International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 10 Issue 08, August-2021

Detectron2 Object Detection & Manipulating Images using Cartoonization

Allena Venkata Sai Abhishek
Dept of Computer Science and Engineering
GITAM University
Visakhapatnam, India

Sonali Kotni
Dept of Computer Science and Engineering
GITAM University
Visakhapatnam, India

***Abstract*— In today's world, there is a rapid increase in the autonomous vehicle. There are various levels of autonomous vehicles depending upon the degree of autonomy-for the lower degree of autonomy driver has more power and functionality for managing, on coming to the fully automated vehicle like Tesla are expected to have full control over the functions. These advances cooperate to plan the vehicle's position and its nearness to everything around it. Because of this, there is popularity for these vehicles, since they give a great deal of advantages to individuals utilizing them. We use the Facebook AI Research software system that implements object detection algorithms, Caffe2 deep learning framework for advanced object detection by offering speedy training. We have also manipulated images to derive insights addressing the issues companies face when making the step from research to production. We have implemented detectron2 object detection for faster detection of objects. There is labeling of the object & we used manipulation of images using cartoonization.**

Keywords*— *Mask R-CNN; Retina Net; Faster R-CNN; RPN; Fast R-CNN, R-FC; Classification; Deep Learning; Grayscale;

I. INTRODUCTION

In this automation, the information is gathered by the on-board sensors without any communication. Moreover, these automated vehicles can communicate with each other In this automation, the data is accumulated by the on-board sensors with no correspondence [1]. Besides, these computerized vehicles can speak with one another and can share data about the climate. . We utilize the Facebook AI Research programming framework that executes object location calculations, Caffe2 profound learning structure for cutting edge object discovery by offering expedient preparing.

The objective of Detectron is to offer a –

- High-quality,
- High-execution
- Codebase for object location research.

It is intended to be adaptable to help fast execution and assessment of novel exploration.

II. DATA COLLECTION

Then input data for our model is image type. We give an input image in either JPEG or PNG format. The input image is used for manipulation of using various cartoonization techniques. Then we import the necessary libraries for the uploading of the images.

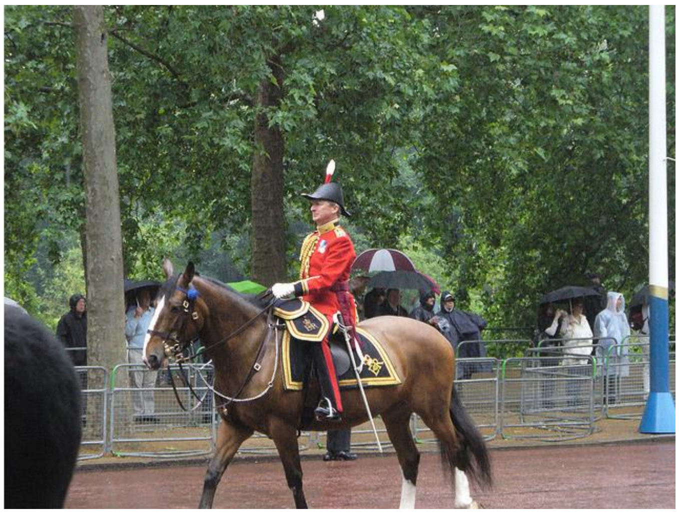


Fig. 1. Input Image

III. PROPOSED MODEL

The proposed model take up input as images in the format of PNG, or JPEG, using multiple libraries in which, we use the Detectron2 for the faster object detection of the objects using various object detection algorithms such as Mask R-CNN; Retina Net; Faster R-CNN; RPN; Fast R-CNN, R-FC; Classification; Deep Learning; Grayscale. We take the backbone & proposal that we crop & wrap and we implement all the box, mask, key points, dense pose and semantic segmentation, while clubbing it and generating labels and we detect the object using a box [4]. We have also manipulated images by grey scaling, cartoonizing, applying bilateral & Gaussian filtering, to derive insights addressing the issues companies face when shifting from research to production.

