Efficient Joined Pyramid Network Applied to Semantic Segmentation for GPU Embedded System

Soto-Orozco Oscar Alejandro
Division de Estudios de Posgrado e Investigacion
Tecnologico Nacional de Mexico/IT Chihuahua
Chihuahua Chih, Mexico
oscar.soto@ieee.org

Ramirez-Quintana Juan Alberto
Division de Estudios de Posgrado e Investigacion
Tecnologico Nacional de Mexico/IT Chihuahua
Chihuahua Chih, Mexico
Jarq1910@gmail.com

Reyes -Porras Jesus Alonso
Division de Estudios de Posgrado e Investigacion
Tecnologico Nacional de Mexico/IT Chihuahua
Chihuahua Chih, Mexico
alonsoreyes17@gmail.com

Corral- Saenz Alma Delia
Division de Estudios de Posgrado e Investigacion
Tecnologico Nacional de Mexico/IT Chihuahua
Chihuahua Chih, Mexico
adcorral@itchihuahua.edu.mx

Abstract—Fully Convolutional Networks (FCN) are the best methods for semantic segmentation. However, these networks are implemented in computers with high processing capabilities since they are computationally complex. For this reason, many strategies have emerged in the literature to propose novel efficient FCN for embedded applications. In order to contribute with this effort, we propose a network called Efficient Joined Pyramid Network (EJPNet), which is an efficient FCN that reduces the number of activations maps with pointwise convolutions in the encoder and also reduces the number of parameters with an Efficient Joint Pyramid Upsampling (JPU) decoder. EJPNet and other four state of the art efficient semantic segmentation methods were implemented on a Graphic Processing Unit (GPU) embedded system to obtain the performance in precision, number of parameters, memory footprint, Floating Point Operations per Second (FLOPS), and time processing. According to the results, EJPNet not only has the best number of parameters, FLOPS, and time processing, but also this network has one of the best precisions. Then, considering these results, EJPNet is a feasible semantic segmentation method for embedded systems applications.

Keywords—Semantic Segmentation, Fully Convolutional Networks, Deep Learning, Computer Vision.

I. INTRODUCTION

Semantic Segmentation (SS) consists in detecting objects previously defined in a digital image and it is applied to technologies such as autonomous vehicles, human-machine interaction, augmented reality, robotics, etc. Fully Convolutional Networks (FCN) are the most common solution for SS because they report the best performance in this task. However, FCNs are computationally complex, and they require expensive computers to be processed because FCNs are Convolutional Neural Networks (CNN) that have a set of deconvolutional layers to generate the objects segmentation instead of fully connected layers for classification [1]. For these reasons, new approaches that consist in developing new strategies for FCN computation, have emerged in the literature to reduce the amount of calculus, the memory footprint, the power consumption, the number of parameters, and the multiplications in the convolutions [2]. From these strategies, several FCN have been proposed, for example, Dimension-wise Convolution Efficient Network (DICENet) [4], Spatial Pyramid of Dilated Convolutions (ESPNet V1, V2) [3][5], Fast Fully Convolutional Network (FastFCN) [6], and Efficient Residual Factorized Network (ERFNet) [7]. These efficient FCNs allow semantic segmentation applications in embedded systems as phones, smart surveillance systems, embedded autonomous driving, Internet of things, etc.

According to [4][5][6], ESPNetV2 is the best network for embedded applications like edge devices, mobile phones, and embedded GPUs, because this network reports one of the best precisions in SS, consumes the lowest power when an image is propagated, and has a reduced number of parameters. On the other hand, FastFCN is an FCN that replaces the dilated deconvolutions layers of the decoder with an Efficient Joint Upsampling Module (JPU) to reduce the computation complexity and memory footprint during the object segmentation.

Then, based on the architecture of ESPNetV2 and the properties of the JPU module, this paper proposes an FCN called Efficient Joined Pyramid Network (EJPNet) which is an extremely efficient semantic segmentation network, inspired in the architecture of ESPNetV2, the low computational cost of the JPU module, and the capability to make representations with low memory using the pointwise convolution. The experiments were developed on an embedded system based on the NVIDIA Jetson TX2 GPU, that can be applied in real-world applications such as robotics, autonomous driving, and mobile applications, were resources like memory, capability of FLOPS, and power consumption are limited.

