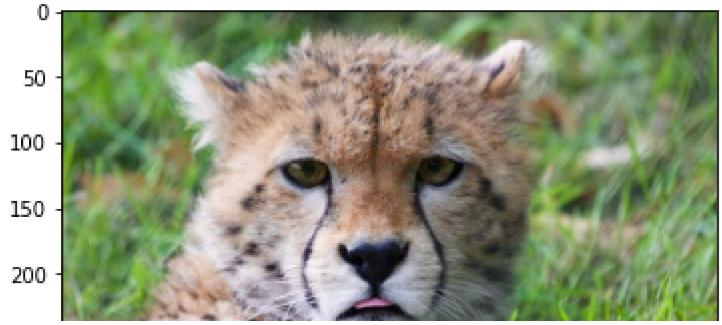
Why: Most of the neural network models assume a square shape input image, which means that each image needs to be checked if it is a square or not, and cropped appropriately.
Cropping can be done to select a square part of the image, as shown. While cropping, we usually care about the part in the center. Images can be resized to a smaller or larger size and scaled to have a certain range of pixel values.

```
from PIL import Image

image = Image.open('/content/drive/MyDrive/Animals/Cheetah/animal-africa-wilderness-zoo.jpg')

print(f"Original size : {image.size}")
```

```
Original size : (500, 333)
```

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

```
<matplotlib.image.AxesImage at 0x7f998d2528e0>
```




```
animal_resized = image.resize((int(image.size[0]/4), int(image.size[1]/4)))
animal_resized.save('animal-africa-wilderness-zoo_jpg_400.jpeg')
Image.open('/content/animal-africa-wilderness-zoo_jpg_400.jpeg')
```

2. Normalization

What is Normalization: Normalization, which is also known as contrast stretching or histogram stretching, is a technique in image processing that alters the range of pixel intensity values. This method is useful in enhancing photographs that suffer from low contrast because of glare or other issues. In other domains of data processing, like digital signal processing, it is referred to as dynamic range expansion.

Why:

Normalization is used to improve the model's performance, the pixel values are converted to a range between 0 and 1, or -1 and 1.

The objective is to ensure uniformity in the dynamic range of a collection of data, signals, or images so as to prevent mental exhaustion or distraction. As an illustration, a newspaper would endeavor to ensure that all the pictures in a particular edition have a comparable grayscale range.

Normalization transforms an n-dimensional grayscale image:

$$I : \{\mathbb{X} \subseteq \mathbb{R}^n\} \rightarrow \{\mathrm{Min}, . . , \mathrm{Max}\}$$

with intensity range(Min, Max), into a new image:

$$I_N : \{\mathbb{X} \subseteq \mathbb{R}^n\} \rightarrow \{\mathrm{newMin}, . . , \mathrm{newMax}\}$$

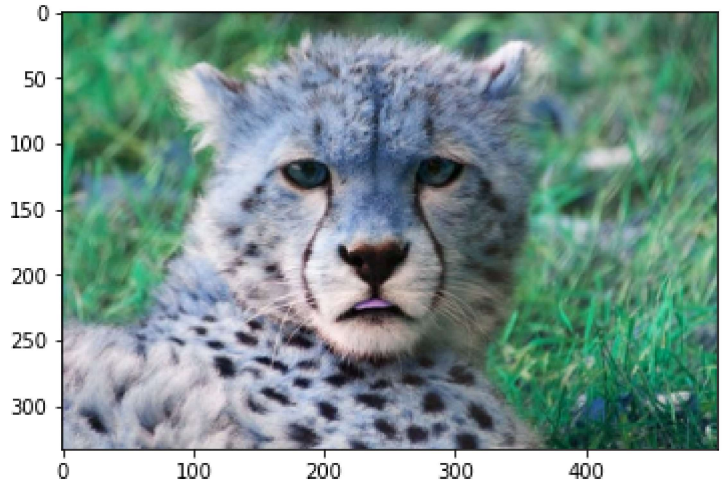
with intensity values in the range (newMin,newMax)

The linear normalization of a grayscale digital image is performed according to the formula:

$$I_N = (I - \text{Min})\frac{\text{newMax} - \text{newMin}}{\text{Max} - \text{Min}} + \text{newMin}$$