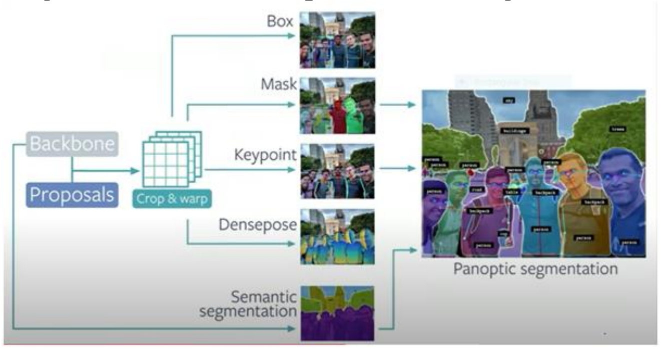


Fig. 2. Proposed model framework

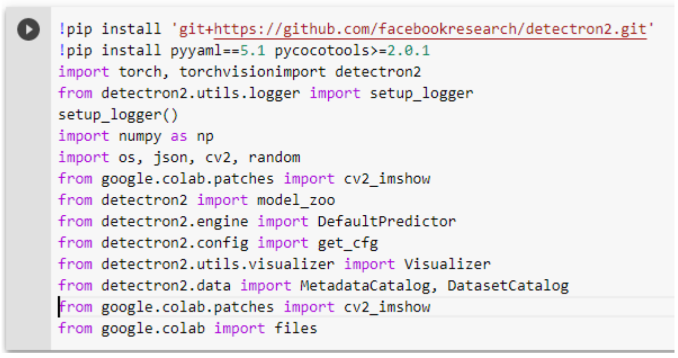
IV. METHODOLOGY

The detectron2 framework is initially imported using the git command from the Github repository of the detectron2.

A. Installing & Importing the required Dependencies:

We install & import the required dependencies that are as follows-

a. pyyaml
b. CUDA
c. Torch
d. Torchvision
e. detectron2 logger
f. Numpy
g. JSON
h. OpenCV
i. Random
j. detectron2 utilities

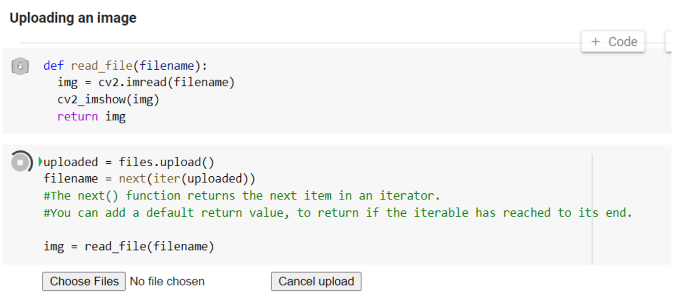


```
!pip install 'git+https://github.com/facebookresearch/detectron2.git'
!pip install pyyaml==5.1 pycocotools>=2.0.1
import torch, torchvisionimport detectron2
from detectron2.utils.logger import setup_logger
setup_logger()
import numpy as np
import os, json, cv2, random
from google.colab.patches import cv2_imshow
from detectron2 import model_zoo
from detectron2.engine import DefaultPredictor
from detectron2.config import get_cfg
from detectron2.utils.visualizer import Visualizer
from detectron2.data import MetadataCatalog, DatasetCatalog
from google.colab.patches import cv2_imshow
from google.colab import files
```

Fig. 3. Dependencies Installed & Imported

B. Uploading an Image:

We define a function read_file function to read the file. Then input data for our model is image type. We give an input image in either JPEG or PNG format. The input image is used for manipulation of using various cartoonization techniques. Then we import the necessary libraries for the uploading of the images.

