AI6126 Advanced Computer Vision Assignment 2

Chen Yongquan (G2002341D)

Nanyang Technological University

Question 1

The authors of the original batch normalization paper noted that normalizing the inputs of a layer can limit the range of representations a layer can learn. The paper gave the example of normalizing the inputs of a sigmoid activation layer, which would then constrain them around the linear portion of the sigmoid function centered on 0. In convolutional neural networks where the ReLU activation function is most often used, normalizing inputs would mean that approximately half of all inputs would always be deactivated. Resultingly, this would affect the capacity of the network and it may not be able to train properly (Ioffe & Szegedy, 2015).

Thus, the authors proposed scaling and shifting the normalized value with the trainable γ and β parameters respectively. This allows the network the flexibility of adjusting the scale of the inputs further in proportion to the change in total loss with respect to this adjustment. In doing so, the network can even learn to reverse the batch normalization process if that is the optimal approach.

For instance, given input $x: N \times D$,

performing normalization $\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$,

then scaling and shifting $y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$,

where $\gamma_j = \sqrt{\sigma_j^2 + \varepsilon}$ and $\beta_j = \mu_j$,

we would arrive back at the original inputs before batch normalization.

Question 2

The main improvement of Faster R-CNN over Fast R-CNN is the use of a region proposal network instead of the selective search algorithm. In doing so, Fast R-CNN not only provides speed improvements orders of magnitude over Fast R-CNN, it also allows for the region proposal method to be trained which in turn translates to better mean average precision at test time. In the paper, using a region proposal network to generate 300 proposals resulted in a mAP of 59.9%, in contrast to using the selective search algorithm which generated 2000 proposals but only had a mAP of 58.7% (Ren, et al., 2016). Meanwhile, using a K40 GPU on a VGG-16 backbone network, using a region proposal network gave a total testing time of only $198ms$, giving 5 frames per second, while the selective search algorithm took an average of $1.5s$ by itself on top of $320ms$ for feature extraction and classification using Fast R-CNN with VGG-16, giving only 0.5 frames per second. When using the Zeiler and Fergus model as the backbone network, the testing rate was further improved, allowing for up to 17 frames per second real-time processing.

In the original paper, the loss function was defined to be the sum of a normalized classification loss and a normalized bounding-box regression loss with the latter being scaled by a hyperparameter λ, so that both losses are roughly equally weighted:

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i L_{cls}(p_i, p_i^*) + \lambda\frac{1}{N_{reg}}\sum_i p_i^* L_{reg}(t_i, t_i^*)$$

By default, λ was set to be 10 which gave the best mAP of 69.9%, while using values of 0.1, 1 and 100 gave 67.2%, 68.9% and 69.1% respectively. N_{cls} is the mini-batch size while N_{reg} is the number of anchor locations. p_i is the predicted probability that anchor i is an object, while p_i^* is the ground-truth label and takes a value of 1 when the anchor is an object and vice-versa. Thus, the bound-box regression loss is not considered when p_i^* is 0. t_i is the predicted box transform parameters for the box's center coordinates, width and height from the anchor box, represented as a vector:

$$t_i = (t_x, t_y, t_w, t_h)$$

Similarly, $t_i^*$is the transform parameters for the ground-truth box corresponding to that anchor box. The classification loss is further defined to be the log loss of two classes, one being the probability that anchor i is an object and the other that it is not:

$$L_{cls}(p_i, p_i^*) = -\log\left(\frac{\exp(p_{ij})}{\sum_j^2 \exp(p_{ij})}\right)$$

The bounding-box regression loss is the same robust loss function used in Fast R-CNN:

$$L_{reg}(t_i, t_i^*) = \sum_{j\in\{x,y,w,h\}} \text{smooth}_{L1}(t_{ij} - t_{ij}^*)$$

$$\text{smooth}_{L1}(x) = \begin{cases} 0.5x^2 & if\ |x| < 1 \\ |x| - 0.5 & otherwise \end{cases}$$

Question 3

In RoIPool, feature maps for each region proposals are extracted from the image features produced from the convolution backbone network by "snapping" the feature map grid to the image features grid. For instance, given a $3 \times 640 \times 480$ input image from which we obtain a $512 \times 20 \times 15$ feature map after convolution, and we want to extract the features for the region proposal at $(440, 264)$ of dimension 128×160. Because the feature map stride is 32, we cannot align the proposal grid onto the feature map grid perfectly. Thus, quantization is performed on the bounding box coordinates by dividing the coordinates by the stride and rounding off to a discrete value corresponding to the grid coordinates that are "snapped to" i.e. $(440, 264) \div 32 = (13.75, 8.25) \approx (14, 8)$. Similarly, because the extracted features are of dimension $512 \times 4 \times 5$, when subdividing them into a $512 \times 2 \times 2$ grid for max pooling, subregions along the y-dimension cannot be perfectly divided, resulting in differently sized subregions or omitted feature map cells, depending on the method of quantization. Consequently, because of the misalignments introduced by quantizing the region proposal bounding box coordinates, the segmentation masks produced by the network would also be less pixel-accurate.

