Semantic Segmentation and Object Detection in a single pipeline using CNNs

Iyer Venkatesh

***Abstract*— Pixel-wise Semantic Segmentation and detection of objects in real time is of utmost importance in many applications. This project aims to train E-Net [1] network using deep learning library Tensorflow for Semantic Segmentation and detection of objects in an image using OpenCV. Based on the Semantic Segmentation prediction, Object Detection of dynamic classes are done. These dynamic detected classes are passed to a classification network and the prediction of this network gives a confirmation whether the prediction of both the deep networks matches or not. This project is an experiment that follows a different pipeline that does Object Detection on segmented images while the general pipeline is about detecting objects on raw images.**

I. Introduction

Deep learning has seen huge success lately in the areas of handwritten digit recognition, speech recognition and classifying images. Lately, there has been a consistent rise in pixel-wise semantic labeling due to the excessive usage of augmented reality wearable, home devices, and self-driving vehicles. Semantic Segmentation has been made possible due to the large image repositories and high-performance computing systems such as GPUs and large-scale distributed clusters. In order to do pixel-wise segmentation, network architectures like SegNet [2] or fully convolutional networks [3] have been proposed based on VGG16 [4] architecture. These networks come with a huge number of parameters and long inference time and hence become unusable for many mobile or battery powered applications. The E-Net network consists of a fast and compact encoder-decoder architecture optimized for fast inference and high accuracy. E-Net uses less time and less number of parameters compared to the state of art networks

Object Detection is probably the most profound aspect of computer vision due to the number of practical use cases. Object Detection has been widely used for face detection, vehicle detection, pedestrian counting, web images, security systems, and driver-less cars. Accurate deep learning algorithms and methods such as R-CNN [5], Fast R-CNN [6], Faster R-CNN [7], RetinaNet [8] have been brought to practice since the year 2012. The above-mentioned methods are highly accurate and quite fast but require a lot of mathematical and deep learning framework understanding. Before the inception of such complex yet faster architectures, identifying objects was made possible by the use of algorithms supported in OpenCV, a popular computer vision library.

The motivation behind this project is to utilize the already achieved localization of an object through Semantic Segmentation, detect the dynamic object and to classify the detected part using a classification network. Also, it is an experiment which uses less number of trainable parameters compared to an object detection network.

Section II is an overview about the workflow of the project, section III is an overview about the fundamentals of the project, section IV is about the process of Semantic Segmentation, section V is about the Object Detection of dynamic objects, section VI is about the classification of detected dynamic objects, sectionVII is about the classification results, section VIII is about future work of this project and section IX is a list of references used for the project.

II. Workflow

This section is an overview of the steps involved in this project. Raw images are passed through the Semantic Segmentation network which gives out the class predictions. On the predictions, dynamic objects are detected using contours. Using the contour co-ordinates, the dynamic objects are cropped out from the raw image and are classified using a classification network. Fig. 1 shows the workflow for the project

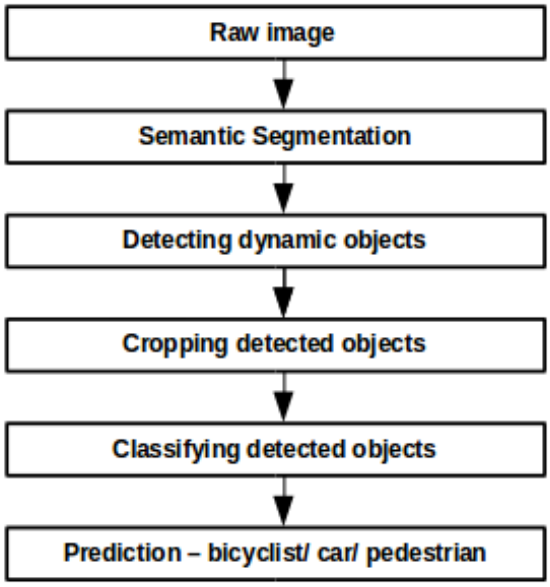

Fig. 1: Workflow of the project

III. Fundamentals

A. Dataset

The dataset used for this project is CamVid. The Cambridge-driving Labeled Video Database (CamVid) is a

