Hypercolumns for Semantic segmentation

Charic Daniel Farinango Cuervo

1. Introduction

Semantic segmentation is one of the most important tasks in computer vision. Its importance comes from the necessity of understanding the context in a scene. This is very useful for different applications that involve handling scenes in real life in real-time. Some examples of such applications are self-driving cars, manufacturer robots, among others. The task of semantic segmentation can be defined as a fine-grained inference by making dense predictions to infer labels for each pixel and return labeled regions. If done correctly, one can delineate the contours of all the objects appearing on the input image.
However, the case in semantic segmentation is somewhat different. As a result of classifying each pixel, the task is more challenging, largely due to complex interactions between neighboring as well as distant image elements, the importance of global context, and the interplay between semantic labeling and instance-level detection [1].
Recognition algorithms based on convolutional networks (CNNs) typically use the output of the last layer as a feature representation. However, information in this layer may be too coarse spatially to allow precise localization which is fundamental in semantic segmentation. On the other hand, earlier layers may be precise in localization but will not capture semantics [2]. To take advantage of both representations, B. Hariharan et al. had the idea of representing pixels by a set of features along different layers, shallow and deep ones, of the CNN. They defined the hypercolumn at a pixel as the vector of activations of all CNN units above that pixel.
In the original paper, the authors used the hypercolumn representation to perform semantic segmentation and keypoint predictions for joints localization. The purpose of this project is to understand and implement this approach. Moreover, it will be applied to a different CNN called SqueezeNet [3] to observe its performance.

2. Related work

2.1 Pedestrian detection

The authors combine subsampled intermediate layers with the top layer for pedestrian detection [4].

2.2 Hypercolumns for semantic segmentation

As it was previously mentioned, the idea of hypercolumns was introduced for the first time in the paper “Hypercolumns for Object Segmentation and Fine-grained Localization” [2]. To perform semantic segmentation, the authors predicted a heatmap on an expanded box of a bounding box. This heatmap encodes the probability that a particular location is inside the object. As a result, they predict a 50x50 heatmap that is resized to the size of the expanded

bounding box and splat onto the image (task of assigning a probability to each of the 50x50 locations or, in other words, of classifying each location.) [2]. The authors used the hypercolumn idea to predict such heatmaps and locations.

3 Problem setup

According to [2], recognition algorithms use the output of the last layer of the CNN. The last layer is the most sensitive to category-level semantic information and the most invariant to "inconvenient" variables such as pose, illumination, precise location among others. As the current task is fine grained, these inconvenient variables are precisely of interest.
The information that is generalized over in the last layer is present in intermediate layers, but intermediate layers are also much less sensitive to semantics.
For instance, bar detectors in early layers might localize bars precisely, but cannot discriminate between bars that are horse legs and bars that are tree trunks [2]. Therefore, reasoning at multiple levels of abstraction and scale could be helpful. As such, the hypercolumns lend a hand in defining these broad descriptors.
One observation that might arise is the source of the features that are being sampled. These features are extracted from an already defined CNN such as R-CNN [2] or VGG. This concept can be applied to other networks. Particularly, the SueezeNet [3] has a strategy in which it downsamples late in the network so that convolution layers have large activation maps. Hence, it would be interesting to observe if sampling features from these large activation maps would benefit the hypercolumn approach, given that it continuously upsample the feature maps to obtain the pixel representations. Moreover, according to a benchmark performed in [5] SqueezeNet has a higher accuracy than VGG.
Ultimately, the goal of this project is to explore and learn how to implement hypercolumns for semantic segmentation. Moreover, to observe its behavior after applying the hypercolumns approach with a SqueezeNet.

3.1 Hypercolumn

The hypercolumn is a set of features which describes the same pixel representation towards the layers of a CNN. For each location (pixel), the hypercolumn is formed by extracting features from a set of layers by taking the outputs of the units that are "above" the location. However, because of subsampling and pooling operations in the CNN, these feature maps may not be at the same resolution as the input or the target output size. So, finding which unit lies above a particular location is ambiguous.
To get around this, each feature map should be resized to a particular size with bilinear interpolation. Then, features from some or all of the feature maps in the network are concatenated into one long vector for every location which is called the hypercolumn at that location [2]. This explanation can be observed in Figure 1.

