Lab 4: Class Activation Maps and Weakly Supervised Semantic Segmantation

by Yicong Hong, ENGN8536

September 28, 2020

1 Introduction

In this lab, we are going to learn about an interesting and powerful idea called Class Activation Maps (CAM), which can be used to visualize (and control) the attention of convolutional networks over images, as well as assisting other tasks such as weakly supervised object localization and semantic segmentation.

Quote from the original paper of CAM:

"*In this work, we revisit the global average pooling layer, and shed light on how it explicitly enables the convolutional neural network (CNN) to have remarkable localization ability despite being trained on image level labels. While this technique was previously proposed as a means for regularizing training, we find that it actually builds a generic localizable deep representation that exposes the implicit attention of CNNs on an image.*"

— Bolei Zhou, et al. "Learning Deep Features for Discriminative Localization." [2]

Quote from the paper of weakly supervised semantic segmentation with CAM:

"*This work mainly explores the advantages of scale equivariant constrains for CAM generation, formulated as a self-supervised scale equivariant network (SSENet). Specifically, a novel scale equivariant regularization is elaborately designed to ensure consistency of CAMs from the same input image with different resolutions. This novel scale equivariant regularization can guide the whole network to learn more accurate class activation.*"

— Yude Wang, et al "Self-supervised Scale Equivariant Network for Weakly Supervised Semantic Segmentation." [1]

This lab consists of two tasks: In the first task, we will implement a very simple CAM network and train for Cifar-10 classification. In the second task, we will see how to extend the same network to achieve weakly supervised semantic segmentation with some slight modifications on the inputs and the loss function. If you want to have a concrete understanding of the methods, please read the two papers above.

Note that you need to use the provided dataset (pre-processed CIFAR-10) for this lab. You can download the data from this Google Drive Link or this Baidu Netdisk Link (extraction code: "`gudu`"). You can also find the dataset in our shared data folder in StuGPU2 server, named "`cifar-10-cam-seg-data`". Once you download the dataset, extract it in the "`data`" folder.

Your network can be set to one of the two modes: **CAM** for the class activation map in task 1 and **SEG** for the semantic segmentation in task 2. Please don't change these names.

2 Task 1: Implement the Class Activation Mapping (4 marks)

Before starting to write your network, you need to first go through the provided code in `main.py`, `train.py` and `model.py` to understand the overall framework. You also need to read `dataloader.py` to understand what data will be delivered when iterating through the dataloader.

2.1 Build the CAM Model (2.5 marks)

Quote from the paper of CAM [2]:

"*A class activation map for a particular category indicates the discriminative image regions used by the CNN to identify that category (e.g., CAM for class Australian terrier in Figure 1) ... The network largely consists of convolutional layers, and just before the final output layer (softmax in the case of categorization), we perform global average pooling on the convolutional feature maps and use those as features for a fully-connected layer that produces the desired output (categorical or otherwise).*" Please refer to Section 2 of the paper for more details.

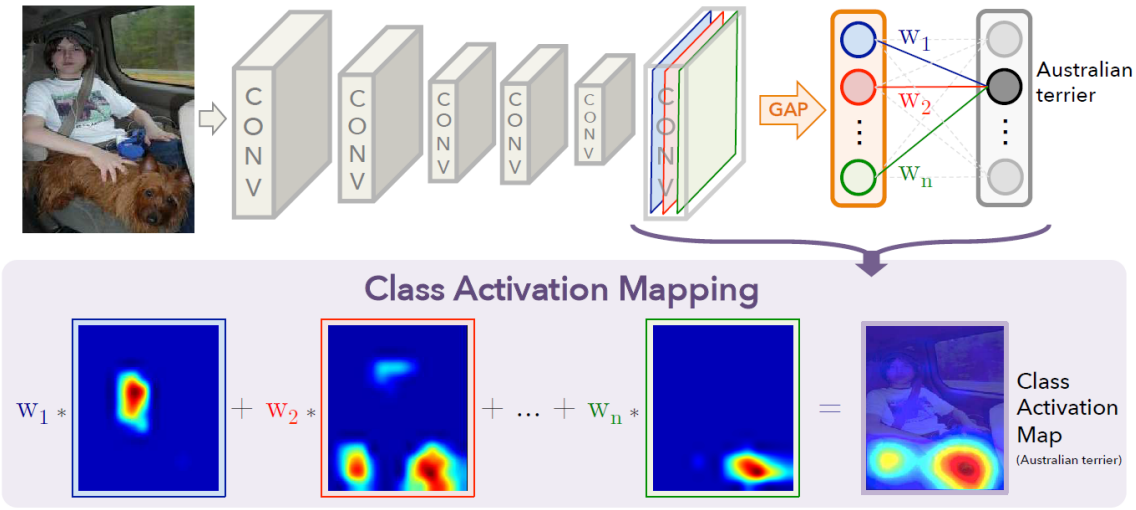

Figure 1: Class Activation Mapping: the predicted class score is mapped back to the previous convolutional layer to generate the class activation maps (CAMs). The CAM highlights the class-specific discriminative regions. [2]

