RAMAIAH UNIVERSITY OF APPLIED SCIENCES

ASSIGNMENT

Course Code	**CSE308A**
Course Name	**Computer Vision**
Programme	**B.Tech**
Department	**Computer Science**
Faculty	**FET**

Name of the Student	**SHUBHAM AGARWAL**
Reg. No	**17ETCS002175**
Semester/Year	**7th /2017**
Course Leader/s	**Dr Aruna Kumar SV**

Declaration Sheet				
Student Name	SHUBHAM AGARWAL			
Reg. No	17ETCS002175			
Programme	B.Tech		Semester/Year	7th /2017
Course Code	CSE308A			
Course Title	Computer Vision			
Course Date	****	to	****	
Course Leader	Dr Aruna Kumar SV			

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		Date	19/01/2021
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	7 th/2017		
Course Code		Course Title	Computer Vision
Course Leader(s)	Dr. Divya BS, Dr. Subarna Chatterjee and Dr. Aruna Kumar SV		

Questions		Marking Scheme	Marks		
			Max Marks	First Examiner Marks	Moderator
1					
	1.1	Introduction to Segmentation and Creation of Dataset.	3		
	1.2	Identify and explain the appropriate pre-processing techniques	5		
	1.3	Identify and explain the appropriate Segmentation techniques	5		
		Question 1 Max Marks	**13**		
2	2.1	Perform preprocessing on the images of the created dataset	**5**		
	2.2	Perform segmentation to segment the image.	**5**		
	2.3	Results and Discussions.	**2**		
		Question 2 Max Marks	**12**		
Total Assignment Marks			**25**		

Course Marks Tabulation

Question	First Examiner	Remarks	Moderator	Remarks
1				
Marks (Max 10)				

Signature of First Examiner **Signature of Moderator**

Contents

List of Figures

1. Question 1

Solution to Part A Question No 1:

1.1 Introduction to Segmentation and Creation of Dataset.

Image Processing or more specifically, Digital Image Processing is a process by which a digital image is processed using a set of algorithms. It involves a simple level task like noise removal to common tasks like identifying objects, person, text etc., to more complicated tasks like image classifications, emotion detection, anomaly detection, segmentation etc.

Image Segmentation is the process by which a digital image is partitioned into various subgroups (of pixels) called Image Objects, which can reduce the complexity of the image, and thus analysing the image becomes simpler.

It is the process of dividing an image into different regions based on the characteristics of pixels to identify objects or boundaries to simplify an image and more efficiently analyze it.

The goal of image segmentation is to cluster pixels into salient image regions, i.e., regions corresponding to individual surfaces, objects, or natural parts of objects.

The goal of segmentation is to simplify and/or change the representation of an image into something that is more meaningful and easier to analyze. Image segmentation is typically used to locate objects and boundaries (lines, curves, etc.) in images. More precisely, image segmentation is the process of assigning a label to every pixel in an image such that pixels with the same label share certain characteristics.

Dataset creation:

Dataset is the collection of specific data. These dataset are further classified as training and Testing datasets. Image dataset is a collection of same category of images.

Dataset creation can be done either in 2 ways:

1. By create a folder and adding all the images to this folder, then using imread function one can read images stored in this folder. Make sure all the images have same extension

2. By using python scripts: Setting the path and convert to RGB and append the image file to a Python list then converting this list to np array and using this array an image dataset. Make sure all the images have same extension.

Here a car dataset is created this data set contains 16 images from the ai.stanford.edu cars dataset. A folder with 16 .jpg images is created and stored in the google drive, later this folder is accessed and all the images present in the folder are read and converted to numpy array.

```
Imagepaths = glob.glob("/content/drive/MyDrive/Segmentation_images/Car/*.jpg",recursive='true')
```

```
#opening image and converting it array:
OriginalImage = np.array([np.asarray(Image.open(img)) for img in Imagepaths])
```

Figure 1: fetching the images from the directory and converting them to np array.

In the above code: car folder stored in the google drive is fetched. Images present in this folder are converted to array.

1.2 Identify and explain the appropriate pre-processing techniques.

In machine learning projects in general, one usually go through a data preprocessing or cleaning step. one spend a good amount of time in cleaning up and preparing the data before building appropriate learning model. The goal of this step is to make data ready for the ML model to make it easier to analyze and process computationally, as it is with images. Based on the problem solving and the dataset in hand, there's some data massaging required before one feed images to the ML model.

Image processing could be simple tasks like image resizing. In order to feed a dataset of images to a convolutional network, they must all be the same size. Other processing tasks can take place like geometric and color transformation or converting color to grayscale and many more.

The acquired data are usually messy and come from different sources, hence they need to be standardized and cleaned up. preprocessing is used to conduct steps that reduce the complexity and increase the accuracy of the applied algorithm.

The primary challenges faced in the processing of color images are the variety and enormity of the color intensity gamut along with the processing of the spectral characteristics of the different color components therein. To be precise, the task of color image processing involves a vast amount of processing overhead since color intensity information is generally manifested in the form of admixtures of different color components.

Various Image preprocessing are:

- **Converting to Gray Scale:**

Convert color images to grayscale to reduce computation complexity: in certain problems it useful to lose unnecessary information from different images to reduce space or computational complexity.

A grayscale (or graylevel) image is simply one in which the only colors are shades of gray. The reason for differentiating such images from any other sort of color image is that less information needs to be provided for each pixel. Often, the grayscale intensity is stored as an 8-bit integer giving 256 possible different shades of gray from black to white. If the levels are evenly spaced then the difference between successive graylevels is significantly better than the graylevel resolving power of the human eye.

- **Standardize images:**

Images must be preprocessed and scaled to have identical widths and heights before fed to the learning algorithm. When an image is resized, its pixel information is changed. an image is reduced in size, any unneeded pixel information will be discarded It helps in reducing the number of pixels from an image and that has several advantages. OpenCV provides several interpolation methods for resizing an image.

- **Data augmentation:**

Another common pre-processing technique involves augmenting the existing dataset with perturbed versions of the existing images. Scaling, rotations and other affine transformations are typical

- **Image Transformation:**

Transformation is a function. A function that maps one set to another set after performing some operations.

$$G(x, y) = T\{ f(x, y) \}$$

F(x,y) = input image on which transformation function has to be applied. G(x,y) = the output image or processed image. T is the transformation function. Transformation can be in spatial domain or frequency domain.

Image enhancement is the simplest and most attractive area of DIP. In this stage details which are not known, or we can say that interesting features of an image is highlighted. Such as brightness, contrast, etc... Enhancing an image provides better contrast and a more detailed image as compare to non-enhanced image.

There are three basic gray level transformation.

- Linear : Identity transition is shown by a straight line. In this transition, each value of the input image is directly mapped to each other value of output image.
- Logarithmic: During log transformation, the dark pixels in an image are expanded as compare to the higher pixel values. The higher pixel values are kind of compressed in log transformation.
- Power – law: This type of transformation is used for enhancing images for different type of display devices.

- **Noise in images:**

Image noise is random variation of brightness or color information in the images captured. It is degradation in image signal caused by external sources. Images containing multiplicative noise have the characteristic that the brighter the area the noisier it. But mostly it is additive.

- Gaussian Noise is a statistical noise having a probability density function equal to normal distribution
- Salt Noise: Salt noise is added to an image by addition of random bright all over the image.
- Pepper Noise: Salt noise is added to an image by addition of random dark all over the image.

- **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.

Many preprocessing techniques can be used to get images ready to train the machine learning model. In some projects, one might need to remove the background color from images to reduce the noise. Other projects might require that brighten or darken images. In short, any adjustments that one need to apply to the dataset are considered a sort of preprocessing.

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications. Hence open cv library is used to perform the preprocessing on the various images.

1.3 Identify and explain the appropriate Segmentation techniques

Segmentation partitions an image into distinct regions containing each pixels with similar attributes. To be meaningful and useful for image analysis and interpretation, the regions should strongly relate to depicted objects or features of interest. Meaningful segmentation is the first step from low-level image processing transforming a greyscale or colour image into one or more other images to high-level image description in terms of features, objects, and scenes. The success of image analysis depends on reliability of segmentation, but an accurate partitioning of an image is generally a very challenging problem.

One can various image segmentation algorithms to split and group a certain set of pixels together from the image. By doing so, one actually assigning labels to pixels and the pixels with the same label fall under a category where they have some or the other thing common in them.

Image Segmentation is very widely implemented in Python, along with other classical languages like Matlab, C/C++ etc. More likey so, Image segmentation in python has been the most sought after skill in the data science stack.

Segmentation techniques are either contextual or non-contextual. There are various image segmentation methods available, some of those technique are :

- **Thresholding :**

  Thresholding is the simplest non-contextual segmentation technique. With a single threshold, it transforms a greyscale or colour image into a binary image considered as a binary region map. The

binary map contains two possibly disjoint regions, one of them containing pixels with input data values smaller than a threshold and another relating to the input values that are at or above the threshold. The former and latter regions are usually labelled with zero (0) and non-zero (1) labels, respectively. The segmentation depends on image property being thresholded and on how the threshold is chosen.

Generally, the non-contextual thresholding may involve two or more thresholds as well as produce more than two types of regions such that ranges of input image signals related to each region type are separated with thresholds.

Different types are:

1. cv2.THRESH_BINARY
2. cv2.THRESH_BINARY_INV
3. cv2.THRESH_TRUNC
4. cv2.THRESH_TOZERO
5. cv2.THRESH_TOZERO_INV

- **Edge based segmentation:**

  With this technique, detected edges in an image are assumed to represent object boundaries, and are used to identify these objects. Sobel and canny edge detection algorithms are some of the examples of edge based segmentation techniques.

- **Masking and Contour:**

  A mask is a filter. Concept of masking is also known as spatial filtering. Masking is also known as filtering. In this concept we just deal with the filtering operation that is performed directly on the image.

  A contour is a closed curve joining all the continuous points having some color or intensity, they represent the shapes of objects found in an image

- **Morphological methods based segmentation:**

  It is the methodology for analysing the geometric structure inherent within an image. In this technique the output image pixel values are based on similar pixels of input image with is neighbours and produces a new binary image. This method is also used in foreground background separation.

  The base of the morphological operation is dilation, erosion, opening, closing expressed in logical AND, OR. This technique is mainly used in shape analysis and noise removal after thresholding an image.

- **Graph based segmentation techniques:**

  Graph-based approaches treat each pixel as a node in a graph. Edge weights between two nodes are proportional to the similarity between neighbouring pixels. Pixels are grouped together to form segments or a.k.a superpixels by minimising a cost function defined over the graph.

- **Clustering based segmentation techniques:**

  Starting from a rough initial clustering of pixels, gradient ascent methods iteratively refine the clusters until some convergence criterion is met to form image segments or superpixels. These type of algorithms aim to minimise the distance between the cluster centre and each pixel in the image.

2. Question-2

Solution to Question No 2:

2.1 Perform pre-processing on the images of the created dataset.

The aim of pre-processing is an improvement of the image data that suppresses unwilling distortions or enhances some image features important for further processing, and analysis task.

Below is the list of methods applied on the user created car datasets:

- Resizing image
- Power log transformation (gamma correction)
- Grayscale conversion
- Histogram Equalization

The above preprocessing steps are applied to the dataset by reading the images using imread function and converting the images to array form, but before this one needs to import libraries:

Import libraries

```
[1]  import cv2
     import glob
     import matplotlib.pyplot as plt
     import numpy as np
     from PIL import Image
     import random
```

Now one needs to extract the images stored in a particular folder:

Extract Images from Google Drive

In this step, store the image path from car dataset dataset into a variable.

```
[2]  Imagepaths = glob.glob("/content/drive/MyDrive/Segmentation_images/Car/*.jpg",recursive='true')
```

Now images must be converted to array form, below is the code:

Open the Images and store it in a List.

use the above varialbe to get the images and load the images to a array.

```
[3]  #opening image and converting it array:
     OriginalImage = np.array([np.asarray(Image.open(img)) for img in Imagepaths])
```

Here the first preprocessing step applied:

Resizing is done to have identical widths and heights. When an image is resized, its pixel information is changed.

