Name:- Niket Shah **|** **Branch:-** Computer Engineering **|** **Div:-** A

CV : Facial and Emoji Recognition

Pre Processing Steps required on the image:-

Image processing is divided into analogue image processing and digital image processing.

Analogue Image Processing:-The analogue image processing is applied on analogue signals and it processes only two-dimensional signals. The images are manipulated by electrical signals. In analogue image processing, analogue signals can be periodic or non-periodic

Examples of analogue images are television images, photographs, paintings, and medical images etc.

Digital Image Processing:- Digital image processing is the use of computer algorithms to perform image processing on digital images.

As a subfield of digital signal processing, digital image processing has many advantages over analogue image processing. It allows a much wider range of algorithms to be applied to the input data. The aim of digital image processing is to improve the image data by suppressing unwanted distortions and/or enhancement of some important image features so that our AI-Computer Vision models can benefit from this improved data to work on.

An image is nothing more than a two-dimensional array of numbers(or pixels) ranging between 0 and 255. It is defined by the mathematical function f(x,y) where x and y are the two co-ordinates horizontally and vertically.The value of f(x,y) at any point is giving the pixel value at that point of an image.

The steps to be taken are :-

- **Read image**
- **Resize image**
- **Remove noise(Denoise)**
- **Segmentation**
- **Morphology(smoothing edges)**

Read Image:-

In this step, we store the path to our image dataset into a variable then we created a function to load folders containing images into arrays.

```
import os
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import cv2

# defining global variable path
image_path = "Path to your dataset"

''function to load folder into arrays and
then it returns that same array'''
def loadImages(path):
    # Put files into lists and return them as one list of size 4
    image_files = sorted([os.path.join(path, 'train', file)
         for file in os.listdir(path + "/train") if
file.endswith('.png')])

    return image_files
```

Name:- Niket Shah | **Branch:-** Computer Engineering | **Div:-** A

Resize image:-

Some images captured by a camera and fed to our AI algorithm vary in size, therefore, we should establish a base size for all images fed into our AI algorithms.

In this step in order to visualize the change, we are going to create two functions to display the images the first being a one to display one image and the second for two images. After that, we then create a function called processing that just receives the images as a parameter.

```
# Display one image
def display_one(a, title1 = "Original"):
    plt.imshow(a), plt.title(title1)
    plt.xticks([]), plt.yticks([])
    plt.show()

# Display two images
def display(a, b, title1 = "Original", title2 = "Edited"):
    plt.subplot(121), plt.imshow(a), plt.title(title1)
    plt.xticks([]), plt.yticks([])
    plt.subplot(122), plt.imshow(b), plt.title(title2)
    plt.xticks([]), plt.yticks([])
    plt.show()

# Preprocessing
def processing(data):
    # loading image
    # Getting 3 images to work with
    img = [cv2.imread(i, cv2.IMREAD_UNCHANGED) for i in data[:3]]
```

```
# Preprocessing
def processing(data):
    # loading image
    # Getting 3 images to work with
    img = [cv2.imread(i, cv2.IMREAD_UNCHANGED) for i in data[:3]]


    print('Original size',img[0].shape)
    # ---------------------------------
    # setting dim of the resize
    height = 220
    width = 220
    dim = (width, height)
    res_img = []
    for i in range(len(img)):
        res = cv2.resize(img[i], dim, interpolation=cv2.INTER_LINEAR)
        res_img.append(res)

    # Checcking the size
    print("RESIZED", res_img[1].shape)

    # Visualizing one of the images in the array
    original = res_img[1]
    display_one(original)
```

Remove noise:-

Still, inside the function Processing() we add this code to smooth our image to remove unwanted noise. We do this using **Gaussian blur**.

Gaussian blur is the result of blurring an image by a Gaussian function. It is a widely used effect in graphics software, typically to reduce image noise. The visual effect of this blurring technique is a smooth blur resembling that of viewing the image through a translucent screen, distinctly different from the bokeh effect produced by an out-of-focus lens or the shadow of an object under usual illumination. Gaussian smoothing is also used as a pre-processing stage in computer vision algorithms in order to enhance image structures at different scales.

```
# ---------------------------------
# Remove noise
# Gaussian
no_noise = []
for i in range(len(res_img)):
    blur = cv2.GaussianBlur(res_img[i], (5, 5), 0)
    no_noise.append(blur)


image = no_noise[1]
display(original, image, 'Original', 'Blured')
#---------------------------------
```