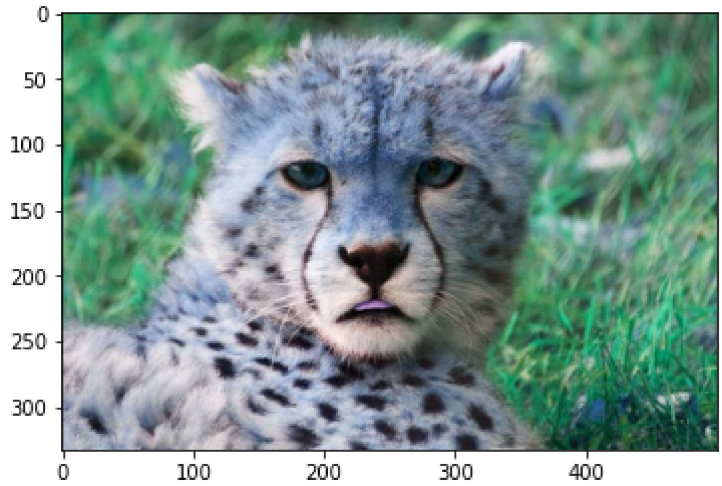
```
plt.imshow(image)
```

```
<matplotlib.image.AxesImage at 0x7f997d2bd2e0>
```




```
norm_image = (image - np.min(image)) / (np.max(image) - np.min(image))
plt.imshow(image)
```

```
<matplotlib.image.AxesImage at 0x7f997d20e5e0>
```



3. Data Augumentation

Data augmentation is a technique used to increase the amount of data by adding slightly modified copies of already existing data or newly created synthetic data from existing data.
Common operations used for data augmentation for images:.

- Rotation.
- Shearing.
- Zooming.
- Cropping.
- Flipping.
- Changing the brightness level.

There are two types of augmentation:

Offline augmentation - Used for small datasets. It is applied in the data preprocessing step.

Online augmentation- Used for large datasets. It is normally applied in real-time.:

Shifting

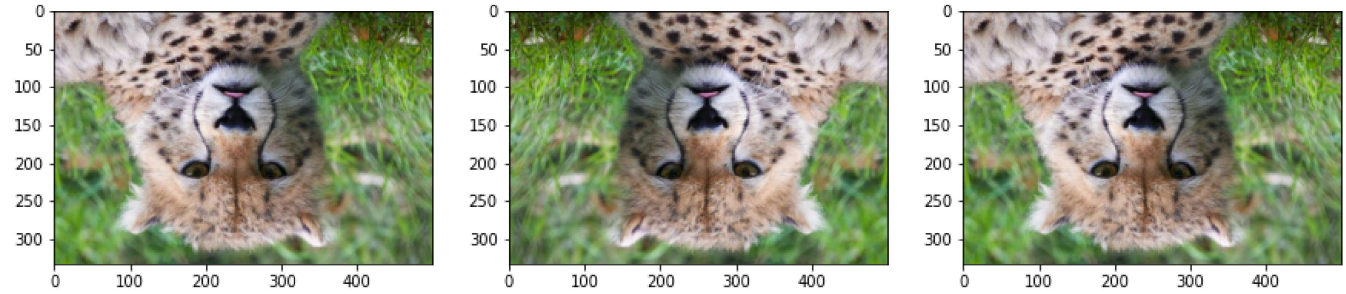
This involves shifting image pixels either horizontally or vertically.

```
import cv2

from numpy import expand_dims
from tensorflow.keras.utils import load_img
from tensorflow.keras.utils import img_to_array
from keras.preprocessing.image import ImageDataGenerator


samples = expand_dims(image, 0)


datagen = ImageDataGenerator(width_shift_range=[-200,200])


it = datagen.flow(samples, batch_size=1)
fig, im = plt.subplots(nrows=1, ncols=3, figsize=(15,15))

for i in range(3):

    # convert to unsigned integers
    image = next(it)[0].astype('uint8')

    # plot image
    im[i].imshow(image)
```

Flipping

This reverses the rows or columns of pixels in either vertical or horizontal cases, respectively.

```
datagen = ImageDataGenerator(horizontal_flip=True, vertical_flip=True)


it = datagen.flow(samples, batch_size=1)
fig, im = plt.subplots(nrows=1, ncols=3, figsize=(15,15))

for i in range(3):

    # convert to unsigned integers
    image = next(it)[0].astype('uint8')

    # plot image
    im[i].imshow(image)
```