Resize Image:

Some images may be captured by a camera and may vary in size, therefore, Resizing needs to be done , and establish a base size for all images which can be used for image segmentation

```
[4]  #Resizing the images:
     for i in range(len(OriginalImage)):
       OriginalImage[i] = cv2.resize(OriginalImage[i], (512,512),interpolation=cv2.INTER_NEAREST)
```

To display the images, matplot function is used:

▼ Display Original Image:

```
[5]  # Displaying the Orginal Image:
     plt.figure(figsize=(15,12))
     for i, img in enumerate(OriginalImage[0:16]):
         plt.subplot(4,4,i+1)
         plt.axis('off')
         plt.grid(False)
         plt.imshow(img)
         plt.suptitle("Original Image", fontsize=20)
     plt.show()
```

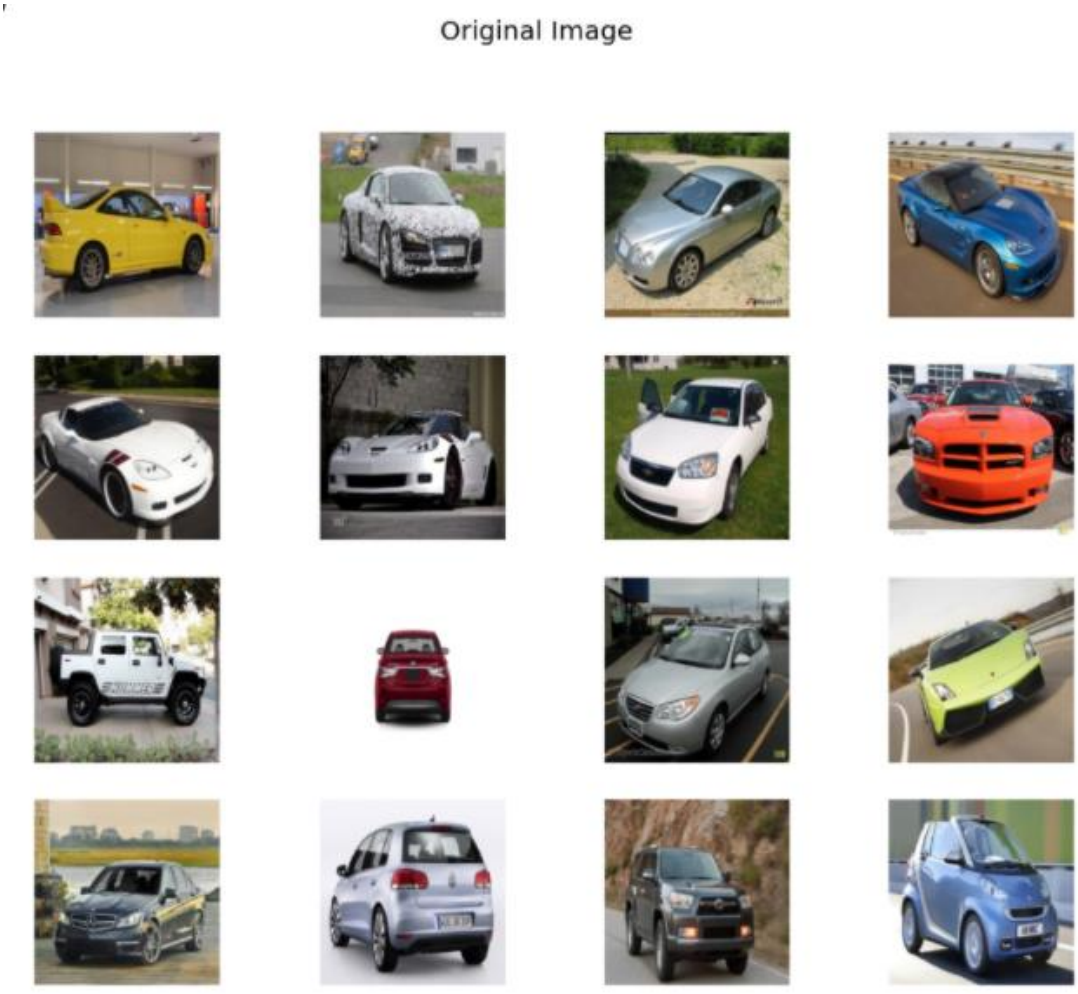


Figure 2: Original Images

Here second preprocessing technique is applied : Gamma correction:

Pre-Processsing:

Power Law Transformation (gamma correction):

```
[6]  logTransformedImage= np.array([np.array(255*(img / 255) ** 2.55, dtype = 'uint8')  for img in OriginalImage])

     # Display Gray-Scale Images:
     plt.figure(figsize=(15,12))
     for i, img in enumerate(logTransformedImage[0:16]):
         plt.subplot(4,4,i+1)
         plt.axis('off')
         plt.grid(False)
         plt.imshow(img)
         plt.suptitle("Gamma correction with gamma = 2.55", fontsize=20)
     plt.show()
```

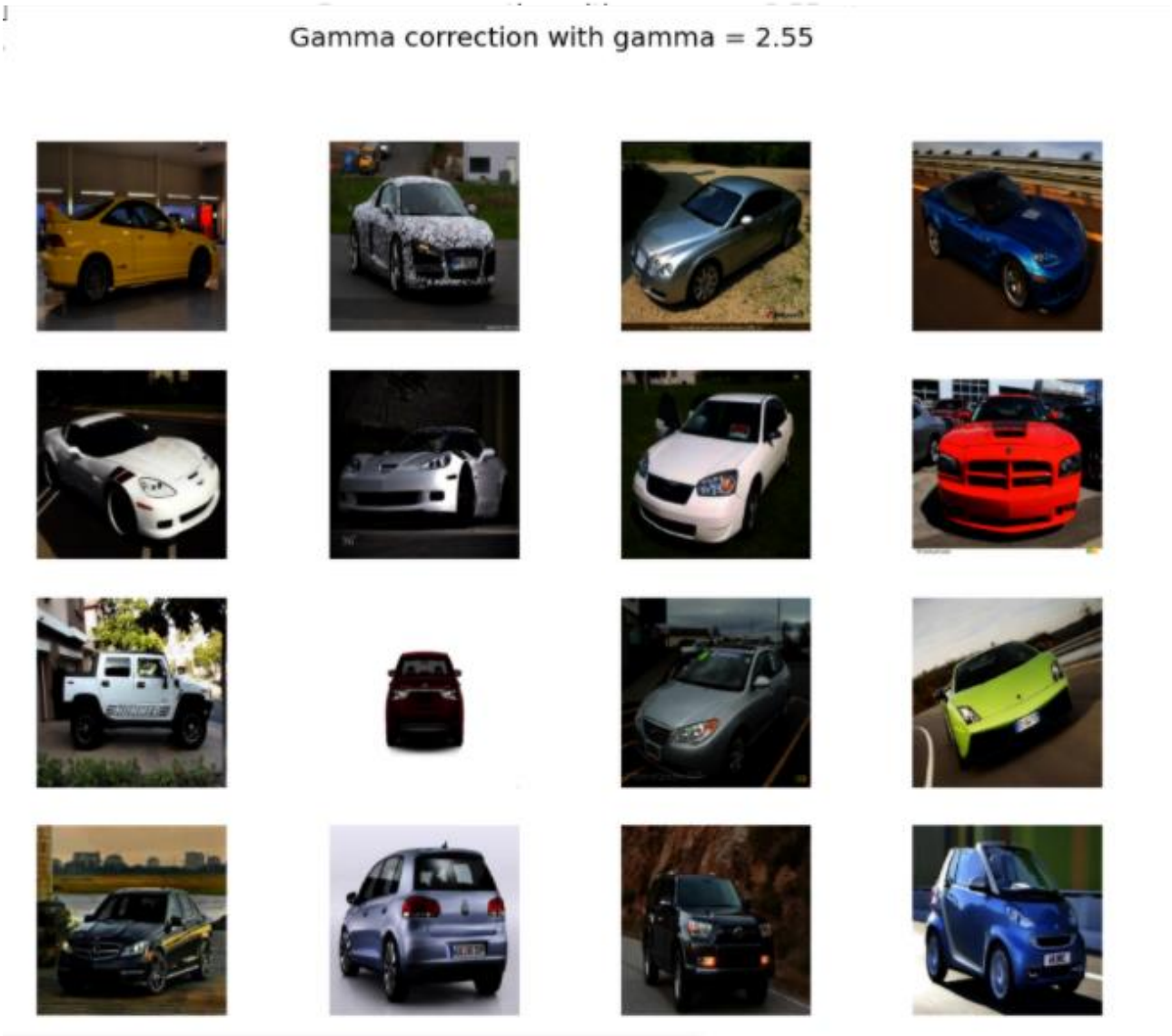


Figure 3: Gamma corrected Images

Gamma correction is also known as the Power Law Transform. During log transformation, the dark pixels in an image are expanded as compare to the higher pixel values.

After the Gamma correction, images are converted to Gray scale:

Grayscaling is the process of converting an image from other color spaces e.g RGB, CMYK, HSV, etc. to shades of gray. It varies between complete black and complete white. The conversion from a RGB image to gray is done with COLOR_RGB2GRAY.

Convert transformed Images to GrayScale Images

use the inbuilt function to convert them to GrayScale and then plot the GrayScale Images

```
[7]  #converting the original images to GrayScale:
     grayImage = np.array([cv2.cvtColor(img,cv2.COLOR_RGB2GRAY) for img in logTransformedImage])

     # Display Gray-Scale Images:
     plt.figure(figsize=(15,12))
     for i, img in enumerate(grayImage[0:16]):
         plt.subplot(4,4,i+1)
         plt.axis('off')
         plt.grid(False)
         plt.imshow(img,'gray')
         plt.suptitle("Grayscale", fontsize=20)
     plt.show()
```

