Semantic Segmentation with CRF

Jingjia Meng
jingjiam@andrew.cmu.edu

Atieno Ogayo
aogayo@cs.cmu.edu

Jacob Rast
jrast@andrew.cmu.edu

1 Introduction

Semantic segmentation is a common computer vision task. Several works have explored this task using different approaches from probabilistic methods, to deep neural nets and innovative combinations of different approaches. The conditional random field (CRF) is one of the approaches used to increase the performance of deep neural networks as they adhere poorly to object boundaries. With CRF, the pixel prediction at the boundaries of different objects is improved. The DeepLab v2 model hit the top of the leaderboard when incorporating the CRF in their model. However, the current SOTA, DeepLabv3+, doesn't use a CRF layer and we are interested in figuring out why they got rid of it.

In this project, we sought to find out whether a CRF layer always improves the accuracy of segmentation models by adding it to some of the top models. We trained several backbone models to collect the probabilities of pixel assignments as unary potential and then we feed these unary potentials to a pre-trained CRF block to make predictions. We evaluated the performance of the models with and without the CRF layer by using mIoU as our metric. We find that the CRF layer can indeed improve the accuracy of some of these models by up to 6%. The improvement is greater when the CRF layer is jointly trained with the deep neural network. In the absence of joint training, the performance of some of the models deteriorates by up to -2%.

2 Dataset and Task

2.1 Task

Semantic segmentation is the task of assigning a semantic label to every pixel in an image. It is used to identify objects in an image by classifying the pixels in the region of the image in a single class. The task has been widely studied in computer vision, where researchers have used both deep learning and probabilistic methods such as conditional random fields. It is still an active area of research and has datasets such as ADE20K, Cityscapes, and PASCAL VOC dedicated to its study. Semantic segmentation is a structured prediction problem as for every input image, a sequence of pixel labels representing the objects in the image are predicted.

2.2 Dataset

The dataset we used for image semantic segmentation is based on PASCAL VOC 2012. The dataset is available at [1]. In this task, we assigned a label to each pixel in the image such that the pixels belonging to the same object were labeled to the same class. Usually, there are multiple objects (classes) in one image consisting of multiple pixels, so the prediction is a structured prediction for this task. To make predictions, we implemented a convolutional neural network to produce unary potentials (probabilities if without a Conditional Random Field layer). We then applied an edge potential connecting every pixel to describe the affinity of each pixel with its neighbor to increase the prediction accuracy. The evaluation metric we are planning to use is Mean region Intersection upon Union (mIoU), which is defined as

$$M_{IoU} = \frac{1}{n_{cl}} \sum_i \frac{n_{ii}}{t_i + \sum_j n_{ji} - n_{ii}}$$

where n_{ij} is the correct label for pixel i belonging to class j. n_{cl} refers to the total number of different classes. $t_i = \sum_j n_{ij}$ is the total number of pixels of class i.

3 Related Work

Graphical models such as Conditional Random Fields (CRF) have been used successfully in segmentation tasks. Krahenbuhl and Koltun's [5] developed a highly efficient approximate inference algorithm that achieved a speed-up to 0.2 seconds for variational inference from 36 hours for MCMC inference.

Deep learning approaches have also been used in the task. Through Unet, [8] built upon "fully convolutional network" (FCN) [6] by introducing large number of feature channels, which allow the network to propagate context information to higher-resolution layers. This enabled the efficient use of training data and yielded more precise segmentations. [7] introduces the Global Convolution Network (GCN) where they deploy a larger kernel (and effective receptive field) to address classification and localization in segmentation. To incorporate suitable global features,[9] propose pyramid scene parsing network (PSPNet) which extends the pixel-level feature to the specially designed global pyramid pooling. The and global clues together make the final prediction more reliable. The current state-of-the-art model, DeepLab V3+,[3] is an improvement on DeepLab [2], but unlike DeepLab, it doesn't employ graphical models.

Although deep learning models based on CNNs have improved semantic segmentation, CNN-based models are limited by the downsampling employed to learn hierarchical features such that pixel-level details are lost in this process, resulting in segmentation masks that poorly adhere to object boundaries [4]. Probabilistic post-processing algorithms such as pRGR [4] and CRFs have been used successfully used to overcome this limitation. [10] implemented CRFs using both detached and end-to-end training, with end-to-end training performing better.

