RAMAIAH UNIVERSITY OF APPLIED SCIENCES

ASSIGNMENT

Course Code	CSE407A
Course Name	Computer Vision
Programme	B.Tech
Department	CSE
Faculty	FET

Name of the Student	K Srikanth
Reg. No	17ETCS002124
Semester/Year	7th/ 4th Year
Course Leader/s	Dr. Subarna Chatterjee

Declaration Sheet				
Student Name	K Srikanth			
Reg. No	17ETCS002124			
Programme	B.Tech		Semester/Year	7^{th}/ 4^{th} Year
Course Code	CSE407A			
Course Title	Computer Vision			
Course Date		to		
Course Leader	Dr. Subarna Chatterjee			

Declaration

The assignment submitted herewith is a result of my own investigations and that I have conformed to the guidelines against plagiarism as laid out in the Student Handbook. All sections of the text and results, which have been obtained from other sources, are fully referenced. I understand that cheating and plagiarism constitute a breach of University regulations and will be dealt with accordingly.

Signature of the Student	Srikanth K	Date	24/11/2021
Submission date stamp (by Examination & Assessment Section)			

Signature of the Course Leader and date	Signature of the Reviewer and date

Engineering and Technology			
Ramaiah University of Applied Sciences			
Department	Computer Science and Engineering	Programme	B. Tech.
Semester/Batch	7th/2018		
Course Code	**CSE407A**	Course Title	Computer Vision
Course Leader(s)	Dr. Aruna Kumar S V., Dr. Divya BS and Dr. Subarna Chatterjee		

Questions		Marking Scheme	Marks		
			Max Marks	First Examiner Marks	Moderator
1					
	1.1	Implementation of noise reduction filter	5		
	1.2	Implementation of histogram equalization filter	7		
	1.3	Reduction of the salt-and-pepper noise from the input image	4		
	1.4	Extraction of the gradient parts from the input image	4		
	1.5	Conclusion and Recommendations	3		
	Question 1 Max Marks		**23**		
2	2.1	Introduction to Segmentation and Creation of Dataset.	3		
	2.2	Identification and explanation of the appropriate pre-processing techniques	5		
	2.3	Identification and explanation of the appropriate Segmentation techniques	2		
	Question 2 Max Marks		**10**		
3	3.1	Pre-processing on the images of the created dataset	**5**		
	3.2	Segmentation of the images.	**7**		
	3.3	Results and Discussions.	**5**		
	Question 3 Max Marks		**17**		
Total Assignment Marks			**50**		

Course Marks Tabulation				
Question	**First Examiner**	**Remarks**	**Moderator**	**Remarks**
1				
2				
3				
Marks (Max 50)				

Signature of First Examiner **Signature of Moderator**

Question 1

Q-1.1

Introduction

Noise filtering is a set of processes that is performed to remove the noise contained with the data acquired on construction and infrastructure sites. The 3D data obtained need to be processed, and the contained noise should be eliminated or reduced. The refined point cloud then better represents the 3D objects and elements and the geometric information is then processed more accurately and reliably.

Now, let's try to pass our input image and through all kinds of different noise filters.

- **Arithmetic Mean Filter**

  An arithmetic mean filter operation on an image removes short tailed noise such as uniform and Gaussian type noise from the image at the cost of blurring the image. The arithmetic mean filter is defined as the average of all pixels within a local region of an image.

  The arithmetic mean is defined as:

$$\bar{x} = \frac{1}{n}(x_1 + \cdots + x_n)$$

  Pixels that are included in the averaging operation are specified by a mask. The larger the filtering mask becomes the more predominant the blurring becomes and less high spatial frequency detail that remains in the image.

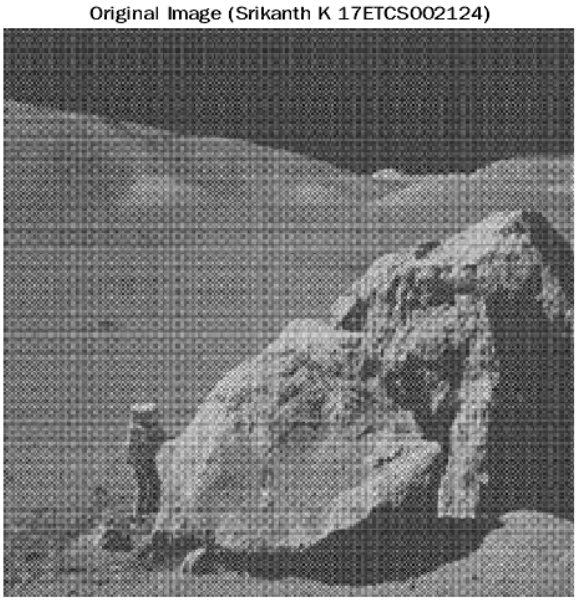
Name: Srikanth K Registration Number: 17ETCS002124

MATLAB Code

```matlab
% Srikanth K 17ETCS002124
x=imread('Q1_1.tif');
x1=imresize(x,[256,256]);
figure;imshow(x1)
title('Original Image (Srikanth K 17ETCS002124)')
x2=im2gray(x1);
figure;imshow(x2);title('Gray Image (Srikanth K 17ETCS002124)');
[m,n]=size(x2);
for i=1:m-3
    for j=1:n-3
        a=x(i:i+3,j:j+3);
        v(i,j)=sum(sum(a));
    end
end
y=mat2gray(v);
figure;imshow(y);
title('Output: Arithmetic Mean Filter (Srikanth K 17ETCS002124)');
```

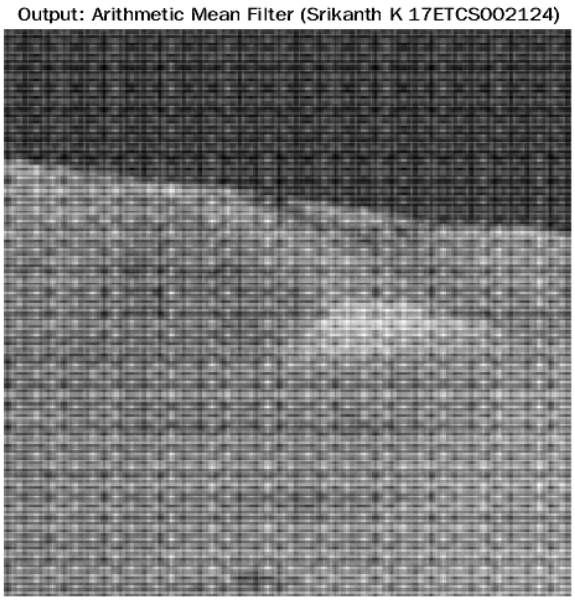
Output

Original Image



Name: Srikanth K Registration Number: 17ETCS002124

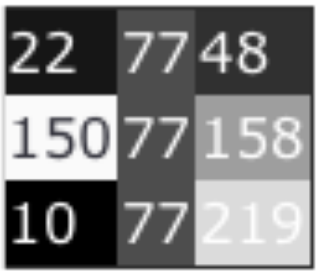
Arithmetic Mean Filter Output



- **Max and Min Filter**

  The "Max - Min" filter blurs the image by replacing each pixel with the difference of the highest pixel and the lowest pixel (with respect to intensity) within the specified window size.