dataset with a collection of videos with object class semantic labels and with complete metadata. This dataset provides ground truth labels that associate each pixel with one of the 11 classes that are the sky, building, pole, road-marking, road, pavement, tree, sign symbol, fence, car, pedestrian, bicyclist and unlabelled. This database addresses the need for experimental data to quantitatively evaluate emerging deep learning algorithms. The videos in this database are filmed using a fixed-position CCTV-style camera and the data was recorded from the perspective of a driving automobile. This dataset contains 367 (480x360) images for training and 101 (480x360) images for validation. The original CamVid dataset consists of 32 classes but this experiment is performed using the 11 classes version.

B. Tensorflow

Tensorflow deep learning library is an open source library offering tools for developing programs which work on different platforms like CPU and GPU. This library is created by Google and also it supports most of the applications in the area of machine learning and deep learning.

C. Keras

Keras is a high-level neural networks API which has the base of Python and is capable of running on top of Tensorflow, CNTK or Theano. Keras was developed with a focus on enabling faster experimentation. It is designed to support both convolutional networks and recurrent networks, as well as the combinations of both. Keras runs seamlessly on both CPU and GPU's.

D. OpenCV

Open Source Computer Vision Library (OpenCV) is an open source computer vision and machine learning software library. OpenCV provides a common infrastructure for computer vision applications to accelerate the use of machine learning algorithms in commercial products. OpenCV provides more than 2500 optimized algorithms for state-of-the-art computer vision and machine learning algorithms. As this library is enabled with OpenCL, it can take advantage of the hardware acceleration of underlying heterogeneous compute platforms.

IV. Pixel-wise Semantic Segmentation

E-Net architecture is used for pixel-wise Semantic Segmentation. This architecture consists of two blocks, they are the initial block and the bottleneck block.

A. Network Architecture

Convolution is done using 13, 3x3 filters with stride 2. This generates 13 feature maps. MaxPooling is performed on the input using 2x2 windows which generate 3 feature maps. In total, we get 16 feature maps using the convolution and MaxPooling on the inputs after concatenation. Fig.2 shows the initial block of E-Net architecture.

There are four types of bottleneck block that E-Net architecture uses, they are:

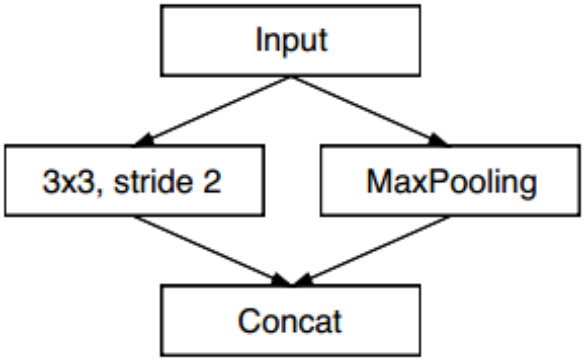

Fig. 2: E-Net initial block[1]

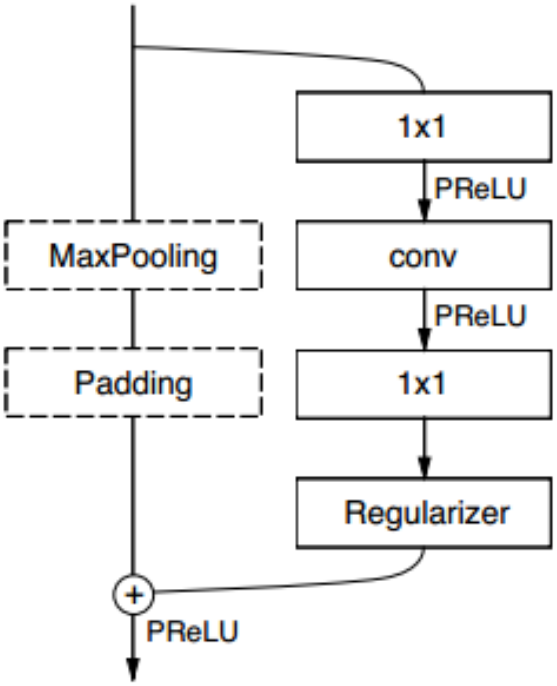

Fig. 3: Basic bottleneck block[1]