4 Methods

The insight for this is because, from the graphic model, pixels in the center of an object have all other pixels in the same class. All surrounding edge potentials are predicting the center one to the same class. In other words, the edge potential actually encourages the pixel next to each other to be in the same class. So, there is no significant difference for the pixel surrounded by the same kind. However, for the pixels at the edges, the preferred class from pixels at both sides varies. The edge potential adds more information to the pixel about which class to be assigned. To conduct a belief propagation through an image graphic model is complicated in terms of the interconnection between every pixel and the loopy paths. In the baseline work, they used a mean field approximation to simplify the conditional dependencies. They assumed all pixels in the image are fully connected and used a high dimensional Gaussian distribution to approximate the conditional probability. Theoretically, the assignment for one pixel is strongly correlated to its neighbor pixels, and the "long-distance" interactions can be ignored as compared to the "short-distance" interaction. So, they made a further assumption that the approximation is only accounting for the nearest several pixels. In this case, they transform this conditional random field problem into a convolutional problem. They used a total number of M high-dimensional Gaussian filters to calculate the probabilities. By doing the mean-field approximation, greatly increases the inference speed than using loopy belief propagation. The pseudocode for Mean-field approximation as a common CNN is as follows.

```
$\bar{Q}_i^{(m)}(l) \leftarrow \sum_{j \neq i} k^{(m)} (\mathbf{f}_i, \mathbf{f}_j) Q_j(l)$ for all $m$
$\dot{Q}_i(l) \leftarrow \sum_m w^{(m)} \tilde{Q}_i^{(m)}(l)$                    ▷ Message Passing
$\hat{Q}_i(l) \leftarrow \sum_{l' \in \mathcal{L}} \mu(l, l') \bar{Q}_i(l')$          ▷ Weighting Filter Outputs
$\dot{Q}_i(l) \leftarrow U_i(l) - \dot{Q}_i(l)$                          ▷ Compatibility Transform
$Q_i \leftarrow \frac{1}{2_i} \exp\left(\dot{Q}_i(l)\right)$                       ▷ Adding Unary Potentials
```

5 Code Overview

5.1 Inference

Our code is based on the CRFasRNN and the pytorch-segmentation repos. To reproduce the results of CRFasRNN, we modified the code to toggle between using the CRF layer and not using it. When not using the CRF layer, we used the unary potentials (logits) and pass them to the image labeling method. This is achieved by changing the forward method in *crfasrnn_model.py* from repo1. We comment out the forward through *crfrnn* layer and just return *out*, see appendix B.1. To use the backbones provided by GCN2, DeepLab, PSPNet, and UNet, we modify the same forward method by making it to take unary potentials (which we obtain from these backbones) in addition to the image, see appendix B.2.

5.2 Data Loading and Preprocessing

The CRFasRNN repo only shows how to do inference on one example. To efficiently use the available computing resources and time, we wrote a data loading and preprocessing pipeline that could be used to do batch inference. We were only able to use batch inference in without CRF mode, as the current implementation of the CRF layer doesn't support batched inference. The data loading and inference pipeline can be found in *evaluate.py* in the code submission.

In order to integrate the CNN from other works (referred to as backbones from now on) with the CRF layer from the *crfasrnn* model, several problems need to be fixed. First, we had to bring the CRFasRNN module as a submodule of the pytorch-segmentation repo. We created an inference file, *inference_final.py* in the code submission, where we import the necessary files from *crfasrnn* submodule. Secondly, these backbones' model outputs have mismatched dimensions with the input of the CRF layer. The outputs unary potentials/logits) from the other backbones had the same dimensions as the input image whereas the CRF layer expects a 500*500 image. To reconcile this mismatch, when passing the image through the backbone's forward method, we first resize it to 500*500 as the backbones could take images of any dimension. After we get the final prediction from the CRF layer, we resize it to its original dimension so that we can make a fair evaluation. In order to properly save the image with the correct dimension, we modified the save_image function by passing the original image size as a function argument. These modifications can be found in *inference_final.py* file in the code submission.

