Machine Vision Homework

Multi-Object Traffic Segmentation

Igor Racca

K45DZH

Budapest University of Technology and Economics
Autonomous Vehicles Control Engineering MSc

Budapest University of Technology and Economics
Autonomous Vehicles Control Engineering MSc

INTRODUCTION

PROJECT

This subject of this homework image segmentation using neural networks. More specifically, in this project, the goal is to perform image segmentation on multi traffic objects.

TASK

For this task, the neural network used is the ENet [1]. This neural network model will be described in more detail further. The source-code implemented takes traffic images as input, and using deep learning, it performs the instance segmentation for several objects, the classes will also be more detailed in the next chapter.

DEVELOPMENT

ENVIRONMENT

The development was made on Linux environment (Ubuntu 20.04.1), due to better compatibility with libraries and packages.The desktop graphical user interface (GUI) Anaconda Navigator was used to launch and manage conda packages. Through the anaconda navigator, the application Jupyter Notebook was used, where the code can be easily run, which makes the development more efficient, organized, as well as good for image visualizations. The programming language utilized was Python 3.8.5, together with OpenCV 3.4.8 which stands for Open Source Computer Vision Library and is an open source computer vision and machine learning software library. Additional libraries and packages such as NumPy, imutils and Matplotlib were also used.

DATA

The dataset used for training and validating the neural network is the Cityscapes Dataset [2], which is dataset consisting of diverse urban street scenes across 50 different cities at varying times of the year as well as ground truths for several vision tasks including semantic segmentation, instance level segmentation (TODO), and stereo pair disparity inference, it provides dense pixel level annotations for 5000 images at 1024 * 2048 resolution pre-split into training (2975), validation (500) and test (1525) sets. Label annotations for segmentation tasks span across 30+ classes commonly encountered during driving scene perception.

The classes on this dataset that the neural network model was trained are the following:

- Unlabeled (i.e., background)
- Road
- Sidewalk
- Building
- Wall
- Fence

- Pole
- TrafficLight
- TrafficSign
- Vegetation
- Terrain
- Sky
- Person
- Rider
- Car
- Truck
- Bus
- Train
- Motorcycle
- Bicycle

On the web, it is possible to find a pre-trained model of the ENet trained in another dataset (CamVid).

NEURAL NETWORK

As mentioned before, the neural network considered in this project is the ENet, this neural network is faster and requires less capacity than a SegNet, for example.

The ENet semantic segmentation architecture is shown in Figure 1.

Name	Type	Output size
initial		16 × 256 × 256
bottleneck1.0	downsampling	64 × 128 × 128
4× bottleneck1.x		64 × 128 × 128
bottleneck2.0	downsampling	128 × 64 × 64
bottleneck2.1		128 × 64 × 64
bottleneck2.2	dilated 2	128 × 64 × 64
bottleneck2.3	asymmetric 5	128 × 64 × 64
bottleneck2.4	dilated 4	128 × 64 × 64
bottleneck2.5		128 × 64 × 64
bottleneck2.6	dilated 8	128 × 64 × 64
bottleneck2.7	asymmetric 5	128 × 64 × 64
bottleneck2.8	dilated 16	128 × 64 × 64
Repeat section 2, without bottleneck2.0		
bottleneck4.0	upsampling	64 × 128 × 128
bottleneck4.1		64 × 128 × 128
bottleneck4.2		64 × 128 × 128
bottleneck5.0	upsampling	16 × 256 × 256
bottleneck5.1		16 × 256 × 256
fullconv		$C \times 512 \times 512$