- Basic bottleneck block - A convolution of 1x1 is applied in order to reduce the dimensions. Then, a convolution of 3x3 and then again a 1x1 convolution for expansion.
- Downsampling bottleneck block - If the bottleneck is downsampling, a MaxPooling layer is added to the main branch. The 1x1 projection is replaced with a 2x2 convolution with stride 2. The second convolution of 2x2 is then applied. Again, a 1x1 convolution is applied for expansion. Zero padding in order to match the number of feature maps.
- Dilation bottleneck block - A 1x1 convolution is applied to reduce the dimensions. Dilated convolution of 3x3 and with a dilation rate. Then a convolution of 1x1 for expansion.
- Asymmetric bottleneck block - A 1x1 convolution is applied to reduce the dimensions. Then, a sequence of asymmetric convolutions of 1x5 and 5x1. In order to do

an expansion, we apply a 1x1 convolution.
Batch Normalization[9] and PreLu[10] activation is applied in between all convolutional layers. Fig. 3 shows the bottleneck block of E-Net architecture.

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

Fig. 4: E-Net Architecture

Fig. 4 shows the entire bottleneck architecture of E-Net. The initial stage consists of a single block as shown in Fig.1. Stage 1 consists of 5 bottleneck blocks, while stage 2 and 3 have the same structure with only one exception in stage 3, that is, it does not downsample the input at the beginning. The first 3 stages in the E-Net architecture are the encoders. Stages 4 and 5 work as the decoders. In the part of the decoder, the MaxPooling step is replaced with MaxUnpooing and the padding is replaced with spatial convolution without bias. Pooling indices were not used in the upsampling module as the initial block was operated on 3 channels of the output frame, while the final output has C feature maps according to the number of object classes. A bare full convolution was placed as the last module of the network due to which the decoder takes up more processing time.

B. Pixel-wise semantic segmentation predictions

Fig. 5 shows the results after training the network. The color encoding for the main classes are as follows: yellow - bicyclists, maroon - cars, green - pedestrians, blue - buildings

C. Mean IoU score and inference time

- Achieved mean IoU score of 42% after training the network for 300 epochs
- Inference time of less than a second per testing image