5.3 mIOU Calculation

Our baseline model included PyTorch implementations of inference only. Training and evaluation needed to be reimplemented. Rather than adapting the implementation of the mIOU metric from another project, a standalone re-implementation of the mIOU loss was selected.

mIOU is implemented in the TorchMetrics package as JaccardIndex, however the documentation is quite poor. The following idiosyncrasies needed to be learned through trial-and-error:

- The number of classes in mIOU calculation should be the total classes that appear in ground truth and prediction, not the total number of possible classes.
- The class index should not be higher than input parameter *num_classes*
- Ground truth labels include a white outline. This is not assigned to an class in the literature and should be ignored.
- The background class is black. It is not clear if this should be used for mIOU calculations or not. In our implementation, it is included.

Helper function "to_unique" in Appendix A is required to map classes indexes appearing in the dataset to the range from 1 to num_classes. Additionally, white image outlines are ignored. This function runs in O(n) time.

Helper function "metric" in Appendix A computes the mIOU for a single pair of predicted and target images.

Function "get_miou" in Appendix A computes the average mIOU for all images in a directory, where the target and predicted images share the same name.

6 Experiments

Generally, the existing fully convolutional nets have a relatively good prediction on the pixels at the center of an object, however, the prediction at the pixels close to the edge does not work so well. Incorporating an edge potential describing the affinity between pixels will help with labeling these pixels at the edge.

The first problem we need to solve for this project is that it is always difficult to train the CRF and backbone CNN nets jointly. And the existing work with training is on different ML frameworks that are beyond our availability. In this case, since the unary potentials only describe the probabilities of a pixel being assigned to each class by the information of the pixel itself, and the binary factor describes the correlation between pixels, it potentially allows for separating the training process. And by comparing the results with the whole model trained jointly and disjointly, the evaluated metrics do not decrease much. At this point, we trained our backbone fully convolutional layer and the CRF layer separately.

We used the pre-trained baseline full model to make the prediction and calculated the mIoU metrics. Then we figured out a way to remove the crf layer and made predictions based on just the FCN8s neural nets and calculate the metrics. Since the model had been fully trained jointly, rather than training the model on our own, we downloaded the pre-trained weights. We used the existing inference code to generate labels for each pixel and we wrote a standalone script to evaluate mIoU. Since the code was run on CPU, it took around 6s per segmentation prediction. The results can be found in the first column in table1. The FCN-CRF model outperformed the mere FCN model by around 6 percent. This result provided us with a general insight and understanding of the pre-existing work. By checking the output images, we found that the images predicted with the crf layer had a higher mIoU than the model without the crf layer. The main difference between these predicted images was the CRF has a more precise boundary pixel prediction.

The next step we took is trying to improve the model performance by getting a better unary potential. We did a review of the benchmark work on different neural networks. The Unet has a better performance on medical image segmentation while the deeplab v3 is the current SOTA. Since these works are reported and it is easily available from the libraries, so, we get downloaded these models instead of implementing them from scratch. Since the available computational resources are limited, we ran several pilot runs and found that after 20 epochs, most of the models were almost converged. So, we trained every model for 20 epochs to get the pre-trained model. The training processes were conducted on both GPU and CPU. It took approximately 1.5 hours on CPU for each epoch, while it took approximately 10 minutes per epoch on GPU training.

Then we had to integrate the models altogether and conduct the inference. There were two possible approaches that could potentially lead to the final result. One is to export the unary potentials for each figure and export the potential matrices for every image. The other method is just to save the model and the weights. Just following the standard procedure for engineering, we implemented the latter approach. We integrated these two models and generate the final results shown in table 1 and Figure 1. The results are evaluated from a subset of the val dataset which consists of 362 images. From the figure, we found that the baseline model has the highest mIoU than any other models even without the crf layer. The interpretation for this is the obtained weight is being trained for more epochs than any other model. But for other models trained with the same epoch, the PSPNet has the highest performance and the Unet is very hard to train. By adding the edge potential, all of the models have a worse mIoU value. It is probably because that binary potential was not trained simultaneously, and there is a mismatch of the value of these two. A better result would be obtained if we trained the CRF and the CNN layers jointly.

Table 1: Results using various backbones.