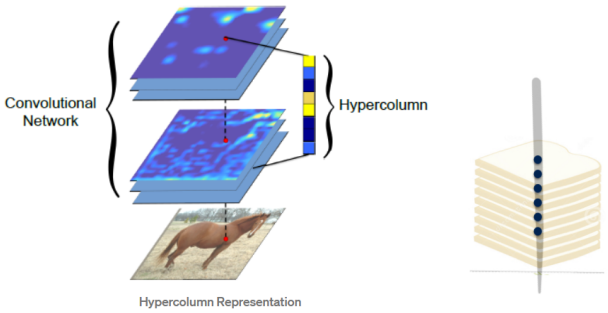

Figure 1. Hypercolumn concept

Mathematically, an hypercolum is defined as:

$$\mathbf{f}_i = \sum_k \alpha_{ik} \mathbf{F}_k$$

Where the feature map is defined by F and the upsampled feature map for the ith location by fi. αik depends on the position of i and k in the box and feature map respectively.

As an example, using a pool2 (256 channels), conv4 (384 channels) and fc7 (4096 channels) from the architecture of AlexNet would lead to a 4736 dimensional vector (hypercolumn).

Because these feature maps are the result of convolutions and poolings, they do not encode any information about where in the bounding box a given pixel lies. Location is an important feature i.e. head is more likely to be in the top. The simplest way to get a location-specific classifier is to train separate classifiers for each of the 50x50 locations [2]. However, doing so has three problems:

1. The amount of data going through one classifier is small, which might lead to overfitting: dividing input images in 50x50 regions causes less training examples for each classifier.
2. Training these many classifiers is computationally expensive.
3. While the classifier should vary with location, the classifier should change slowly: two adjacent pixels that are similar to each other in appearance should also be classified similarly.

The solution is to create a coarse K x K grid where each cell is a classifier (a function gk( )). Then apply bilinear interpolation (to classify a pixel) by defining a function hi at each pixel as a linear combination of the nearby grid functions. These functions output a probability Pik (kth classifier) for ith pixel.

$$p_i = \sum_k \alpha_{ik} g_k(\mathbf{f}_i) = \sum_k \alpha_{ik} p_{ik}$$

At test time, all classifieds (KxK) are runned on all pixels. If K=5, there would be 25 classifiers applied to each pixel when inferring.
Now, there is a new problem related to efficiency. Upsampling each feature map and then classifying is expensive. To address this problem, the authors re-define the solution as: if fi is

the hypercolumn at location i, then fi will be composed of blocks f (j) i corresponding to the jth feature map output for pixel i. A linear classifier w will decompose similarly. Then, the jth term in the decomposition corresponds to a linear classifier on top of the upsampled jth feature map. Now, since the upsampling is a linear operation, the most important conclusion is that the classifier can be applied first and then upsampling action. This is represented as:

$$\mathbf{w}^{(j)T}\mathbf{f}_i^{(j)} = \sum_k \alpha_{ik}^{(j)} \mathbf{w}^{(j)T}\mathbf{F}_k^{(j)}$$

Figure 2 shows the convolution applied to the feature maps and then the resampling of each result.

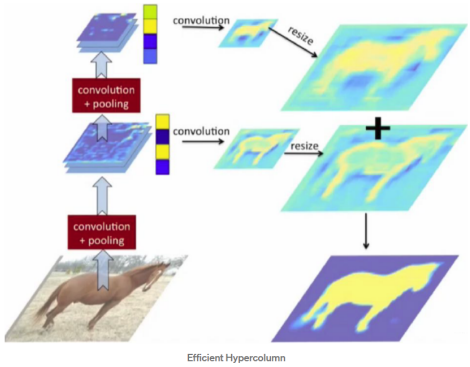

Figure 2. Efficient feature extraction to form Hypercolumns

The process can be performed as a CNN. For each feature map, one convolutional layer should be stacked. Each such convolutional layer has KxK channels, corresponding to the KxK classifiers to train. Then, the outputs of all these layers are upsampled using bilinear interpolation and sum [2].
Finally, these outputs are passed through a sigmoid, and combine the KxK heatmaps using equation 3 to give the final output. The final hypercolumn classifier can be described as in the figure 3.