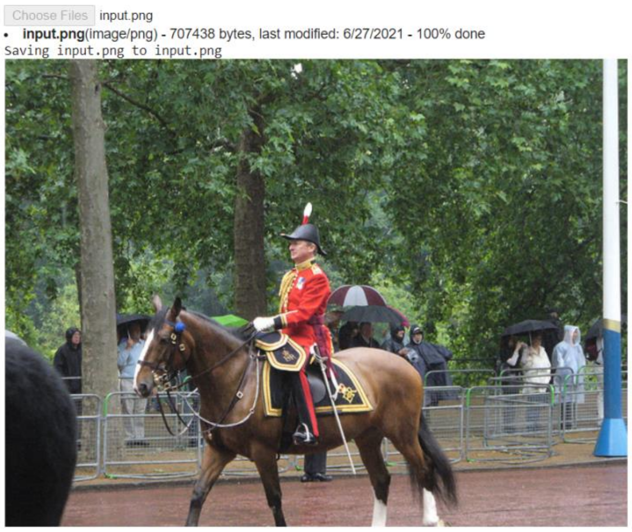


Uploading an image

```
def read_file(filename):
  img = cv2.imread(filename)
  cv2_imshow(img)
  return img
```

```
uploaded = files.upload()
filename = next(iter(uploaded))
#The next() function returns the next item in an iterator.
#You can add a default return value, to return if the iterable has reached to its end.

img = read_file(filename)
```

Fig. 4. Image uploading



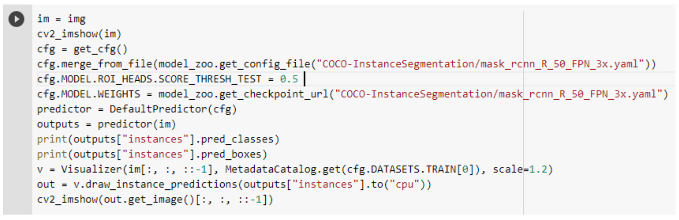
```
Choose Files input.png
• input.png(image/png) - 707438 bytes, last modified: 6/27/2021 - 100% done
Saving input.png to input.png
```

Fig. 5. Image that has been uploaded

C. Detecting& Labelling of Image:

We detect & label the objects of the image. We take the backbone & proposal that we crop & wrap and we implement all the box, mask, key points, dense pose and semantic segmentation, while clubbing it and generating labels and we detect the object using a box. A mask is applied on the image, finding the key points[2]. Then we use dense pose & semantic segmentation to finally display the image with labeled box.

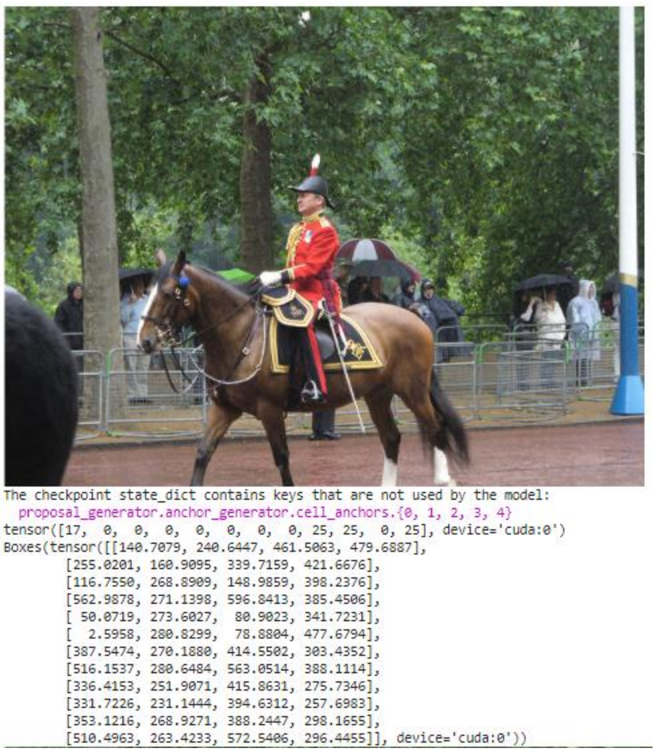
a. We get the Image from uploading or either from the MS-COCO dataset.
b. We have created a detectron2 configuration and a detectron2 Default Predictor for the running of the inference on a particular image.
c. We also add the model - specific configuration like, Tensor Mask, etc. here as we are not running a model in detectron2's core library.
d. We set a certain threshold for this model.
e. We find a model from detectron2's model zoo.
f. Last and final step is to visualize our processed image.



```
im = img
cv2_imshow(im)
cfg = get_cfg()
cfg.merge_from_file(model_zoo.get_config_file("COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_3x.yaml"))
cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.5
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_3x.yaml")
predictor = DefaultPredictor(cfg)
outputs = predictor(im)
print(outputs["instances"].pred_classes)
print(outputs["instances"].pred_boxes)
v = Visualizer(im[:, :, ::-1], MetadataCatalog.get(cfg.DATASETS.TRAIN[0]), scale=1.2)
out = v.draw_instance_predictions(outputs["instances"].to("cpu"))
cv2_imshow(out.get_image()[:, :, ::-1])
```