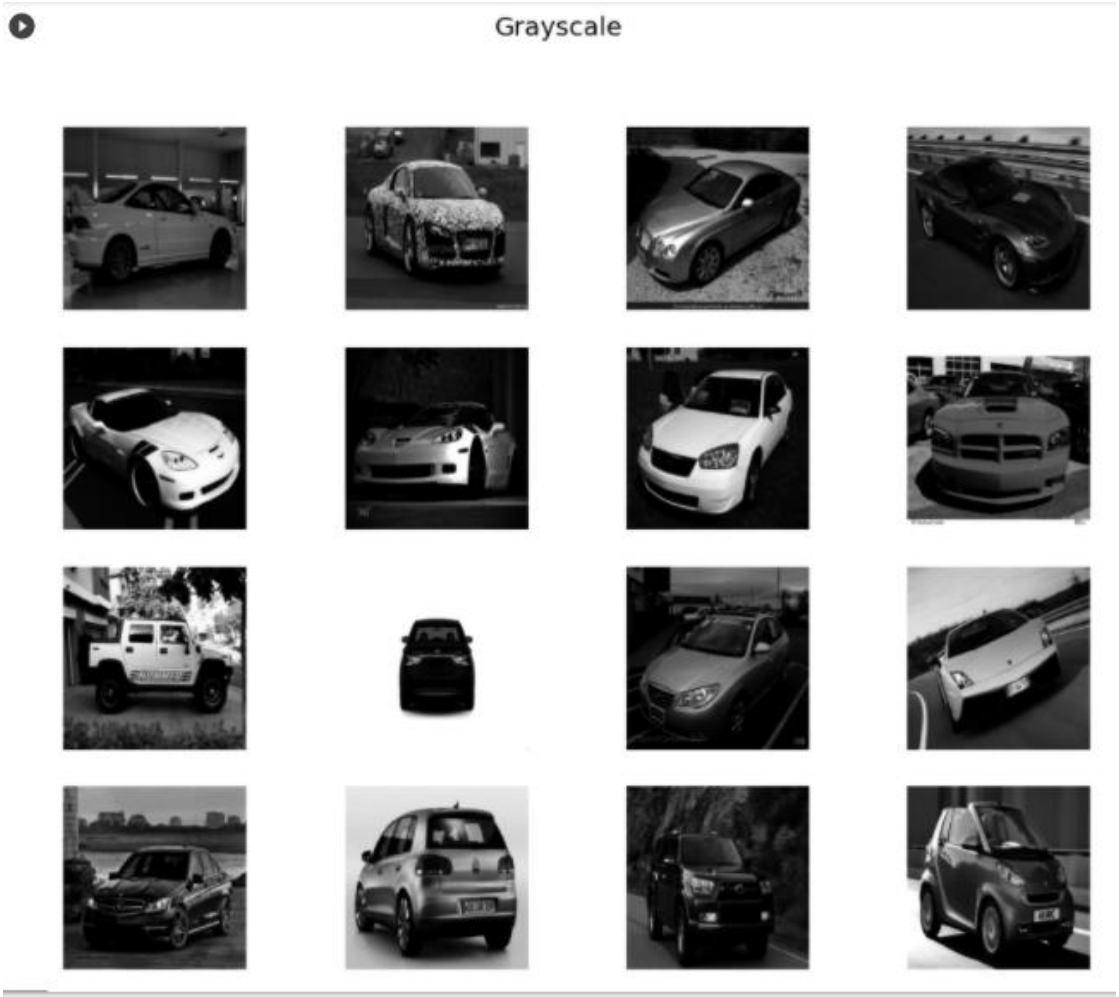


Figure 4: Grayscale Images

After coveting to Grayscale images , Histogram Equalization is applied to the grayscale images:

Histogram equalization is a method in image processing of contrast adjustment using the image's histogram. This method usually increases the global contrast of many images, especially when the usable data of the image is represented by close contrast values.

Histogram equalization is good when histogram of the image is confined to a particular region. It won't work good in places where there is large intensity variations where histogram covers a large region, ie both bright and dark pixels are present.

OpenCV has a function to do this, cv2.equalizeHist(). Its input is just grayscale image and output is our histogram equalized image.

Histogram Equalization

```
equilized = np.array([cv2.equalizeHist(img) for img in grayImage])
plt.figure(figsize=(15,12))
for i, img in enumerate(equilized[0:16]):
    plt.subplot(4,4,i+1)
    plt.axis('off')
    plt.grid(False)
    plt.imshow(img,'gray')
    plt.suptitle("Histogram Equalisation", fontsize=20)
plt.show()
```

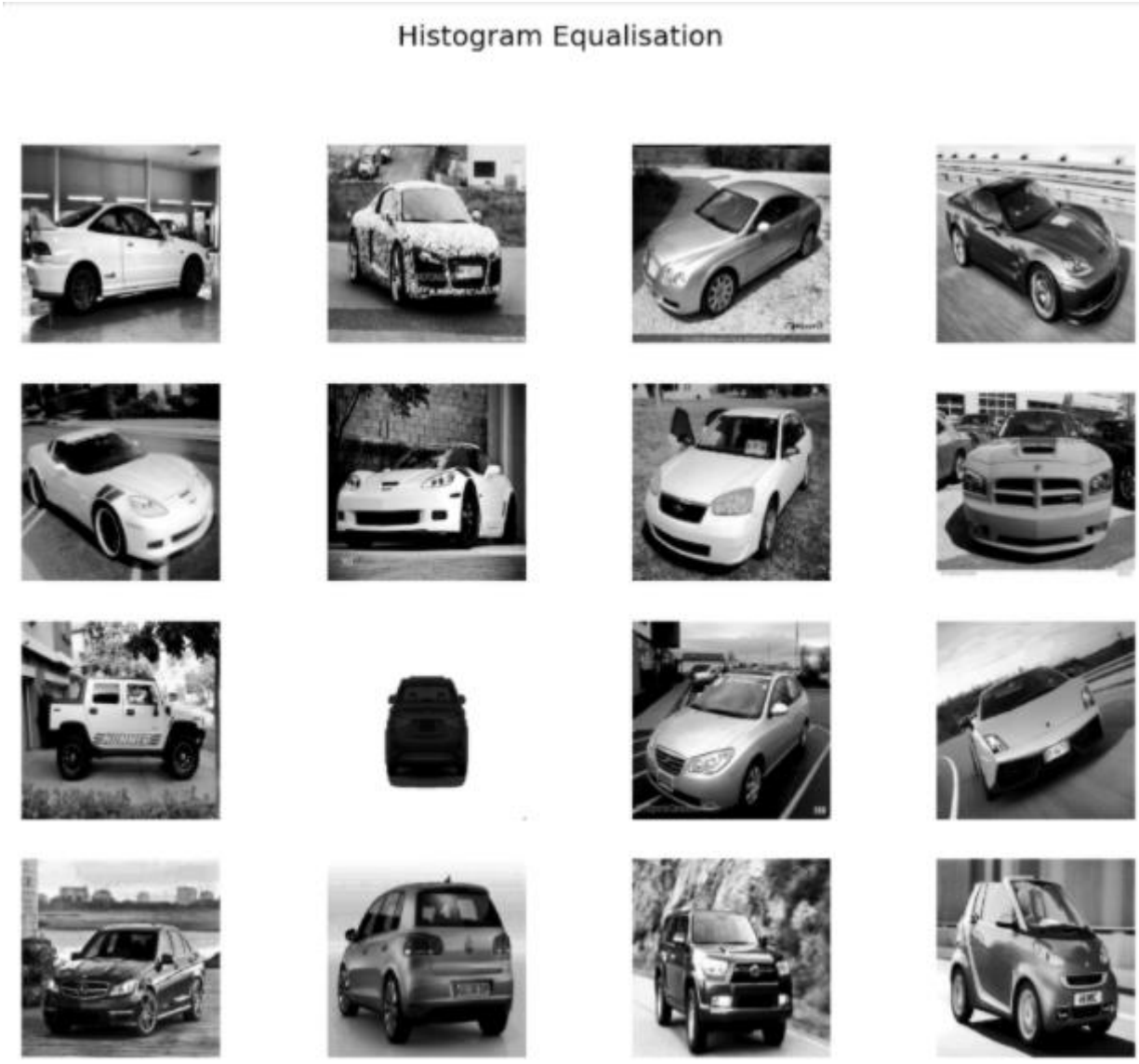


Figure 5: Histogram Equalized Images

This completes the required Preprocessing steps. One can even add or remove noise from the images under preprocessing stem , opencv has different function to add and remove noise, function like Medianblur and Gaussianblur adds noise to the images, where as fastNlMeansDenoising helps in removing noise from gray scale images. One can also add illumination or adjust saturation and brightness of a particular image to improve image information quality hence these can also be done under Image preprocessing.

Now lets jump to the Image segmentation

2.2 Perform segmentation to segment the image.

As discussed earlier, there are various segmentation methods available, one needs to select the appropriate segmentation method and apply it on the image, here Binary Thresholding is considered, the process is described as:

Thresholding is a technique in OpenCV, which is the assignment of pixel values in relation to the threshold value provided. In thresholding, each pixel value is compared with the threshold value. If the pixel value is smaller than the threshold, it is set to 0, otherwise, it is set to a maximum value (generally 255). In Computer Vision, this technique of thresholding is done on grayscale images. So initially, the image has to be converted in grayscale color space.

The basic Thresholding technique is Binary Thresholding. For every pixel, the same threshold value is applied. If the pixel value is smaller than the threshold, it is set to 0, otherwise, it is set to a maximum value.

Opencv provides different threshold function The function used is cv2.threshold. OpenCV provides different styles of thresholding and it is decided by the fourth parameter of the function.

Different types are:

- cv2.THRESH_BINARY: If pixel intensity is greater than the set threshold, value set to 255, else set to 0.
- cv2.THRESH_BINARY_INV: Inverted or Opposite case of cv2.THRESH_BINARY.
- cv.THRESH_TRUNC: If pixel intensity value is greater than threshold, it is truncated to the threshold. The pixel values are set to be the same as the threshold. All other values remain the same.
- cv.THRESH_TOZERO: Pixel intensity is set to 0, for all the pixels intensity, less than the threshold value.
- cv.THRESH_TOZERO_INV: Inverted or Opposite case of cv2.THRESH_TOZERO.

  Performing Segmentation on Equalized images: Binary Thresholding is used:

Segmentation:

Binary Thresholding

using opencv tool to apply thresholding on the set of images

```
[9]  #Thresholding the Images:
     thresh = [cv2.threshold(img, 130, 255, cv2.THRESH_BINARY_INV)[1] for img in equilized]

     #displaying the Thresholding Images:
     plt.figure(figsize=(15,12))
     for i, img in enumerate(thresh[0:16]):
         plt.subplot(4,4,i+1)
         plt.axis('off')
         plt.imshow(img,'gray')
         plt.suptitle("Threshold", fontsize=20)
     plt.show()
```

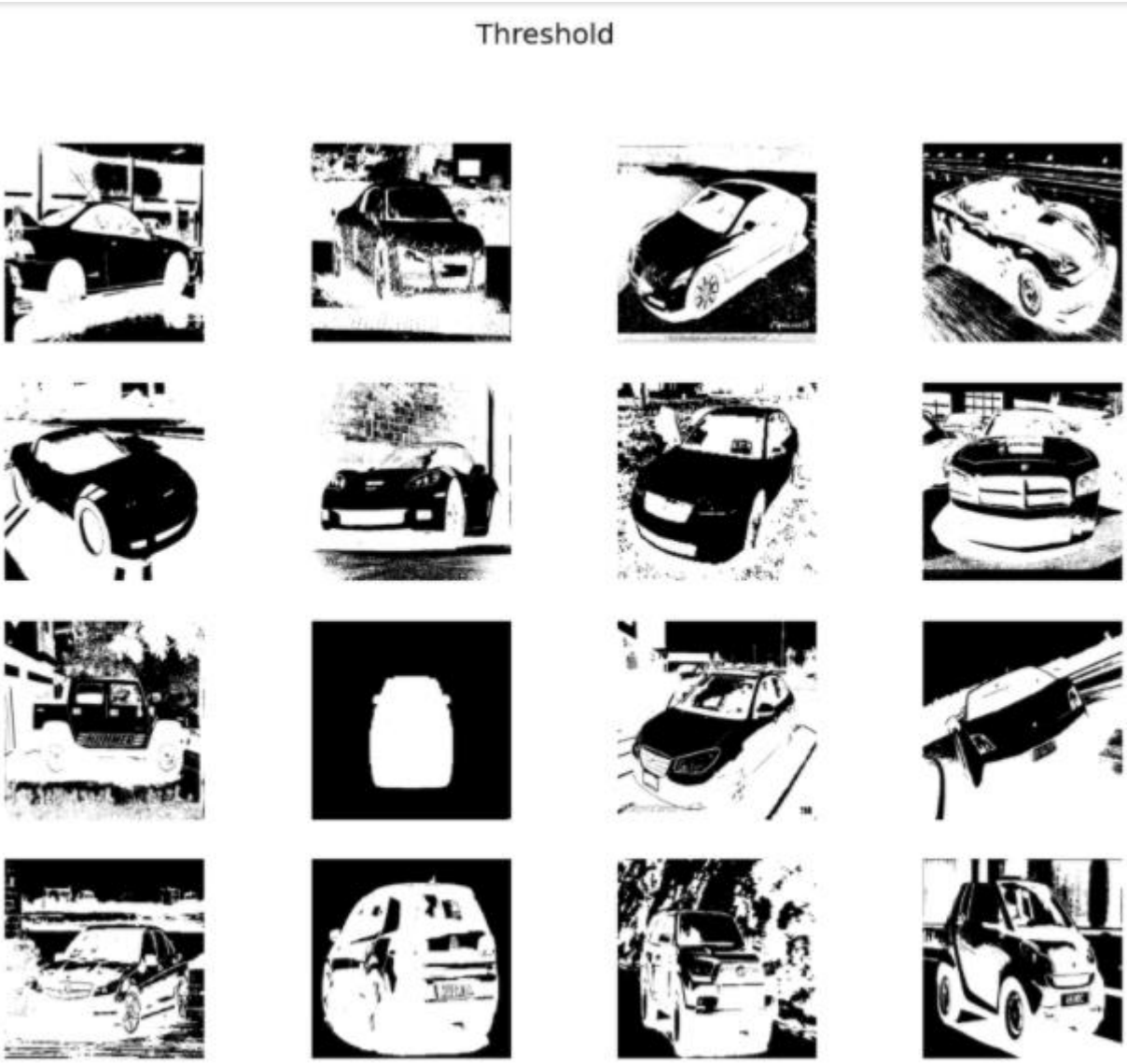


Figure 6: Thresholding Images

The threshold images are sent for edge detection, Canny edge detection method is used:

The Canny edge detector is an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images It is a multi-stage algorithm

The Canny edge detection algorithm is composed of 5 steps:

1. Noise reduction;
2. Gradient calculation;
3. Non-maximum suppression;
4. Double threshold;
5. Edge Tracking by Hysteresis.