  For example, given the grayscale 3x3 pixel neighborhood;



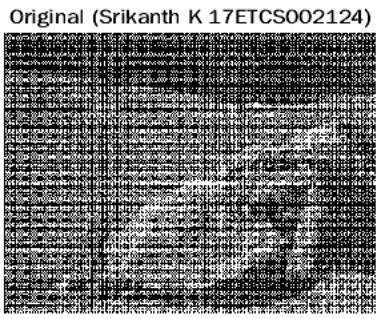
  The centre pixel would be changed from 77 to 209 as it is the difference between the brightest pixel 219 and the darkest pixel 10 within the current window.

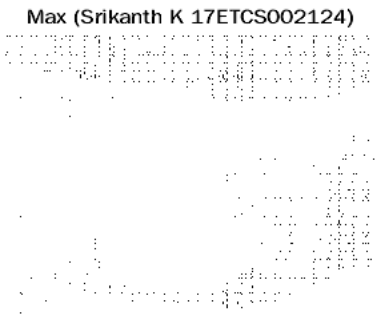
  It is interesting to note that at a small window size (2 or 3) this filter is an effective edge detection filter aswel.

MATLAB Code

```
% Srikanth K 17ETCS002124
Original=imread('Q1_1.tif');
BW = im2bw(Original,0.6);
minf=@(x) min(x(:));
maxf=@(x)max(x(:));
min_Image=nlfilter(BW,[3 3],minf);
max_Image=nlfilter(BW,[3 3],maxf);

subplot(2,2,1), imshow(BW), title('Original (Srikanth K 17ETCS002124)');
subplot(2,2,2), imshow(min_Image), title('Min (Srikanth K 17ETCS002124)');
subplot(2,2,3), imshow(max_Image), title('Max (Srikanth K 17ETCS002124)');

```

Output





- **Alpha trimmed Mean Filter**

This function will filter the image by the nonlinear alpha-trimmed mean method and works for only monochrome, 8 bit per pixel and 24 bit per pixel images. The alpha-trimmed mean filter is based on order statistics and varies between a median and a mean filter. It is used when an image contains both short and long tailed types of noise (e.g., both Gaussian and

salt and pepper noise). To define the alpha-trimmed mean filter, all pixels surrounding the pixel at the coordinate (x,y) in the image A which are specified by an input N x N size square mask A(i) are ordered from minimum to maximum value. The alpha-trimmed mean filter is given as,

$$Alpha - TrimmedMean(A) = \frac{1}{N*N-2P} \sum_{i=P}^{N*N-P} A_i$$

MATLAB Code

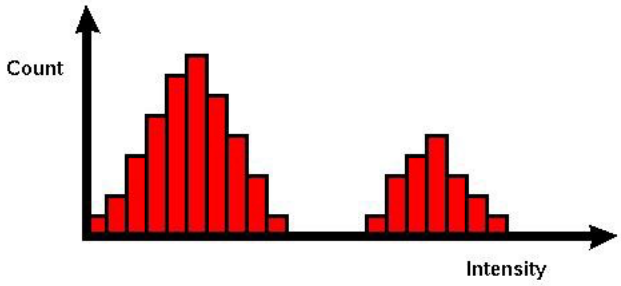
```
% Srikanth K 17ETCS002124
data=imread('Q1_1.tif');
figure,imshow(data);, title('Original Image (Srikanth K 17ETCS002124)');
data=im2gray(data);
data=im2double(data);
masksize=2;
d=4;
figure,imshow(data)
[ro col]=size(data);
temp1=[];
graber=0;
akkumulator=[];
for i=1:ro;
    for j=1:col;
        for m=-masksize:masksize;
            for n=-masksize:masksize;
                if (i+m>0 && i+m<ro && j+n>0 && j+n<col && ...
                        masksize+m>0 && masksize+m<ro && ...
                        masksize+n>0 && masksize+n<col)

                    temp1=[temp1 data(i+m,j+n)];

                end
            end
        end

        temp1=sort(temp1);
        lenth=length(temp1);
        for k=((d/2)-1):(lenth-(d/2))
            akkumulator=[akkumulator temp1(k)];
        end
        akkumulator=sum(akkumulator);
        reformedimage(i,j)=(akkumulator) / (25-d);

        akkumulator=[];
        temp1=[];

    end
end

figure,imshow(reformedimage), title('Alpha Trimmed Mean Filter Image (Srikanth K 17ETCS002124)');
```

Output

Q-1.2

What is a Histogram of An Image ?

A histogram of an image is the graphical interpretation of the image's pixel intensity values. It can be interpreted as the data structure that stores the frequencies of all the pixel intensity levels in the image.



Histogram of an Image

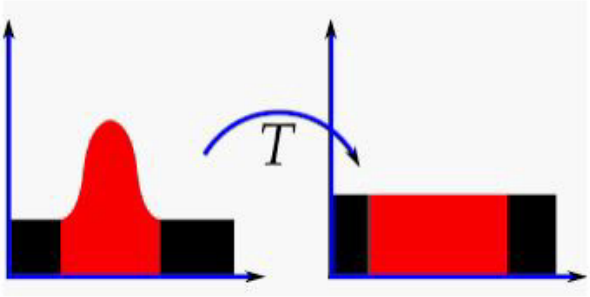
Histogram of an Image ?

As we can see in the image above, the X-axis represents the pixel intensity levels of the image. The intensity level usually ranges from 0 to 255. For a gray-scale image, there is only one histogram, whereas an RGB colored image will have three 2-D histograms — one for each color. The Y-axis of the histogram indicates the frequency or the number of pixels that have specific intensity values.

Name: Srikanth K Registration Number: 17ETCS002124

What is Histogram Equalization?

Histogram Equalization is an image processing technique that adjusts the contrast of an image by using its histogram. To enhance the image's contrast, it spreads out the most frequent pixel intensity values or stretches out the intensity range of the image. By accomplishing this, histogram equalization allows the image's areas with lower contrast to gain a higher contrast.

