Report

This Repport is better viewed as a notebook

Time spent on code and report

About 35h

Hardware used

- Everything was done on my Linux PopOS 20.04 laptop with a quadro T2000 GPU

Test image and Results

- Some test images (and video) have been collected and are located in **TestImages**
- Both computer vision and DeepLearning approaches contain result images on this nano dataset

Computer vision result :

Deep Learning result :

Computer vision folder contains also a result video :

```
%%HTML
<video width="800" height="600" controls>
  <source src="./02_AutoGrabCut_Cpp/TestVideoCppResult/video.mp4" type="video/mp4">
</video>
```

```
<IPython.core.display.HTML object>
```

Traditional Computer Vision Approach

Result folder :

```
#!pip install ipyplot
import ipyplot

from glob import glob
listofImageNames = glob('02_AutoGrabCut_Cpp/TestImagesCppResults/*.jpg', recursive=True)

ipyplot.plot_images(listofImageNames, img_width=200)
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

Build and Run instructions:

Please checkout the README in AutoGrabCut_Cpp folder 02_AutoGrabCut_Cpp

Approach :

Idea & Assumptions :

- Sky region are low entropy areas (no much edge)

- Sky in located in the top 33% percent of the image
- There is a strong separation between the sky and the rest of the image : there is an edge

The approach is based on the GrabCut algorithm from OpenCV : "GrabCut": interactive foreground extraction using iterated graph cuts

It is usually used in an interactive manner (the user select the ROI). Here the area of interest in statically define as the top 33% of the image.

GrabCut from OpenCV uses 4 labels, here the top 33% of the image is labelled as "probable foreground" and the rest of the image as "probable background".

A graph minCut then occurs to optimize a split between foreground / background, hopefully sky and the rest of the image

Edges are not Sky :

- In order to help the grabCut algorithm, edges are detected and removed from the initialization. The initialization mask become "the top 33% of the image, but not where there is an edge"
- This assumption is also exploited as a final stage, to remove from the final binary mask every strong edges of the images.
- -> This assumption seems decent enough to be also exploited on deep learning mask, as a post processing step

Make it faster :

- GrabCut is slow. To speedup the process, the image is resized in two stages pyramid. The grabCut is run first on the smaller resolution to get a first mask approximated. This first result then feed a second GrabCut stage to obtain the final mask.
- In between steps, the intermediary mask is processed with the same "edge trick" and eroded as used as "annotation" to the second GrabCut pass.

Final stage :

- the output mask from the last GrabCut is resize to the full scale image resolution. The final mask is processed with morphological operations and blur for de-noising and smoothing.
- Once again the edges from image are exploited to exclude those area from the sky mask.

Subjective Performances and Obvious limitation :

- This is fast enough to process video of moderate resolution (720p) at near real time
- It work great in case of un-occluded sky and strong sky / land delimitation.
- However, it struggle when the conditions deviate too much from the assumptions.
- GrabCut algorithm has some internal random initialization and may give unstable result on static image, from run to run
- The 33% sky in image is a too strong assumptions, and must be manually adjusted in various cases.
- Vegetation branches is hard (you see the sky through branch). Thin structures, such as poles and wires, are hard
- Are cloud in the sky (class) ?

Objective Performance :

This remains to be done on annotated dataset. However, the repository contains a script to process the whole COCO dataset (or ADE20K) and save binary image masks.

Things I tried and did not work :

- Histogram back-projection : The idea was to get the color statistic of the segmented sky to further refine the mask. On video, it is unstable, and the mask tends to flash.

When it works, it performs great around contrasted objet, such as tree branch. It basically perform a color based segmentation. The idea was to only trust it on mask edge, in order to refine them.

Things I did not tried, but wish I did :

- Machine learning color segmentation (KNN, regression, ...) to categorize pixel based on color value.
- Machine learning pixel segmentation based on feature+color. (colors + texture)
- block based features classification
- Connected component post processing / filtering (remove region based on size...)
- Some kind of region growing (floodfill, watershed) from color priors (like photoshop magic wand)
- explore colorspace

Deep Learning Approach

```
import ipyplot

from glob import glob
listofImageNames = glob('01_DeepLearning/TestImagesUnetResults/ResNet18/*.jpg', recursive=

ipyplot.plot_images(listofImageNames, img_width=200)
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

Dataset Preparation

00_Dataset contains scripts and notebook to generate binary mask images from Coco and ADE20K dataset

both dataset mask are exported in a single folder (one per dataset) along with link to rgb images

COCO

A pure python script read the annotation, get the proper labels for a given image, and save the mask accordingly

ADE20K

A bit simpler than coco, as RGB color value in annotation images encode the class.