Fig. 6. Detecting& Labeling of Image

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 10 Issue 08, August-2021



```
The checkpoint state_dict contains keys that are not used by the model:
  proposal_generator.anchor_generator.cell_anchors.{0, 1, 2, 3, 4}
tensor([17,  0,  0,  0,  0,  0,  0,  0, 25, 25,  0, 25], device='cuda:0')
Boxes(tensor([[140.7079, 240.6447, 461.5063, 479.6887],
        [255.0201, 160.9095, 339.7159, 421.6676],
        [116.7550, 268.8909, 148.9859, 398.2376],
        [562.9878, 271.1398, 596.8413, 385.4506],
        [ 50.0719, 273.6027,  80.9023, 341.7231],
        [  2.5958, 280.8299,  78.8804, 477.6794],
        [387.5474, 270.1880, 414.5502, 303.4352],
        [516.1537, 280.6484, 563.0514, 388.1114],
        [336.4153, 251.9071, 415.8631, 275.7346],
        [331.7226, 231.1444, 394.6312, 257.6983],
        [353.1216, 268.9271, 388.2447, 298.1655],
        [510.4963, 263.4233, 572.5406, 296.4455]], device='cuda:0'))
```

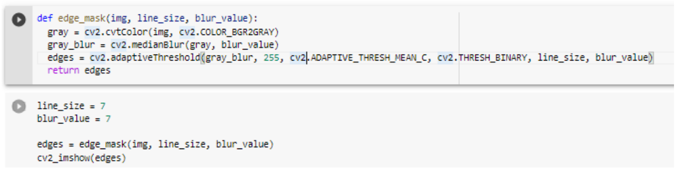
Fig. 7. Detecting& Labeling of Image Computations

D. Manipulating the Image:

We use the input image and we manipulate it by using the following techniques to derive insights [3]. Manipulating of an image can be done in many ways. Here the image is manipulated by cartoonizing the image which involves in adding a cartoon effect to the image and the image can be filtered by using various filters. The steps for manipulating an image are briefly described below.

a. <u>Creating an edge mask function</u>

Initially an image is uploaded from device or from a dataset. Then an edge mask is created. When creating an edge mask,the thickness of an image's edges is given first consideration when producing an edge mask. The cv2.adaptiveThreshold () method will be used to identify the edge of a picture. To determine the threshold for smaller areas of the picture, we utilize the cv2.adaptiveThreshold () function. As a result, different thresholds are obtained for various parts of the same picture. It will highlight the black edges surrounding the image's objects.



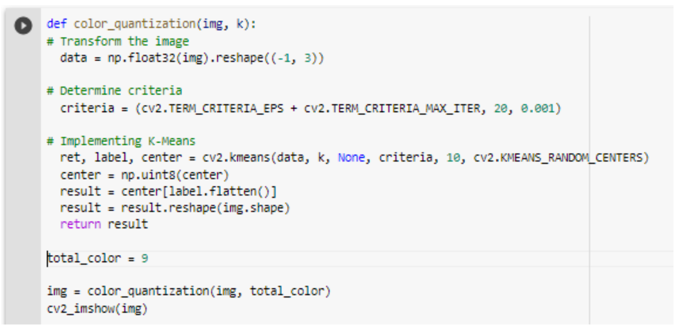
```
def edge_mask(img, line_size, blur_value):
  gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
  gray_blur = cv2.medianBlur(gray, blur_value)
  edges = cv2.adaptiveThreshold(gray_blur, 255, cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY, line_size, blur_value)
  return edges
```

```
line_size = 7
blur_value = 7

edges = edge_mask(img, line_size, blur_value)
cv2_imshow(edges)
```

Fig. 8. Detecting& Labeling of Creating an edge mask function

b. <u>Converting into grayscale</u>

Secondly, the image is converted to grayscale. Here the image consists of two colors i.e. black and white. During the process of gray scaling and image, the noise is compressed from the image to reduce the number of detected edges that are not required. cv2.adaptiveThreshold () defines the line size of the edge. The thicker borders that will be highlighted in the image will have a higher line size.