	CRFasRNN	**DeepLab3+**	**PSPNet**	**UNet**
Without CRF	75.89	49.63	56.47	17.05
With CRF	81.63	48.58	53.82	23.83

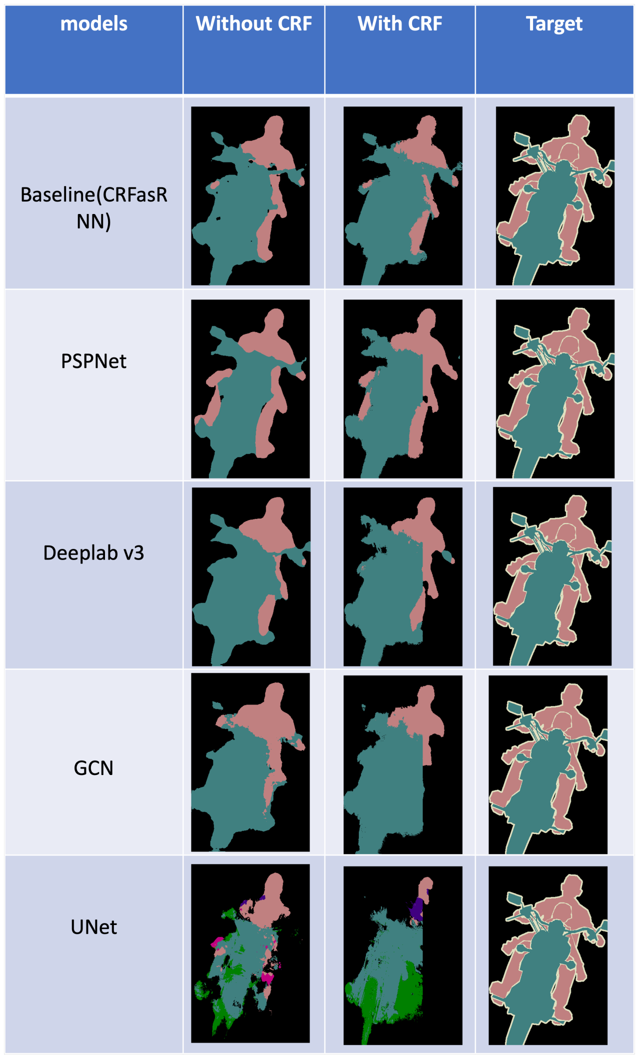

Figure 1: **Model outputs**

7 Timeline

A rough breakdown of our timeline is as follows. Note that all sessions were collaborative with all group members. Approaches taken include pair programming and asynchronous development.

Day 1, December 1 (4h x 3 people): Ideation, project proposal, CRF as RNN code analysis, literature review, project proposal writeup

Day 2, December 1 (1h x 3 people): Attend office hours and receive instructor feedback.

Day 3, December 5 (4h x 3 people): Getting both projects (CRF as RNN and PyTorch Segmentation) to run. Modifying CRF as RNN to report loss. Create detailed plan to finish project.

Day 4, December 6 (4h x 3 people): Write a dataloader for CRF as RNN. Re-implement intersection-over-union metric. Begin training models for unary potentials. Modify CRF as RNN to calculate both unary and binary potentials.

Day 5, December 7 (2h x 3 people): Integration of disjoint sections.

Day 6, December 8 (2-3h x 3 people): Creation of executive summary

8 Research Log

Our group proposed three experiments. We achieved results in two out of three.

1. Reproduce ablations reported in CRF as RNN [10] both with and without edgewise potentials.
2. Compare the performance of DeepLab v3+ [3] both with and without edgewise potentials. To achieve this, we proposed adding a CRF loss module.
3. Manually reimplement the CRF module and variational inference using Python.

We fully accomplished the first two experiments. While we were able to achieve our aims, the research was not fully a linear path. Some deviations from our initial plan occured due to unexpected difficulties.

Concretely, we encountered difficulties in the following areas:

1. Model training
2. CRF training

For model training, we lacked the computational resources to fully train all 8 models and binary variants. Instead, a pretrained model was used for the baseline (CRF as RNN) and trained all other unary models for just 20 epochs.

For CRF training, we expected PyTorch implementation of CRF as RNN [10] to include training code. However, it only included code for inference. As a result, we were unable to perform end-to-end training of semantic segmentation models.

