Conditional Random Fields as Recurrent Neural Networks

1 Citation

Zheng, Shuai, et al. "Conditional random fields as recurrent neural networks." Proceedings of the IEEE international conference on computer vision. 2015.

`https://www.cv-foundation.org/openaccess/content_iccv_2015/papers/Zheng_Conditional_Random_Fields_ICCV_2015_paper.pdf`

2 Abstract

In semantic segmentation, we want to label each image pixel with a label. The best approaches use Convolutional Neural Networks (CNN) to label the pixels and Conditional Random Fields (CRF) to smooth the labels. We model a CRF as a Recurrent Neural Network (RNN) so the whole system is end-to-end trainable. We set state of the art on the PASCAL VOC dataset.

3 Introduction

Since object recognition CNNs are designed to produce a single label for each image and use max-pooling which summarizes groups of pixels, they don't work off the shelf for image segmentation. If you modify the network to do per-pixel prediction, they can sometimes produce non-smooth outputs. This why you need a CRF. A CRF models the pixels as a graph where each pixel has a unary energy (dependent on whether its label is correct) and a pairwise label with its neighbors (dependent on whether they map). By minimizing the energy using mean field inference, you can smooth the labels. The problem is that this is separate from the neural network, so you cannot get wins from end-to-end training. In this paper, we formulate the mean field inference as a RNN and thus enable end-to-end training.

4 Related Work

Other work has used CNNs and CRFs separately to do semantic segmentation. Some people have looked into turning CRFs into neural networks.

5 Conditional Random Fields

Suppose the set of labels is $\mathcal{L} = \{l_1, l_2, ..., l_L\}$ and let X_i be the predicted label for pixel $i \in \{1...N\}$ in image $\mathbf{I}$. We can define the energy of the labeling $\mathbf{x} \in \mathcal{L}^N$ as:

$$E(\mathbf{x}) = \sum_i \psi_u(x_i) + \sum_{i<j} \psi_p(x_i, x_j)$$

$psi_u(x_i)$ is the inverse likelihood of the pixel having its predicted label and the pairwise term is the sum of M Gaussian kernels $k_G^{(m)}$ applied on the feature vectors $\mathbf{f}_i, \mathbf{f}_j$ (from the CNN) for the pixels.

$$\psi_p(x_i, x_j) = \mu(x_i, x_j) \sum_{m=1}^{M} w^{(m)} k_G^{(m)}(\mathbf{f}_i, \mathbf{f}_j)$$

To minimize the energy, we use mean field inference. We formulate this algorithm using components that would be present in a neural network. It is defined as follows.

Let $U_i(l) = -\psi_u(X_i = l)$. We first normalize these values by applying a softmax.

$$Q_i(l) = \frac{\exp(U_i(l))}{\sum_j \exp(U_j(l))} \text{ (for all } i)$$

Next, we repeat the following steps in a loop until convergence.

We do message passing by applying M Gaussian filters over the Q values. We use the permutohedral lattice implementation to do this efficiently. We use two Gaussian kernels - a spatial kernel and bilateral (color related?) kernel.

$$\tilde{Q}_i^{(m)}(l) = \sum_{j \neq i} k^{(m)}(\mathbf{f}_i, \mathbf{f}_j) Q_j(l) \text{ (for all } m)$$

After applying the Gaussian filters, we need to compute a weighted sum of filter outputs (the weights are learned). We use one set of kernel weights for each class label. This is because some classes may care about the spatial kernel more than the bilateral kernel (e.g. bicycle detection cares about bilateral pixel location because color is quite informative, while TV detection probably cares most about spatial kernel because the colors in a TV vary greatly).

$$\check{Q}_i(l) = \sum_m w^{(m)} \tilde{Q}_i^{(m)}(l)$$

Next, we do a compatibility step where we consider how label pairs interact. Here $\mu(l, l')$ can be represented by a $|\mathcal{L}| \times |\mathcal{L}|$ matrix.

$$\hat{Q}_i(l) = \sum_{l' \in \mathcal{L}} \mu(l, l') \check{Q}_i(l)$$

Next, we add the unary potentials and normalize:

$$\breve{Q}_i(l) = U_i(l) - \hat{Q}_i(l)$$
$$Q_i(l) = \frac{\exp(\breve{Q}_i(l))}{\sum_j \exp(\breve{Q}_j(l))}$$

6 CRF as RNN

One iteration of the mean field inference algorithm is denoted as f_θ for parameters θ. We can then formulate the algorithm as an RNN as follows. We run the algorithm for T iterations.

$$H_1(t) = \begin{cases} \text{softmax}(U) & t = 0 \\ H_2(t-1) & 0 < t \leq T \end{cases}$$

$$H_2(t) = f_\theta(U, H_1(t), I) \text{ for } 0 \leq t \leq T$$

$$H_1(t) = \begin{cases} 0 & 0 \leq t < T \\ H_2(t) & t = T \end{cases}$$

This CRF-RNN does not use a Long Short-Term Memory (LSTM) because T is small so we don't run into vanishing/exploding gradient problems. This is end-to-end trainable.

For our CNN, we used the FCN-8s network, which is a VGG network modified to do pixel-level predictions.

7 Implementation Details

The compatibility transform parameters are initialized with the Potts model. Kernel width and hieght come from cross validation. We train with learning rate 10^{-13} and momentum 0.99 since we process one image at a time. We set $T = 5$ at train time and $T = 10$ at test time. We train with the usual average per-pixel cross entropy loss. We tried replacing the normalization step of CRF-RNN with ReLU, but it didn't help.

8 Experiments

We evaluate on PASCAL VOC. We compare against a plain FCN-8s, FCN-8s and CRF combination (separate, not end-to-end trainable), and all state of the art algorithms. We beat them all.

We also tried adding Microsoft COCO do our training set, and this helped a little.

Using different filter weights for each kernel gave a 1.8% boost. Using an asymmetric compatibility transform gave a 0.9% boost. Using $T = 10$ at test time gave a 0.2% boost. End-to-end training gave a 3.4% boost. Fine-tuning the FCN-8s along with training the CRF-RNN gave a 1% boost. Using MS COCO gave a 3.3% boost. Our best model got 74.7% vs. 72.7% state of the art. The metric here is mean intersection-over-union.

9 Conclusion

CRF-RNN is an end-to-end trainable CRF. Combining this with FCN-8s gives state of the art semantic segmentation performance.