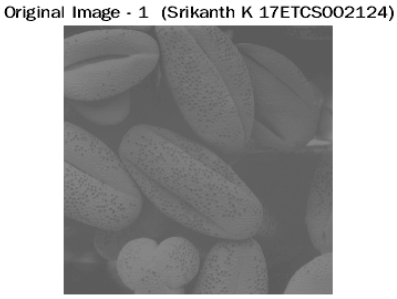


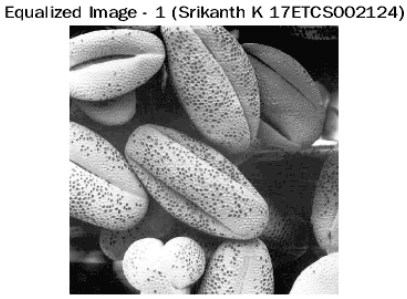
MATLAB Code

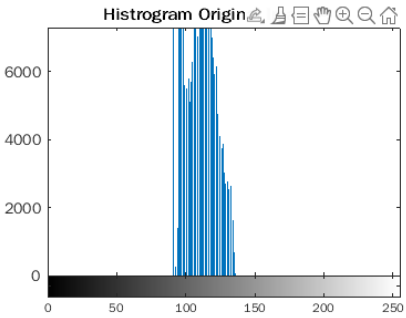
```
% Srikanth K 17ETCS002124

% Image Q1_2
a = imread('Q1_2.tif');
heq = histeq(a)
subplot(221);imshow(a);title('Original Image - 1  (Srikanth K 17ETCS002124)')
subplot(222);imshow(heq);title('Equalized Image - 1 (Srikanth K 17ETCS002124)')
subplot(223);imhist(a);title('Histrogram Original Image - 1')
subplot(224);imhist(heq);title('Histrogram Equalized Image - 1 (Srikanth K 17ETCS002124)')

% Image Q1_3
a = imread('Q1_3.tif');
heq = histeq(a)
subplot(221);imshow(a);title('Original Image - 2  (Srikanth K 17ETCS002124)')
subplot(222);imshow(heq);title('Equalized Image - 2 (Srikanth K 17ETCS002124)')
subplot(223);imhist(a);title('Histrogram Original Image - 2')
subplot(224);imhist(heq);title('Histrogram Equalized Image - 2 (Srikanth K 17ETCS002124)')


```

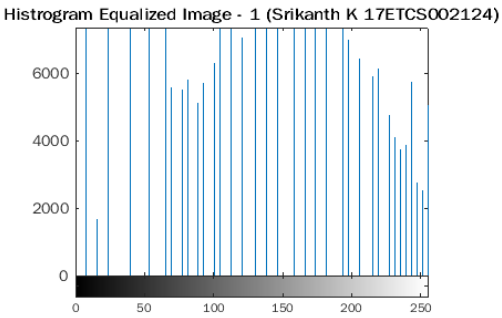
Name: Srikanth K Registration Number: 17ETCS002124

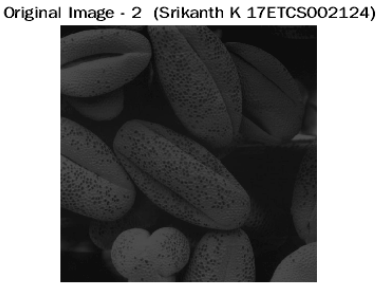
Output

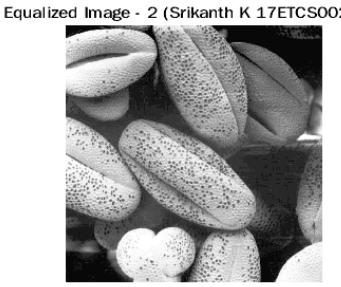


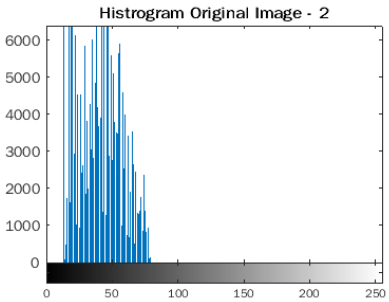


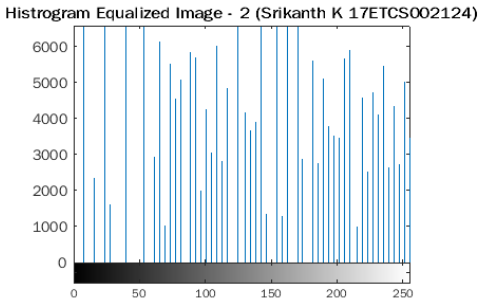












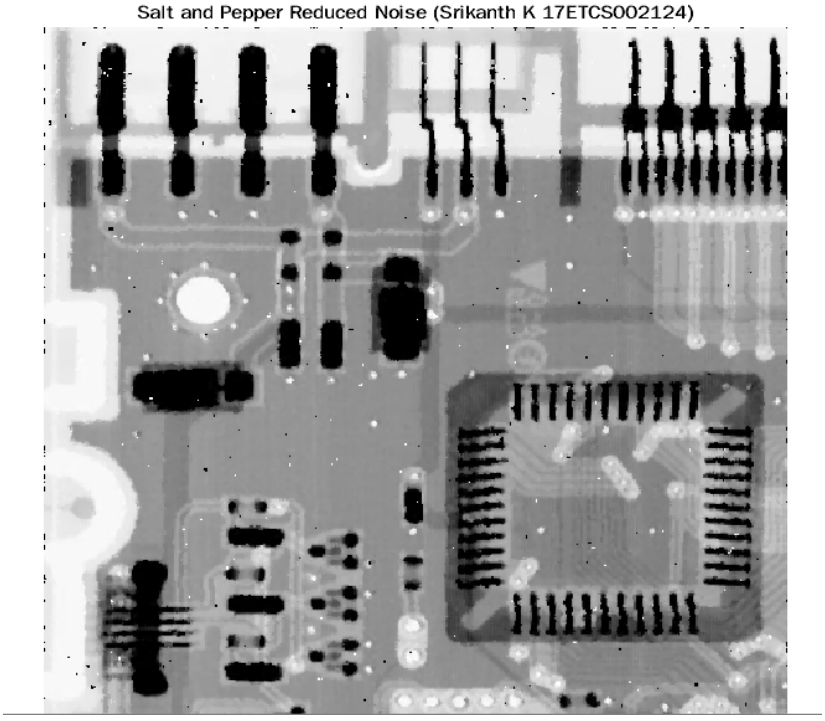
Q-1.3

Salt and pepper noise is an impulse type of noise in images. We consider salt-and-pepper noise, for which a certain amount of the pixels in the image are either black or white (black or white dots). Normally if there is black dots in the image we called it pepper noise and if there is white dots in the image we called it salt noise. This noise is generally caused by errors in data transmission, failure in memory cell or analog-to-digital converter errors.

