**Solutions to cARscan Programming task for Computer Vision Engineer**

**Note:**

1. I have used traditional image processing, machine learning and deep learning techniques respectively for the three different models below.

2. Please find the output images in respective folders.

**Models:**

1. Blur detection
2. Bad lighting detection
3. Semantic segmentation for Background removal / replacement

**Brief Description of the Models:**

1. **Blur detection**

I have used a simple image processing technique to classify blurry and non-blurry (sharp) images.

Steps / pseudo code is as follows:

1. Read the input image
2. Convert the input image to grayscale
3. Convolve the grayscale image with the Laplacian kernel
4. Take the variance (standard deviation squared) of the output of convolution operation
5. If variance < pre-defined threshold (here, set to 100), classify as ‘Blurry’

else, classify as ‘Non-Blurry’

**Output**:

Of the 10 given images, 9 are non-Blurry and 1 Blurry (view5).

Also, additionally 1 blurry image downloaded from the internet was shown to detect blurry correctly. The “blurriness” score of the images is given on the images itself in ‘output/blurry’ folder.

View1: Non Blurry View2: Blurry Image1 (downloaded): Blury



**Further improvement:**

This model can be further improved by carefully selecting the threshold level on a larger dataset. Also, FFT based models can be used to improve robustness of the model. If the dataset is significantly large, deep learning models can be explored too.

1. **Bad lighting detection**

For this task I have used a machine learning model to classify bad lighting and good lighting images from their grayscale histograms.

Steps / pseudo code is as follows:

* + 1. Read the input images
    2. Convert the input images to grayscale
    3. Calculate the histogram of each image with bins=256
    4. Train a Random Forest Classifier with X = histograms and y = labels to classify between bad lighting (1) and good lighting (0) images.
    5. Evaluate the model on performance metrics such as accuracy and f1-score.

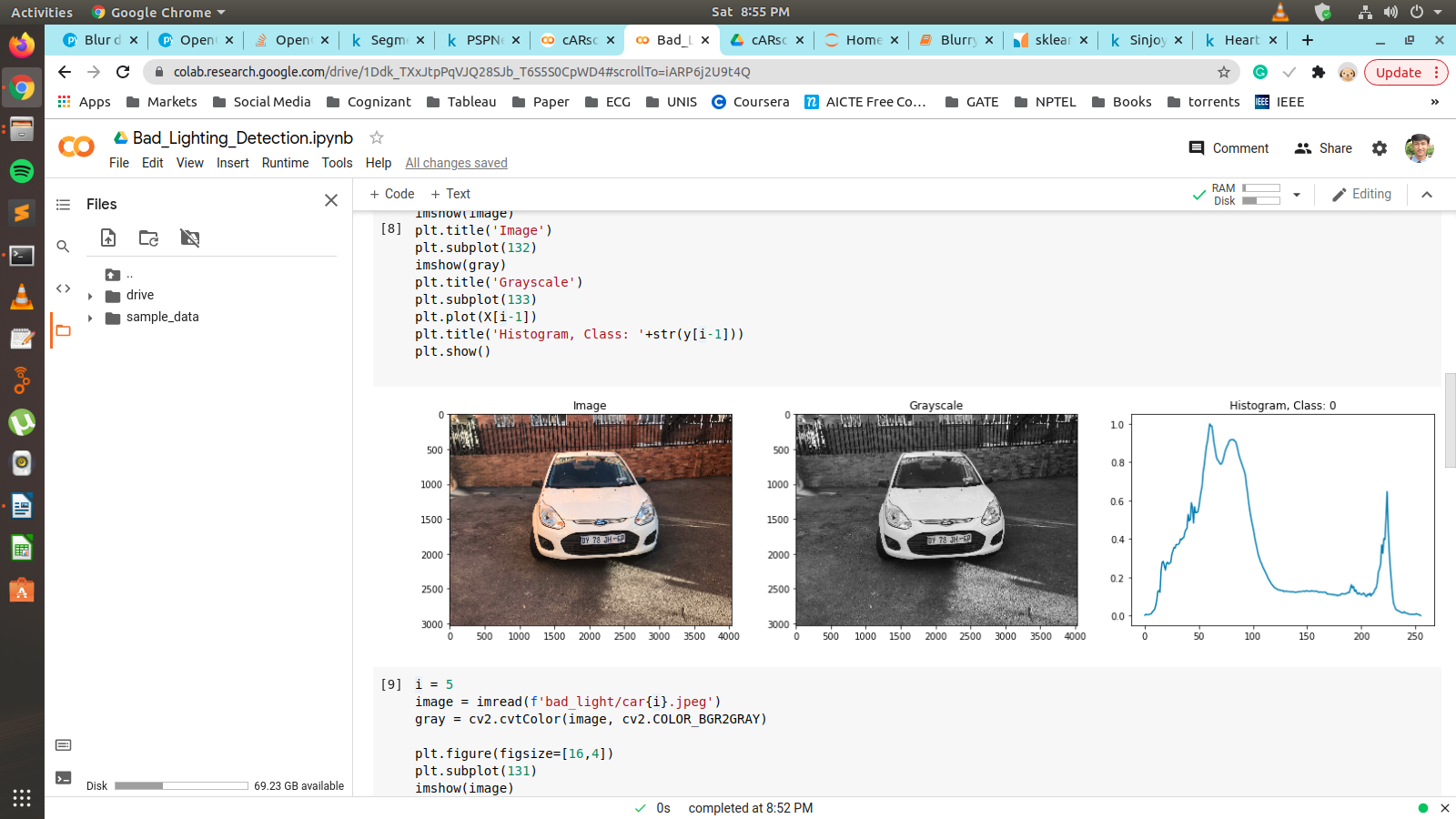
**Output**:

The dataset is created by labelling the given 10 car images as ‘good lighting’ (0) and labelling 5 bad lighting images downloaded from the internet as ‘bad lighting’ (1).

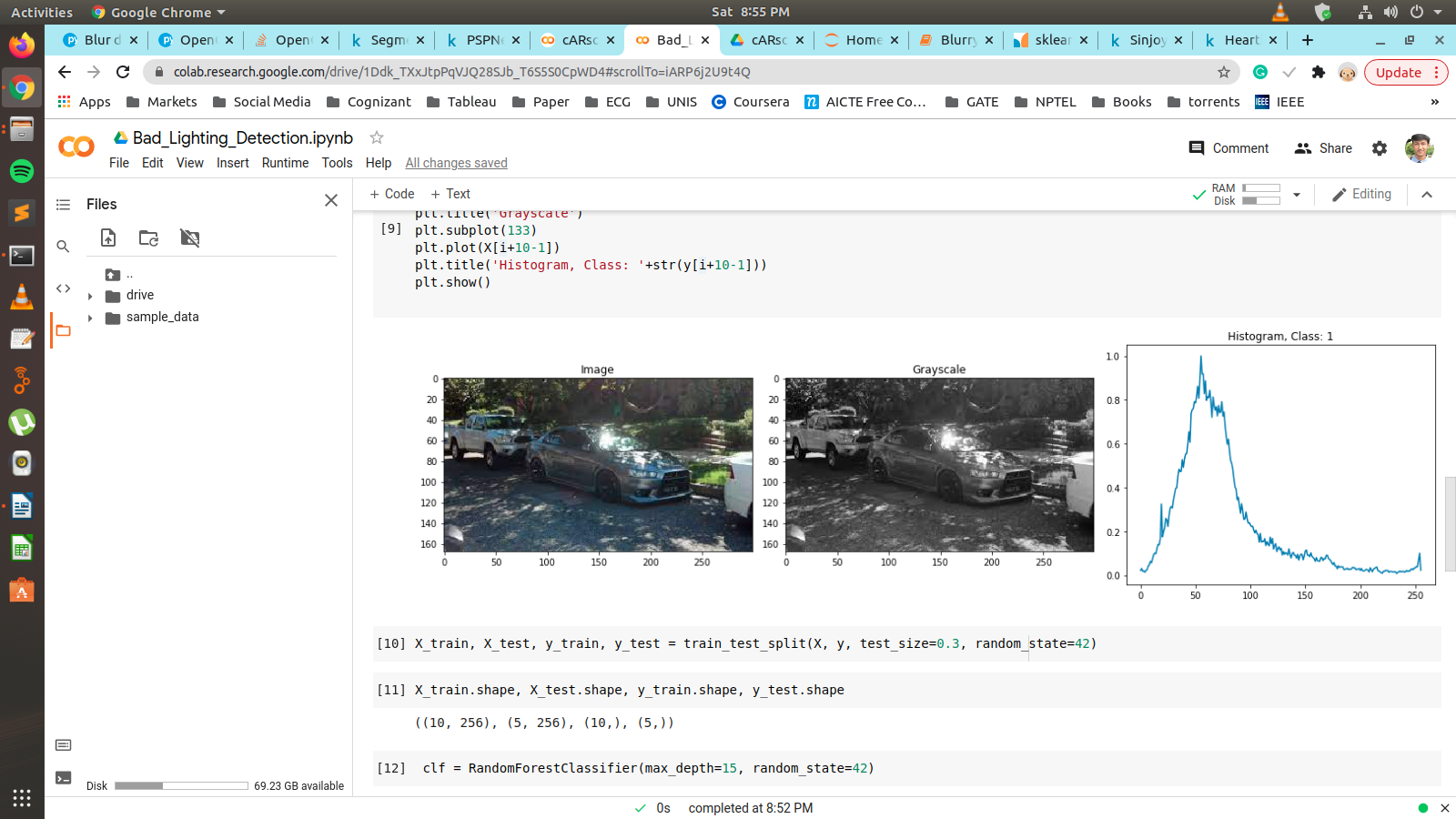
The train / test split was 70-30 % i.e., 10 train images and 5 test images.

