Semantic segmentation for learning hockey broadcast image representation

Philippe Blouin-Leclerc and Stéphane Caron
Laval University
{philippe.blouin-leclerc.1, stephane.caron.9}@ulaval.ca

May 10th 2019

Abstract

In this project, we propose a novel way to recognize key locations within hockey broadcast images using semantic segmentation and convolutional neural networks (CNN). The semantic representation of an image could then be used for many applications such as mapping a broadcast image into a 2D plan. All the codes that aimed to realized that project and that article are hosted on that GitHub repository.

1 Introduction

Computer vision is a growing field that changed the faces of many applications in robotic, security or even health sectors. Another field that is currently having more and more interests in such computer vision applications is the sports analytics. The professionnal sports clubs are using analytics and data to understand as more as they can the game and the performances of their players and their opponents. To increase that understanding, you often need a bunch of experts analyzing a bunch of data. Computer vision is well suited to extract that data because it allows the detection of many events simultaneously, which otherwise may have been done by many humans. In order to extract that data properly, it's way more interesting to map those events on the field (or the ice). To do so, it's often necessary to understand the general representation of the moment (or the image), which could be done by a computer vision task called semantic segmentation.
In his work, Homayounfar et al. (2017) present a methodology where he uses different cues on the field such as lines, corners, circles and so on to train another model that position the field in a 2 dimensionnal plan. In our project, we tried to improve the cues detection technic by training different semantic segmentation models and gain insights on how to train such models. The next step following that project will be to also use that representation in order to map players into a 2 dimensionnal plan.
In the next section, we will start by giving some background about the task of semantic segmentation (section 2.2), then we'll present our methodology (section 3.2), the dataset we used for our experiments (section 4.1), the results we had (section 5.1) and finally a short conclusion (section 6).

2 Background

2.1 Define the task

Before getting into more details in the 3.2 section, let's first define the task we will try to learn in this project. Semantic segmentation is a computer vision task where the model learns the general representation of an image by attributing a label to each and every pixels. Intuitively, to make pixel-wise predictions, you first need to have attributed a class to each pixel in the image (see figure 1), this is called a **mask**.

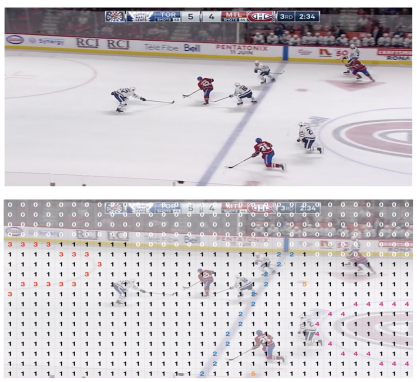

Figure 1: Example of mask where each pixel of an image is associated with one class.

2.2 Complete workflow

Once we have a class to every pixel, we now have the output we would like our model to predict. Similarly to other multiclass learning problems, we will have to *one-hot encode* the output of the model so that the output for every pixel will be a vector with the same length as the number of classes. If we look at it matrix-wise instead of pixel-wise, we can say that we will have *one-hot encoded matrix* as the output of the model. Each of those matrix. have the same width and height as the input image, will be related to a specific class and the values predicted by the model will correspond to probabilities to be part of that class.

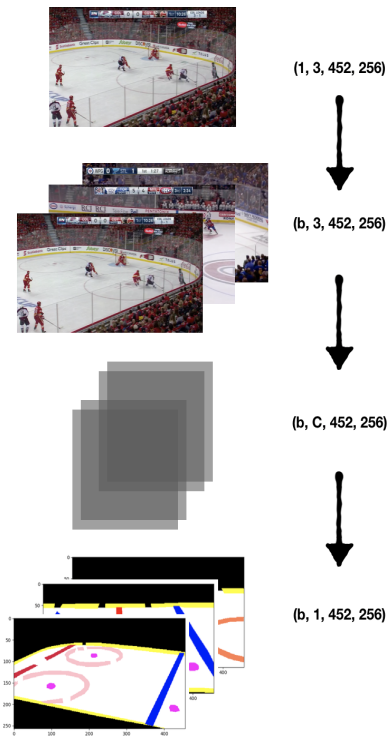

Figure 2: Summary of the workflow behind the semantic segmentation learning task.

The figure 2 summarizes this workflow where we start from multiple images and ends up with multiple predictions. As such, we start with one RGB image, then we have b images inside one minibatch. At the end of the model, each of those images are *one-hot* encoded to C dimensions, corresponding to the number of classes to predict. Finally, we apply a `softmax` or any other function that select a predicted class, so that we no longer have C dimensions, but only 1.

3 Methodology

3.1 Semantic segmentation

For this problem, the use of semantic segmentation was priorized. We uses fully convolutional network to achieve our semantic segmentation. One of the main problem of this task, is that the output need to have the same dimensions as the input. So the tricks to uses max pooling to reduce the dimension of the image while incresasing the number of channel is more challenging, since a we need to apply a upsampling technique in the network. We try two architecture, one of them uses the max pooling with the upsampling and the other one keep the same dimensions along the network.
Another challenge that we add in this task, was to deal with the presence of the class imbalance. We try to have loss function that were meaningful for this task and that take this challenge into account. We mainly try two types of loss function.
Like it was discuss in the dataset section, the amount of data was problematic for training our model. We then uses the technique of data augmentation that still make sens for the type of image that we wanted to apply the semantic segmentation on.
Finally, we discuss about the training details that we uses for our FCN.