MATLAB Code

```
% Srikanth K 17ETCS002124
im = imread('Q1_4.tif');
image = imnoise(im,'salt & pepper',0.01);
M = medfilt2(image);
imshow(M), title('Salt and Pepper Reduced Noise (Srikanth K 17ETCS002124)');
```

Output



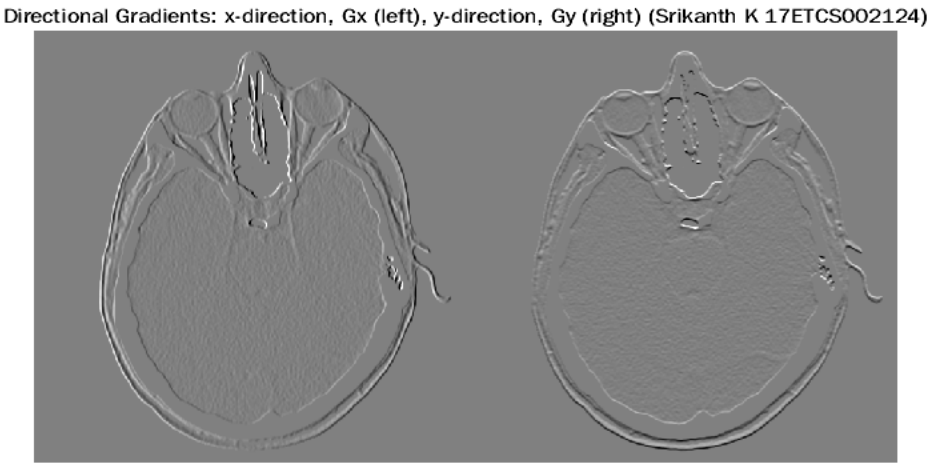
Q-1.4

An image gradient is a directional change in the intensity or color in an image. The gradient of the image is one of the fundamental building blocks in image processing. For example, the Canny edge detector uses image gradient for edge detection. In graphics software for digital image editing, the term gradient or color gradient is also used for a gradual blend of color which can be considered as an even gradation from low to high values, as used from white to black in the images to the right.

MATLAB Code

```
% Srikanth K 17ETCS002124
I = imread('Q1_5.tif');
[Gx, Gy] = imgradientxy(I,'prewitt');
figure
imshowpair(Gx, Gy, 'montage');
title('Directional Gradients: x-direction, Gx (left), y-direction, Gy (right) (Srikanth K 17ETCS002124)')
axis off;

```

Output



Q-1.5

We first implemented the noise reduction function and ran it over Q1_1.tif to produce a denoised image. We performed histogram equalization on Q1_2.tif and Q1_3.tif to enhance contrast. We reduced the salt and pepper noise from the image Q1_4.tif. We at the end extracted gradient parts from the image Q1_5.tif. One of the fundamental challenges in image processing and computer vision is image denoising. What denoising does is to estimate the original image by suppressing noise from the image. Image noise may be caused by different sources (from sensor or from environment) which are often not possible to avoid in practical situations. Therefore, image denoising plays an important role in a wide range of applications such as image restoration, visual tracking, image registration, and image segmentation. Histogram equalization is used to enhance contrast. A histogram of an image is the graphical interpretation of the image's pixel intensity values. It can be interpreted as the data structure that stores the frequencies of all the pixel intensity levels in the image. It is not necessary that contrast will always be increase in this. There may be some cases where histogram equalization can be worse. In those cases the contrast is decreased. histogram equalization allows the image's areas with lower contrast to gain a higher contrast. Salt-and-pepper noise is a special form of noise sometimes also known as impulse noise. This noise can be caused by sharp and sudden disturbances in the image signal. It presents itself as sparsely occurring white and black pixels. An effective noise reduction method for this type of noise is a use of filter among which Median filter is quite popular. Image gradients can be used to extract information from images. Gradient images are created from the original image (generally by convolving with a filter) for this purpose. Each pixel of a gradient image measures the change in intensity of that same point in the original image, in a given direction. To get the full range of direction, gradient images in the x and y directions are computed.

One of the most common uses is in edge detection. After gradient images have been computed, pixels with large gradient values become possible edge pixels. The pixels with the largest gradient values in the direction of the gradient become edge pixels, and edges may be traced in the direction perpendicular to the gradient direction. One example of an edge detection algorithm that uses gradients is the Canny edge detector. Image gradients can also be used for robust feature and texture matching.

As we have seen from the above examples, The aim of these image enhancement techniques is to improve the information in images for human viewers, or to provide “better “input for other automated image processing tasks. There is no general theory for determining what a good image enhancement technique is, each technique has a specific purpose.

Question 2

Q-2.1

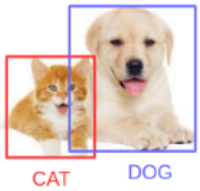
Introduction

Let’s understand image segmentation using a simple example. Consider the below image:

There’s only one object here – a dog. We can build a straightforward cat-dog classifier model and predict that there’s a dog in the given image. But what if we have both a cat and a dog in a single image?

We can train a multi-label classifier, in that instance. Now, there's another caveat – we won't know the location of either animal/object in the image. That's where image localization comes into the picture (no pun intended!). It helps us to identify the location of a single object in the given image. In case we have multiple objects present, we then rely on the concept of object detection (OD). We can predict the location along with the class for each object using OD.

Image Localization



Object Detection

The Different Types of Image Segmentation

1. **Region-based Segmentation**
2. **Edge Detection Segmentation**
3. **Image Segmentation based on Clustering**

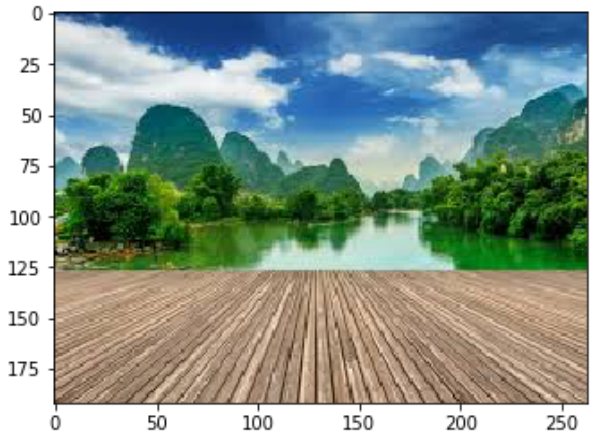
Now let's look at the first two image segmentation techniques.

Region- Based Segmentation

One simple way to segment different objects could be to use their pixel values. An important point to note – the pixel values will be different for the objects and the image's background if there's a sharp contrast between them. In this case, we can set a threshold value. The pixel values falling below or above that threshold can be classified accordingly (as an object or the background). This technique is known as Threshold Segmentation. If we want to divide the image into two regions (object and background), we define a single threshold value. This is known as the global threshold. If we have multiple objects along with the background, we must define multiple thresholds. These thresholds are collectively known as the local threshold.