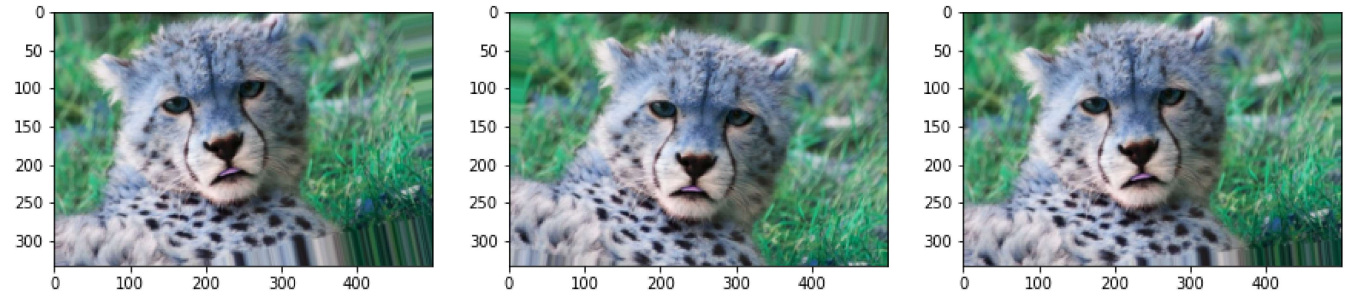


Rotation

This process involves rotating an image by a specified degree.

```
datagen = ImageDataGenerator(rotation_range=20, fill_mode='nearest')


it = datagen.flow(samples, batch_size=1)
fig, im = plt.subplots(nrows=1, ncols=3, figsize=(15,15))

for i in range(3):
```

```
# convert to unsigned integers
t_image = next(it)[0].astype('uint8')

# plot image
im[i].imshow(t_image)
```

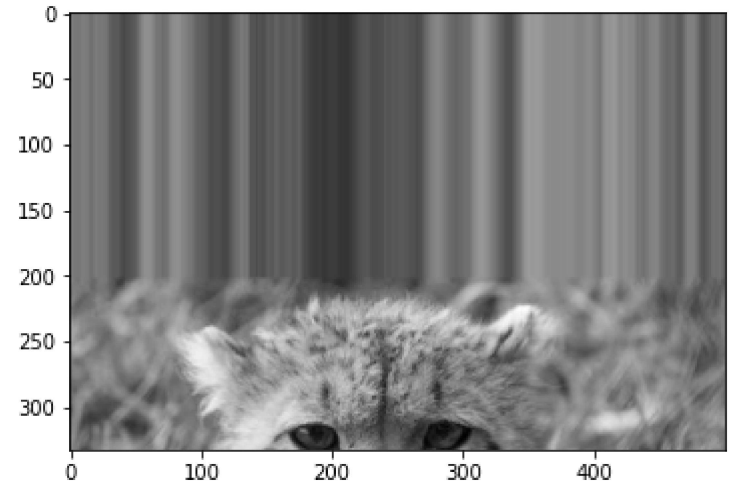


4. Greyscale

Grayscale is simply converting images from colored to black and white.
Why: It is normally used to reduce computation complexity in machine learning algorithms.

```
import skimage


gray_image = skimage.color.rgb2gray(image)
plt.imshow(gray_image, cmap = 'gray')
```

```
<matplotlib.image.AxesImage at 0x7f997ce20070>
```