OpenCV puts all the above in single function, cv2.Canny().

Canny Edge Detection:

```
[10] #canny Edge Detection:
     edges = [cv2.dilate(cv2.Canny(img, 0, 255), None) for img in thresh]

     #dsiplaying the Canny Edge:
     plt.figure(figsize=(15,12))
     for i, img in enumerate(edges[0:16]):
         plt.subplot(4,4,i+1)
         plt.axis('off')
         plt.grid(False)
         plt.imshow(cv2.cvtColor(img, cv2.COLOR_GRAY2RGB))
         plt.suptitle("Edges", fontsize=20)
     plt.show()
```

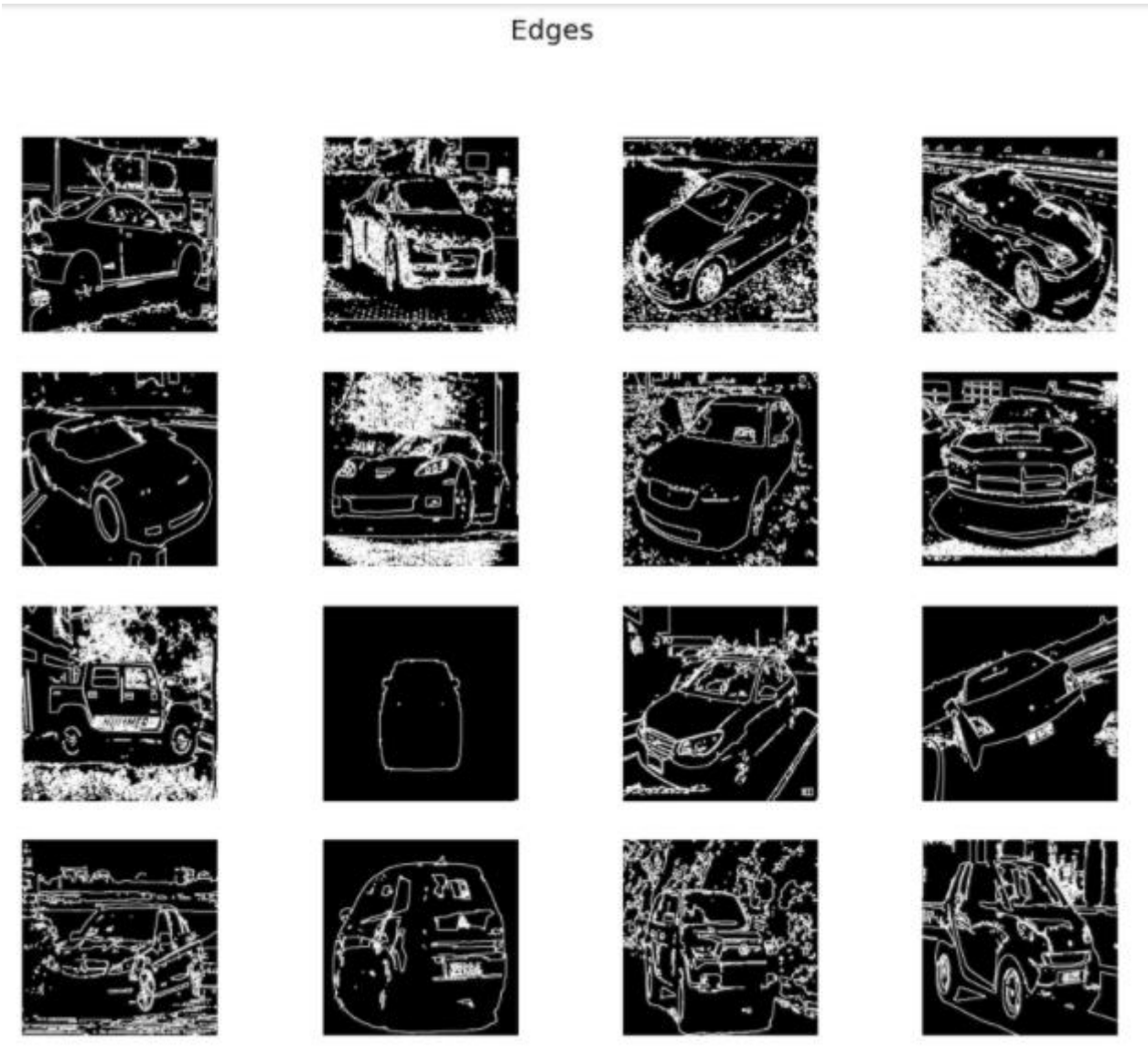


Figure 7: Edge Detected Images Using Canny edge method

A contour is a closed curve joining all the continuous points having some color or intensity, they represent the shapes of objects found in an image. Contour detection is a useful technique for shape analysis and object detection and recognition.

- For better accuracy, use binary images. So before finding contours, apply threshold or canny edge detection.
- In OpenCV, finding contours is like finding white object from black background. So remember, object to be found should be white and background should be black.

Applying Masks

applying the mask to the edge detected images, and then segmenting them

```
[11] #mask:
     masked=[]
     segmented=[]

     for i, img in (enumerate(edges)):
       if(i==16):
         continue
       cnt = sorted(cv2.findContours(img, cv2.RETR_CCOMP, cv2.CHAIN_APPROX_SIMPLE)[-2], key=cv2.contourArea)[-1]
       mask = np.zeros((512,512), dtype='uint8')
       masked.append(cv2.drawContours(mask, [cnt],-1, 255, -1))
       dst = cv2.bitwise_and(OriginalImage[i], OriginalImage[i], mask=mask)
       segmented.append(cv2.cvtColor(dst, cv2.COLOR_BGR2RGB))
```

Displaying Image Masks:

```
[12] #displaying Masked Images:
     plt.figure(figsize=(15,12))
     for i, masking in enumerate(masked[0:16]):
       plt.subplot(4,4,i+1)
       plt.axis('off')
       plt.grid(False)
       plt.imshow(masking, cmap='gray')
     plt.suptitle("Mask", fontsize=20)
     plt.show()
```

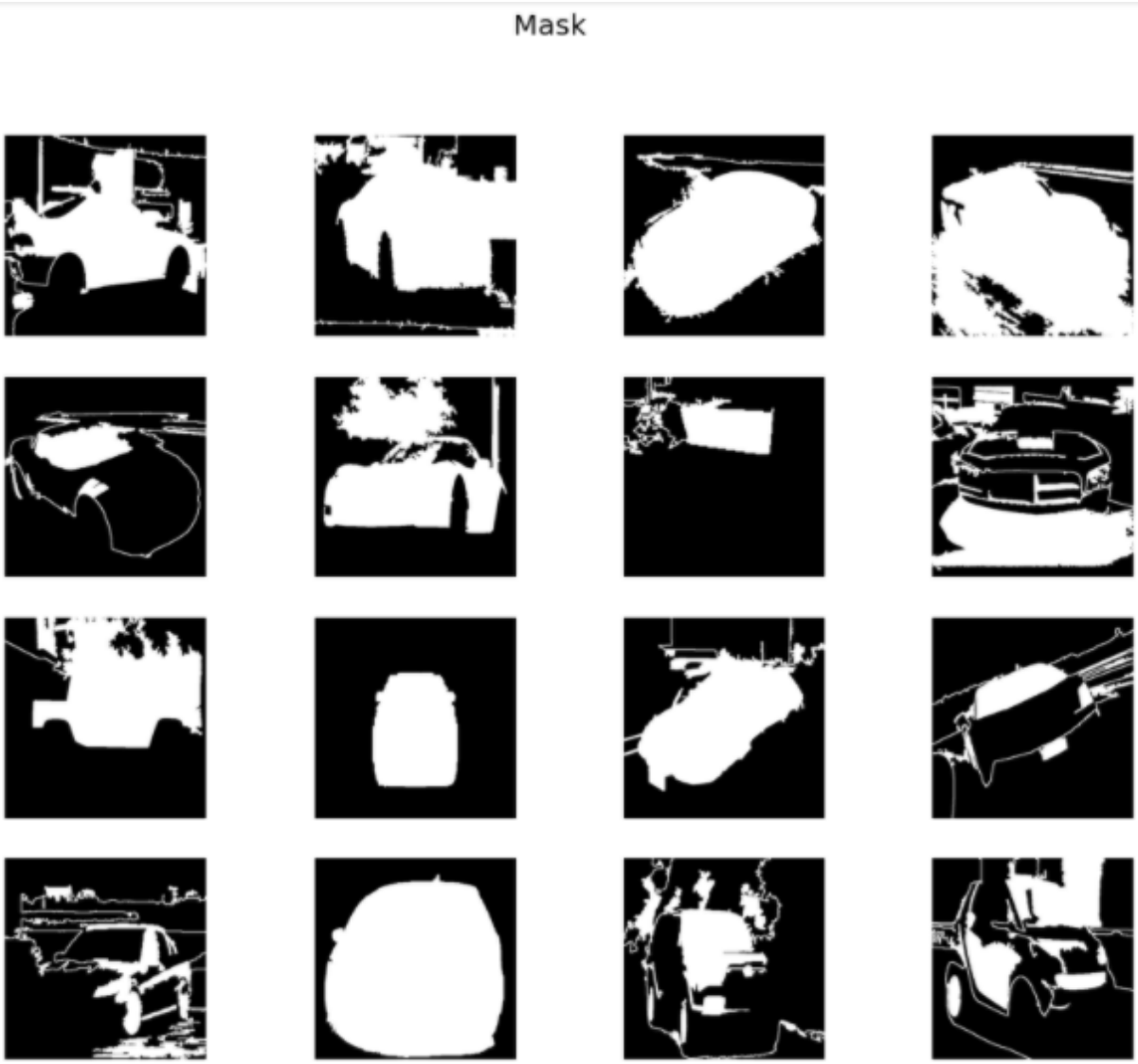


Figure 8: Image Mask

The final segmented Images are displayed below:

Displaying the Segemented Images:

using the plot function displaying the coroulred segmented objects

```
[13] plt.figure(figsize=(15,12))
     for i, img in enumerate(segmented[0:16]):
       plt.subplot(4,4,i+1)
       plt.axis('off')
       plt.grid(False)
       plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))

     plt.suptitle("Segmented Image", fontsize=20)
     plt.show()
```