The rest of the paper is organized as follows: Sect 2 describes the proposed network EJPNet. Sect 3 describes the embedded system based on GPU. Finally, Sections 4 and 5 report the results and conclusions respectively.
II. EFFICIENT JOINED PYRAMID NETWORK

The EJPNet is an autoencoder network based on the FCN architecture, divided into encoder and decoder blocks, as Figure 1 shows.

The input is a video frame or an image $I(x, y)^{RGB}$, where $(x, y)$ are the pixel position, $E(x, y)^i$ are the feature maps generated in the encoder, and $S(x, y)$ is the segmentation output. The encoder is divided into the next layers: Pointwise convolutions channel, Convolution / Batch normalization / PReLU (CBR), and Extremely Efficient Pyramid of Depth-wise (EESP). The decoder is to get the objects segmentation, and it has the next layers: JPU module, dropout (DC), and a nearest neighbor interpolation. $S(x, y)$ is the segmentation output divided into 30 classes, each represents an object with semantic meaning.

Figure 2 shows the general structure of EJPNet, and the next subsections describe each part of the network architecture.

A. Encoder

Point convolution Channel (PC). this layer is a regular Convolution given by:

$$\sum_{p=1}^{p_1} F_p^{i-1} \Theta_1 K_p = F_p^i$$

(1)

where $\Theta_1$ is the dilation rate (the spaces between the feature kernel values), $p$ is the space summation index, $F_p^{i-1} \in \mathbb{R}^{W \times H \times M}$ is the output of the previous layer, where $W$ is the width of $F_p^{i-1}$, $H$ is the height, and $M$ is the depth. $K \in \mathbb{R}^{h \times n \times M}$ is the feature kernel with the size of $n \times n \times M$, and $F_p^i$ is the feature maps of the $i$-th convolutional layer.

PC layer is a Pointwise Convolution defined by (1), where $n=1, M=3, F_p^{i-1} = I(x, y)^{RGB}$ and $i=1$. The kernel of the PC layer combines the RGB features in one channel, which generates better information than a grayscale image/frame and helps to reduce the depth of the kernels in the next layers. This reduction impacts in the memory and the FLOPS usage.

Convolution, Batch normalization, and PReLU (CBR). It is a layer presented in the ESPNetV2 network to downsample the feature maps of the previous layer. Also, CBR appears three times throughout the network ($i=2,6$ and $13$).

The CBR layers are dilated convolution defined as (1), but with $n=3$ and an extended kernel $K_9 \times 9 \times M$ which dilation rate $\Theta_1$ is 3. This kernel helps to learn features with an extended field, and with fewer parameters. Additionally, to reduce the internal covariance of the network and accelerate the training, CBR layers have a batch normalization defined as follows:

$$\overline{F_p} = \frac{F_p^i - \mu}{\sqrt{\sigma^2}}$$

(2)

where the variance is defined as $\sigma^2 = \frac{1}{M} \sum_{m=1}^{M} (F_p^i - \mu)^2$, $\mu = \frac{1}{M} \sum_{m=1}^{M} F_p^i$, and $m$ represents the depth index.

For this layer, $F_p^{i-1}$ are the feature maps where the first CBR is $i=2$, the second CBR is $i=6$, and the third CBR is $i=14$. This normalization accelerates the convergence, even when the features are not correlated. With this normalizing, the values of the CBR module can be processed with the same range [7].

Figure 1. Autoencoder structure.

Figure 2. EJPNet model architecture.
Each CBR layer has a Parametric ReLU activation function (PReLU), which is given by:

\[
f(\mathcal{F}_p^i) = \begin{cases} 
\lceil \mathcal{F}_p^i \rceil & \text{if } \mathcal{F}_p^i > 0 \\
q_i \mathcal{F}_p^i & \text{if } \mathcal{F}_p^i \leq 0
\end{cases}
\]  

(3)

where \(q_i\) is a slope for negative coefficients of \(\mathcal{F}_p^i\) [10]. Then, PReLU learns negative activations because according to [8][9][10][11], these activations improve the performance. Figure 3 shows the shapes of ReLU and PReLU.