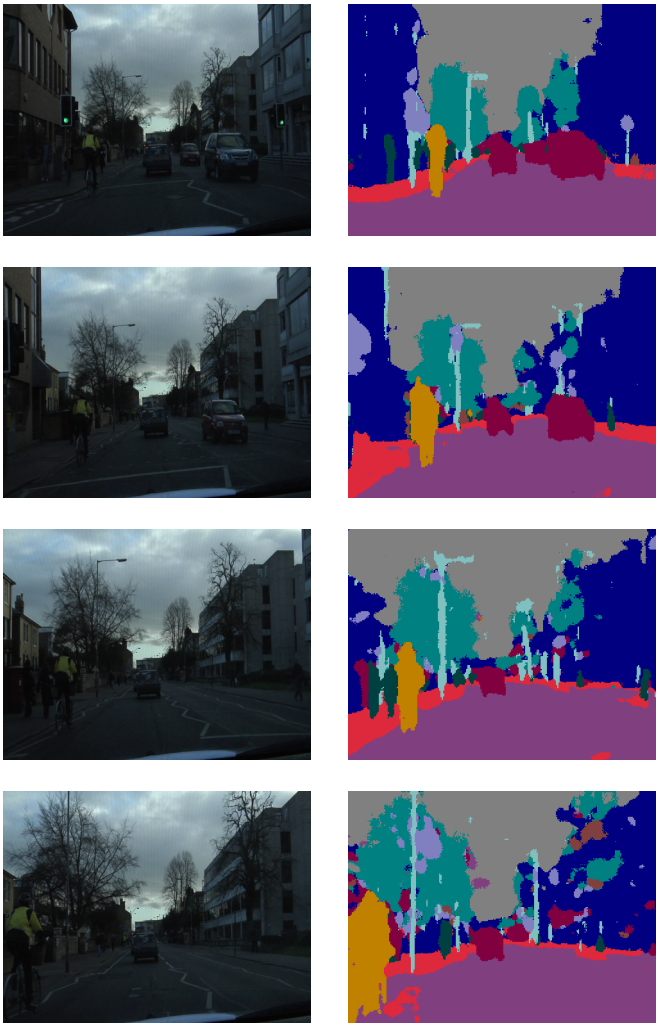

Fig. 5: E-Net predictions

D. Experiments using other networks

Other networks were also tried for Semantic Segmentation. AdapNet [11] and DeepLabV3 [12] were trained for 70 and 200 epochs respectively using pre-trained weights of ResNet [13]. Mean IoU score of 43% and 38% were achieved for both the networks. Results are shown in Fig.6. The left side of the figure is the ground truth and the right side is the prediction of the network after training.

The color encoding for main classes are as follows: light blue - bicyclists, violet - cars, dark green - pedestrians, red - buildings

V. Object Detection

Object Detection in computer vision is intensively used in the area of autonomous or driver-less cars. By using Object Detection, an autonomous car can perceive the world around it by creating a digital map using sensory devices like cameras, radars, and lasers. The task in this project is to detect dynamic objects in an image. Dynamic objects such as cars, bicyclists and pedestrians among the 11 classes are detected using OpenCV algorithms.

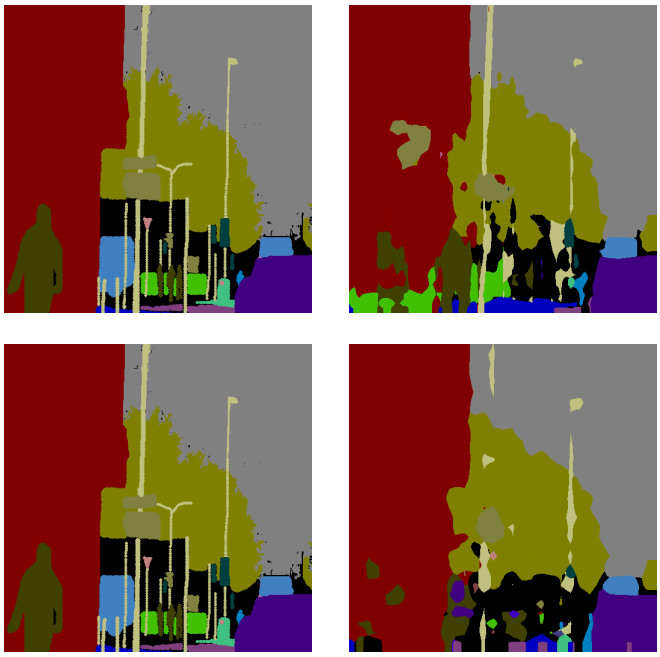

Fig. 6: Predictions of AdapNet (above) and DeepLabV3 (below) networks

A. *Plotting contours*

The initial step is to detect the contours of the dynamic objects in images. Contours can be explained as a curve joining all the continuous points (along with the boundary), having the same color or intensity. Contours are extensively used in the area of shape analysis, Object Detection and recognition. A mask is created wherever the color code of a class matches in the segmented predicted image. In order to detect and draw accurate contours, the segmented predicted image is converted to a binary image by applying a threshold to the image. As a result, a black and white image which depicts the white part as desired classes or the mask and the black part as the background or undesired classes is achieved. Also, a contour threshold area of 300 is set in order to ignore the detection of objects that are too small in the image. Fig. 7 shows the prediction and the plotted contours of required classes (car, pedestrian, and bicyclist). Fig. 8 shows the result of detecting all objects in an image, and without setting a contour threshold area.