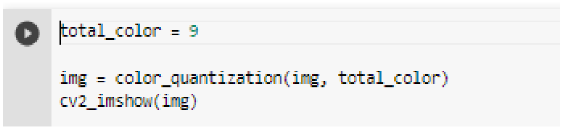


```
def color_quantization(img, k):
# Transform the image
  data = np.float32(img).reshape((-1, 3))

# Determine criteria
  criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 20, 0.001)

# Implementing K-Means
  ret, label, center = cv2.kmeans(data, k, None, criteria, 10, cv2.KMEANS_RANDOM_CENTERS)
  center = np.uint8(center)
  result = center[label.flatten()]
  result = result.reshape(img.shape)
  return result

total_color = 9

img = color_quantization(img, total_color)
cv2_imshow(img)
```

Fig. 9. Converting into grayscale

c. <u>Reducing color palette</u>

Color Quantization: This method reduces the number of colors in the image and gives it a cartoon effect. When presenting output with a finite number of colors, color quantization is accomplished using the K-means clustering method. K-means is an unsupervised machine learning algorithm that performs clustering. From the word, K means number of clusters and Means refers to the variance. We can determine the number of colors in the output picture using different values of K. So, here for the present image the number of colors is reduced to 9.



```
total_color = 9

img = color_quantization(img, total_color)
cv2_imshow(img)
```

Fig. 10 . Reducing color palette

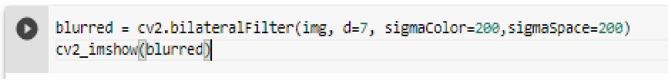
d. <u>Bilateral Filtering</u>

The bilateral filter is the next approach for decreasing picture noise. It decreases the image's blurriness and sharpness. Consider a 3D bilateral filter that is processing an image's edge region. Each pixel value is replaced by a weighted average of neighboring pixel values in a bilateral filter. In order to retain edges, it uses a variety of pixel intensities.

For bilateral filtering, there are three key requirements. They are:

- **d** :Diameter of each pixel neighborhood
- **sigmaColor:** A higher value for the parameter implies that colors from further away in the pixel neighborhood will be blended together, resulting in bigger semi-equal color regions.

- **sigmaSpace:** As long as the pixels' colors are close enough, a higher value of the parameter implies that they will affect each other.

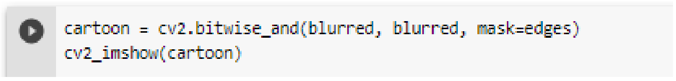


```
blurred = cv2.bilateralFilter(img, d=7, sigmaColor=200,sigmaSpace=200)
cv2_imshow(blurred)
```

Fig. 11 . Bilateral Filtering

e. <u>Combining edge mask with the colored image - Adding Cartoon Effect</u>

Finally the edge mask is combined with the color-processed image. Here cv2.bitwise_and function is used. Bitwise operations are performed on the image to get the output. Now you can see how an image can be converted into a cartoon. So, come on and have a try by converting your images into a cartoon.



```
cartoon = cv2.bitwise_and(blurred, blurred, mask=edges)
cv2_imshow(cartoon)
```

Fig. 12. Combining edge mask with the colored image - Adding Cartoon Effect

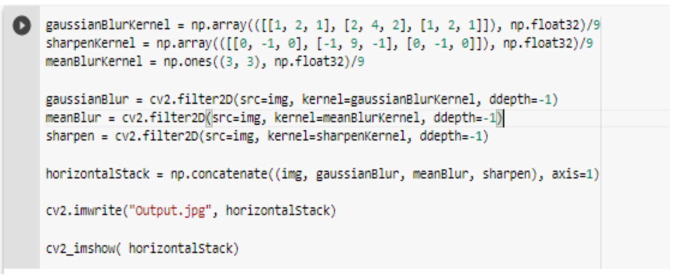
f. <u>Filtering the Image</u>

Apart from using bilateral filter for filtering the image, Gaussian Blur, sharpen and mean Blur kernel can be used in filtering an image.