Instead, we simply appended the pre-trained CRF module from CRF as RNN [10] to several models as a black box.

As a result of this, our final results is an apples-to-oranges comparison. We could not perform a fair comparison of CRF as RNN to other models.

Appendix A Code for Metric Calculation

```
def to_unique(array1, array2):
    # 256 is a large number, ignore the white (black class) and assign it to the largest label
    unique_map = {255:256}
    unique_idx = 0

    processed1 = []
    processed2 = []

    for i in array1.flatten().numpy():
        if i not in unique_map:
            unique_map[i] = unique_idx
            unique_idx += 1

        processed1.append(unique_map[i])

    for i in array2.flatten().numpy():
        if i not in unique_map:
            unique_map[i] = unique_idx
            unique_idx += 1

        processed2.append(unique_map[i])

    for (idx, j) in enumerate(processed1):
        if j == 256 or j == 255:
            processed1[idx] = unique_idx

    for (idx, j) in enumerate(processed2):
        if j == 256 or j == 255:
            processed2[idx] = unique_idx

    return torch.Tensor(processed1), torch.Tensor(processed2), unique_idx


def metric(image, target):
    p_image, p_target, n = to_unique(image, target)
    jaccard = torchmetrics.JaccardIndex(task="multiclass", num_classes=n, ignore_index=n)

    return jaccard(p_image, p_target)


def get_miou(target_dir, output_dir):
    target_names = os.listdir(target_dir)
    output_names = os.listdir(output_dir)

    t = torchvision.transforms.ToTensor()

    iou = []

    for i in tqdm(range(len(output_names))):
        name = output_names[i]
        if name[0] == ".":
            continue
        target_path = os.path.join(target_dir, name)
        output_path = os.path.join(output_dir, name)

        target = Image.open(target_path)
        target_size = target.size
        output = Image.open(output_path).resize(target_size)

        target = t(target)
        output = t(output)

        score = metric(output, target)
        iou.append(score)

    miou = np.array(iou)

    return np.mean(miou), np.std(miou)
```

Appendix B Inference

B.1 Modified Forward Method for CRFasRNN

```
    def forward(self, image):
        out = super(CrfRnnNet, self).forward(image)
        # Plug the CRF-RNN module at the end
        return self.crfrnn(image, out)
        # return out #when not using the crf
```

B.2 Modified Forward Method for other Models

```
    def forward(self, image, unary):
        #out = super(CrfRnnNet, self).forward(image)
        # Plug the CRF-RNN module at the end
        return self.crfrnn(image, unary)
```

References

[1] URL: http://host.robots.ox.ac.uk/pascal/VOC/voc2012/.

[2] Liang-Chieh Chen et al. *DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs*. 2016. eprint: arXiv:1606.00915.

[3] Liang-Chieh Chen et al. *Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation*. 2018. eprint: arXiv:1802.02611.

[4] Philipe Ambrozio Dias and Henry Medeiros. “Semantic Segmentation Refinement by Monte Carlo Region Growing of High Confidence Detections”. In: *ArXiv* abs/1802.07789 (2018).

[5] Philipp Krähenbühl and Vladlen Koltun. “Efficient Inference in Fully Connected CRFs with Gaussian Edge Potentials”. In: (2012). eprint: arXiv:1210.5644.

[6] Jonathan Long, Evan Shelhamer, and Trevor Darrell. *Fully Convolutional Networks for Semantic Segmentation*. 2014. DOI: 10.48550/ARXIV.1411.4038. URL: https://arxiv.org/abs/1411.4038.

[7] Chao Peng et al. *Large Kernel Matters – Improve Semantic Segmentation by Global Convolutional Network*. 2017. eprint: arXiv:1703.02719.

[8] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. DOI: 10.48550/ARXIV.1505.04597. URL: https://arxiv.org/abs/1505.04597.

[9] Hengshuang Zhao et al. *Pyramid Scene Parsing Network*. 2016. DOI: 10.48550/ARXIV.1612.01105. URL: https://arxiv.org/abs/1612.01105.

[10] Shuai Zheng et al. “Conditional Random Fields as Recurrent Neural Networks”. In: (2015). DOI: 10.1109/ICCV.2015.179. arXiv: 1502.03240.