The test set contains 3 good lighting and 2 bad lighting images.

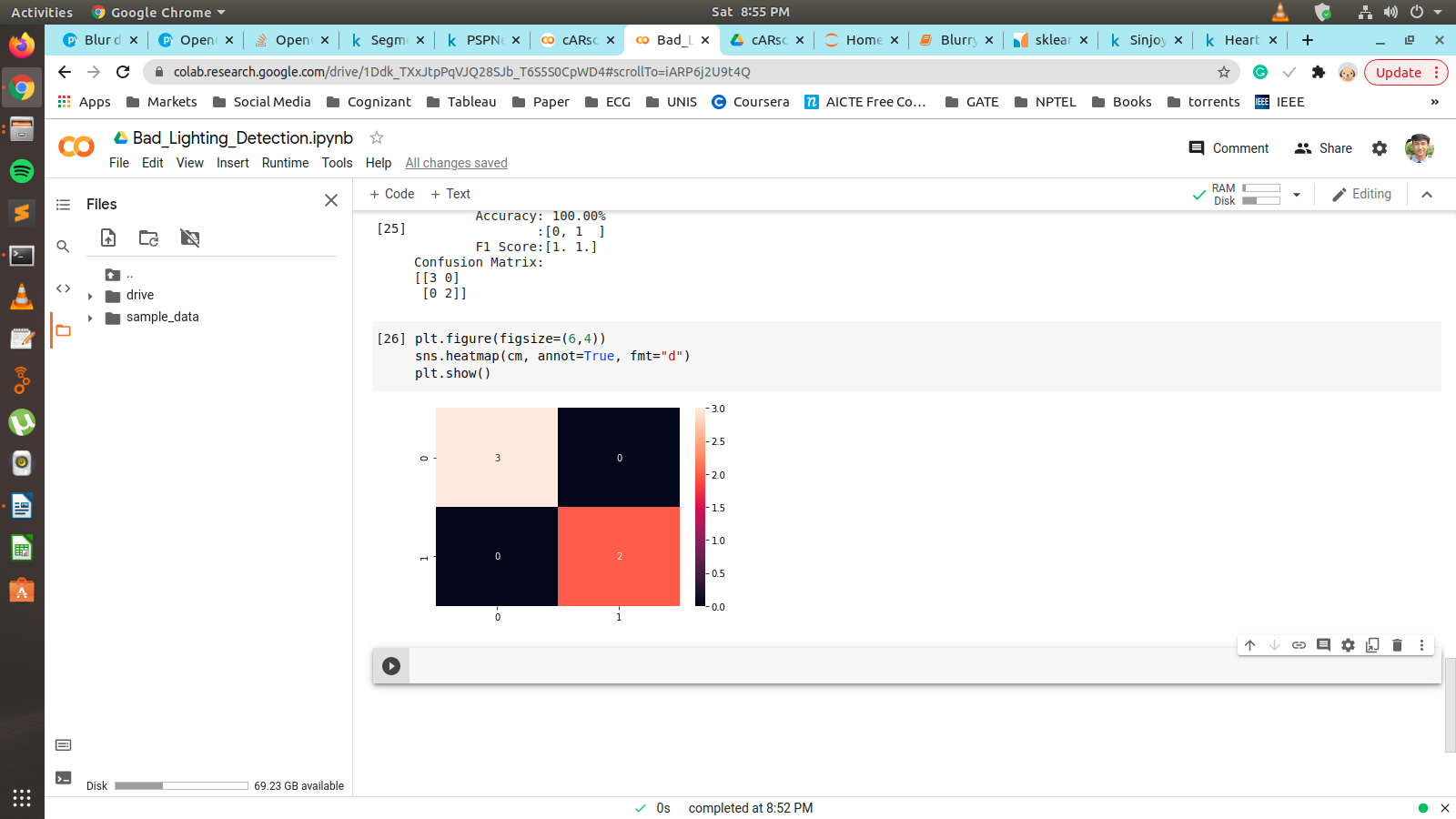
The evaluation of the model on the test set gives 100% accuracy and f1-score 1.0 on both classes.