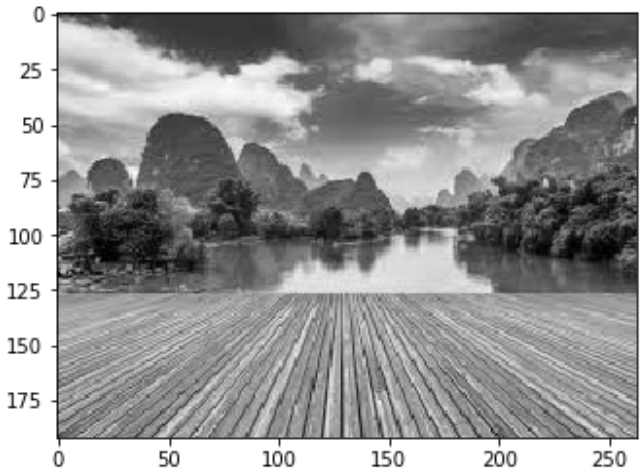
Name: Srikanth K Registration Number: 17ETCS002124

Input Image



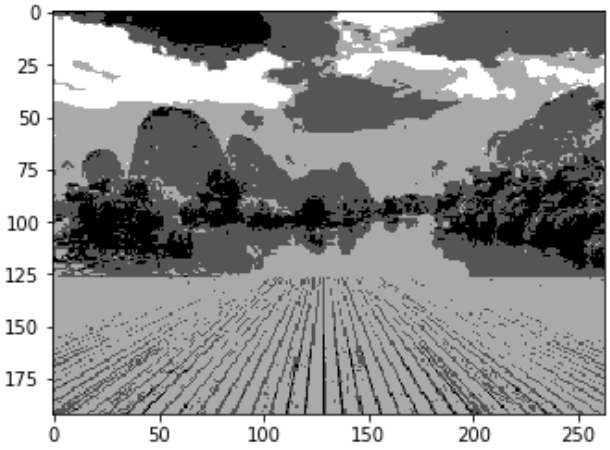
It is a three-channel image (RGB). We need to convert it into grayscale so that we only have a single channel.

Greyscale Image



Now, we want to apply a certain threshold to this image. This threshold should separate the image into two parts – the foreground and the background. The height and width of the image is 192 and 263 respectively. We will take the mean of the pixel values and use that as a threshold. If the pixel value is more than our threshold, we can say that it belongs to an object. If the pixel value is less than the threshold, it will be treated as the background.

Segmented Image



There are three different segments in the above image. You can set different threshold values and check how the segments are made. Some of the advantages of this method are:

- Calculations are simpler
- Fast operation speed
- When the object and background have high contrast, this method performs really well

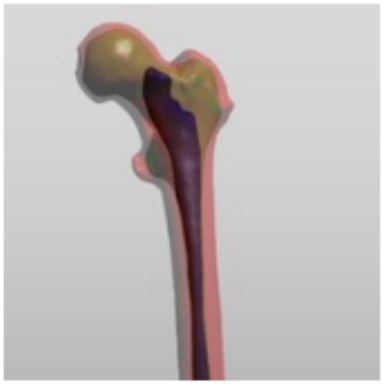
But there are some limitations to this approach. When we don't have significant grayscale difference, or there is an overlap of the grayscale pixel values, it becomes very difficult to get accurate segments.

Q-2.2

Image segmentation is the process of dividing an image (a digital image) into multiple disjoint regions, where pixels in the same region share similar properties or belong to the same object. Image segmentation changes the representation of an object, making it meaningful and easier to analyze.

Image segmentation has variety of applications, such as:

- **Medical Imaging :** Image segmentation is applied extensively in medical applications to locate tumours and other pathologies, measure tissue volume, diagnosis and surgery planning, simulation, and navigation.
- **Recognition :** Image segmentation is applied in face recognition to identify multiple faces in an image, fingerprint recognition to highlight the contours and edges of a fingerprint and in iris recognition.
- **Object detection :** Image segmentation is applied in pedestrian detection, to identify the pedestrians on a road, brake light segmentation, and locating objects in satellite images.



Dataset creation is a long and drawn-out process which involves collection of the data samples (such as images, or any other required data) and to label the images in an appropriate manner which depends on the problem that is being tackled. Dataset creation plays an important role in segmentation algorithms that involve neural networks. Errors during the creation of a dataset significantly impacts the performance of the segmentation model, often leading to disastrous consequences.

Pre-Processing Techniques

Image preprocessing techniques help to prepare the image for segmentation. Almost always, the image to segment that is obtained in a production environment are not as “perfect” as the ones that are available during testing or deployment. Often, the images received will have noise, objects in the image will not be aligned properly, the image can be blurry, etc. This section will discuss about pre-processing techniques that are usually applied on image segmentation and image detection tasks during training in a neural network-based algorithm.

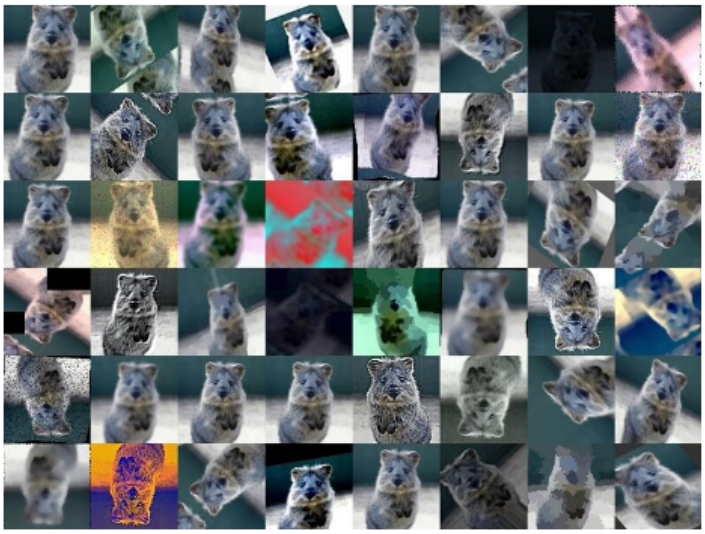
The primary focus on pre-processing techniques in the training phase of a neural network-based segmentation algorithm is to augment the dataset and try to represent real world scenarios during training. Such techniques include:

- **Scale, translation, and rotation**

  The images in the dataset can be scaled (resized), translated or rotated. This makes the network less sensitive to the size, location, and rotation of the object in the image.

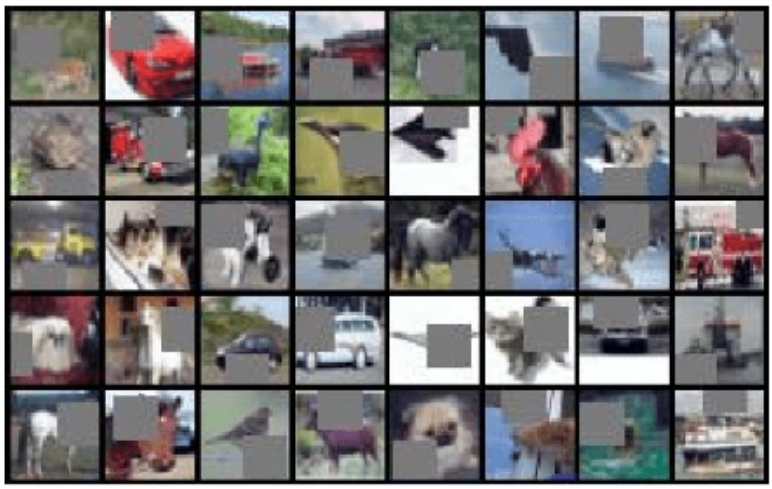
- **Pixel Operations**

  Such operations include color shifting, color whitening, blurring, denoising etc. These operations further enhance the dataset.



