Exercise Sheet 3: Detection and Segmentation

Due on 23.05.2025, 10:00

Pingchuan Ma (P.Ma@lmu.de)

Task 1: Object Detection (12P)

We have covered detection and segmentation in the lecture. Ideally, we would train a network like FCN on a dataset with ground-truth labels (e.g., bounding boxes or segmentation maps). However, due to limited compute and time, we will instead use the intermediate outputs of a pretrained classification network. This is feasible because such networks learn local object features to make global predictions. In this task, we will enable detection using a pretrained classifier, without adding parameters or retraining, only by leveraging gradient-based class activation maps (e.g., Grad-CAM), *assuming the target classes were seen during its original training.*

1. 1. Load the ResNet50[1] (or other variants) with its pretrained (on ImageNet-1k) weights and set it to eval() mode with 'torch.hub'. You don't necessarily move it to cuda(), cpu would be enough for this task. **Print out the name and layers contained in the model.** (2P)

2. For this, we will need to use *Grad-CAM* (https://arxiv.org/pdf/1610.02391) to estimate the location of objects from class scores. It uses two things: the *activation maps* and the *backpropegated gradient with respect to the class* that we are interested in. This will return us a weighted activation map that highlights the "important" pixels in the latent space. **read the paper or just get the gist of it with LLM**. (2P)

3. **Implement functions or code snippet to store the intermediate activation (forward pass) and gradient (backward pass) for the 'layer 4'** (see Fig.1), as it still has spatial resolution . Hint: you can search sth called 'hook'. (3P)

4. **Finish the code snippet of 'calculate_grad_cam',** where your input should be the activation and the gradient, and it should return a heatmap has high value on the location where the object is. **Visualize the Grad-CAM Heatmap and bounding box for our 'Chihuahua.jpg'** (3P)

[1] https://pytorch.org/hub/pytorch_vision_resnet/

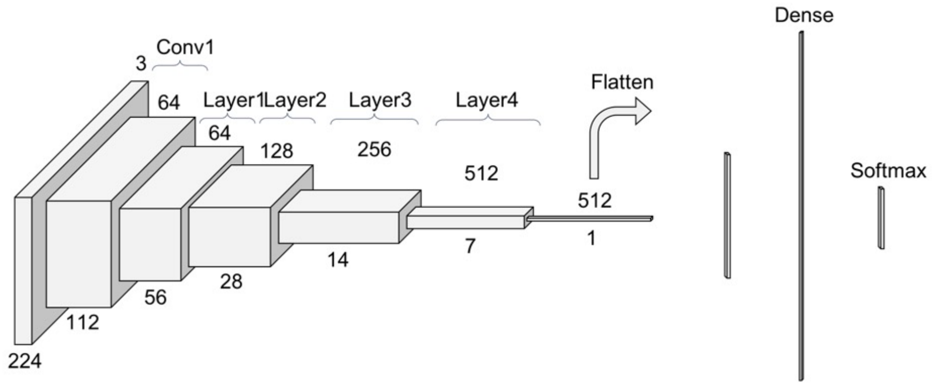

Figure 1: An overview of Resnet.

5. You can do another forward/backward pass, to **store the gradient for a different class, and write about how are the results are different.** (1P)

6. Similarly you can **do the same for all possible top-5 ranked prediction.** (1P)

Task 2: Segmention with a classification network (7P)

Similarly, we can turn the ResNet into a segmentation network without altering the model structure and additional training. We know from the task.1 that model has some internal knowledge about objects, but in a much lower spatial resolution space. Let's recall FCN in the lecture, which performs dense, per-pixel classification by transforming standard CNN classifiers into segmentation models. In a nutshell, we can do the following to achieve this:

1. **Load the pretrained ResNet and select the last conv layer.** This layer captures high-level semantic features useful for localization.(1P)

2. **Extract only activations during the forward pass.**(1P)

3. We use the classifier's FC layer weights as class templates (instead of the class gradient in the previous task). **For a given class, compute a weighted sum of the feature maps using the corresponding FC weights. This gives a class activation map, showing where in the image the model "looked" to make its prediction.** (2P)

4. Resize the low-resolution CAMs (e.g., from 7×7) to the input size (e.g., 224×224) using bilinear interpolation. This yields coarse, class-specific heatmaps. **Apply a threshold to the CAMs to produce binary masks for the most confident regions for each class.** You can combine multiple class masks to build a multi-class segmentation map (e.g., top-3 predicted classes). (2P)

5. **Visualize the segmentation** by color-coding the masks and blending them with the original image for interpretability.(1P)