Good Lighting image



Bad Lighting image



Confusion matrix

**Further work:**

Robustness of the model can be verified using a larger dataset and further hyperparameter tuning such max\_depth can be done if necessary.

1. **Semantic segmentation for Background removal / replacement**

For the task of background removal / replacement, I have used a deep learning technique (U-Net) for the semantic segmentation of the input image to predict the foreground masks. These masks are then used for the background removal / replacement step.

Steps / pseudo code is as follows:

* + 1. Generate the ground truth masks for the input images
    2. Read the images and masks and resize the images to (128, 128)
    3. Normalize the images to (0,1)
    4. Split into train / test sets (70-30 split)
    5. Train a U-Net model to perform pixel-wise classification of foreground / background pixels for the semantic segmentation task.
    6. Evaluate the model on performance metrics such as IoU, Dice Co-efficient and accuracy.
    7. Filter the foreground based on the predicted semantic masks

**Output:**

The dataset is created by generating the masks JSONs using VGG Image Annotator tool. The JSONs used to generate the actual masks using Python code. The output images are stored in the ‘images’ folder in the respective sub-folders.

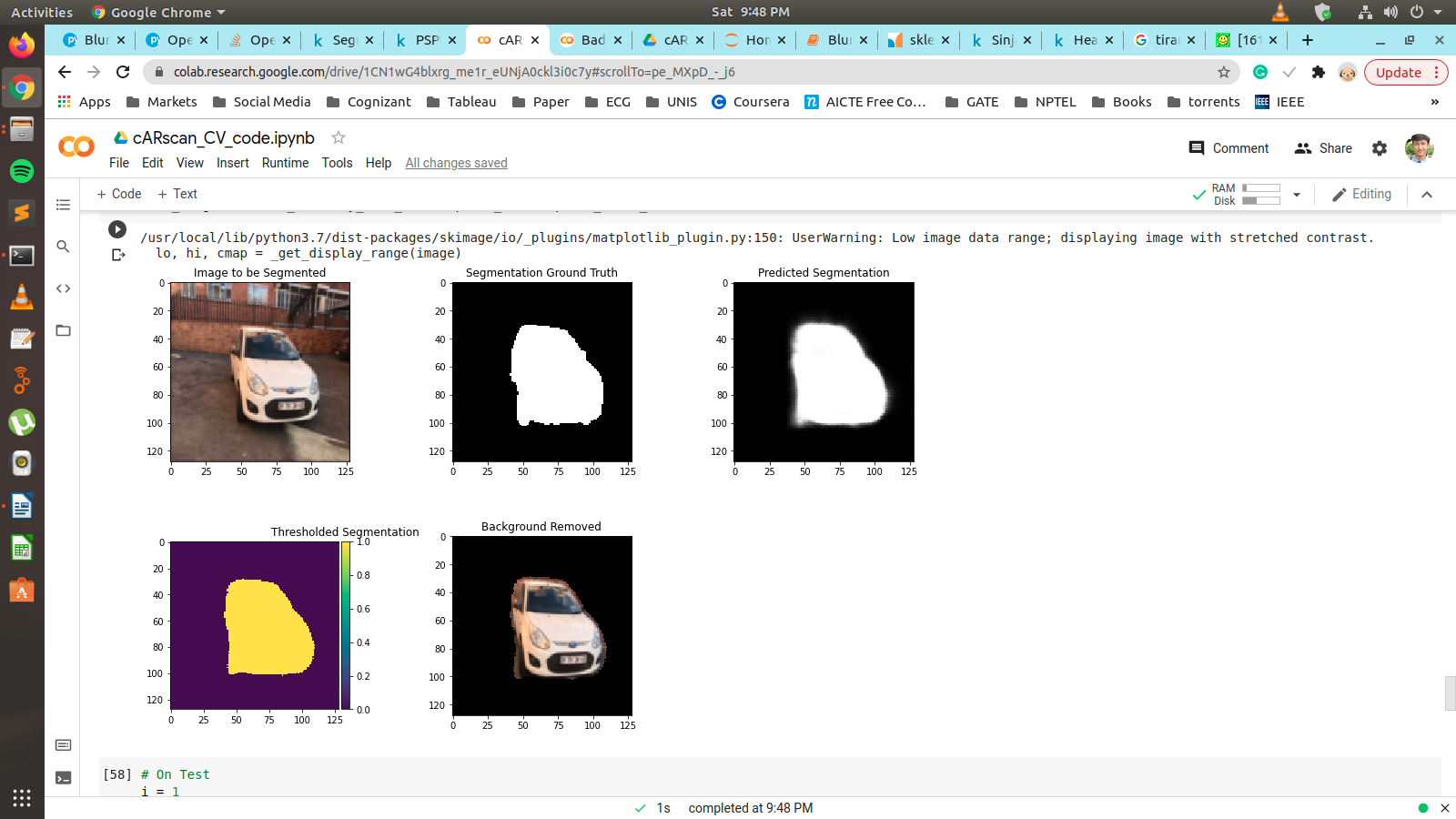
The U-Net model is evaluated on the test set and gives the following metrics:

- loss (binary\_crossentropy): 0.0510

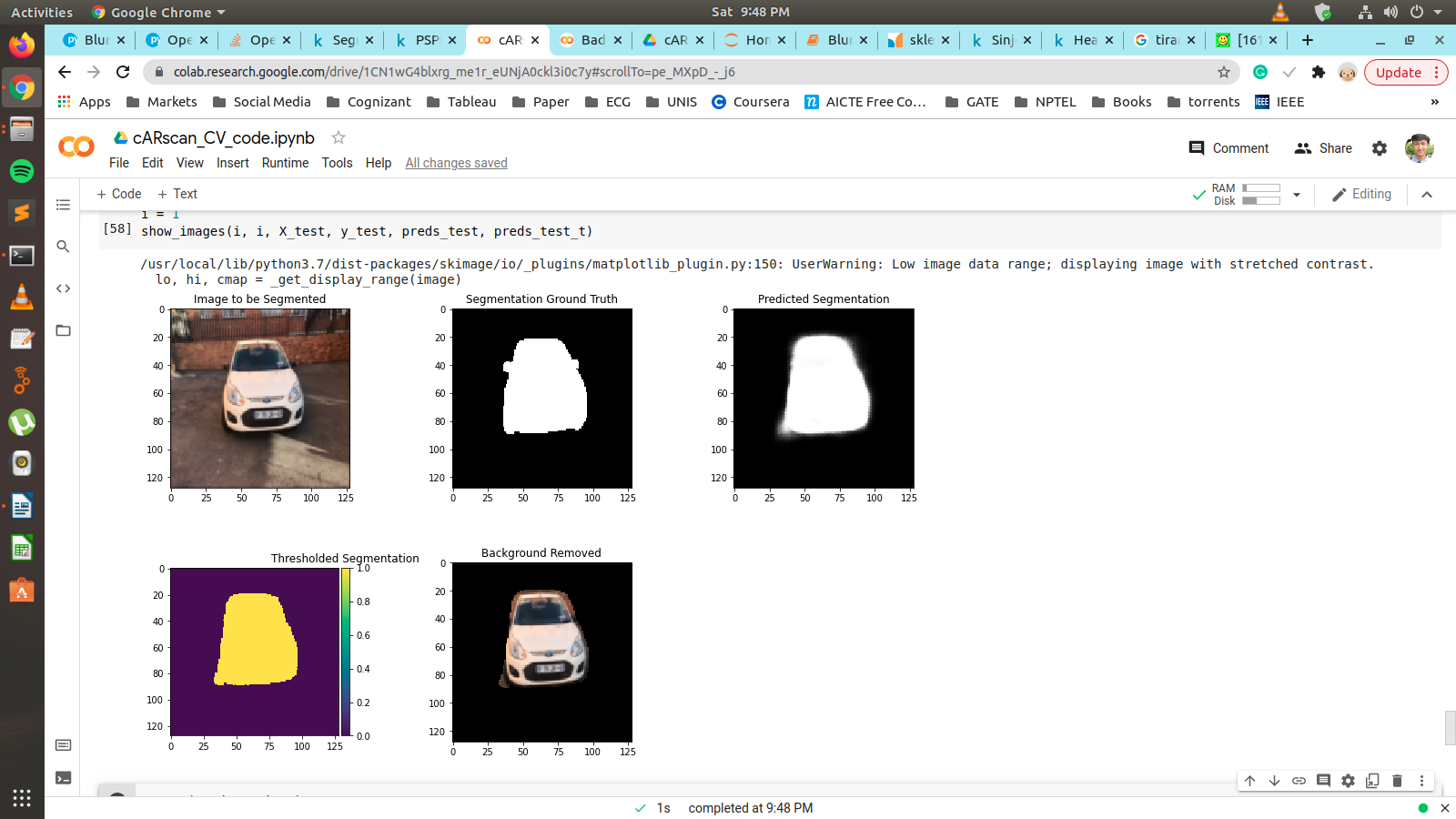
- iou\_coef: 0.8765

- dice\_coef: 0.9342

- accuracy (pixel-wise): 0.9787



Train set image



Test set image

**Further improvement:**

Deep learning models generally perform better on larger datasets. Having said that, we can explore collect more data or try data augmentation techniques to improve the dataset.

Further, different loss functions like Dice loss, IoU loss, Focal loss and Jaccard’s loss can be used to optimize the U-Net model.

Lastly, state-of-the-art bigger and deeper models such as the Tiramisu model using the Fully-Connected DenseNets and having 100 layers can also be tried out.

**Note:** Here, the images are resized to 128x128 pixels to run on the classic U-Net model. Custom U-Net maybe designed to run directly on input image sizes for better segmentation.

**Time to process the images:**

All the above models take less than 100ms to predict the output images.

**How can we run this on mobile devices?**

C++ can be used for implementing on mobile app. The models can be pre-trained in Python and saved as JSON or h5 files which can be imported in C++ for prediction task in the mobile app.

A simple pipeline for running on mobile devices:

1. Capture image using camera on device.

2. Read the image and pre-process (normalization) as needed.

3. Run the already stored model on the pre-processed image.

4. Display the output

5. Store the output logs on device for future debugging.

**Would you able to write or convert the same code into C++ if needed?**

Yes

**Are you able to write custom code for blur/light models without OpenCV?**

Yes