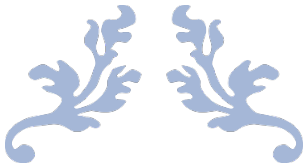

COMPUTER VISION

PROJECT

FEBRUARY 16, 2021

BUZATU RARES TUDOR
STUPARU MARIA ANDREEA
Automatic Control and Computer Science, Bucharest

Table of Contents

Abstract

The purpose of this paper is to explain in detail the implementation of the project for Computer Vision. The purpose of this project is to implement some vision algorithm(s) on a given video, the choice of the algorithm being determined by us. That being said, we chose to implement an image segmentation with multiple classes in Python using machine learning and then try to expend and particularize this segmentation on the provided video.

Introduction

Image segmentation represents the method in which an image is split into several classes representing different kind of objects / entities. This procedure is done nowadays using machine learning and more specifically deep neural networks. In this project we implement this procedure by machine learning, more specifically with an encoder-decoder U-Net network. After the segmentation we want to get the depth of objects on our segmented video and also to try to detect objects from the segmentation labels and also track them.

Workflow

In the section below, we explain in detail the steps that we did in order to implement the ideas above. The Steps are:

1. Extract the frames
2. Manually label the frames and augment them
3. Train the network
4. Fed all the frames on the network and reconstruct the video
5. Fed all the frames through the depth estimation network and reconstruct the video
6. Identify the blobs of segmentation from the frames
7. Track the blobs of segmentation and reconstruct the video

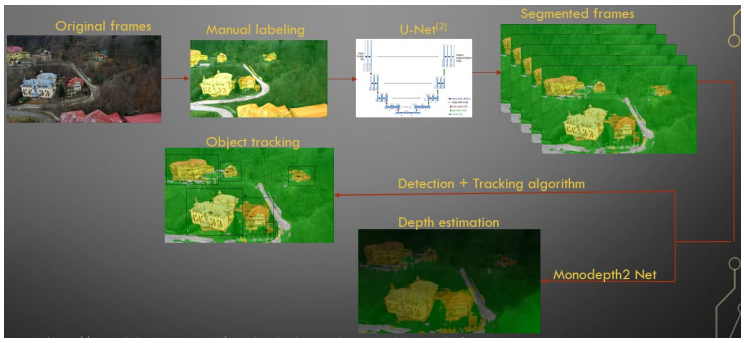

Figure 1: Workflow

Extract the frames

For this step, we have extracted all the frames from the video and then chose 36 frames that in our opinion captured the most elements of the video. The resolution of the frames was 1088x1920, so

a resize to 1088x1920 was necessary. The initial resolution probably came from a 4K video that was downscale by 2.

Labeling and augmentation

For the labeling we used the frameSeg.py [1] software provided by the professor and split the pixels into 4 classes: vegetation, residential, roads and cars. The software added a padding of 70 pixels around the frames, so a crop was necessary in order to get the frames back to the original resolution. To get more frames for the training, we also augmented the segmented frames by cropping them and flipping them. We obtain a total of 100 segmented frames. Here is an example of one of these frames:

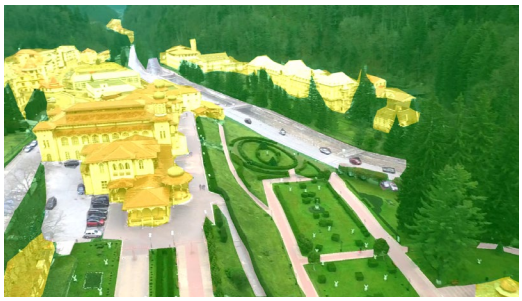

Figure 2: Segmented frame

Network training

The U-Net network that we used is a pre-trained network also provided by the professor. The network is called SaveUAV [2]. The network was pretrained on frames with 12 classes, so the output shape was (12,1). We modified this output by changing the last layer to a layer of 2D convolution with output shape equal to (4,1). We trained multiple configurations of the training involving the learning rate, the number of epochs, the layers state, but the bet configuration is this one:

1. 15 epochs with the learning rate of 0.01 and all layers but the last one frozen.
2. 15 more epochs with the learning rate of 0.0001 and all layers unfrozen.

The frames were split in an 80 / 20 proportion of training / validation. Here is the plot of loss function and the dice_coeff functions values:

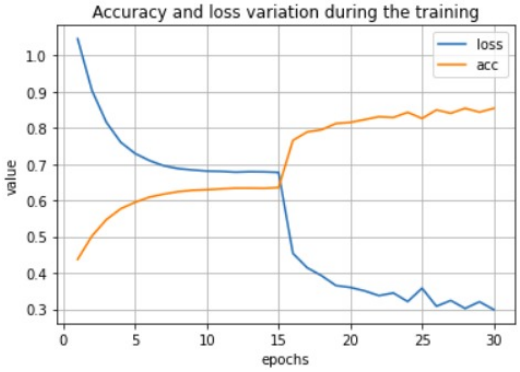

Figure 3: Loss and Accuracy values

Because the network was too big (2 million parameters) we had to use Google Colab [3] in order to run the training. Our PCs were not able to handle it and the RAM memory was exceeded quickly.