Gaussian blurring to an Image: This approach utilizes a Gaussian filter that performs a weighted average. The Gaussian blurs weights pixel values based on their distance from the kernel's centre. The weighted average is less affected by pixels that are further away from the centre.

Median Blurring to an Image: Each pixel in the source picture is replaced by the median value of the image pixels in the kernel region in median blurring.

Sharpening an Image: A 2D-convolution kernel can be used to sharpen a picture. Create a custom 2D kernel first, then apply the convolution operation to the picture with the filter 2D ( ) method.



```
gaussianBlurKernel = np.array(([[1, 2, 1], [2, 4, 2], [1, 2, 1]]), np.float32)/9
sharpenKernel = np.array(([[0, -1, 0], [-1, 9, -1], [0, -1, 0]]), np.float32)/9
meanBlurKernel = np.ones((3, 3), np.float32)/9

gaussianBlur = cv2.filter2D(src=img, kernel=gaussianBlurKernel, ddepth=-1)
meanBlur = cv2.filter2D(src=img, kernel=meanBlurKernel, ddepth=-1)
sharpen = cv2.filter2D(src=img, kernel=sharpenKernel, ddepth=-1)

horizontalStack = np.concatenate((img, gaussianBlur, meanBlur, sharpen), axis=1)

cv2.imwrite("Output.jpg", horizontalStack)

cv2_imshow( horizontalStack)
```

Fig. 13. Filtering the Image

V. RESULTS

We were successfully able to detect the objects of the images and we were able to label it according to the predefined dataset, & we manipulate the images as shown below:

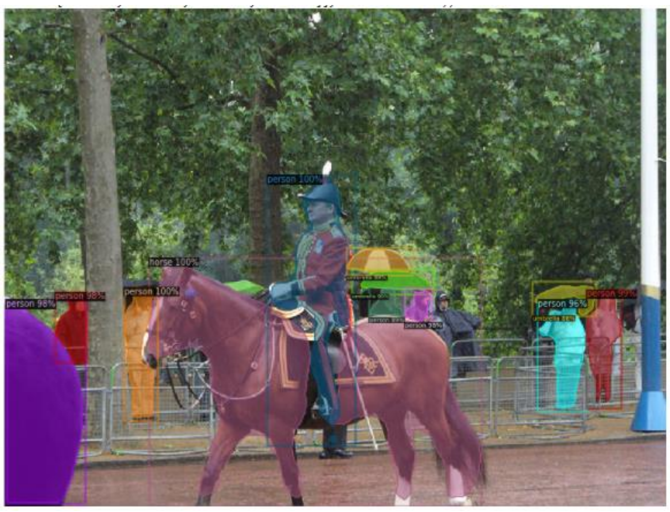


Fig. 14. Detecting & labeling of the objects

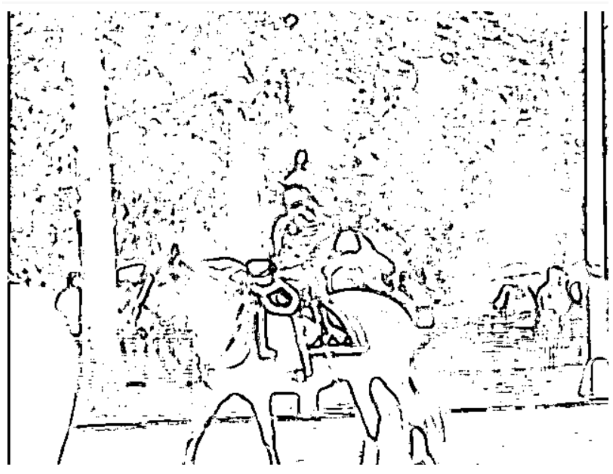


Fig. 15. Converting into grayscale

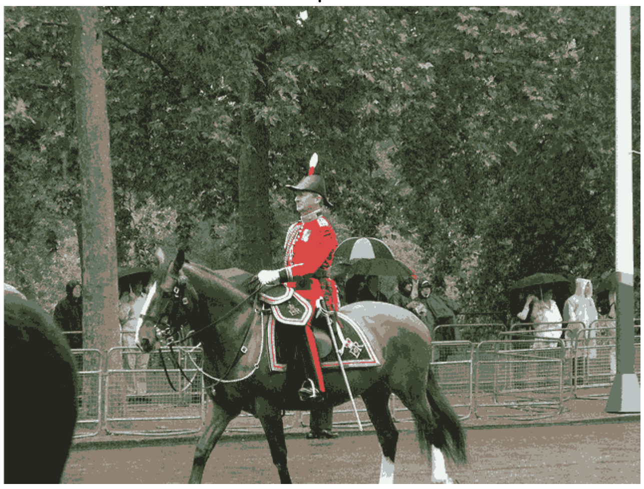


Fig. 16. Color Quantization

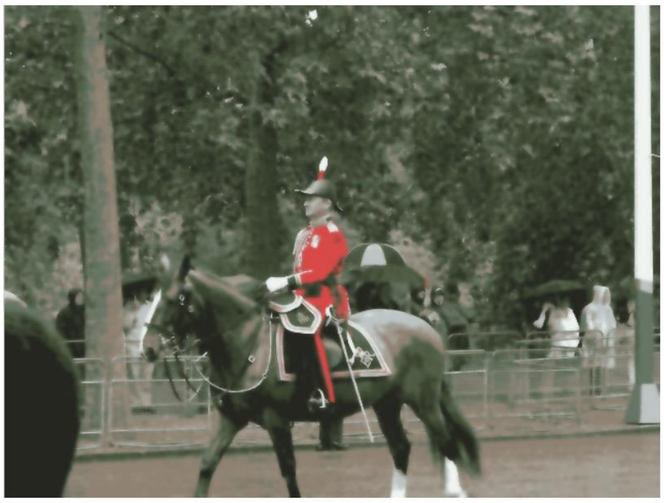


Fig. 17. Bilateral Filtering

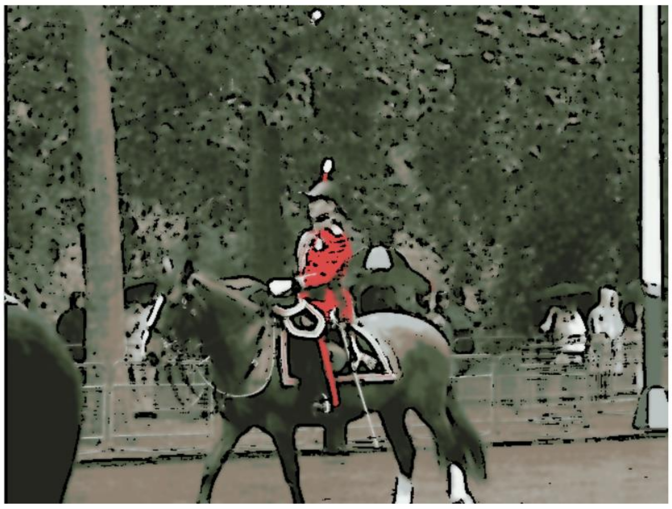


Fig. 18. Adding Cartoon Effect

Fig. 19. Filtering the Image using Gaussian, sharpen and Mean Blur filters

VI. CONCLUSION

In this study, We fine tuned a framework that comprised of the superlative model for the object detection application practices, for which we have developed and implemented, an advanced object detection by offering speedy training, the FAIR software system that implements object detection algorithms like, Mask R-CNN, Retina Net, Faster R-CNN, RPN, Fast R-CNN, R-FCN & it uses Caffe2 deep learning framework for it. We have also manipulated images by grey scaling, cartoonizing, applying bilateral & Gaussian filtering, to derive insights addressing the issues companies face when making the step from research to production.

VII.REFERENCES

[1] https://analyticsvidhya.com/blog/2018/01/facebook-launched-detectron-platform-object-detection-research/
[2] https://towardsdatascience.com/image-labelling-using-facebooks-detectron-4931e30c4d0c
[3] Archana B. Patankar; Purnima A. Kubde; Ankita Karia (Aug. 2016). Image cartoonization methods IEEE
[4] Vung Pham; Chau Pham; Tommy Dang (2020). Road Damage Detection and Classification with Detectron2 and Faster R-CNN 20511275 IEEE