B. *Crop*

The area of contours is cropped from the original image using the contour coordinates obtained from the mask. In order to maintain a balanced ratio of sizes in between a car, bicyclist, and pedestrian, all the cropped original images are re-sized to 64x64 size. These cropped and re-sized images of contours shall be used further as training images in a classification network. Fig. 9 shows the cropped and re-sized images.

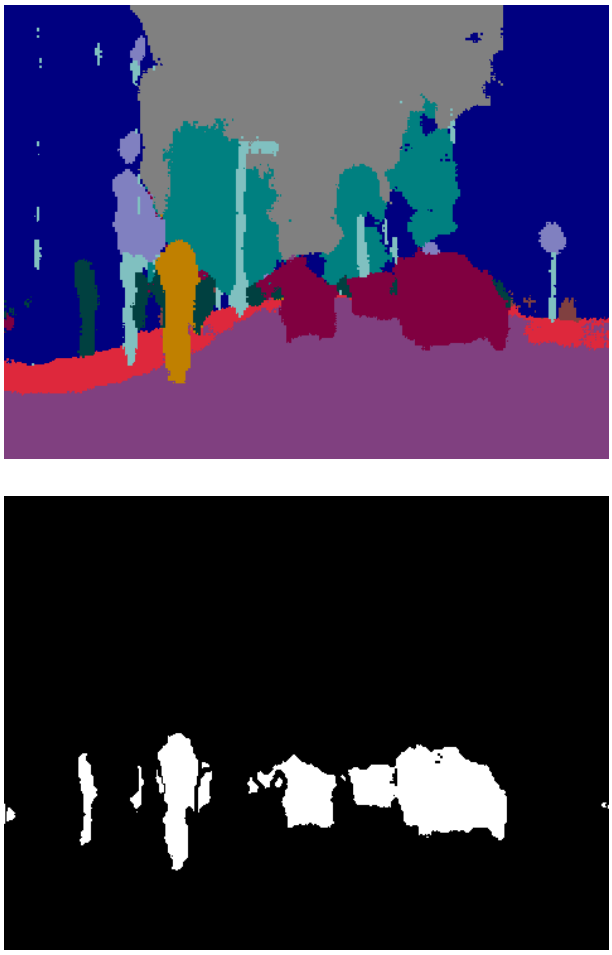

Fig. 7: Segmented image (above) and plotted contours (below)