Entire video testing

In order to segment the whole video, we took all the frames from it and fed them in batches of 5 to the network. The output of the network is composed of the labels needed for the segmentation, so with them we were able to compute the visually segmentation. Here is an example of a generated segmented frame:

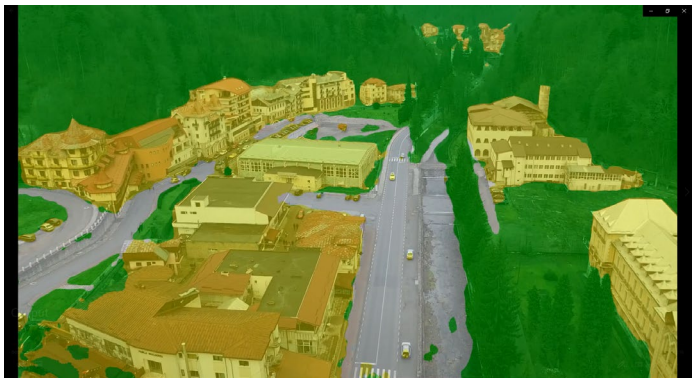

Figure 4: Generated segmented frame

As you can see, the residential areas, the vegetation and the roads are correctly segmented, but the cars are not found because of their small size, the inconsistency in our labeling and the disproportion of the classes' size.

Depth Estimation

For the depth estimation we used an already implemented network named Monodepth2 Net. This network uses 2 networks: one U-Net for the depth estimation and one ResNet 18 for the object location. The network is self-supervised.

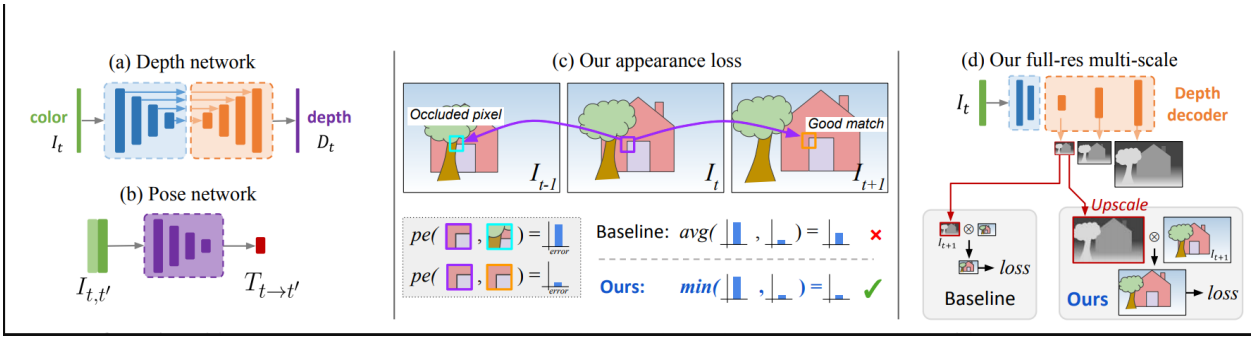

Figure 5: Monodepth2 architecture

This model was generating some indexes that represents the depth of the objects in the video, so with those indexes we were able to modify the colors of the objects in our segmented video in order to add some sense of depth. Here is an example of the frames:

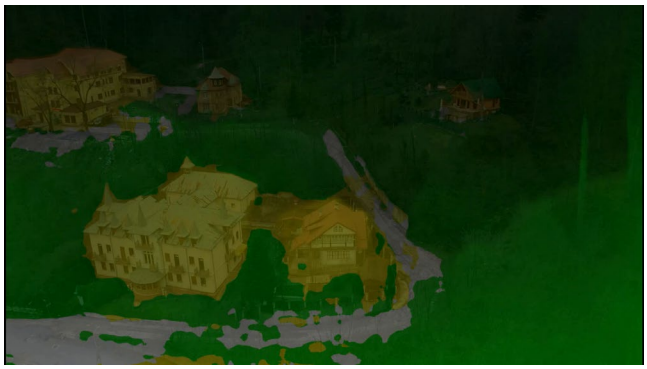

Figure 6: Depth estimated segmented frame

After each frame was combined with the indexes, we just paste together all the frames in order to get the video back.