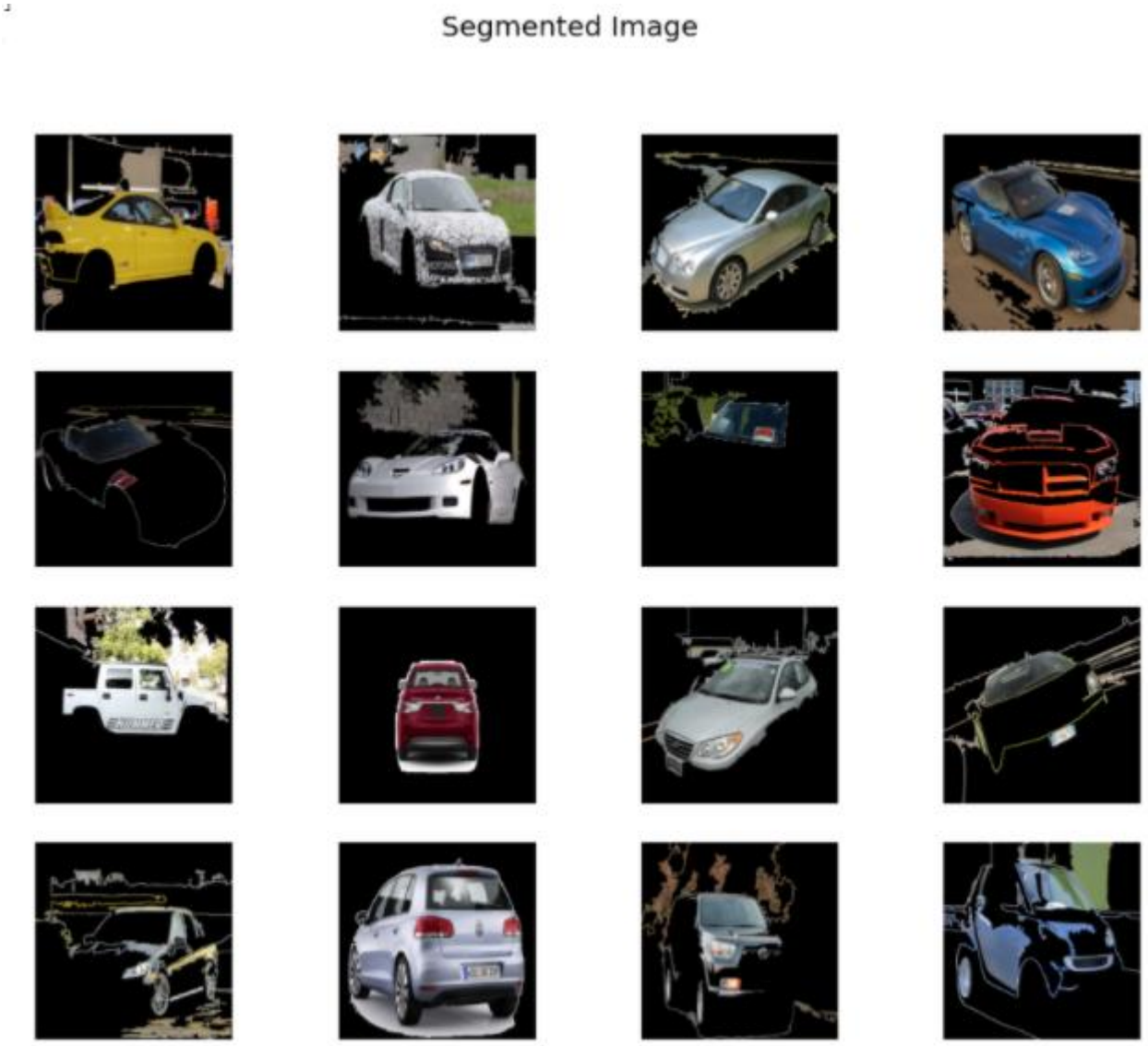


Figure 9: Final Segmented Images

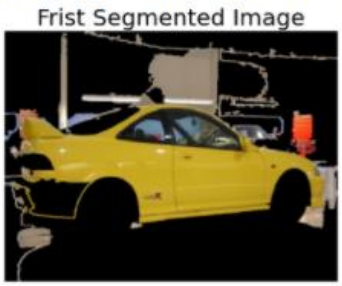
The above images are the final segmented images, the background of the original images are converted to black color, partial objects are detected, but one can see that objects are not 100% properly detected. Therefore one need to select proper pre-processing and segmentation Methods to get appropriate results.

2.3 Results and Discussions.

The segmentation results are shown below,

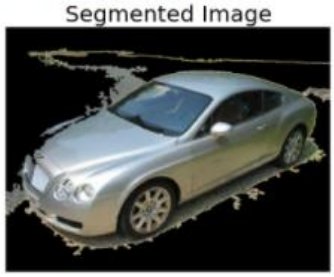
```
segmented[0] = cv2.resize(segmented[0], (2048, 1536),
                          interpolation=cv2.INTER_NEAREST)
plt.imshow(cv2.cvtColor(segmented[0], cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.grid(False)
plt.title("Frist Segmented Image", fontsize=20)
```

```
Text(0.5, 1.0, 'Frist Segmented Image')
```




```
segmented[2] = cv2.resize(segmented[2], (2048, 1536),
                          interpolation=cv2.INTER_NEAREST)
plt.imshow(cv2.cvtColor(segmented[2], cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.grid(False)
plt.title("Segmented Image", fontsize=20)
```

```
Text(0.5, 1.0, 'Segmented Image')
```




```
segmented[14] = cv2.resize(segmented[14], (2048, 1536),
                           interpolation=cv2.INTER_NEAREST)
plt.imshow(cv2.cvtColor(segmented[14], cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.grid(False)
plt.title("Segmented Image", fontsize=20)
```

```
Text(0.5, 1.0, 'Segmented Image')
```

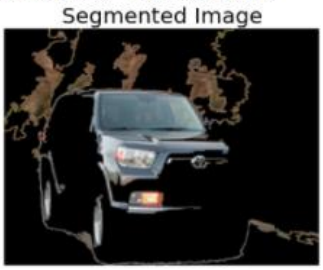


Figure 10: Displaying individual Segmented images

Some of the images were segmented perfectly but in some of the images only the edges of the required object were considered as segmented part. This might be due to the multiple objects in the image or due to using different contour attributes.

The segmentation process can be improved by using different preprocessing methods like adding and removing noise from the images, Changing the saturating and contrast of images, one can also transform images to frequency domain and perform preprocessing using Fourier transformation.

The various different segmentation methods like Edge base Segmentation, Region based segmentation Clustering and watershed-based segmentation can be applied on the images to get higher accuracy.

The segmentation process can be improved by using deep learning segmentation methods like usage of Convolutional Neural Networks, Fully Convolutional Networks or Ensemble learning. One of the popular image segmentation models is Mask R-CNN (Regional Convolutional Neural Network). It is an Instance segmentation model. It has become a standard while doing image segmentation and it is used widely while performing image segmentation.

3. Bibliography

- https://www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/ImageProcessing-html/topic3.html
- http://www.cs.toronto.edu/~jepson/csc2503/segmentation.pdf
- https://www.analytixlabs.co.in/blog/what-is-image-segmentation/
- https://www.analyticsvidhya.com/blog/2019/04/introduction-image-segmentation-techniques-python/
- https://medium.com/analytics-vidhya/create-your-own-real-image-dataset-with-python-deep-learning-b2576b63da1e
- https://towardsdatascience.com/deep-learning-prepare-image-for-dataset-d20d0c4e30de
- https://ai.stanford.edu/~jkrause/cars/car_dataset.html#:~:text=Overview,or%202012%20BMW%20M3%20coupe.
- https://freecontent.manning.com/the-computer-vision-pipeline-part-3-image-preprocessing/
- https://www.tutorialspoint.com/dip/image_transformations.html
- https://medium.com/image-vision/noise-in-digital-image-processing-55357c9fab71
- https://www.tutorialspoint.com/dip/histogram_equalization.htm
- https://opencv.org/about/
- https://www.easytechjunkie.com/what-is-image-segmentation.htm
- https://medium.com/analytics-vidhya/image-segmentation-techniques-using-digital-image-processing-machine-learning-and-deep-learning-342773fcfef5
- https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_contours/py_contours_begin/py_contours_begin.html
- https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_thresholding/py_thresholding.html

The notebook for image preprocessing and segmentation can be found Here and on GitHub Here

Import libraries

In [1]:

```
import cv2
import glob
import matplotlib.pyplot as plt
import numpy as np
from PIL import Image
import random
```

Extract Images from Google Drive

In this step, store the image path from car dataset dataset into a variable.

In [2]:

```
Imagepaths = glob.glob("/content/drive/MyDrive/Segmentation_images/Car/*.jpg",recursive='
true')
```

Open the Images and store it in a List.

use the above varialbe to get the images and load the images to a array.

In [3]:

```
#opening image and converting it array:
OriginalImage = np.array([np.asarray(Image.open(img)) for img in Imagepaths])
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:2: VisibleDeprecationWarning
: Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-
tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to do th
is, you must specify 'dtype=object' when creating the ndarray
```

Resize Image:

Some images may be captured by a camera and may vary in size, therefore, Resizing needs to be done , and establish a base size for all images which can be used for image segmentation

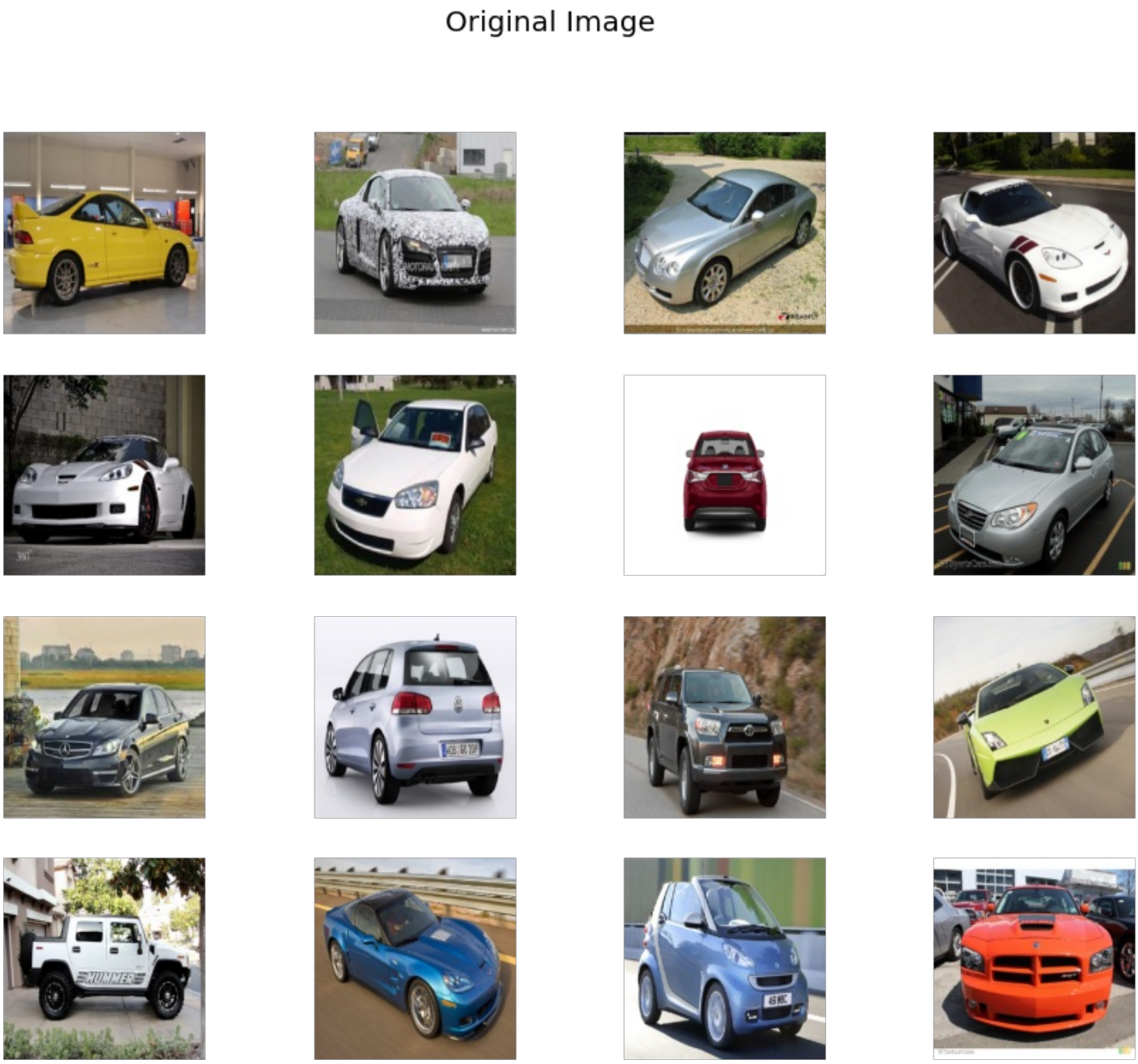
In [4]:

```
#Resizing the images:
for i in range(len(OriginalImage)):
  OriginalImage[i] = cv2.resize(OriginalImage[i], (512,512),interpolation=cv2.INTER_NEARE
ST)
```