**Extremely Efficient Pyramid of Depthwise (EESP).** These layers are presented in ESPNetV2 and their aim is to project the feature maps into features interpreted with different receptive fields. EESP comprehends a PC sublayer and a set of depthwise dilated convolution sublayers.

The input of the PC sublayer is \(f(\mathcal{F}_p^i)\), and the operation is defined by (1), where \(n=1\) and \(M\) is given by the depth of the previous layer. These operations project the high-dimensional feature maps into low-dimensional spaces and replace a 3x3 convolution to reduce calculus complexity.

Next, the depthwise dilated separable convolutions are used to map the features into different receptive fields. These convolutions use different dilation rates and are configured in parallel as follows:

\[
\sum_{m=1}^{M} F_{p,l}^{i-1} \oplus I, K_p = F_{p,l}^i
\]  

(4)

where \(F_{p,l}^{i-1}\) is the PC sublayer of EESP, \(F_{p,l}^i\) is the feature map of the \(l\)-th receptive field, and \(n=3\). The \(F_{p,l}^i\) feature maps obtained from the parallel depthwise convolutions branches are concatenated to obtain a single map as output called \(C(x,y,m)\). This process generates gridding artifacts, which are removed with the Hierarchical Feature Fusion (HFF) method [12].

Finally, a group pointwise convolution defined as (1), where \(n=1\) is used again to reduce the depth of \(C(x,y,m)\) [13]. Then, the output of this sublayer is a map called \(E(x,y)^i\).

With the group pointwise and depth-wise dilated separable convolutions, the total complexity (TC) of the EESP layer is reduced by the factor shown in:

\[
TC = \frac{nM + n^2n^2B}{nM + (n^2 + d)dB}
\]  

(5)

where \(B\) is the number of parallel branches, \(g\) is the number of groups in the pointwise convolution, and \(n, M, g\) are the kernel dimensions. According to the definitions in [3], the EESP unit learns 7 times fewer parameters than the ESP of ESPNet V1. Figure 4 shows the scheme of the EESP unit.

For the first EESP layer, the PC sublayer is \(i=3\), the depthwise sublayer is \(i=4\), the dilation rates are \(l\{}4,5,6\}, and the last PC sublayer is \(i=5\). For the second EESP layer, the PC sublayer is \(i=7\), the depthwise sublayer is \(i=8\), the dilation rates are \(l\{}7,8,9\}, and the last PC sublayer is \(i=9\). For the third EESP layer, the PC sublayer is \(i=10\), the depthwise sublayer is \(i=11\), the dilation rates are \(l\{}1,3,6,8\} and the last PC sublayer is \(i=12\).

**B. Decoder**

The encoder outputs \(E(x,y)^i \in \mathbb{R}^{w \times h \times M}\) are feature maps whose dimensions are lower than \(I(x,y)^R \in \mathbb{R}^{w \times h \times M}\), where \(w<h\) and \(h<H\). Then, the decoder module compresses the three \(E(x,y)^i\) outputs into one feature map and generates a resolution analysis to get a segmentation map with the same size as the input \(I(x,y)^R\).

In the classical FCN, the decoder is a set of transposed convolutions and upsampled operations layers that take \(E(x,y)^i\) to generate the objects segmentation. However, the decoder of EJPNet is a Joint Pyramid Upsampling (JPU), which changes the deconvolutional layers of ESPNetV2 to reduce the computational complexity.

**Joint Pyramid Upsampling (JPU).** This layer was proposed in the Fast FCN network, and the objective of the JPU module is to transfer the structural details of the desired output \(O(x,y)\) to the feature maps used to build the segmentation. With this layer, we can improve the precision because the feature maps learn some properties of the ground truth. Also, JPU...
increases the time processing and reduces the parameters and FLOPS because the operations have less computational complexity than transposed convolutions.