Object detection

For the object detection we wanted to use just the previous segmentation and to not use any artificial intelligence methods in order to detect the residential areas from the video. The method we came with implies 2 basic Computer Vision techniques: erosion and dilation. We binarized the segmented frames, taking just the segmentation for the residential areas from the label's files. Here is an example of one binarized frame:

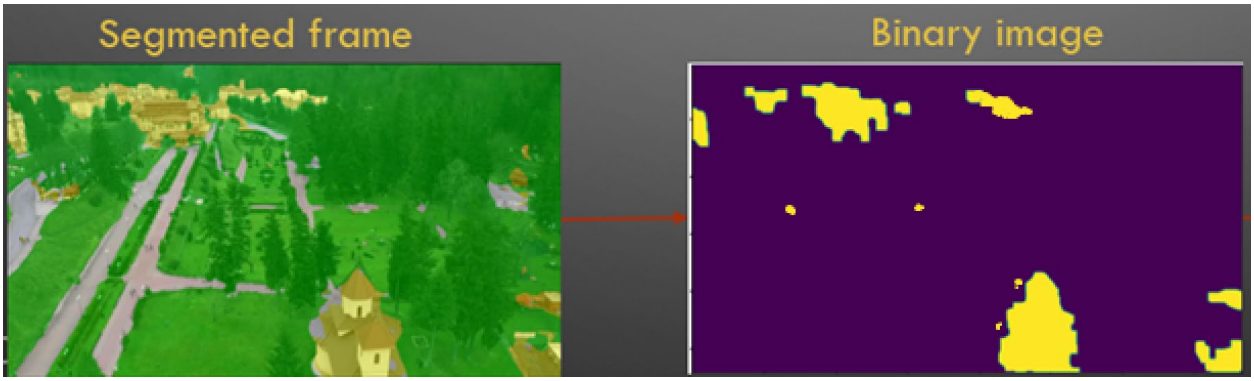

Figure 7: Binarized image

In order to get rid of the small areas and to split the large areas into small areas that represent individual houses we applied first an erosion with the kernel size of (50,50) and then a dilation with a kernel of (40,40) in order to get rid of the artifacts produced by erosion. After that, we took each created blob and draw for each one a square around them with the coordinates (X_{min}, Y_{min}), (X_{max}, Y_{max}) of the blob. Here is a frame with these squares drawn:

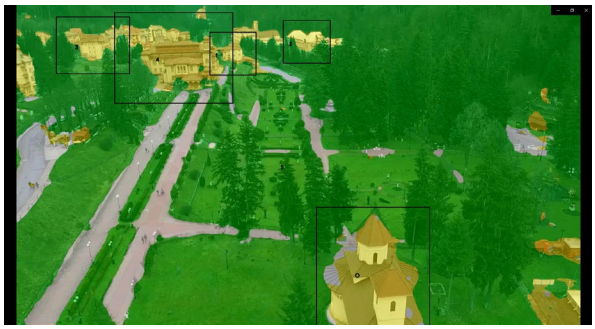

Figure 8: Boxes on residential areas

Object tracking

Now that we have the bounded boxes, in order to track the objects, we used the mass' center of each bounded box and associate each box with a number. If in the next frame we had a center around that previous position we gave that new object the same tag, else, we gave it a different tag. So, we basically computed the Euclidean distance between each new object and old object, took the minimum distance for each one and if the distance was smaller than a threshold, we gave it the same tag. The tags are the number present in the boxes from the figure above.

Conclusions

The accuracy of our results is pretty big, so the frames are really pleasant to watch, the objects in them are well defined and tracked, and the areas are segmented correctly. The depth gave our frames a nice touch and is the shapes look like they would look in reality. The only part that went not so well was the car segmentation, but that is reasonable having in mind the size of the cars in the video. An overall impression about this project is that we learned a lot of things about implementing machine learning and computer vision on videos and we consider these procedures to be extremely intuitive and useful. So, we really enjoyed what we did and what we learned.

References

[1] Frame Segmentation tool [https://github.com/onorabil/frameSegmentation]

[2] Alina Marcu, Vlad Licăreț, Dragoș Costea, Marius Leordeanu, *'Semi-supervised Segmentation of Aerial Videos with Iterative Label Propagation'*, [https://sites.google.com/site/aerialimageunderstanding/semantics-through-time-semi-supervised-segmentation-of-aerial-videos]

[3] C. Godard, O.M. Aodha, M. Firman, G. Brostow, "Digging Into Self-Supervised Monocular Depth Estimation", arXiv:1806.01260v4 [cs.CV] 17, Aug 2019, https://arxiv.org/pdf/1806.01260.pdf

[4] Marius Leordeanu, Slanic Moldova Video, https://drive.google.com/drive/folders/1P5JwPclNqvBhDRyIK_kAP_lx8gt-YXoW