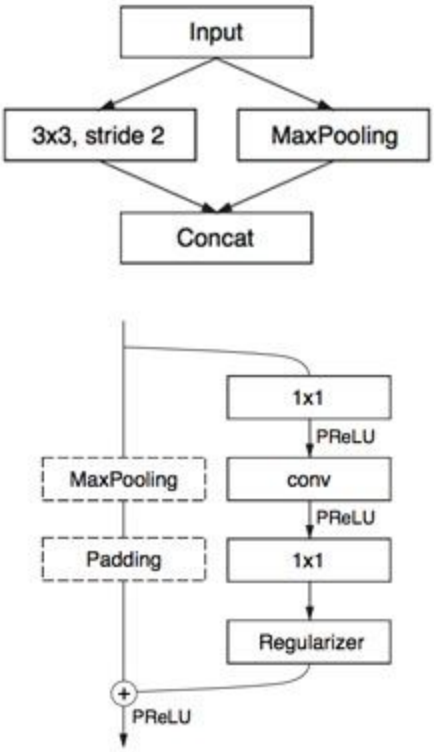

Figure 1. - ENet structure

As said before, there are pre trained models of th ENet available in their repository, for this project, a pre-trained model on the CityScapes dataset was used. In the next subsection, it is shown in the source code how the model is loaded and used to perform the segmentation on the image inputs.

INPUT AND OUTPUTS

The project contains a folder called *imageInputs* where the input images are stored. When the script is run, all the imagem within this folder are loaded and processed.

The script outputs all the images (both the input and the output, which is the image containing the segmentation filter), as well as the legend colors for each class. A transparent mask is applied so it is possible to see both the original image below the

segmentation mask.

SOURCE CODE

For the sake of simplicity and better understanding, the source code file will be explained as it is structured in the IPYNB file, and not the entire code will be illustrated here. The IPYNB extension is a notebook document which can be used in Jupyter Notebook, it offers an interactive computational environment. The source code is explained into the cells, where which section will be described in detail.

In the 1st cell, the required libraries are imported (Figure 2.)

```
In [1]: import numpy as np
        import imutils
        import time
        import cv2

        from os import walk

        %matplotlib inline
        from matplotlib import pyplot as plt
```

Figure 2.
1st fragment of the code

In the 2rd cell, the parameters are defined, the path for the classes, colors, model of the neural network, and constants. (Figure 3.)

```
In [2]: params = {
            'model' : 'enet-cityscapes/enet-model.net',
            'classes': 'enet-cityscapes/enet-classes.txt',
            'colors': 'enet-cityscapes/enet-colors.txt',
            'imagesPath': 'imagesInput'
        }
        WITDH = 500
```

Figure 3.
2nd fragment of the code

In the 3rd cell, the color and image paths are processed and added to a list, and the colors regarding the segmentation classes are set. (Figure 3.)

In [3]:
```
CLASSES = open(params["classes"]).read().strip().split("\n")

COLORS = open(params["colors"]).read().strip().split("\n")
COLORS = [np.array(c.split(",")).astype("int") for c in COLORS]
COLORS = np.array(COLORS, dtype="uint8")
```

In [4]:
```
# initialize the legend visualization
legend = np.zeros(((len(CLASSES) * 25) + 25, 300, 3), dtype="uint8")

# loop over the class names and colors colors
for (i, (className, color)) in enumerate(zip(CLASSES, COLORS)):
    # draw the class name and thee color on the legend
    color = [int(c) for c in color]
    cv2.putText(legend, className, (5, (i * 25) + 17),cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)
    cv2.rectangle(legend, (100, (i * 25)), (300, (i * 25) + 25), tuple(color), -1)
```

Figure 4.
3nd and 4th fragment of the code

The 5rd cell is where all the processing happens. It will be splitted in two figures to provide better visualization. The first part of this fragment is regarding the neural network, where the pretrained model is loaded, the images are read, and the lists of input and output images to be displayed later are initialized. It's important to notice that a blob is created for each image, and after that, it is forwarded through the ENet. (Figure 5.)