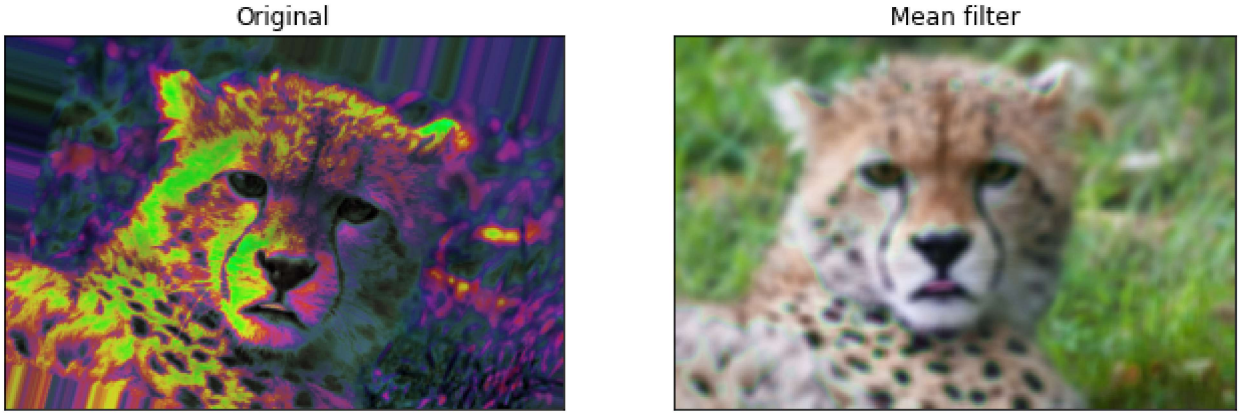


5. Image Filtering

For Python, the Open-CV and PIL packages allow you to apply several digital filters. Applying a digital filter involves taking the convolution of an image with a kernel (a small matrix). For example, you can filter an image to emphasize certain features or remove other features. Image processing operations implemented with filtering include smoothing, sharpening, and edge enhancement.

```
import numpy as np
import cv2
from matplotlib import pyplot as plt
from PIL import Image, ImageFilter
%matplotlib inline
```

```
image_filter = cv2.imread('/content/drive/MyDrive/Animals/Cheetah/animal-africa-wilderness-zo
image_filter = cv2.cvtColor(image_filter, cv2.COLOR_BGR2HSV) # convert to HSV
figure_size = 9 # the dimension of the x and y axis of the kernal.
new_image = cv2.blur(image_filter,(figure_size, figure_size))
plt.figure(figsize=(11,6))
plt.subplot(121), plt.imshow(cv2.cvtColor(image, cv2.COLOR_HSV2RGB)),plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.subplot(122), plt.imshow(cv2.cvtColor(new_image, cv2.COLOR_HSV2RGB)),plt.title('Mean filt
plt.xticks([]), plt.yticks([])
plt.show()
```



Morphological Operations

These are mathematical operations that are used to pull important information from images, like structures and objects. These operations can also be used to enhance the image by removing the noise (errosion) and increasing the size of shapes (dilation). These operations are done using a structuring element, which is a shape that depends on the use case. Important Terminology :

1. Structuring Element : It is a matrix that is moved over an image and its shape is used to extract or modify the useful information in the image.
2. Fit - The pixels in the structuring element overlaps with the objects.
3. Hit - One or more pixels in the structuring element overlap with the pixels of the object being searched
4. Miss - None of the pixels in the structuring element overlap with the pixel being searched. The following figure explains the concept of fit, miss and hit.

Source of Image : Medium Article Types of Morphological Operations : In these operations the value of the resultant pixel depends on the type of morphological operation being used after moving the sthe structuring element.

1. Erosion : It is used to decrease the size of the shapes in the image. Hence can be thought of as removing the noise so that we only have the required shape as one connected object. After the structuring element is placed over the image, the value of the centre pixel is replaced with the minimum value of pixels in the structuring element. In a binary image this value becomes 0 or black. Because the pixel will be replaced with zero objects in the image become smaller.
2. Dilation : As the name suggests it is used to dilate the white pixels or increase the size of objects in the image. In this the value of the centre pixel is replaced by the maximum value of the pixel in the structuring element. This will increase the number of 1s in the image matrix in the case of binary image.

Applications of Morphological Operations:

1. Object Recognition : By using erosion and dilation with a specific size for structuring element in a particular order, depending on the use case, features in an image can be extracted.
2. Image Segmentation : They can also be used to separate features in the image. For instance if there is a binary image where black (pixel value 0) is background and white (pixel value 1) is object, erosion can be used to separate the objects. Vice versa if white is the background.
3. Image Enhancement : It can also be used to remove the noise and improve the sharpness of the image.
4. Image Restoration : They can also be used to repair damaged structures in image or the information that was lost from the image because of poor quality of camera or compression. Compound Operations : As mentioned above image processing algorithms use morphological operations that are a sequence of erosion and dilation. These are called compound operations and they are of the following types :
5. Closing : Dilation followed by erosion
6. Opening : Erosion followed by dilation. The best way to understand image processing concepts is by using images. The following figure explains the two types of compound operations.

```
import cv2
from google.colab.patches import cv2_imshow
import numpy as np

img = cv2.imread('/content/animal-africa-wilderness-zoo.jpg',0)
cv2_imshow(img)
kernel = np.ones((5,5),np.uint8)
erosion = cv2.erode(img,kernel,iterations = 1)
dilate = cv2.dilate(img,kernel,iterations = 1)

cv2_imshow(erosion)
cv2_imshow(dilate)
```

```
from google.colab.patches import cv2_imshow
ret, bw_img = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)

# converting to its binary form
bw = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)
erosion = cv2.erode(bw_img,kernel,iterations = 1)
dilate = cv2.dilate(bw_img,kernel,iterations = 1)

cv2_imshow(bw_img)
cv2_imshow(erosion)
cv2_imshow(dilate)
```