The inputs of JPU are $E(x, y)^i$, $i=\{5,9,12\}$. The output is a segmentation map $F_p^i \in \mathbb{R}^{h \times w \times M}$ with $i=17$.

Figure 5 represents the process of JPU. The inputs $E(x, y)^{5,9,12}$ are convolved with (1) and $n=3$, forming three feature maps $F_{pb}^i$, where $b$ is the feature map index that represents the branch of each $E(x, y)^i$ input. Then, $b=\{5,9,12\}$. These maps are upsampled using the function $h$ to generate one feature map $F_p^{14}$ which is the approximation of $E(x, y)^{5,9,12}$ to $O(x, y)$. This upsampling is defined as follows:

$$F_p^{14} = \{E(x, y)^{5,9,12}\} \rightarrow h$$

$$\hat{h} = \text{argmin}_{h \in \mathcal{H}} \| O(x, y) - h(E(x, y)^i) \|,$$ (7)

where $h$ is the mapping function that learns the mixed relation between $E(x, y)^{5,9,12}$ and $O(x, y)$, $\mathcal{H}$ is a set of all possible transformation functions, and $\| \cdot \|$ is a predefined distance metric, all are defined in [14]. The resulting maps $\hat{h}$ are then concatenated to generate only one feature map $F_p^{15}$ with ground truth properties. The next step is a depthwise dilated convolution given by (4), where $l = \{1, 2, 4, \text{and } 8\}$ and $n=3$. This step integrates feature maps in different receptive fields and compresses the depth of the feature maps. The output is a set of features $F_{p,1}^{16}$, where $l = \{1, 2, 4, \text{and } 8\}$. These maps are concatenated and finally, a convolution given by (1), and $n=3$ is developed to generate the objects segmentation map $F_p^i$, with $i=17$.

Therefore, JPU can extract multiscale context information from different hierarchical feature maps levels, which generates better performance. This behavior is different from the module ASPP introduced in [15], which only exploits the information in the last feature map.

**Dropout (DC).** This layer ignores randomly the kernels $K_p$ during the training to avoid the overfitting in the learning and the inference [16]. For example, Figure 6 represents the convolutions in the nodes. After the dropout, some kernels are ignored, and therefore, the number of convolution operations decrease. With these convolutions, the network can learn more robust features that are useful in comparison to all set of convolutions. The input of DC is the last CBR layer $f(F_p^i)$, $i=14$, and the DC is defined by (1), where some kernels are zero, while others have $n=3$ to improve the relevant features. Then, the output of this layer is $F_p^i$, with $i=9$.

**Interpolation.** This layer is to find a segmentation mask with the same size as the input. The previous layer generates an output $F_p^i$, with $i=17$, and a size that is less than $W \times H$. Then, it is necessary to up-sample the segmented masks to the size of the input. This up-sampling is developed with bilinear interpolating, and the output is the objects segmentation $S(x, y)$. This interpolation is defined in [1].

III. GPU EMBEDDED SYSTEM FOR THE EXPERIMENTS

Figure 7 shows the general scheme of the embedded system designed for the experiments. This system was focused for embedded applications which have to run on mobile devices, smart surveillance systems, autonomous driving processors, and Internet of things applications. The input is the video frame or the image $I(x, y)^{RGB}$ acquired with a regular IP CCTV or Web camera compatible with HVEC codec.

The embedded board is an NVIDIA Jetson TX2 that has 32 GB storage MMC Memory, 8 GB RAM, 256 CUDA cores with pascal architecture, 1.3 TFLOPS, 1300 MHz, a CPU with four ARM Cortex-A57 cores with 2GHz and two NVIDIA Denver cores with 2GHz. The Jetson was selected because it is a board that has a GPU focused on embedded applications and it is the most popular GPU embedded board in literature.

The embedded system has a monitor connected with the board through HDMI. The operating system is Ubuntu 18.04 for ARM. The semantic segmentation methods were implemented in Python with OpenCV, Numpy, and Pytorch[17]. The networks implemented in this system were Enet, ERFNet, FastFCN, ESPNet, and EJPNet.
Additionally, all the networks were trained with the Adaptive Moment Estimation (ADAM) optimizer, 300 epochs, an initial learning rate of 0.0005, and polynomial rate decay with a power of 0.8.