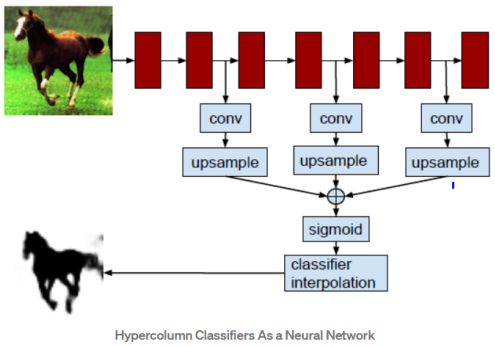

Figure 3. Process as a CNN

3.2 SquezeNet

The original motivation to create the SqueezeNet [3] was to obtain a high accuracy network (compared with networks at the time) while reducing the size of it (fewer parameters). As a result, SqueezeNet consists of 18 deep layers and, for the same accuracy as AlexNet, SqueezeNet can be three times faster and 500 times smaller.

To define this network, the authors defined three strategies.

1. **Replace 3×3 filters with 1×1 filters:** make the majority of filters 1×1, since a 1×1 filter has 9× fewer parameters than a 3×3 filter.
2. **Decrease the number of input channels to 3×3 filters:** For a convolution layer that is composed entirely of 3×3 filters, the total quantity of parameters would be (number of input channels) × (number of filters) × (3×3). So, to maintain a small total number of parameters in a CNN, it is important not only to decrease the number of 3x3 filters (like Strategy 1), but also to decrease the number of input channels to the 3x3 filters.
3. **Downsample late in the network so that convolution layers have large activation maps:** The intuition is that large activation maps (due to delayed downsampling) can lead to higher classification accuracy.

The architecture of the SqueezeNet is based on the Fire block. This block consists of: a squeeze convolution layer (which has only 1x1 filters to apply strategy 1), feeding into an expanded layer that has a mix of 1x1 and 3x3 convolution filters. The dimensions in the Firemodule are: s1x1 (number of filters in squeeze layer), e1x1 (1x1 filters in the expand layer), and e3x3 (3x3 filters in the expand layer). S1x1 should be less than (e1x1+e3x3), so the squeeze layer helps to limit the number of input channels to the 3x3 filters, as per Strategy 2 [3]. This module can be seen in figure 4.

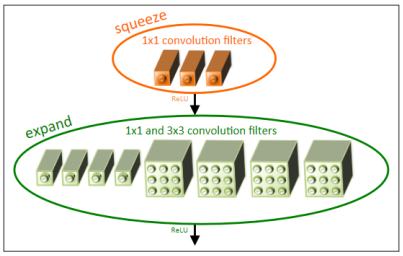

Figure 4. Fire block

4. Experimental setup

To implement the project, a simplified approach was used. Instead of creating a grid of KxK classifiers, I used one fully connected classifier to classify each pixel represented as a hypercolumn.
This project was based in previous implementations of hypercolumns to make semantic segmentation and colorization. These implementations used a VGG network to extract the

features for the hypercolumns. Therefore, as a first step a VGG-11 will be used. In general, the following steps were taken.

1. Use a pre-trained CNN such as VGG-11. Define layers from which features will be extracted to form the hypercolumns.
2. Define classifier as a pixel-wise classifier of two fully-connected layers

3. Extract sparse samples of hypercolumns to pre-train the classifier, this will help to accelerate the training with all training examples:

    a. For each training image, sample 3 pixels from each class
    b. For each pixel, save the pixel feature vector or hypercolumn (i.e. 1x1xZ) along with the identity of that pixel as a label
    c. There are an average of 2.5 classes per image so this process should yield around 10,000 example (feature,label) pairs

4. Convert fully-connected classifiers into a dense classifier with 1x1 convolutions. Converting a fully connected network to a convolutional allows to "slide" the original ConvNet very efficiently across many spatial positions in a larger image, in a single forward pass [6].