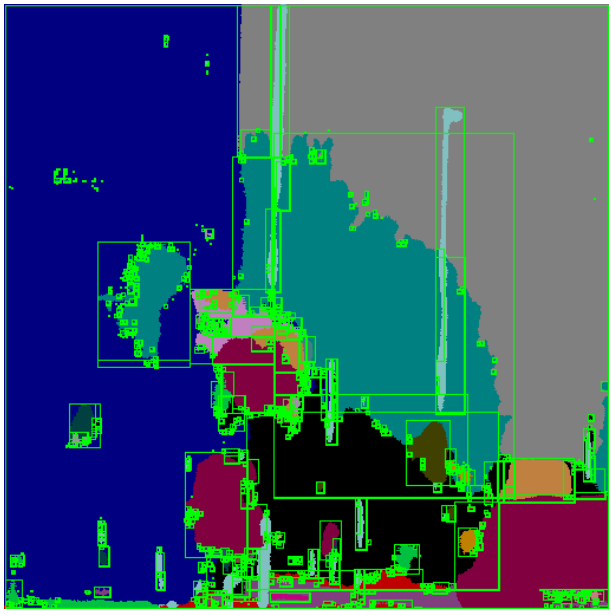

Fig. 8: Detection of all objects and without any contour threshold area

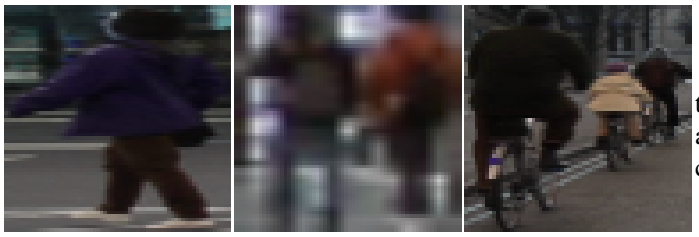

Fig. 9: Cropped and re-sized original images of size 64x64

VI. Classification

This final check is done in order to compare the results of both the deep networks. The data set consists of 250 (64x64) training images per class (pedestrian, bicyclist and car) and 60 validation images per class.

A. Network architecture

The cropped and re-sized images are categorized into three folders which are named as the labels i.e bicyclist, car and pedestrian. Fig. 10 shows the classification network architecture

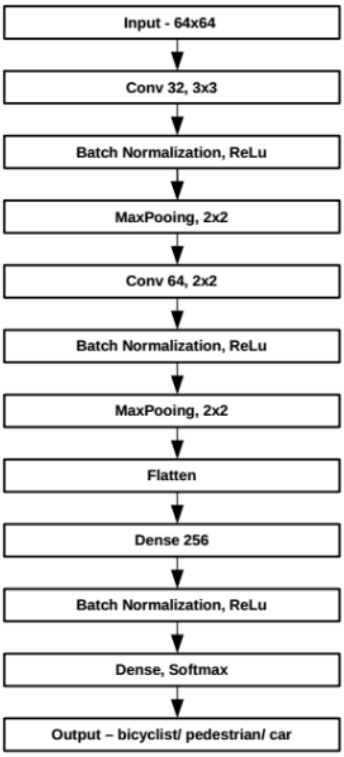

Fig. 10: Classification Architecture

The network architecture consists of two convolutional layers of 32, 3x3 and 64, 2x2 filters respectively. Batch Normalization, ReLu activation and Max Pooling of 2x2 window are done in between both the convolution layers. Lastly, a dense layer with the number of classes and softmax activation gives the prediction probabilities.

VII. Results

This section shows the results that are achieved after training the deep network for 50 epochs with batch size 5 and learning rate of 0.001. Fig. 11 shows the results of the classification.

Fig. 11: Original image (above), classification prediction result (below)

VIII. Future work

This pipeline can be tried with a different and bigger dataset such that more images can be used in order to train the classification network. Model pruning can also be tried in order to reduce the number of model parameters even further. Also, this pipeline can be directly evaluated with other state of art Object Detection architectures and the results can be compared.

References

[1] Adam Paszke, Abhishek Chaurasia, Sangpil Kim, Eugenio Culurciello. ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation

[2] Vijay Badrinarayanan, Alex Kendall, Roberto Cipolla, Senior Member, IEEE. SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation

[3] Jonathan Long, Evan Shelhamer, Trevor Darrell. Fully Convolutional Networks for Semantic Segmentation

[4] Karen Simonyan, Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition
[5] Ross Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation
[6] Ross Girshick. Fast R-CNN
[7] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks
[8] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, Piotr Dollr. Focal Loss for Dense Object Detection
[9] Sergey Ioffe, Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift
[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification
[11] Abhinav Valada, Johan Vertens, Ankit Dhall, Wolfram Burgard. AdapNet: Adaptive Semantic Segmentation in Adverse Environmental Conditions
[12] Liang-Chieh Chen, George Papandreou, Florian Schroff, Hartwig Adam. Rethinking Atrous Convolution for Semantic Image Segmentation
[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Deep Residual Learning for Image Recognition