Display Original Image:

In [5]:

```
# Displaying the Orginal Image:
plt.figure(figsize=(15,12))
for i, img in enumerate(OriginalImage[0:16]):
    plt.subplot(4,4,i+1)
    plt.axis('off')
    plt.grid(False)
    plt.imshow(img)
    plt.suptitle("Original Image", fontsize=20)
plt.show()
```

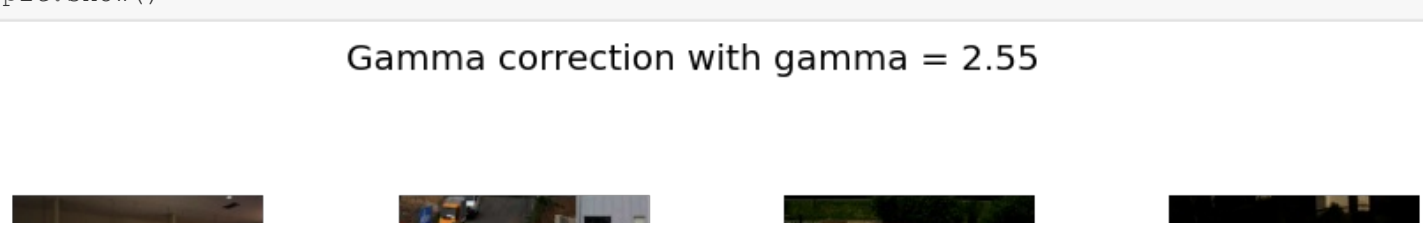


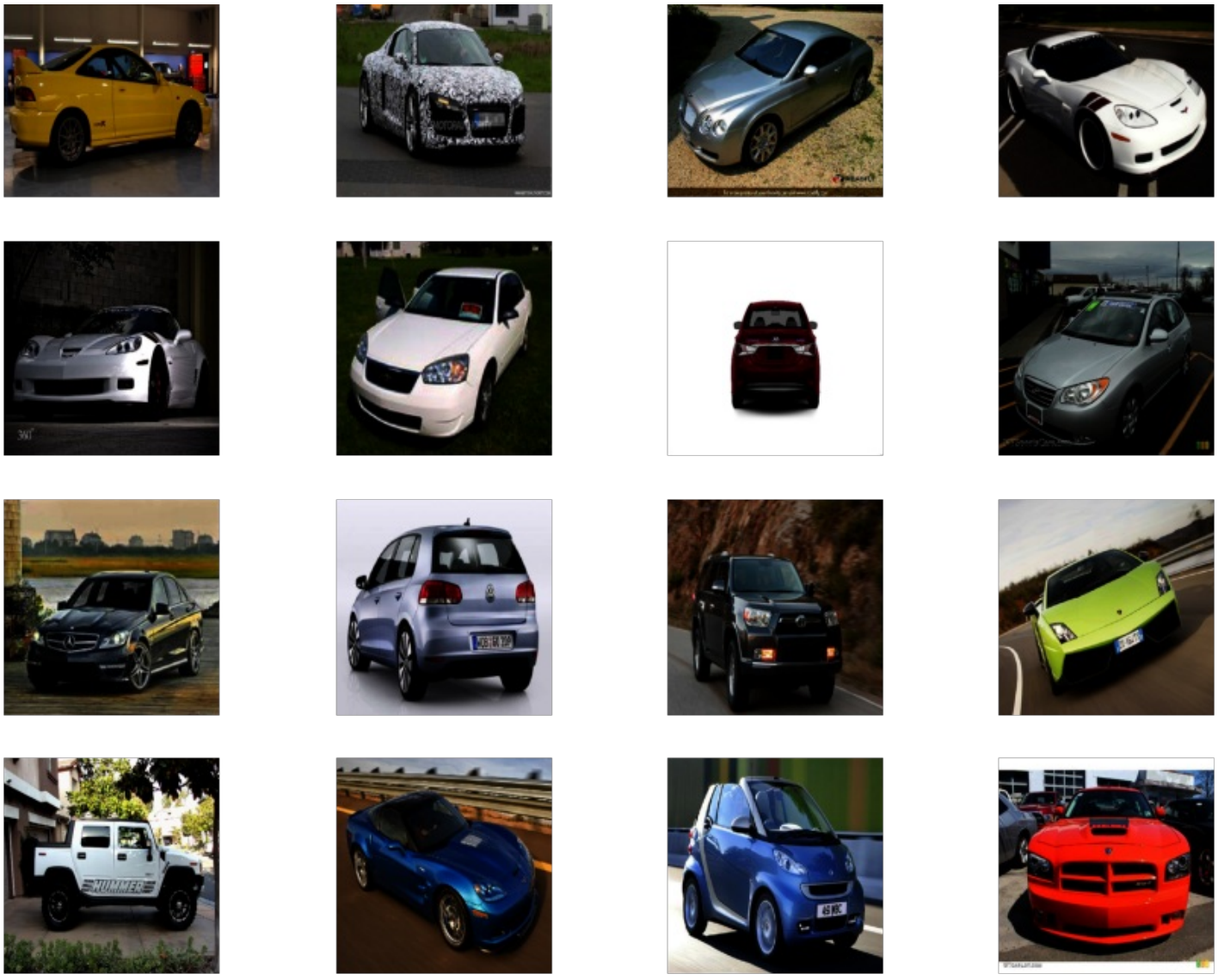
Pre-Processsing:

Power Law Transformation (gamma correction):

In [6]:

```
logTransformedImage= np.array([np.array(255*(img / 255) ** 2.55, dtype = 'uint8')  for i
mg in OriginalImage])

# Display Gray-Scale Images:
plt.figure(figsize=(15,12))
for i, img in enumerate(logTransformedImage[0:16]):
    plt.subplot(4,4,i+1)
    plt.axis('off')
    plt.grid(False)
    plt.imshow(img)
    plt.suptitle("Gamma correction with gamma = 2.55", fontsize=20)
plt.show()
```



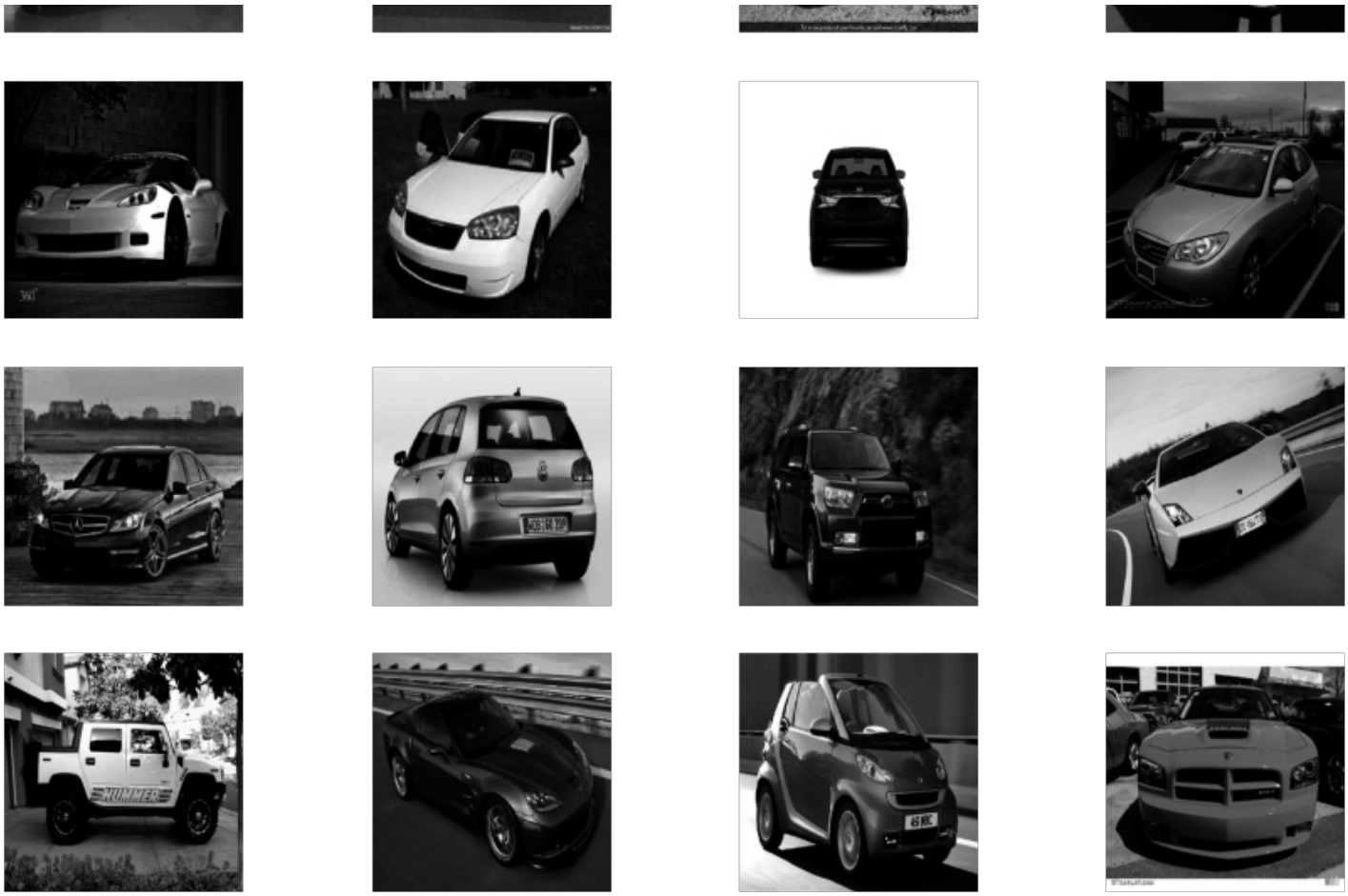


Convert transformed Images to GrayScale Images

use the inbuilt function to convert them to GrayScale and then plot the GrayScale Images

In [7]:

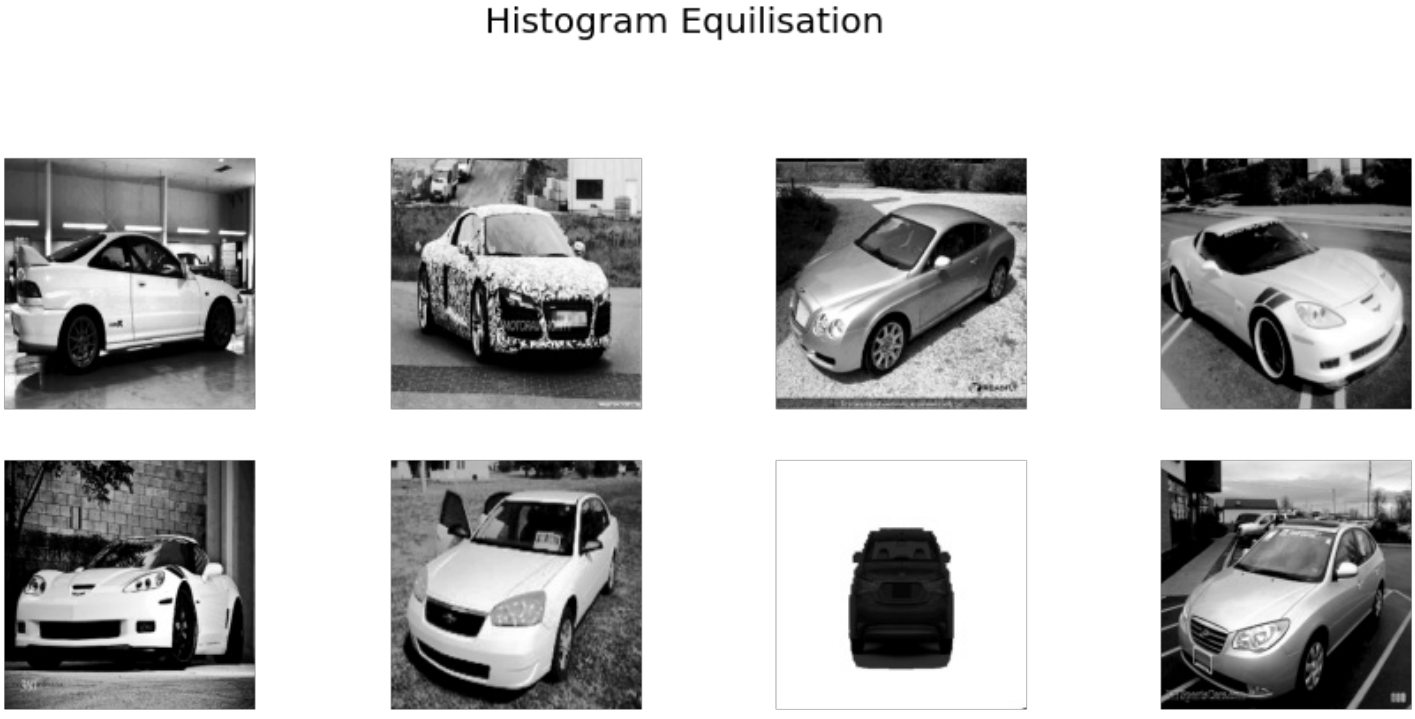
```
#converting the original images to GrayScale:
grayImage = np.array([cv2.cvtColor(img,cv2.COLOR_RGB2GRAY) for img in logTransformedImage
])

# Display Gray-Scale Images:
plt.figure(figsize=(15,12))
for i, img in enumerate(grayImage[0:16]):
    plt.subplot(4,4,i+1)
    plt.axis('off')
    plt.grid(False)
    plt.imshow(img,'gray')
    plt.suptitle("Grayscale", fontsize=20)
plt.show()
```



Histogram Equilization

In [8]:

```
equilized = np.array([cv2.equalizeHist(img) for img in grayImage])
plt.figure(figsize=(15,12))
for i, img in enumerate(equilized[0:16]):
    plt.subplot(4,4,i+1)
    plt.axis('off')
    plt.grid(False)
    plt.imshow(img,'gray')
    plt.suptitle("Histogram Equilisation", fontsize=20)
plt.show()
```



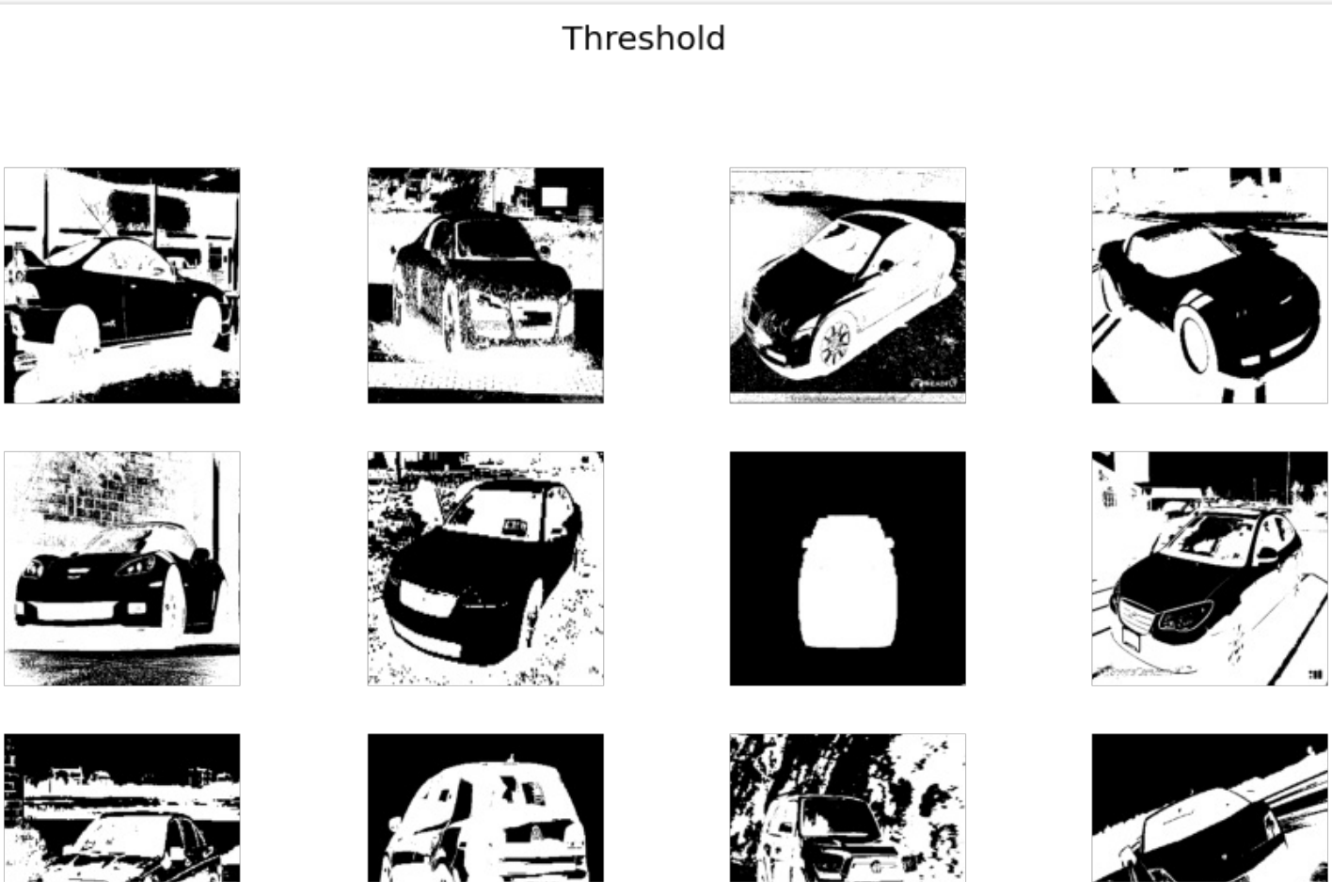
Segmentation:

Binary Thresholding

using opencv tool to apply thresholding on the set of images

In [9]:

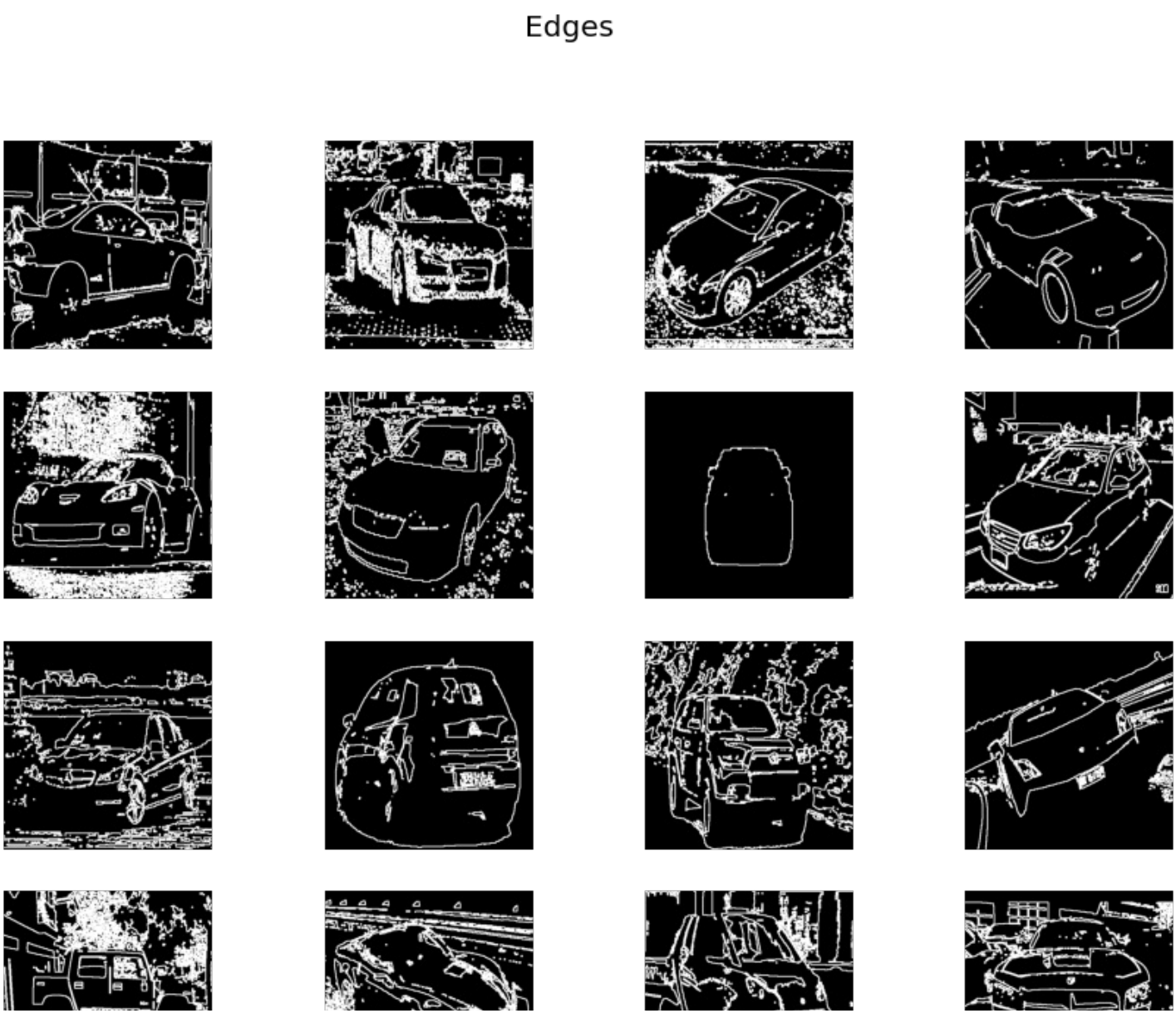
```
#Thresholding the Images:
thresh = [cv2.threshold(img, 130, 255, cv2.THRESH_BINARY_INV)[1] for img in equilized]

#displaying the Thresholding Images:
plt.figure(figsize=(15,12))
for i, img in enumerate(thresh[0:16]):
    plt.subplot(4,4,i+1)
    plt.axis('off')
    plt.imshow(img,'gray')
    plt.suptitle("Threshold", fontsize=20)
plt.show()
```



Canny Edge Detection:

In [10]:

```
#canny Edge Detection:
edges = [cv2.dilate(cv2.Canny(img, 0, 255), None) for img in thresh]

#dsiplaying the Canny Edge:
plt.figure(figsize=(15,12))
for i, img in enumerate(edges[0:16]):
    plt.subplot(4,4,i+1)
    plt.axis('off')
    plt.grid(False)
    plt.imshow(cv2.cvtColor(img, cv2.COLOR_GRAY2RGB))
    plt.suptitle("Edges", fontsize=20)
plt.show()
```



Applying Masks

applying the mask to the edge detected images, and then segmenting them

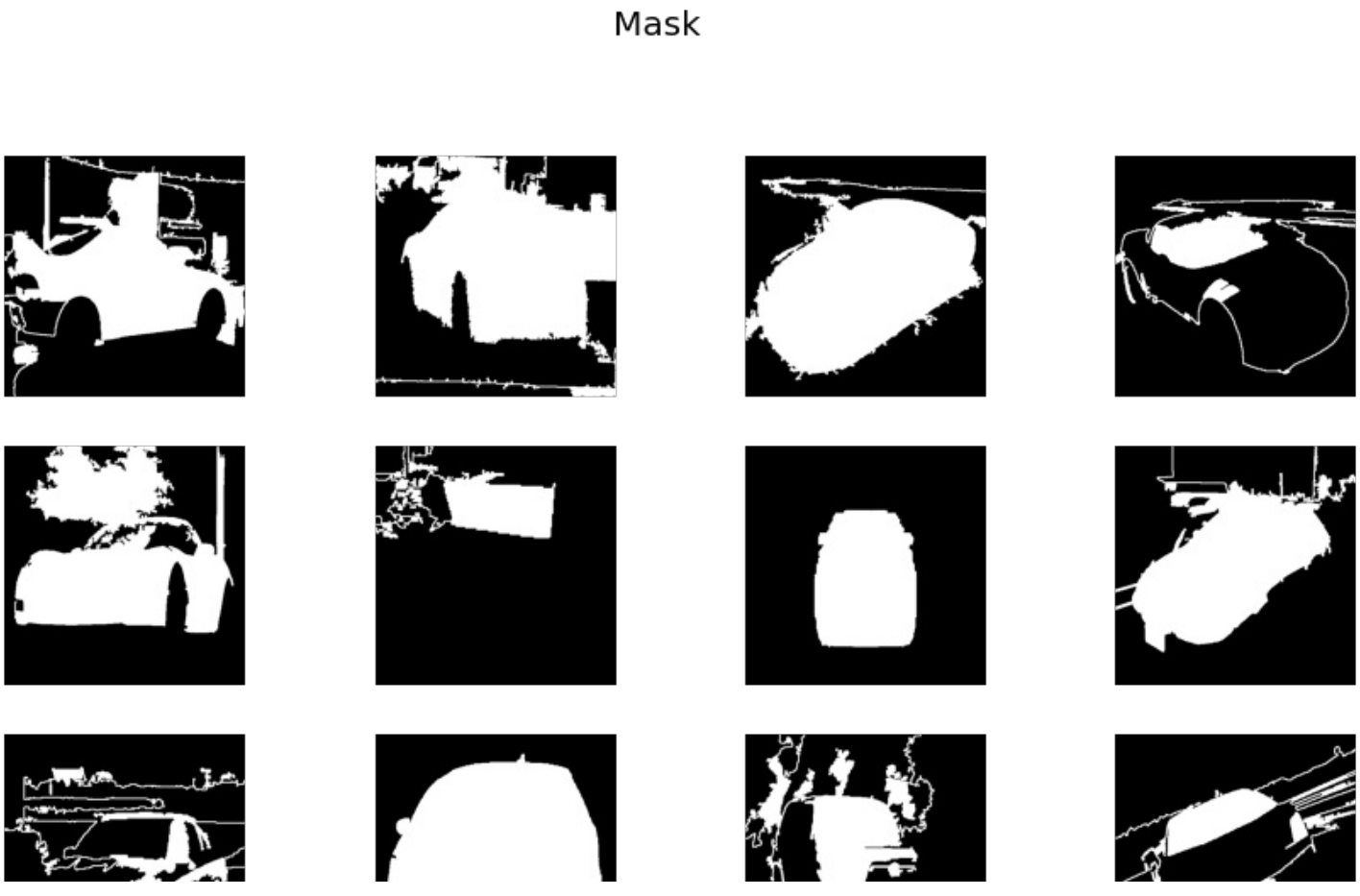
In [11]:

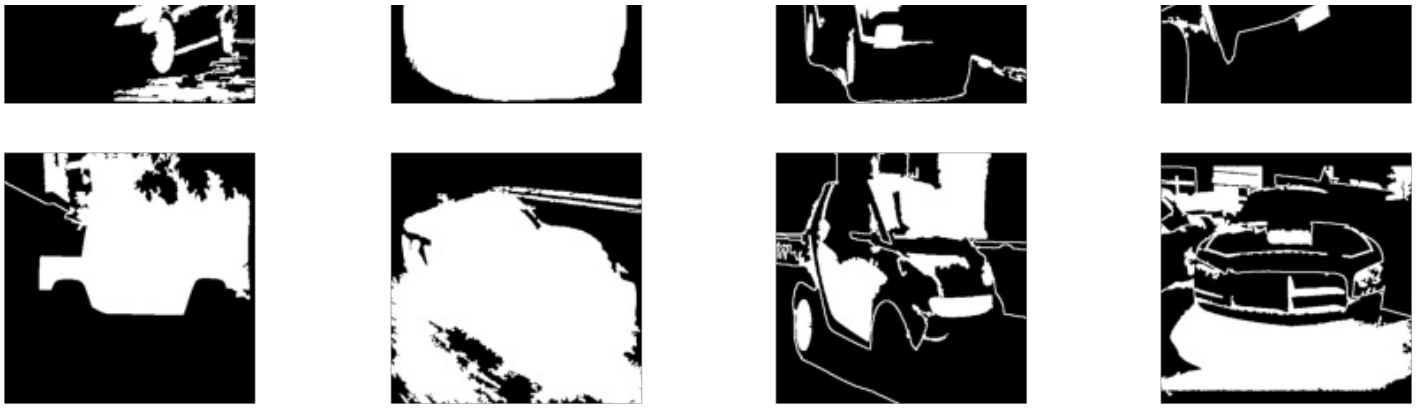
```
#mask:
masked=[]
segmented=[]

for i, img in (enumerate(edges)):
  if(i==16):
    continue
  cnt = sorted(cv2.findContours(img, cv2.RETR_CCOMP, cv2.CHAIN_APPROX_SIMPLE)[-2], key=c
v2.contourArea)[-1]
  mask = np.zeros((512,512), dtype='uint8')
  masked.append(cv2.drawContours(mask, [cnt],-1, 255, -1))
  dst = cv2.bitwise_and(OriginalImage[i], OriginalImage[i], mask=mask)
  segmented.append(cv2.cvtColor(dst, cv2.COLOR_BGR2RGB))
```

Displaying Image Masks:

In [12]:

```
#displaying Masked Images:
plt.figure(figsize=(15,12))
for i, masking in enumerate(masked[0:16]):
  plt.subplot(4,4,i+1)
  plt.axis('off')
  plt.grid(False)
  plt.imshow(masking, cmap='gray')
plt.suptitle("Mask", fontsize=20)
plt.show()
```



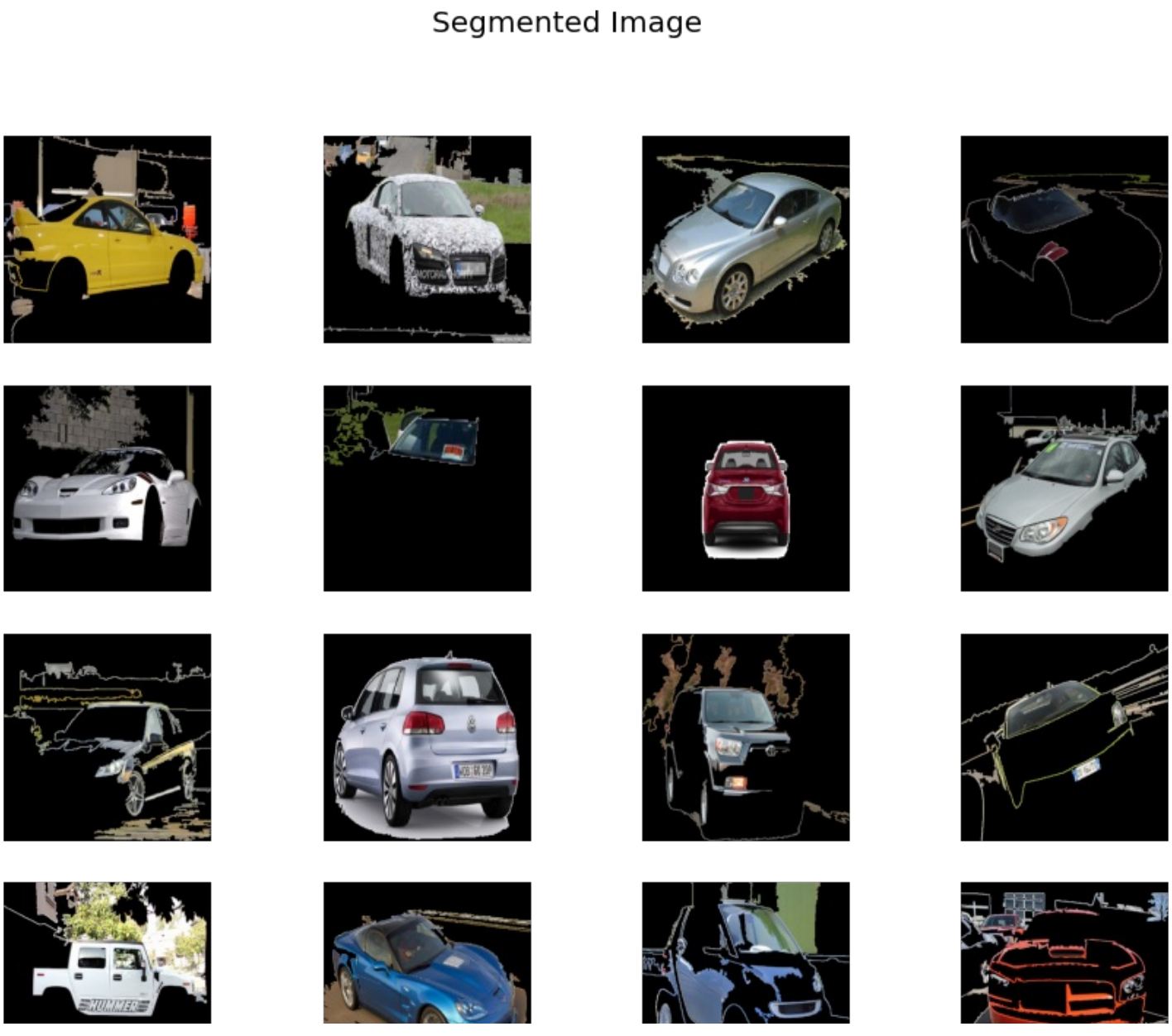


Displaying the Segemented Images:

using the plot function displaying the coroulred segmented objects

In [13]:

```
plt.figure(figsize=(15,12))
for i, img in enumerate(segmented[0:16]):
  plt.subplot(4,4,i+1)
  plt.axis('off')
  plt.grid(False)
  plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))

plt.suptitle("Segmented Image", fontsize=20)
plt.show()
```



In [14]:

```
segmented[0] = cv2.resize(segmented[0], (2048, 1536),
                          interpolation=cv2.INTER_NEAREST)
plt.imshow(cv2.cvtColor(segmented[0], cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.grid(False)
plt.title("Frist Segmented Image", fontsize=20)
```

Out[14]:

```
Text(0.5, 1.0, 'Frist Segmented Image')
```