5. Train the network with all of the training examples (all feature columns formed by all of the pixels)

4.1 VGG-11 setup

Taking as a reference Figure 3, Figure 5 shows the layers from where features were extracted from the VGG-11 network architecture.

- Sampled layers: [0,3,8,13,18]
- Hypercolumns size: 1472 = 64+128+256+512+512

Figure 5. VGG-11 network architecture and extracted feature maps

4.2 SqueezeNet setup

For SqueezeNet, the process of extracting features is different from VGG. Due to the Fire block explained before, the extraction process required to enter the block and concatenate the feature maps from the expand layer. This will return the correct dimension of features. Figure 6 shows the layers from where features were extracted.

- Sampled layers: [0,1,2,4,6,8]
- Hypercolumns size: 1504= 96+128+128+256+384+512

There is no specific rule to pick the layers from where the features will be extracted. The only constraint that I defined was that the total number of features per hypercolumn (dimension) was similar.

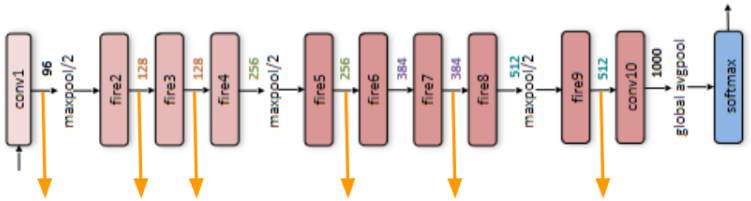

Figure 6. SqueezeNet network architecture and extracted feature maps

4.3 Dataset

For this project the PASCAL Visual Object Classes 2012 was used. It was thought as a dataset to help generate pixel-wise segmentations giving the class of the object visible at each pixel, or "background" otherwise. It has 21 classes including background. It has 1465 training examples and 1450 test examples.

5. Results

Once the implementation was done, the network had to be trained once for the VGG-11 network and again for SqueezeNet. Accuracy and Intersection over Union (IU) were evaluated. The mean accuracy for VGG-11 and SqueezeNet were 67.84 and 67.13 respectively. Figures 7 and 8 show these results.