Build and Run instructions

Setup conda :

```
conda env create -f environment.yml
```

in case it doesn't work, checkout full environmentWithVersion.yml

Project

- A Unet network is trained on ADE20k dataset
- Training part and evaluation part are in distinct notebooks, checkout those for more information
- Result are tested on coco (part of it)

./01_DeepLearning/TrainUnet.ipynb

./01_DeepLearning/EvaluateModel.ipynb

Objective Performance :

Evaluation is on part of the coco validation dataset (trained on ADE20K)

ResNet18 training :

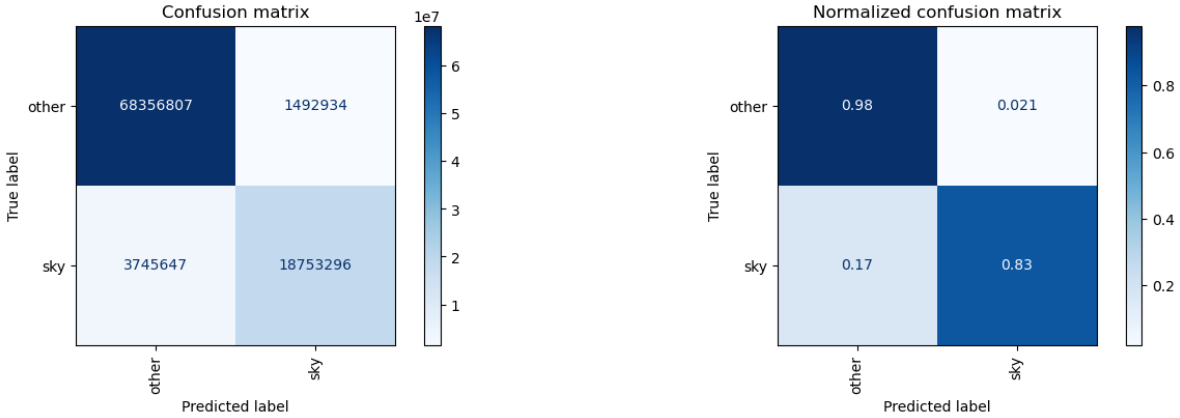

Figure 1: ResNet18

ResNet34 training :

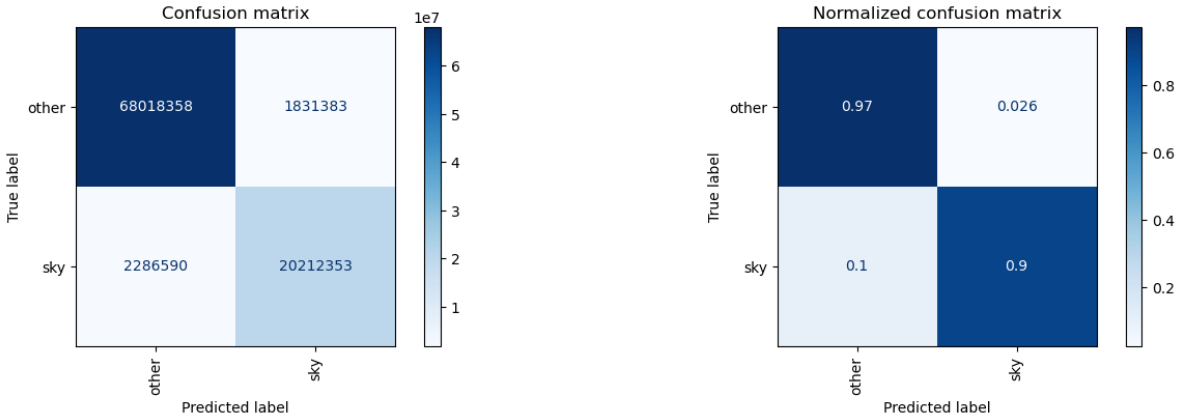

Figure 2: ResNet34

–> Both network performs about the same here, Resnet34 seems slightly better on Sky

Subjective Performances :

- On the test images, resnet18 Unet training seems to perform more consistently compared to resnet34. This is unexpected and does not match the objective performance measures.

Resnet18: Resnet34:

Results discussion

- Trainings converges toward the objectives, as shown by the confusion matrix, and as expected. However, during test on real "big" images, results are a bit inconsistent, or even poor in the ResNet34 case. Additional tests need to be done, but it seems that generalization of the network could be improved.
- I trained the networks with progressive resizing. Great trick to train faster, and fine tune on larger input size.
- There is something to tune (or experiment) between the scaling of the train images and the scaling done in the augmentation to create the batches. (in relationship with the kind of image expected in the final application)
- The dataset (ADE20K) seems of quality for the task, and should be ok to process the test images. However image resolution are on the lower side compared to test images.
- I noted some poor annotations showing up when displaying sample with the "worst loss". Some training images may need to be removed.
- Annotations on ADE20k seems to be of good quality. However contours are a bit rough, and annotations on hard cases (tree branch, thin structures) may be poor.

Some thoughts :

- The discrepancy between the Resnet18 and ResNet34 is bothering me. Ideally, I would retrain the ResNet34 in the same exact conditions as ResNet18 to be sure. Check the progressive resizing.
- Processing the image at different scale (on the mini dataset) give different results : what is the optimal resolution in this case?. Does DeepLab architecture could perform better with multi-scale featues?
- There is no pre processing of the image and no post processing of the mask

Things I wish I did :

- Test an 'off the shelf' segmentation network as a benchmark
- Test DeeplabV3 architecture on this problem
- Test the deeplab tflite model present in the GoPro App
- Quantize the model to test it on Android (ONNX, tflite)
- My GPU has limitations, better rent some remote compute next time.

Future improvements :

- Mixing the Computer vision and deep learning approaches could lead to better and faster results.
- Pre-processing of image and post-processing of mask in case of deep learning to clean up the results
- There is no safe guards, any kind of image can be processed, any type of mask can be accepted. There is cases where we should not even try to run the segmentation depending on the type of scene (underwater, night time, …).
- Video and temporal consistency could be considered in case of video processing
- User interaction has not been considered
- At the end of the day, the binary mask is not what makes the User Experience. Imperfections can be hidden or invisible depending on the expected application.