Connected Components

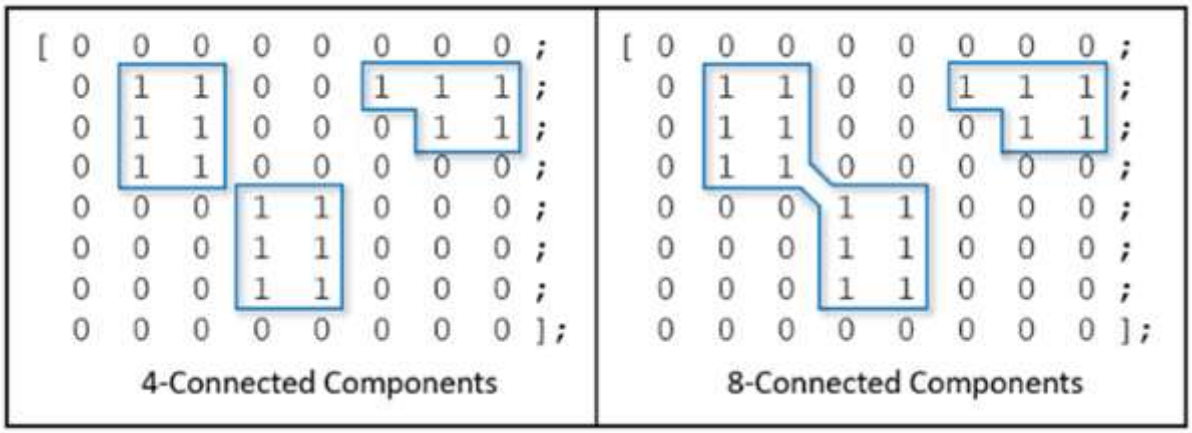
In computer vision, connected components refer to sets of pixels in an image that are connected to each other by some criterion, such as sharing the same color, intensity, or texture. Connected components are important in computer vision for several reasons:

1. Object detection and recognition: Connected components can be used to detect and recognize objects in an image. By identifying groups of pixels that form a connected component, computer vision algorithms can isolate and analyze individual objects within an image.
2. Image segmentation: Connected components can be used to segment an image into regions or objects. By grouping pixels into connected components, computer vision algorithms can separate the foreground from the background, or identify different objects within an image.
3. Feature extraction: Connected components can be used to extract features from an image. By analyzing the properties of connected components, such as their shape, size, and color,