3.1.1 Architecture

The first architecture that was tried is a network called U-Net. The name of this network is due to his shape. This net apply max pooling to reduce the dimensions of the feature map while increasing the number of channel to obtain more expressivity. Four max pooling are applied. After, there is four upsampling that are applied. The dimension of the feature map is then increased while the number of channel is decreased. The upsampling can either be learn or it can be an interpolation. The particularity of U-Net is the uses of skip connection. The skip connection are between the the same dimension feature map before each max pooling and after each upsampling. Those skip connection help to learn how to upsample well the image. A little difference with the figure 3 of the architecture, we used padding on our convolution filter, so the dimensions on the encoder part and the decoder part are the same and no croping is needed.

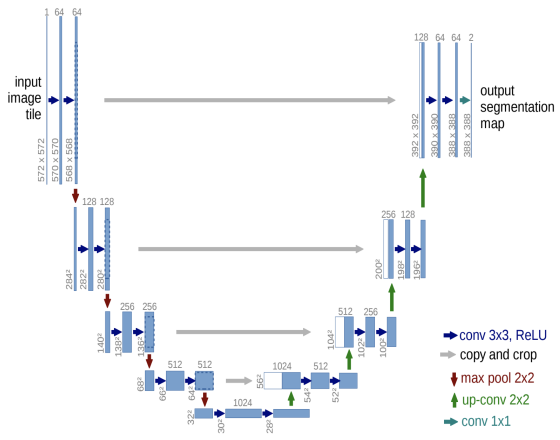

Figure 3: U-Net architecture.

The second architecture that was used is the VGG 16 pretrained with batch norm. We wanted to use a pretrained net because of the low amount of image that we had. This net was built do to image classification, so some modification was needed to adapt it for our task. The max pooling were removed from the net, so the input stay the same size as the output. Furthermore, we kept only the first seven layer of the net. This is mainly due to time computation constraint. A last layer of convolution with kernel size of one was added to the net the have nine channel in output of the net, like our task.

3.1.2 Loss function

As describe earlier, for our model to learn the task, we needed to have a loss function that isn't too much influenced by the class that are more present in the image. Details such as line and face-off dot are as important as the ice. We started by using the cross entropy loss. In this loss, each pixel is given an equal proportion in the loss, so the class that are more present in a batch, will have a bigger impact on the loss. We adjust the weight in the loss function with a formula base on the proportion of the class in the batch as discussed in this paper Paszke et al. (2016).

The second loss function that was used is the dice loss. This loss is calculated for each class and the means on all class is taken afterwards. Here is the formula of the loss.

$$\frac{1}{nb\_class} * \sum_{i=1}^{nb\_class} \left(1 - \frac{2 \sum_{pixels} y_{true} y_{pred}}{\sum_{pixels} y_{true}^2 + \sum_{pixels} y_{pred}^2}\right)$$

This has the particularity to give to each class the same proportion in the loss. This was usefull for recognizing all the class.

3.1.3 Data augmentation

The number of image that we label for our task was not enough to properly train our model for a lot on epoch. Also the labeling task was quite time consuming so that's mainly why we were not able to extract a large amount of data (only 43 images). A way to increase our training data set was to do data augmentation. Since we didn't have a lot of image, we didn't want to apply data augmentation that would modify to much the semantic of the image. We wanted our new image to be as realistic as possible of want a hockey ring look like. We decided to apply an horizontal rotation. As it can be seen it the figure 4 the new image is a image that is possible in the sense of hockey broadcast. This allow us to double the amount of image available for training.

Figure 4: Horizontal rotation

3.1.4 Training details

For the training of our models, we uses a kind of grid search approach with some parameters that we were changing each time. A challenge that we add was that we needed to look at image prediction each time to see if our model was converging well. Since we add 9 classes and didn't had a good and representative measure of accuracy we were not able to rely only on the lowest loss value. Sometimes, the loss was low, but the class were not well predicted and few of them was actually predicted. It happends that with higher loss, we were more satisfy of the result of the prediction on either the training set and the test set. So it was not possible to do a full grid search and take the combination of parameters that gave the lowest loss.

We tried to vary the learning rate, the batch size, the model uses, the loss uses, the weight apply on the cross entropy, the optimizer, whether we uses learned upsampling method on the U-Net or interpolation, the number of epoch and whether normalize the image according to the normalization calculated on imagenet. Our best result is presented in the result section.

3.2 Mapping to 2D plan

The goal of this projet is then to map the image of hockey broadcast in a 2D plan. To do so, a good representation of the ice rink feature must be learn by our algorithm. There is some key point in a hockey ring that are always at the same spot and that is really usefull to map the image in a plan. For example, the joint between the line and the board is an important feature. Be positioning the well recognized pixel on the true pixel value in the 2D map, it's possible to find the good deformation to apply to the image. As for now, we hadn't try anything related to the mapping. This would be try more thoroughly when our model of semantic segmentation will performed better.