First, define the CAM model in `model.py`, which only consist of three components:

1. A backbone convolutional network. Initialize a pretrained ResNet-18 model from torchvision.models, however, only use the model up to (Layer 2 - BasicBlock 1). You can print out the ResNet-18 model to see how each layer is named. Do not freeze it weights. (0.5p)
2. Initialize the Global Average Pooling (GAP) function which pools the entire feature map f_i of each channel i to a single weighting coefficient w_i. (0.5p)
3. Initialize a linear projection layer W_{CLS} which maps the weighting coefficients $\boldsymbol{w}$ (Dim128) to the class dimension (Dim10). (0.5p)

In the forward path of mode `CAM`:

1. Feed the input images into the CAM model following Backbone $\rightarrow$ GAP $\rightarrow W_{CLS}$. (0.5p)
2. Output the classification scores, the feature maps $\boldsymbol{f}$ of the last convolutional layer and the weighting coefficients $\boldsymbol{w}$. Just for a sanity check, there should be 128 feature maps each of dimension 28×28. (0.5p)

2.2 Train the CAM model (1.5 mark)

Now, move to `train.py`, initialize a Cross-Entropy Loss and compute the loss from the ground-truth labels and the output classification scores. (0.5p)

Train the network, make sure your network is in `CAM` mode. You should see the saved checkpoint which records the best network weights founded in `checkpoints` folder. Some suitable hyper-parameters have been provided, feel free to change them if you wish, but large epoch number is not necessary for this task. (1p)

3 Task 2: Weakly Supervised Semantic Segmentation with Class Activation Maps (11 marks)

Quote from the paper of Weakly Supervised Semantic Segmentation with CAM [1]:

"*The fully supervised semantic segmentation task can be generally formulated as* $f_{\theta_s}(x) = y$*, where* x *denotes the input image with the corresponding segmentation ground truth mask* y*.* $f_\theta(\cdot)$ *denotes the CNN based nonlinear mapping function. While for the image-level weakly supervised semantic segmentation, the network is attached with additional global pooling function* $P(\cdot)$ *to solve classification task as* $P(f_{\theta_c}(x)) = l$*, where* l *is image category label of* x*. Ideally, the network parameters* θ_s *and* θ_c *should keep the same if the fully supervision task and weakly supervision task have the same optimization objective ... Suppose there is an affine transformation matrix* A*, the fully supervised segmentation task requires the equivariance of affine transformation as* $f_{\theta_s}(Ax) = Ay$*, while the weakly supervision task focusing on the mapping invariance as* $P(f_{\theta_c}(Ax)) = l$*.*"

Intuitively speaking, the classification network and the segmentation network should focus on the same object region in the image, but just using supervision signal from classification task is too implicit for learning segmentation. Ideally, the segmentation map of an affine-transformed image $f_{\theta_s}(Ax)$ should be equal to the affine-transformed segmentation map of the original image $Af_{\theta_s}(x)$. Such equivariant property could be used to provide strong supervision signal for the segmentation task. Please refer to Section `Approach` of the paper for more details.

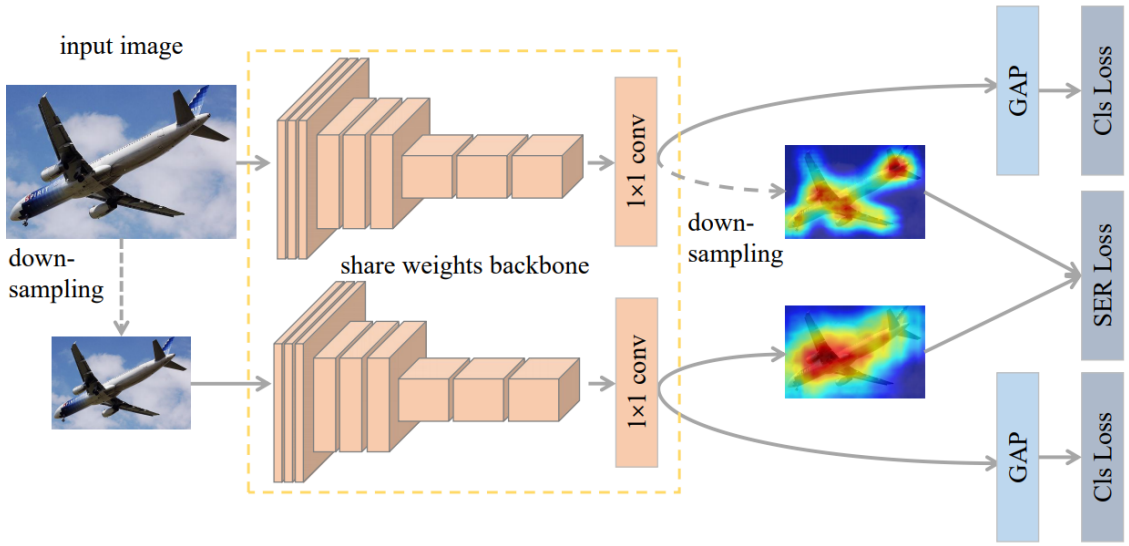

Figure 2: The two-branch architecture of SSENet. The SSENet takes the original and the down-sampled version of images as input, endeavoring to preserve the consistency of CAMs from each branch by scale equivariant regularization loss (SER Loss). GAP denotes the global average pooling layer. Cls Loss denotes the multi-label classification loss. [1]