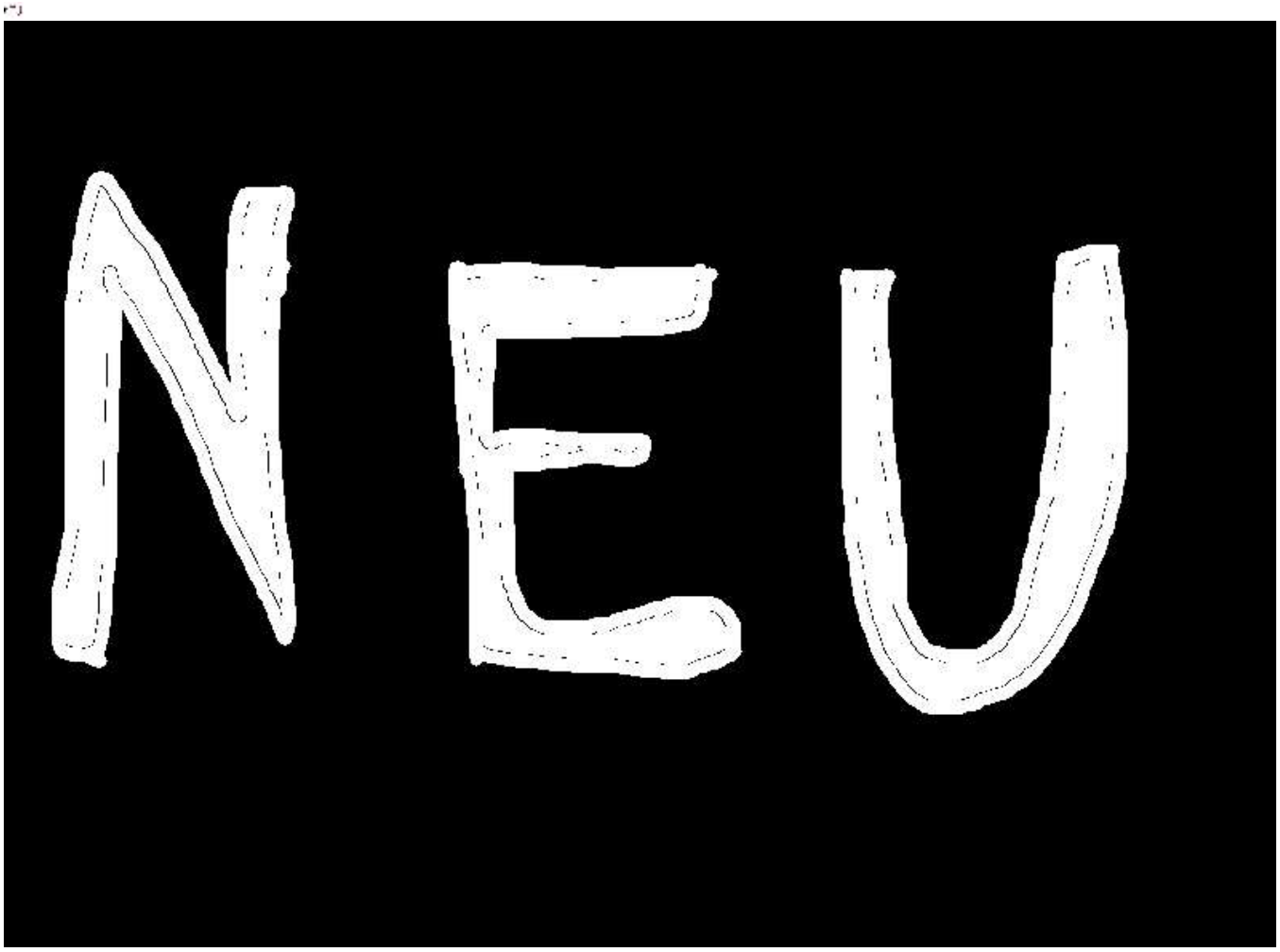
computer vision algorithms can extract meaningful features that can be used for classification or other tasks.

Overall, connected components are a fundamental concept in computer vision that can be used for a wide range of applications, from object detection and recognition to image segmentation and feature extraction.

1. 4-connectivity: If two pixels' edges contact, they are connected. If two pixels are connected in either the horizontal or vertical direction and are both on, they are a single object.
2. 8-connectivity: If two pixels' edges or corners meet, they are connected. If two adjacent pixels are both on and connected in a horizontal, vertical, or diagonal direction, they are a single object.