4 Dataset

4.1 Dataset creation

We applied the methodology described earlier to hockey broadcast images. We generated our dataset, which contains images (inputs) and masks (outputs), by ourselves. To do so, we annotated a total of 43 NHL broadcast images. For the annotation part, we used an open source tool called cvat tool. That tool allowed us to draw polygons around our labels and then associate a class to each pixels in the image.

To choose our labels, we used an iterative approach where we first used our judgment to decide arbitrary categories (ex: ice, crowd, corners, horizontal lines, vertical lines). After that, we used one single image and tried to fit a model using those labels. We then noticed the bahviour of the model regarding those categories and adapted them. For example, we noticed the corners, the horizontal and vertical lines such as the board were difficult to distinguish for the model. As such, we decided to merge them and use one single category called "board". Here are the **9 categories** we selected at the end: ice, boards, crowd, red line, blue lines, goal lines, circles in end zone, circles in neutral zone and dots. The figure 5 is an example of image we extracted and then labeled using our 9 categories.

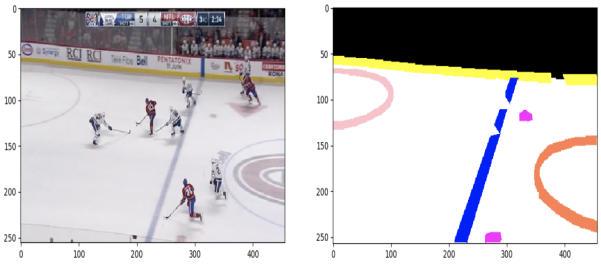

Figure 5: Example of extracted image (left) and the labeling output (right) using the 9 categories.

In the methodology section, we mentionned we had a class imbalance problem. In that section, we tackled this problem by adapting the loss. In the labeling task, we also tackled that class imbalance problem by subjectively drawing larger polygons for rare categories such as dots or circles.

5 Results

With the two neural network architectures, the two loss function, our dataset and all the hyperparameters, we achieve to obtain result that we found good for our little dataset.

5.1 Hyperparameters

To find the hyperparameters that gave the best images predictions on the test set, we did a grid search with a lot of check on the prediction by looking at the image. Our best result was achieve with the adapatation of the pretrained VGG 16. For the optimizer, we choose the stochastique gradient descent with a momentum of 0.9 and a small weight decay. The learning rate started at 0.001 and we decreased it when the loss was not decreasing anymore. The batch size was 4, the dice loss was used, the image was normalize using the normalization of imagenet and we did 20 epochs. Here is the result that we gathered from our best experiments.

Model	Epochs	Loss	Train Loss	Valid Loss
U-Net	50	Dice	0.8263	0.8393
VGG16	20	Dice	0.8387	0.8023

The figure 6 is sample that shows our best results for VGG16 model on test set images.

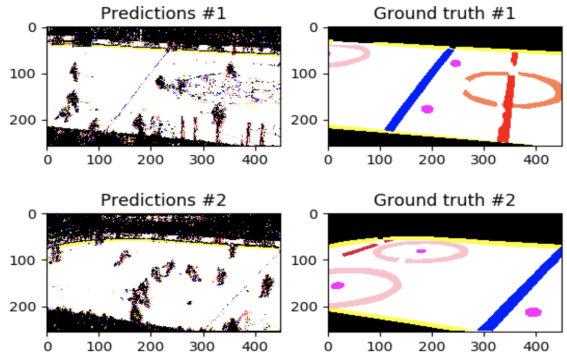

Figure 6: Sample of predictions of the VGG model on the test set.

We see that the upper board seems to be well predicted. The player on the ice are labelled as crowd, but it's not supprising since the player on the beach and the crowd are labbeled as crowd. The line appears a little bit.

6 Conclusion

In conclusion, we can first say that we need more images. The lack of images is even more problematic for a non pre-trained model such as the U-Net model. We can also say that a pre-trained model seems to be a must in our learning task, even if we could have a larger dataset. After a lot of training attempts, we also noticed that those models seems to learn slowly and over a long period of time.
The next steps would be to extract more images. Also, because the pre-trained model was quite successful, we may need to put more focus on using a more specific encoder for that task. In the same idea, we may need to put more efforts on the decoder part of the model, which can hardly be pre-trained in our case. Finally, if we are able to build a adequate semantic segmentation model, we could then use that model to map our images into a 2 dimensional plan.

References

Homayounfar, N., Fidler, S., & Urtasun, R. (2017). Sports field localization via deep structured models. In *2017 IEEE conference on computer vision and pattern recognition, CVPR 2017, honolulu, hi, usa, july 21-26, 2017* (pp. 4012–4020).

Paszke, A., Chaurasia, A., Kim, S., & Culurciello, E. (2016). Enet: A deep neural network architecture for real-time semantic segmentation. *CoRR*.