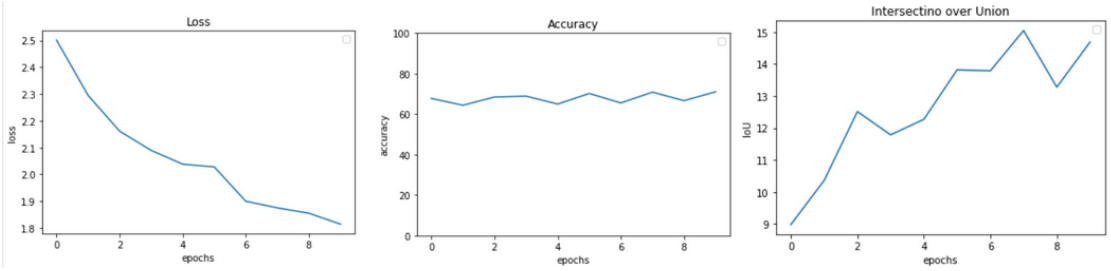

Figure 7. Quantitative results for VGG-11 model

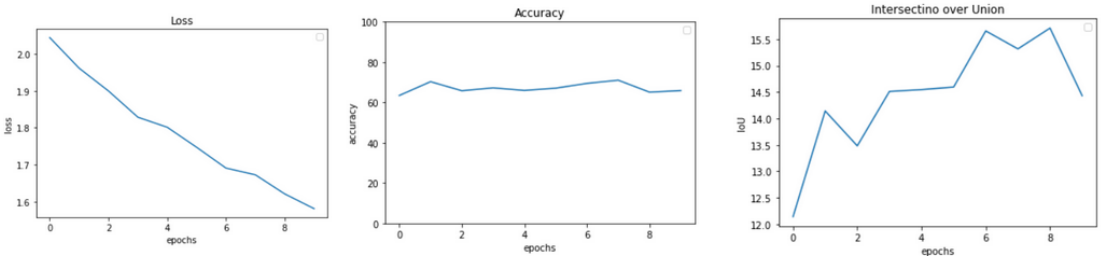

Figure 8. Quantitative results for SqueezeNet model

Qualitative results are shown in Figure 9

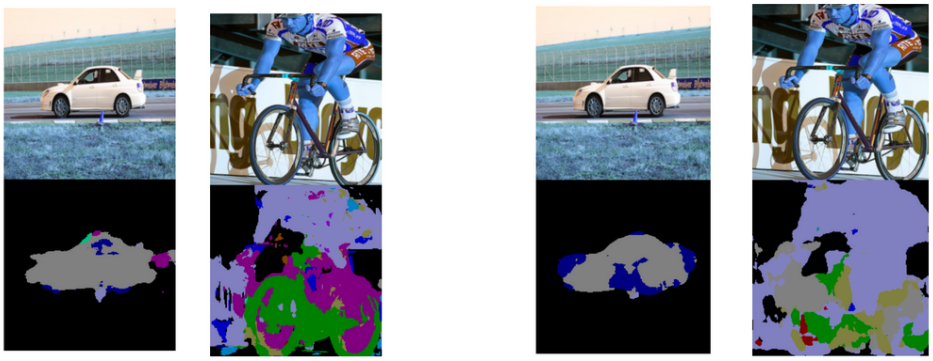

Figure 7. Qualitative results. Left: VGG-11. Right: SqueezeNet

6. Discussion

The hypercolumn idea is a very reasonable and well thought concept. It allows to represent each pixel with a set of features from a broad spectrum of features (corresponding to each layer of a CNN). Moreover, it can be applied to different networks which gives it good versatility. According to different implementations, this idea can slo be used for different tasks such as human pose detection or colorization of images.

Although, VGG-11 net had a better average accuracy, the SqueezeNet had a more consistent accuracy through the epocs. This is shown by both the accuracy and loss figure in Figure 9.
In terms of UI, both models did not perform very well. Although SqueezeNet had higher scores, both models only achieved a maximum score of approximately 0.15 for the last epocs.
Quality results are helpful to visualize in a more interpretable way the results. Both models were tested with two images which can be considered as a simple (car) and hard (man in bicycle) image. For the VGG-11 model, the simple image was classified with 5 different classes. On the other hand, the SqueezeNet model classified it with two classes. However, for the harder image, the first model gave a better outline of the objects than the SqueezeNet model.

7. Conclusions

The following conclusions can be drawn from this project.

- Hypercolumn is an interesting non-complex way to obtain good information about pixels.
- Results did not vary much, probably due to the fact that the implementation does not use a KxK grid of classifiers.
- Although the training in Squeeze net was more consistent, the accuracy for VGG-11 was a slightly better.
- The authors in the original paper used bounding boxes after non-maximal suppression plus high scored boxes.
- Other types of classificators can be tried.

8. Future work

Given the limitations of this project, it would be interesting to implement the KxK filter and observe the performance of both networks.

9. Bibliography

[1] M. Mostajabi, P. Yadollahpourand G. Shakhnarovich, Feedforward semantic segmentation with zoom-out features, CVPR 2015
[2] B. Hariharan, P. Arbelaez, R. Girshick, and J. Malik, Hypercolumns for Object Segmentation and Fine-grained Localization, CVPR 2015
[3] Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., & Keutzer, K. (2016). SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size. *arXiv preprint arXiv:1602.07360*
[4] P. Sermanet, K. Kavukcuoglu, S. Chintala, and Y. LeCun. Pedestrian detection with unsupervised multi-stage feature learning. In CVPR, 2013.
[5] Bianco, S., Cadene, R., Celona, L., & Napoletano, P. (2018). Benchmark analysis of representative deep neural network architectures. *IEEE Access*, *6*, 64270-64277.
[6] Cs231n.github.io. 2021. *CS231n Convolutional Neural Networks for Visual Recognition*. [online] Available at: <https://cs231n.github.io/convolutional-networks/> [Accessed 9 May 2021].