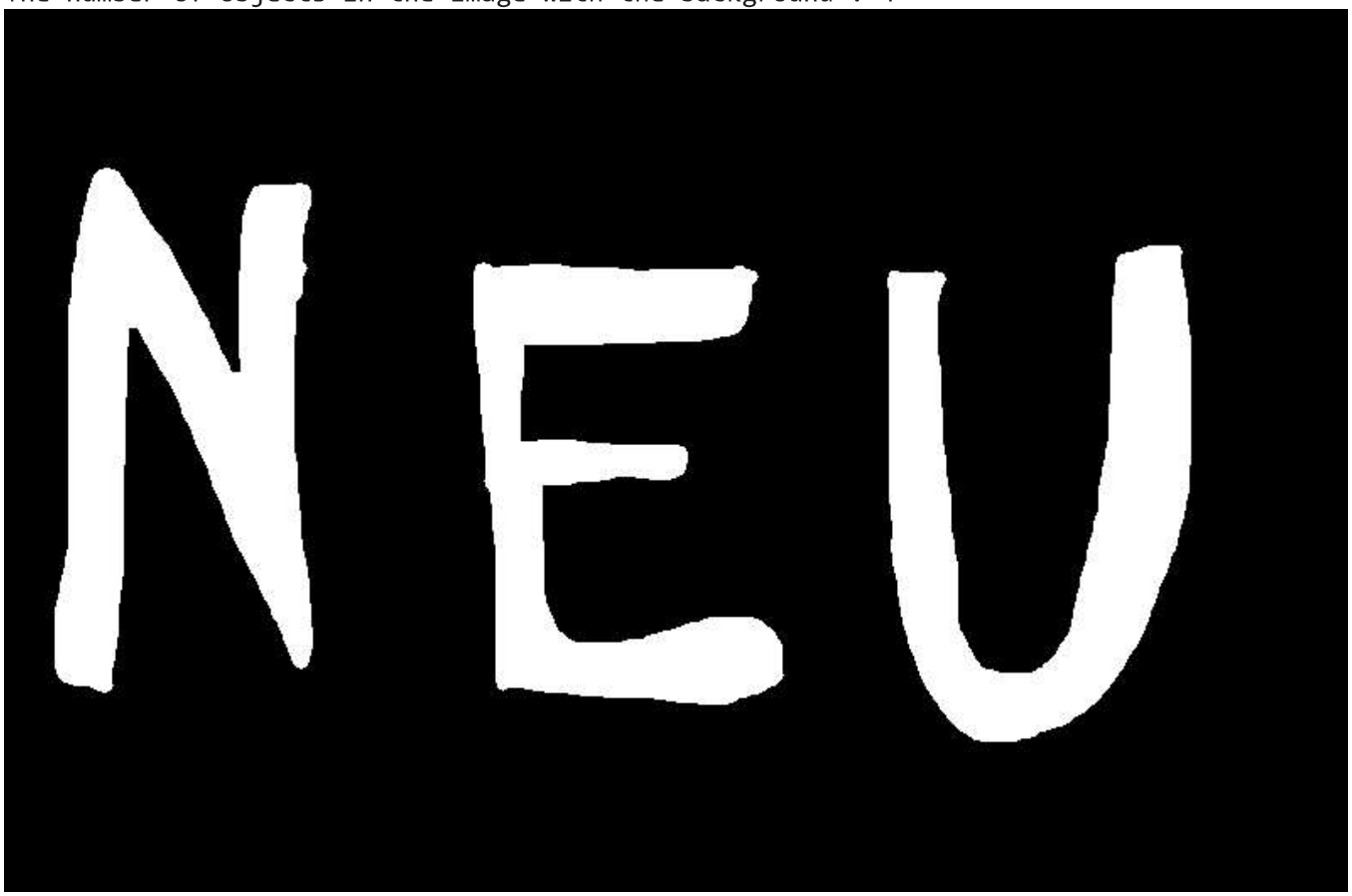

```
from google.colab.patches import cv2_imshow
import cv2
img= cv2.imread("NEU.png")

#img = cv2.resize(img, (int(img.size[0]/4)), int(img.size[1]/4))))

gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
img = cv2.resize(img, (10, 10))
# Threshold the image to create a binary image
ret, binary = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY)
binary = cv2.bitwise_not(binary)
# num_labels, labels, stats, centroids = cv2.connectedComponentsWithStats(binary, connectivit

cv2_imshow(img)
cv2_imshow(binary)
```



As we can see that the binary image has white dots insidw the letters. Hence finding connected components or letters in this case will require some more pre processing. We can use morphological operations discussed above to close the white gaps.

```
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5))
# Perform the closing operation
closing = cv2.morphologyEx(binary, cv2.MORPH_CLOSE, kernel)
num_labels, labels = cv2.connectedComponents(closing, connectivity=8)
print(f"The number of objects in the image with the background : {num_labels}")
cv2_imshow(closing)
```

```
The number of objects in the image with the background : 4
```



We can see that the image has 4 connected objects including the background. If we were to use this to test or train an OCR, the next step would be extracting the three letters accroding to the return value of cv2.connectedComponentsWithStats() and saving it as three different files. The return value will have the pixel coordinates where the detected objets start and end.

Feature Extraction

Feature extraction is a process of selecting and extracting meaningful and informative features from raw data, such as images, that can be used to represent and classify the data. In image processing, feature extraction involves identifying and extracting important patterns or structures from images that can be used to describe their content.

Here are some common methods used for feature extraction from images:

1. Edge detection: Edge detection algorithms identify and extract the boundaries between regions in an image. These boundaries can be used as features for object recognition or segmentation.
2. Histogram-based features: Histogram-based features extract statistical information from the image histogram, such as color or texture information. These features can be used for image classification or clustering.

3. Texture analysis: Texture analysis involves identifying and extracting the patterns or structures that repeat within an image, such as lines, dots, or shapes. These features can be used for image segmentation or classification.
4. Scale-invariant feature transform (SIFT): SIFT is a popular method for detecting and describing key points in an image that are invariant to scaling, rotation, and translation. These key points can be used for object recognition or image matching.
5. Convolutional neural networks (CNNs): CNNs are deep learning models that can learn to extract features automatically from images. These models have been shown to be very effective for a wide range of image processing tasks, including object recognition, image segmentation, and image generation.