IV. RESULTS

This section presents the performance evaluation of EJPNet. The evaluation consists in developing a comparison of FLOPS, time processing, memory, and precision metrics. These metrics were selected because the efficient networks in the literature compare their results using these metrics. The methods implemented were Enet [8], ERFNet [18], FastFCN [14], ESPNet V2 [13] and EJPNet. Enet was selected because is used in all the efficient FCN comparisons in the literature. ERFNet was selected because is the efficient SS model with the best precision. Fast FCN was selected because this network is an FCN that incorporates the JPU module. ESPNet was selected because it is the basis of EJPNet.

The experiments were developed with the Cityscapes dataset because it is the most popular dataset in literature to develop SS experiments. Other datasets as Camvid [19], Mapillary Vistas [20], and MS COCO [21] can be used to SS, but they are not widely used to train FCNs due to the images and their classes do not allow a correct learning generalization in SS applications.

Cityscapes is composed of 5000 images (2975 for training, 500 for validation, and 1525 for test) from 50 different cities and 30 different classes divided into 8 categories that represent urban objects such as vehicles, roads, buildings, cyclists, people, sidewalks, etc. There is one ground truth \( O(x,y) \) for each image of this database, and the objects of the ground truths are represented in false colors, as Figure 8 shows. Cityscape uses the metric of intersection over union (mIoU) to measure the precision of the networks and is given as:

\[
\text{IoU} = \frac{tp}{(tp + fp + fn)} \tag{8}
\]

where \( tp \) are the true positives, \( tn \) true negatives, and \( fn \) are the false negatives. Table 1 shows the results of the average metrics performance with the state of the art efficient FCNs.

The results show that EJPNet presents the best results in Number of parameters, FLOPS, and FPS. Additionally, EJPNet has the second-best place in precision (mIoU) and Memory footprint. ERFNet achieves the best precision, but it has the worst results in Memory use and FLOPS. Enet uses less memory than the other models, but it has the worst precision and it has the highest number of parameters. Figure 8 shows that the results of the networks in some images are similar. However, EJPNet defines better the cars and the truck in the frame Frankfurt 010351, but represents the sidewalks with more noise than the other networks in the frame Frankfurt 014480.

Additionally, according to the experiments, EJPNet has similar mIoU in regular scenarios and scenes with low saturation, shadows, and illumination changes. However, EJPNet, ESPNet, ERFNet, and Enet generate low mIoU when objects that are big or are close to the camera are presented. This low mIoU is because these objects generate over-segmentation. An example of these objects can be shown on the frame Frankfurt 010351 (Figure 8).

V. CONCLUSIONS

This paper proposes the Efficient Joined Pyramid Network (EJPNet) a semantic segmentation FCN that reports the best results in the Number of parameters, FLOPS, and FPS. Also, EJPNet has one of the best results in precision and memory footprint. According to the experiments, the JPU module increases the precision and upsamples the features to generate the object segmentation with fewer parameters, fewer FLOPS,
and best time processing than the other networks. The PC layer of the encoder compresses the color information of the input in one channel, reducing the memory and the FLOPS.

To improve the precision of EJPNet, the PC layer would be trained with different color spaces to find the best color compression patterns. To reduce the memory, we can modify the JPU module to eliminate the third CBR module and the interpolation. Therefore, the future work will be to map the input to other color spaces during the training and analyze the JPU to reduce the depth of the decoder developing a new module that generates an object segmentation with the same resolution as the input. With these improvements, EJPNet can achieve a time processing that allows the development of experiments with IP cameras in real-time applications.

Regard to the GPU embedded system, the experiment shows that our proposed network brings the possibility to design systems for applications such as automatic driving, robotics, and security. Then, considering the performance in precision, memory footprint, FLOPS and time processing, EJPNet is a feasible semantic segmentation method for embedded systems applications.

VI. ACKNOWLEDMENT

This research was funded by Tecnológico Nacional de Mexico (TecNM) under grant 7598.20-P.

REFERENCES