In [5]:
```
# load ENet pretrained model
net = cv2.dnn.readNet(params["model"])
print("ENet model suceffuly loaded.")

files = []
for (dirpath, dirnames, filenames) in walk(params["imagesPath"]):
    files.extend(filenames)
    break

# initialize images list
imagesInput = []
imagesOutput = []

for f in files:
    # load the input image, resize it, and construct a blob
    image = cv2.imread(params["imagesPath"] + '/' + f)
    imagesInput.append(image)
    image = imutils.resize(image, width=WITDH)
    blob = cv2.dnn.blobFromImage(image, 1 / 255.0, (1024, 512), 0, swapRB=True, crop=False)

    # forward into the ENet
    net.setInput(blob)
    start = time.time()
    output = net.forward()
    end = time.time()

    # show the amount of time inference took
    print("ENet output took {:.4f} seconds".format(end - start))
```

Figure 5.
5th fragment of the code - part 1

After the neural network gives the output, the visualization will be processed. In the next fragment of the code (Figure 6.), the the spatial dimensions of the output volume are defined, and the total number of classes with the dimensions of the mask are inferred via the shape of the output array, after that, the argmax gets the most probably class and its mapped with the ID of the corresponding color in the segmentation. Then the NumPy array is used to index to the corresponding visualization color for each pixel. The transparency of the mask is set to 60% so it is possible to see the original image below the segmentation mask. The output mask is added to the output list and the time for each image taken for being inferred by the neural network is printed out.

```
    # show the amount of time inference took
    print("ENet output took {:.4f} seconds".format(end - start))

    # infer the total number of classes along with the spatial dimensions
    # of the mask image via the shape of the output array
    (numClasses, height, width) = output.shape[1:4]

    # get the most probably class
    classMap = np.argmax(output[0], axis=0)

    # map each of the class IDs to its corresponding color
    mask = COLORS[classMap]

    # resize the mask and class map
    mask = cv2.resize(mask, (image.shape[1], image.shape[0]), interpolation=cv2.INTER_NEAREST)
    classMap = cv2.resize(classMap, (image.shape[1], image.shape[0]), interpolation=cv2.INTER_NEAREST)

    # add transparency mask
    output = ((0.4 * image) + (0.6 * mask)).astype("uint8")

    imagesOutput.append(output)
```

```
ENet model suceffuly loaded.
ENet output took 0.6487 seconds
ENet output took 0.3934 seconds
ENet output took 0.3465 seconds
ENet output took 0.3727 seconds
ENet output took 0.3551 seconds
ENet output took 0.3378 seconds
ENet output took 0.3519 seconds
```

Figure 6.
5th fragment of the code - part 2

In the last cell, the list of the images are shown. First the original input image, then the output image with segmentation mask and finally the color that corresponds to each class, so it is possible to visualize. (Figure 7.)

In [6]:
```
# display input, output and legend
for image, output in zip(imagesInput, imagesOutput):
    fig, (ax1, ax2) = plt.subplots(nrows=2, figsize=(13,15))
    # input image
    img_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    ax1.imshow(img_rgb, aspect='auto')
    ax1.set_title('input',fontsize= 20)
    ax1.set(xticklabels=[])
    ax1.set(xlabel=None)
    ax1.set(yticklabels=[])
    ax1.set(ylabel=None)
    ax1.tick_params(left=False)
    ax1.tick_params(bottom=False)
    # output image
    ax2.imshow(output,aspect='auto')
    ax2.set_title('output',fontsize= 20)
    ax2.set(xticklabels=[])
    ax2.set(xlabel=None)
    ax2.set(yticklabels=[])
    ax2.set(ylabel=None)
    ax2.tick_params(left=False)
    ax2.tick_params(bottom=False)

    plt.tight_layout()
    plt.show()

    # legend
    fig, ax1 = plt.subplots(nrows=1, figsize=(13,15))

    ax1.imshow(legend, aspect='auto')
    ax1.set(yticklabels=[])
    ax1.set(ylabel=None)
    ax1.tick_params(left=False)
    plt.show()
```

Figure 7.
6th fragment of the code

VISUALIZATION

For each image in the input, their visualization will be shown as illustrated in Figure 8 and 9.