Canny Edge Detections

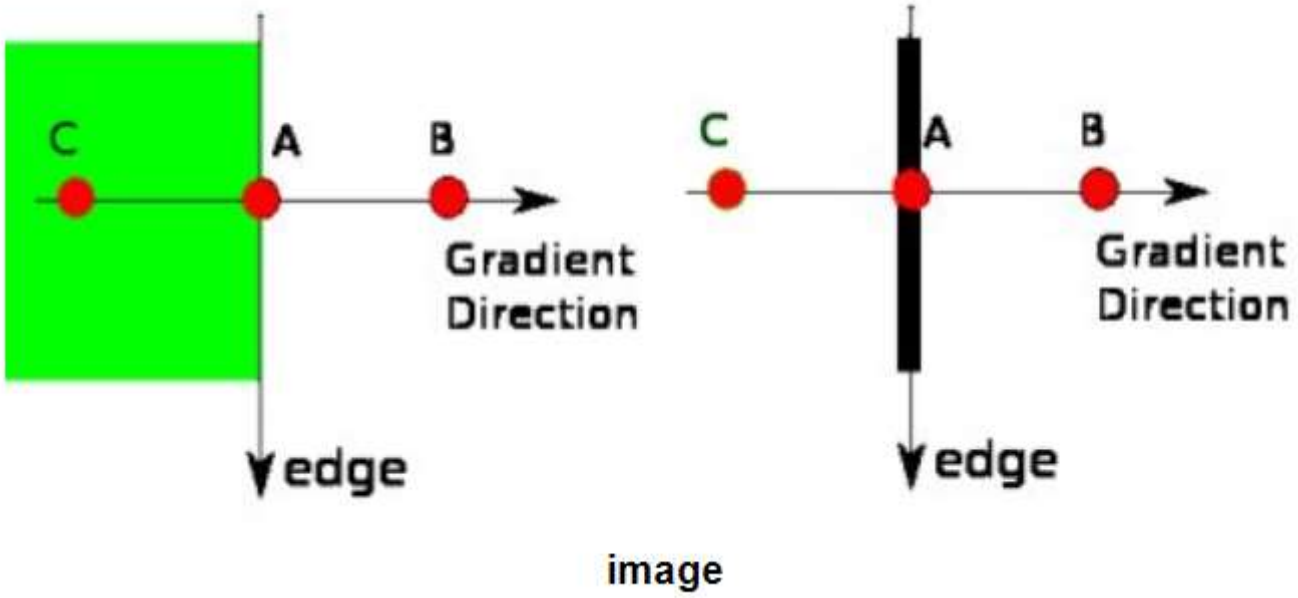
It is a popular edge detection technique developed by John F. Canny.Following are the steps that we need to follow:

1. Remove the noise using Gaussian Blurring
2. Calculate Intensity Gradient:Gradient is the change in direction of intensity level. It can be calculated using the following formula.

$$Edge\_Gradient\ (G) = \sqrt{G_x^2 + G_y^2}$$

$$Angle\ (\theta) = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

3. Non Maximum Suppression This is done to find pixels in the image that might not be the edge of an object. To do this we check if the pixel is a local maximum in its neighbourhood in the direction of the gradient.



image

In this image A is on the edge and the gradient is perpendicular to the edge. If A is the local maxima it is kept and C and B are suppressed.

5. Hysteresis Thresholding

The process of determining which edges are genuine and which ones are not involves utilizing two threshold values, namely minVal and maxVal. Any edges whose intensity gradient exceeds the maximum threshold value (maxVal) are definitely considered as edges, while those whose gradient falls below the minimum threshold value (minVal) are automatically excluded as non-edges. The remaining edges, whose gradients fall between these two threshold values, are evaluated based on their connectivity. If these edges are connected to pixels that are definitely classified as edges, they are deemed to be part of the genuine edges, and if not, they are also excluded.

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
img = cv.imread('NEU.png',0)
edges = cv.Canny(img,100,200)
plt.subplot(121),plt.imshow(img,cmap = 'gray')
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(edges,cmap = 'gray')
plt.title('Edge Image'), plt.xticks([]), plt.yticks([])
plt.show()
```

The MIT License