Segmentation & Morphology:-

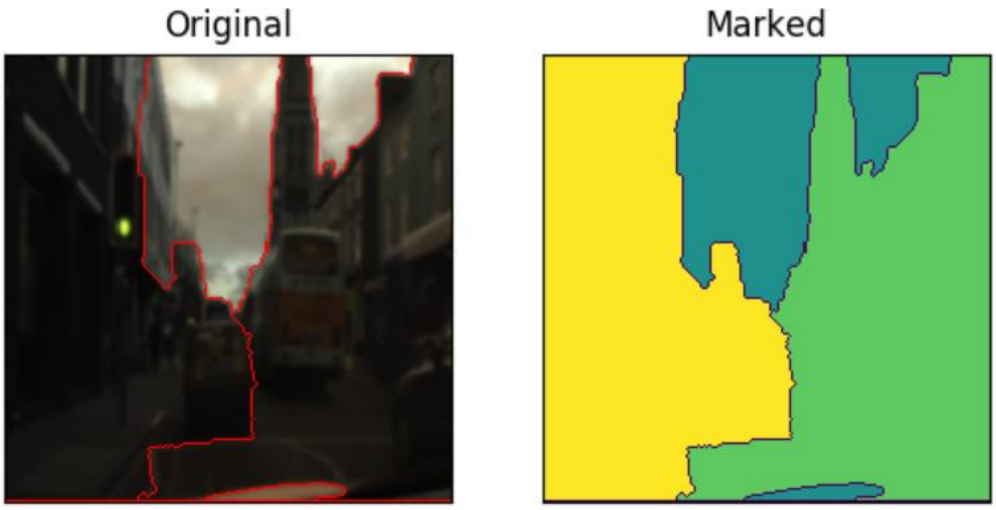
Still, inside the function Processing() we add this code. In this step, we are going to segment the image, separating the background from foreground objects and we are going to further improve our segmentation with more noise removal.

```
# Further noise removal
kernel = np.ones((3, 3), np.uint8)
opening = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel,
iterations=2)

# sure background area
sure_bg = cv2.dilate(opening, kernel, iterations=3)

# Finding sure foreground area
dist_transform = cv2.distanceTransform(opening, cv2.DIST_L2, 5)
ret, sure_fg = cv2.threshold(dist_transform, 0.7 *
dist_transform.max(), 255, 0)

# Finding unknown region
sure_fg = np.uint8(sure_fg)
unknown = cv2.subtract(sure_bg, sure_fg)

#Displaying segmented back ground
display(original, sure_bg, 'Original', 'Segmented Background')
```

Now, we separate different objects in the image with markers.

```
# Marker labelling
ret, markers = cv2.connectedComponents(sure_fg)

# Add one to all labels so that sure background is not 0, but 1
markers = markers + 1

# Now, mark the region of unknown with zero
markers[unknown == 255] = 0

markers = cv2.watershed(image, markers)
image[markers == -1] = [255, 0, 0]

# Displaying markers on the image
display(image, markers, 'Original', 'Marked')
```

Name:- Niket Shah **|** **Branch:-** Computer Engineering **|** **Div:-** A



Final Output

Techniques used for classification in image:-

The 3 main types of image classification techniques**:-**

- Unsupervised image classification
- Supervised image classification
- Object-based image analysis

Unsupervised and supervised image classification are the two most common approaches.

However, object-based classification has gained more popularity because it's useful for high-resolution data.

Name:- Niket Shah **|** **Branch:-** Computer Engineering **|** **Div:-** A

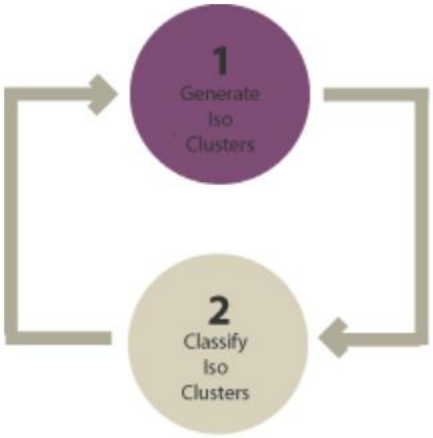
1. Unsupervised Classification

In unsupervised classification, it first groups pixels into “clusters” based on their properties. Then, you classify each cluster with a land cover class.

Overall, unsupervised classification is the most basic technique. Because you don’t need samples for unsupervised classification, it’s an easy way to segment and understand an image.

The two basic steps for unsupervised classification are:

- Generate clusters
- Assign classes



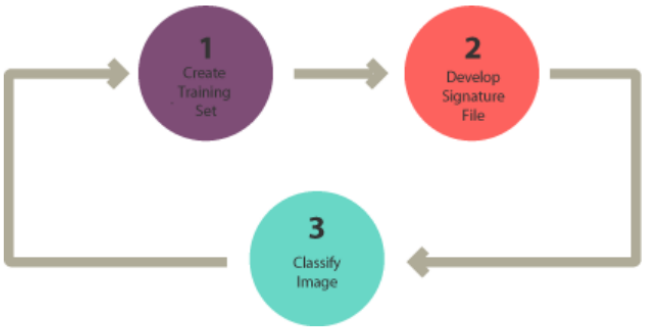
Name:- Niket Shah | **Branch:-** Computer Engineering | **Div:-** A

2. Supervised Classification

In supervised classification, you **select representative samples** for each land cover class. The software then uses these **"training sites"** and applies them to the entire image.

The three basic steps for supervised classification are:

- Select training areas
- Generate signature file
- Classify



Name:- Niket Shah | **Branch:-** Computer Engineering | **Div:-** A

3. Object-Based Image Analysis (OBIA)

Supervised and unsupervised classification is pixel-based. In other words, it creates square pixels and each pixel has a class. But object-based image classification groups pixels into representative vector shapes with size and geometry.

Here are the steps to perform object-based image analysis classification:

- Perform multiresolution segmentation
- Select training areas
- Define statistics
- Classify