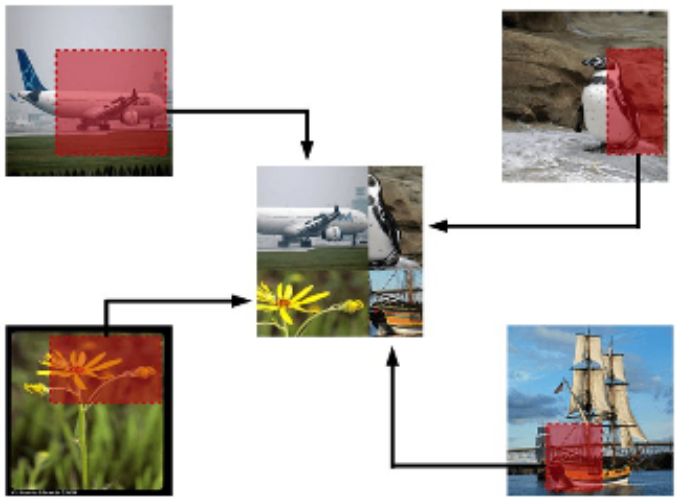
- **Cutout**

  CutOut is an image augmentation technique that cuts out parts of the image. This is a highly effective technique in object detection tasks since the entire object is not always visible in an image, only a part of it is visible.



- **RICAP**

  RICAP crops four training images and patches them to construct a new training image; it selects images and determines the cropping sizes randomly, where the size of the final image is identical to that of the original image. RICAP also mixes class labels of the four images with ratios proportional to the areas of the four images.



- **Patch Gaussian**

  Patch Gaussian is a technique that combined the accuracy improvement of CutOut and robustness provided by additive gaussian noise. It is a simple augmentation scheme that adds noise to randomly selected patches in an input image. In this way, the model gains robustness to adversarial attacks and has a high enough accuracy.

Q-2.3

There are multiple segmentation techniques, which can be grouped into two general classes:

1. Classical methods
2. AI based methods.

There are a wide variety of classical methods, such as

Thresholding methods

Thresholding methods are the simplest techniques of image segmentation. This method clips the pixel values that are over a certain threshold, which generate a binary image. The key to successful segmentation is to select a good threshold value. Methods such as balanced histogram, Otsu's method and k-means clustering fall under this category.

• In brief, Otsu's (Nobuyuki Otsu) method involves calculating a single intensity threshold that separate pixels into background pixels and foreground pixels. This is done by minimizing the inter class intensity variance by exhaustively searching for it. The inter class variance is defined as the weighted sum of the variances of the two classes (background and foreground):

$$\sigma_w^2(t) = w_0(t)\sigma_0^2(t) + w_1(t)\sigma_1^2(t)$$

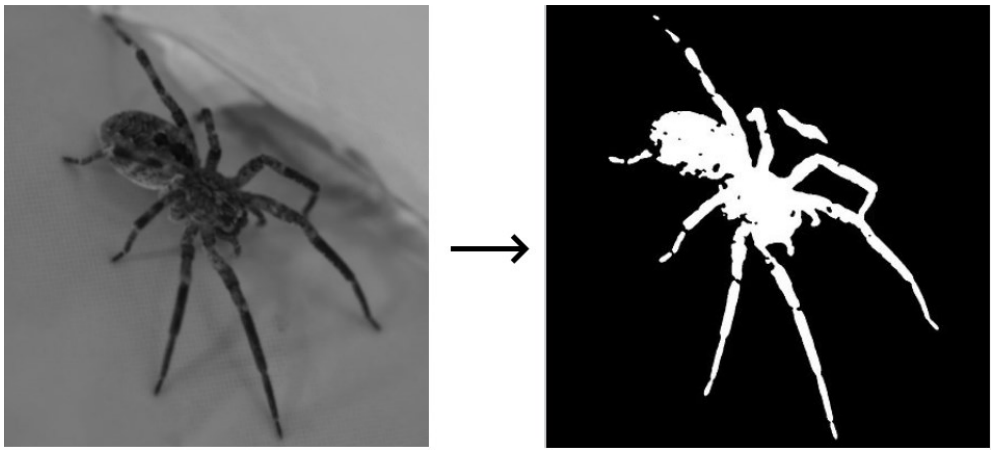
Where,

$$w_0(t) = \sum_{i=0}^{t-1} p(i)$$

$$w_t(t) = \sum_{i=t}^{L-1} p(i)$$

in which w_0 and w_1 are the probabilities of the two classes that are separated by a threshold t and L is the number of bins in the histogram.

Balanced histogram thresholding is a method like Otsu's method, but tries to find a threshold level that divides the histogram of the image into two classes. It does this by "weighing" the histogram and checking which side of the histogram (that is divided by the threshold value) is heavier. It removes the weight from the heavier side till it becomes lighter. This operation is repeated till the edges of the weighing scale meet.



Region growing methods

Region growing methods involve the growth of regions that encapsulate the objects in the image. They operate on the assumption that neighbouring pixels within one region have similar values, and thus, belong to the same object. Some region growing methods are:

- **Seeded region growing:** This method takes a set of seeds as input along with the image. The seeds mark each of the objects to be segmented. The regions are iteratively grown by comparison of all unallocated neighbouring pixels to the regions. The difference between a pixel's intensity value and the region's mean, is used as a measure of similarity. The pixel with the smallest difference measured in this way is assigned to the respective region. This process continues until all pixels are assigned to a region. Since this method relies on an extra input that is the seed, the results that are obtained by this method differ with different seeds.
- **Unseeded region growing:** It is a modified algorithm that does not require explicit seeds. It starts with a single region—the pixel chosen here does not markedly influence the final segmentation. At each iteration it considers the neighbouring pixels in the same way as seeded region growing. It differs from seeded region growing in that if the minimum measure of similarity is less than a predefined threshold, then it is added to the region. If not, then the pixel is considered different from all current regions and a new region is created with this pixel.

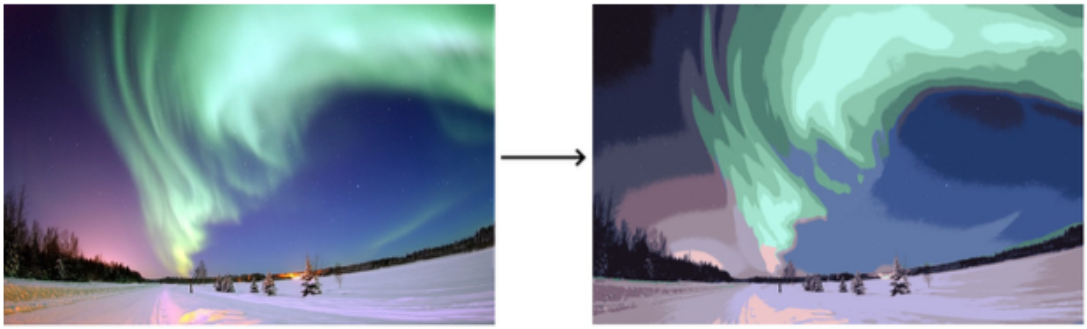
Name: Srikanth K Registration Number: 17ETCS002124

Clustering methods

Clustering methods involve using variants of the K-means algorithm and other clustering algorithms to cluster the pixels of an image into different groups. A typical clustering algorithm has the following steps:

1. Pick some number of clusters.
2. Assign each pixel in the image to one of the cluster in such a way that the distance between the pixel and the centre of the cluster is minimum.
3. Calculate the new cluster centres by taking the mean of all the pixels in the cluster.
4. Repeat the previous steps until no pixel changes cluster.