3.1 Complete the forward path for `SEG` mode (1.5 marks)

Go to `model.py`, in the forward path of mode `SEG`:

1. Feed the original-size input images I_L and their down-scaled copies I_S into the same CAM model that we built in Task 1, following Backbone $\rightarrow$ GAP $\rightarrow W_{CLS}$. (0.5p)
2. Output the classification scores, the feature maps of the last convolutional layer and the weighting coefficients for both the original-size (I_L) and the down-scaled (I_S) inputs. For a sanity check, each feature map of the down-scaled input should be of dimension 14×14. (1p)

3.2 Implement the scale equivariant loss (3 marks)

In `train.py`, implement the following for `SEG` mode:

1. Compute the classification loss for I_L and I_S using the same Cross-Entropy Loss defined in Task 1, denote the losses as L_{cls}^L and L_{seg}^S respectively. (0.5p)
2. Downscale the large feature maps f_L to the same size as the small feature maps f_S (use scale factor 0.5, bilinear interpolation). (0.5p)
3. Do class activation mapping for f_L and f_S. You need to first apply a `Softmax` function over the weigting coefficients, then use those coefficients to weighted sum the corresponding feature maps to get the CAMs for all inputs (see Figure 1). (1p)
4. Compute the Mean Squared Error between the two set of CAMs you get from previous step, denote the loss as L_{seg}. (0.5p)
5. Define the final loss function as $L = (L_{cls}^L + L_{cls}^S)/2 + L_{seg}$. (0.5p)

3.3 Train the CAM model in `SEG` mode (1 mark)

Train the network, make sure your network is in `SEG` mode. Do not forget to change your `experiment ID`, otherwise the checkpoints from previous task will be replaced. (1p)

3.4 Visualization and IoU (2 marks)

Draw the output class activation maps using the provided `visualize.py` script.

1. Read and run `visualize.py` in `CAM` and `SEG` modes to obtain the CAMs and the segmentation maps generated by the two trained networks for 10 selected images. (0.5p)

   The selected images are in `data/test_seg_source` folder. The output images should be saved to the `visualize` folder. You may need to change the paths.

   You can modify the threshold value for binary image to get better results. But the value should be the same in two modes.

2. In your own words, explain what is Intersection over Union (IoU) and how to compute it. Implement IoU calculation in `iou.py`, the function should return IoUs for the 10 segmentation maps output from the previous step. (1p)

   The ground-truth segmentation maps are in `data/test_seg` folder.

3. Put all the visualizations and IoUs in your report. Explain and describe their difference when the network is in `CAM` or `SEG` mode. (0.5p)

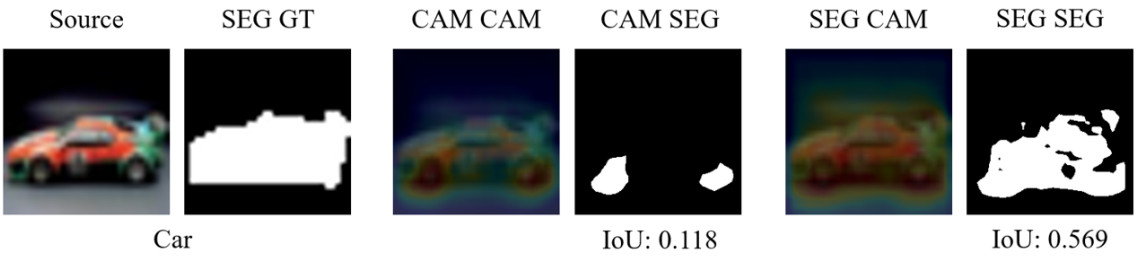

Figure 3: Visualization example.

3.5 Explore other transformations (3.5 marks)

An extended version of scale equivariant regularization is that the transformation A can be any spatial linear transformation, such as rotation, reflection and flipping. In this part, you need to apply a different linear transformation (not scaling) and retrain the above `SEG` network. (1.5p)

Compare the segmentation results quantitatively and qualitatively to the results you obtained from using scaling, can you see any difference? Explain your findings (why the results become better or worse or no difference?). (2p)

4 Submission

1. Please submit a lab report that clearly documents all the experimental results and answers to the questions, and attach all your code in Appendix.
2. The lab will be due on Sunday, 18 Oct 2020, 6:00pm.
   Name your submission as Lab_4_u0000000.pdf, replacing u0000000 with your uni-ID.
3. The full grade of this lab is 15 marks. This Lab worth 7.5% of the total course assessment.

References

[1] Yude Wang et al. "Self-supervised scale equivariant network for weakly supervised semantic segmentation". In: *arXiv preprint arXiv:1909.03714* (2019).

[2] Bolei Zhou et al. "Learning deep features for discriminative localization". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2921–2929.