Figure 8.
Original image and image with the segmentation

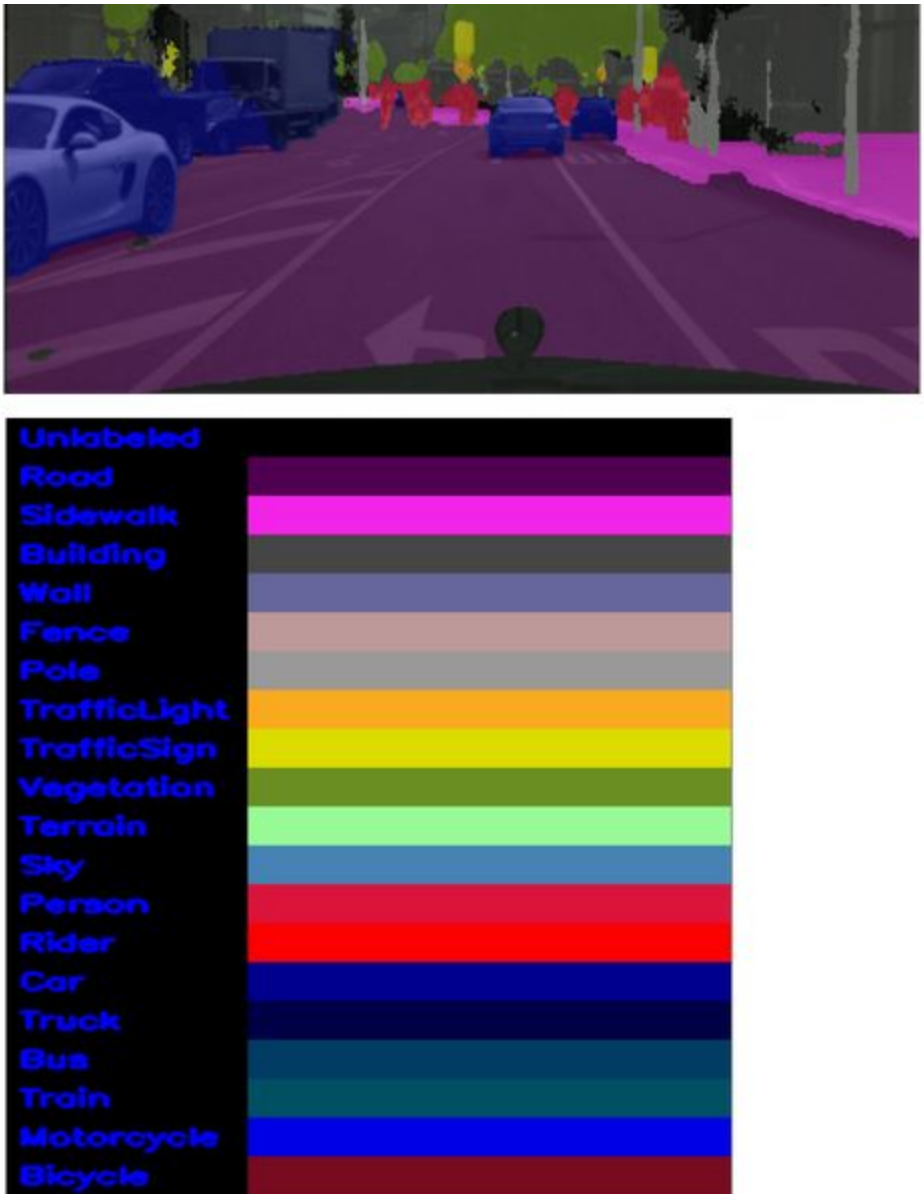

Figure 9.
Segmented image and color legend corresponding to each class

CONCLUSION

It is possible to see that the ENet does not perform very well as the newest techniques. The work with this neural network was published in 2016 best computer vision and deep learning techniques were developed.

The same example can be performed with videos, however, as it can be seen, the time taken to infer each image in the neural network is still big, which makes it not applicable for real-time processing.This code uses the CPU to process, it is possible to configure the OpenCV to run with the GPU, yet, it does not as efficient as other techniques, such as Mask R-CNN and YOLO, which makes it not viable to applications that demand fast response.

REFERENCES

1. A Paszke, A Chaurasia, S Kim, E Culurciello “**ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation**”, *June 2016*
2. **Cityscapes Dataset** [Web]

https://github.com/mcordts/cityscapesScripts