The formula to calculate the distance between a pixel and a cluster varies. Usually it is based-on pixel colour, intensity, texture, etc. or a combination of all of them. Although clustering algorithms are guaranteed to converge, it may not return an optimal solution.



Question 3

Q-3.1

Image segmentation using DeepLabV3

[1]:
```
# installing self developed package to use a helper function
! pip install git+https://github.com/firekind/athena &> /dev/null
```

[2]:
```
import os
import requests

import torch
from torchvision import transforms

import cv2
from PIL import Image
import matplotlib.pyplot as plt

from athena.visualizations.utils import plot_grid
```

Initialization

Creating a Pytorch dataset class:

[3]:
```
class Dataset(torch.utils.data.Dataset):

    urls = [
        "https://github.com/pytorch/hub/raw/master/images/dog.jpg",
        "https://horsej-intellectsolutio.netdna-ssl.com/cdn/farfuture/
↪PpyzjIrU-2dSANOhApE2tPK6e266eDKZZn5u4exkjZQ/mtime:1543626955/files/styles/
↪article_large/public/pictures-videos/articles/40530CM-41.jpg?itok=YWNwgVp_",
        "https://d2gg9evh47fn9z.cloudfront.net/800px_COLOURBOX2022932.jpg",
        "http://www.samacharnama.com/wp-content/uploads/2018/03/CARS-ON-ROAD..2.
↪jpg",
        "https://dkt6rvnu67rqj.cloudfront.net/cdn/ff/
↪CpBJCwYtAGyu75jfNXbfEG6_xAfL4mJOcylI8terr_s/1579830710/public/styles/600x400/
↪public/media/cow_run_5.jpg?itok=MiejaPwP",
        "https://images.assettype.com/
↪swarajya%2F2019-12%2F313732bc-ae64-46ef-a83f-ec67a5845c27%2F640px_UPSRTC_Bus_in_Bareilly_Cant
↪jpg?w=640&q=75&auto=format%2Ccompress",
        "https://loveincorporated.blob.core.windows.net/contentimages/gallery/
↪e02ee640-4804-47a0-bd32-5f3456eaac61-aa2ca5d9-883f-4d12-8fdb-2fa13bc6d1b5-Carpetright-House-H
↪jpg",
        "https://media.istockphoto.com/videos/
↪airplane-flying-in-the-sky-video-id686869508?s=640x640",
        "http://t1.gstatic.com/images?q=tbn:
↪ANd9GcSbrT4zlG2bfuQdMfTlAKI8gXBLyNNY_kKUbYhKeGEhqRSQwjW_OpCJmLq4UMbL95v-N6HLw__mkmM9usfIU6Q",
        "https://static.toiimg.com/thumb/
↪msid-58333548,width-1200,height-900,resizemode-4/.jpg",
```

```
        "https://dkt6rvnu67rqj.cloudfront.net/cdn/ff/
↪Gc3el01F1Z6wFRbfqhsSJHOBAW0BDJMW9I-JKmEOZ_0/1579547029/public/styles/600x400/
↪public/media/1012343_edited-1.jpg?itok=WGl2CMA0",
        "https://i.ytimg.com/vi/Dgc2FBZ6trE/maxresdefault.jpg"
    ]

    def __init__(self, download = False, transform = None, folder = "images"):
        self.samples = []
        self.transform = transform

        # downloading the images if required
        if download:
            self._download_images(Dataset.urls, folder)

        # reading the files in the given folder and storing their paths
        for f in os.listdir(folder):
            self.samples.append(os.path.join(folder, f))

    def __getitem__(self, index):
        # opening the image
        img = cv2.imread(self.samples[index])
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

        # applying preprocessing transforms
        if self.transform is not None:
            img = self.transform(img)

        # returning the image
        return img

    def __len__(self):
        return len(self.samples)

    def _download_images(self, urls, folder="images"):
        # making the directory to store the downloaded images
        os.mkdir(folder)

        # for each url
        for i, url in enumerate(urls):
            # download the image
            res = requests.get(url)

            # save the image into the directory
            with open(os.path.join(folder, str(i)+".jpg"), "wb") as f:
                f.write(res.content)
```

Perform preprocessing on the images of the dataset

The model used for segmenting the image is a deeplab v3 model with a resnet 101 as the encoder / feature extractor. The pretrained model is downloaded using pytorch hub. DeepLabV3 is a model that is an improvement on DeepLabV1 and DeepLabV2 which uses dilated (or atrous) convolutions. The following diagram shows the architecture of the model:

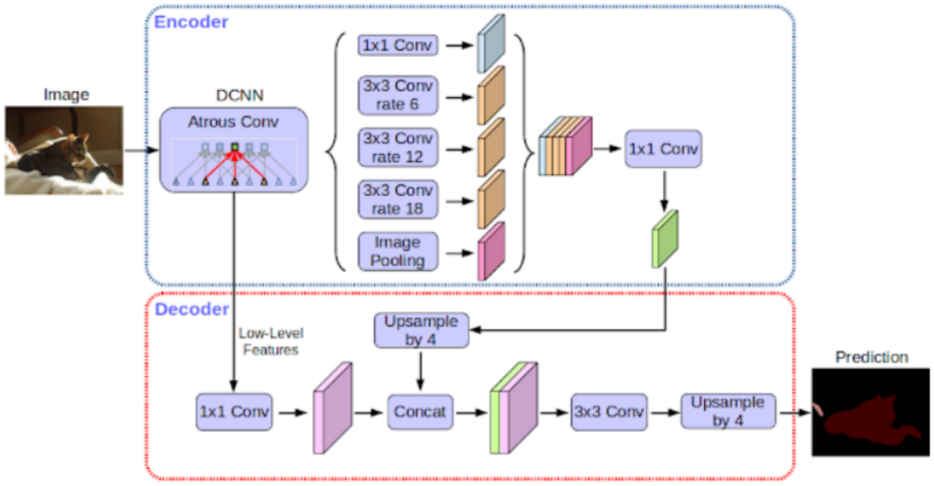


image taken from *https://ai.googleblog.com/2018/03/semantic-image-segmentation-*

As such, the evaluation phase preprocessing techniques only include normalizing the input image using ImageNet's mean and standard deviation. ImageNet's mean and standard deviation since it is calculated based on millions of images.

[4]:
```
# loading the pretrained model from torch hub
model = torch.hub.load(
    'pytorch/vision:v0.6.0',
    'deeplabv3_resnet101',
    pretrained=True
)

# setting the model in eval mode
model.eval()

# perprocessing the image
preprocess = transforms.Compose([
    transforms.ToTensor(), # converting the numpy image to a pytorch tensor
    transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    ) # normalizing the image with ImageNet's mean and std
```

```
])


# moving to GPU if available
device = "cpu"
if torch.cuda.is_available():
    device = "cuda"
    model.to(device)
```

```
Downloading: "https://github.com/pytorch/vision/archive/v0.6.0.zip" to
/root/.cache/torch/hub/v0.6.0.zip
Downloading: "https://download.pytorch.org/models/resnet101-5d3b4d8f.pth" to
/root/.cache/torch/hub/checkpoints/resnet101-5d3b4d8f.pth
```

```
HBox(children=(FloatProgress(value=0.0, max=178728960.0), HTML(value='')))
```

```
Downloading:
"https://download.pytorch.org/models/deeplabv3_resnet101_coco-586e9e4e.pth" to
/root/.cache/torch/hub/checkpoints/deeplabv3_resnet101_coco-586e9e4e.pth
```

```
HBox(children=(FloatProgress(value=0.0, max=244545539.0), HTML(value='')))
```

Loading the dataset:

[5]:
```
dataset = Dataset(download=True, transform=preprocess)
```

Viewing the original images:

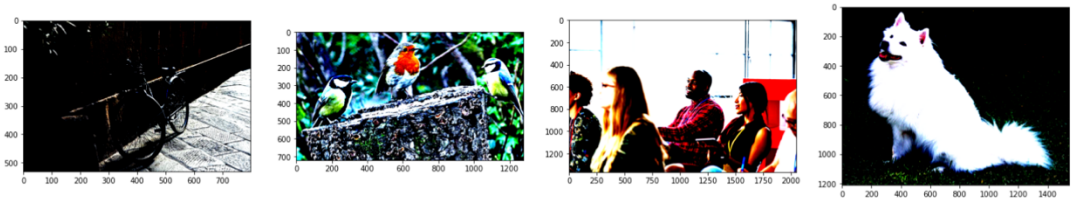
[6]:
```
imgs = []
for path in dataset.samples:
    img = cv2.imread(path)
    imgs.append(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plot_grid(len(imgs), lambda idx, ax: ax.imshow(imgs[idx]), figsize=(25, 30))
```

Viewing the preprocessed images:

[7]:
```
imgs = []
for img in dataset: imgs.append(img)
plot_grid(
    len(imgs),
    lambda idx, ax: ax.imshow(imgs[idx].permute(1, 2, 0).numpy()),
    figsize=(25, 30)
)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
```

Name: Srikanth K Registration Number: 17ETCS002124



```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
```

Some helper functions that will be used later:

[8]:
```
def create_res_images(outputs):
    """
    Creates the segmentation masks from the outputs of the model
    """
    res = []

    # create a color pallette, selecting a color for each of the 21 classes
    palette = torch.tensor([2 ** 25 - 1, 2 ** 15 - 1, 2 ** 21 - 1])
    colors = torch.as_tensor([i for i in range(21)])[:, None] * palette
    colors = (colors % 255).numpy().astype("uint8")
```

```
    for img in outputs:
        # adding the colors to the model outputs
        r = Image.fromarray(img.byte().cpu().numpy())
        r.putpalette(colors)
        res.append(r)

    return res
```

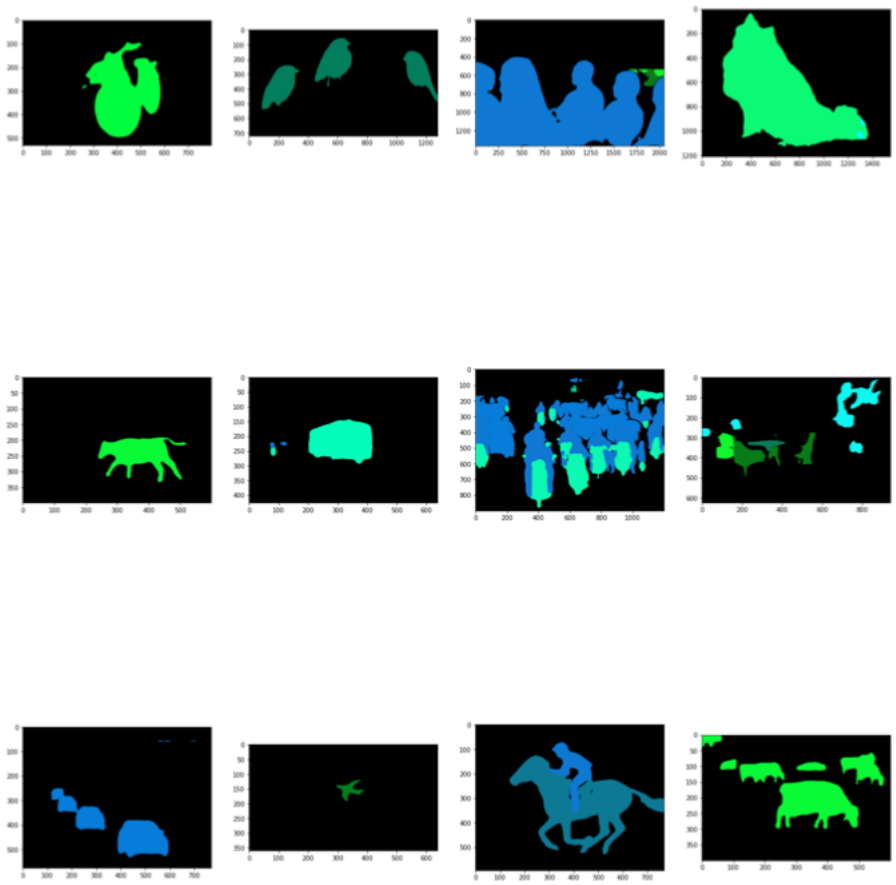
Q-3.2

Running inference

[9]:
```
outputs = []

for img in dataset:
    # casting the image to the device
    img = img.to(device)

    with torch.no_grad():
        # performing inference
        output = model(img.unsqueeze(0))['out'][0]

    # getting the class from predictions
    predictions = output.argmax(0)
    outputs.append(predictions)
```

Visualizing results

[10]:
```
masks = create_res_images(outputs)
plot_grid(len(masks), lambda idx, ax: ax.imshow(masks[idx]), figsize=(25, 30))
```



Q-3.3

The images were successfully segmented. One of the drawbacks of this method is that the model only detects the objects that it was trained on, which is in this case, the 22 classes of the Pascal VOC dataset. If an image contains an object that is not part of the 22 classes, the model will not detect it. This can be a drawback or a feature, depending on how you look at it. Moreover, the segmentation results are not pixel perfect. This is one of the limitations of this model.