Compared to RoIPool, RoIAlign does not perform any quantization or "snapping" to grids. Instead, the RoI is divided into equally sized subregions and directly superimposed onto the feature grid without alignment. In each subregion are regularly spaced sampling points and for each point the values from its four neighboring grid cells on the feature grid are bilinearly interpolated to give the sampled valued. The sampled values for each subregion are then pooled using max or average pooling to give the pooled RoI feature map. RoIAlign is preferred over RoIPool because there is no quantization in any part of the process and thus no misalignment error introduced into the final mask generated. Results from the original paper showed that using RoIAlign with max pooling gave a mask AP of 30.2 compared with 26.9 using RoIPool on the COCO test-dev set. (He, et al., 2018)

Question 4

The encoder-decoder architecture is widely used in semantic segmentation tasks instead of the alternative which is a fully stacked convolutional network. The latter has the problem of a linear increase in effective receptive field size with respect to the number of convolutional layers, so with L 3×3 convolutional layers and stride 1 for all layers, the receptive field after the L layers is $1 + 2L$. Therefore, in order to achieve a receptive field of more than 200, we would need at least 100 convolutional layers. Suppose we perform convolution with a stride and padding of 1 on an input dimension of 400×400, the output size for each layer would remain at 400×400. If we have 200 filters per layer, the memory consumed by the outputs from a single layer would be $200 \times 400 \times 400 \times 4 = 122MB$. When multiplied by the number of layers and a batch size of 20, the memory requirement for performing a single iteration would be $238GB$. This requirement would be even larger when segmenting high resolution images like satellite images and using larger batch sizes, making the stacked convolutional architecture economically unfeasible for most organizations using current technology.

The encoder-decoder architecture alleviates the problem by performing downsampling with the encoding layers then upsampling with the decoding layers, in order to reduce the memory consumption of the middle layers. Downsampling is performed using a combination of pooling and strided convolution, while upsampling is done either by unpooling through the bed of nails method, nearest neighbor, bilinear or bicubic interpolation, or by transposed convolution. Using larger strides in the encoding layers for convolution and pooling not only produce smaller output sizes but also increases the receptive field exponentially for the middle layers.

However, because of the downsampling performed by the encoding layers, there is a loss of spatial information in the middle and decoding layers, which results in misaligned segmentation masks or masks that are not very pixel-accurate. This can be fixed by adding skip connections directly between corresponding downsampling and upsampling layers of similar resolutions which copies high resolution feature maps from the encoding layers to the decoding layers. The original paper did not use padding for convolutions so there is a loss of border pixels with each convolution. The copied feature maps are therefore cropped to match the dimension from the transposed convolution process. By combining the high-resolution features with the upsampled output, successive convolutional layers can learn to assemble a more pixel-accurate output. (Ronneberger, et al., 2015)

Question 5

Resulting receptive field for consecutive 1-dilated, 2-dilated, 4-dilated and 8-dilated 3×3 convolution is:

$$2^i\text{-Dilated Feature Map Size} = \left(2^{i+2} - 1\right)^2$$

$$2^3\text{-Dilated Feature Map Size} = (2^{3+2} - 1)^2 = 31 \times 31$$

Alternatively,

$$R_k = 1 + \sum_{j=1}^{k} \left[\left(F_j - 1\right)\left(D_j\right) \left(\prod_{i=0}^{j-1} S_i \right) \right]$$

$$= 1 + (2 \times 1 \times 1) + (2 \times 2 \times 1) + (2 \times 4 \times 1) + (2 \times 8 \times 1)$$

$$= 31$$

Question 6

Even with dilated convolution, convolutional neural networks can still encounter difficulty in learning to recognize the same object when the test data or real-world sample is of a different geometric variation or transformation of the training data. For instance, the testing input may be of a different scale, in a different pose, from a different viewpoint or has some part deformations. Therefore, we perform data augmentation to obtain more training data that can model some of these geometric transformations. Augmentations include randomly cropping and scaling, adding color jitter or lens distortions, or performing affine transformations, etc. on the original training data. However, having more training data would likewise increase training times and would still only encompass the transformations applied on the original training set, thus it still may not generalize to more complex transformations. Alternatively, on top of training data augmentation, a spatial transformer network can be used in front of the convolutional backbone network, which learns to crop and scale-normalize the object in the input image and subsequently give better classification accuracy in the convolutional network (Jaderberg, et al., 2016).

Deformable convolution is a generalization of dilated convolution that allows for geometric transformations to be addressed in the network parameters themselves by adding 2D offsets to both the convolutional kernel sampling points and the RoI pooling bin positions (Dai, et al., 2017). These offsets can be trained end-to-end through backpropagation, simultaneously with the convolutional filter weights. Because these offsets can be fractional, bilinear interpolation is used for sampling from the input during convolution and for RoI pooling. For deformable RoI pooling, regular RoI pooling is performed first in a separate branch to produce regularly pooled feature maps. These feature maps are then fed into a fully connected layer to produce normalized offsets that are then scaled by a constant $\gamma = 0.1$, to give the actual offsets used for deformed RoI pooling in the main branch. Offset normalization allows for offset learning to be scale invariant to the size of RoI. The added parameters and computation are relatively lightweight compared with using a spatial transformer network or traditional transformation invariant feature engineering algorithms, such as SIFT (Lowe, 1999), in front of the convolutional backbone. Meanwhile, using deformable convolutions gave around 10% better mAP scores relative to those of regular convolutional networks when tested on the COCO test-dev set with class-aware RPN, Faster R-CNN and R-FCN (Dai, et al., 2016).

References

Dai, J., Li, Y., He, K. & Sun, J., 2016. *R-FCN: Object Detection via Region-based Fully Convolutional Networks.* [Online]
Available at: https://arxiv.org/abs/1605.06409

Dai, J. et al., 2017. *Deformable Convolutional Networks.* [Online]
Available at: https://arxiv.org/abs/1703.06211

He, K., Gkioxari, G., Dollár, P. & Girshick, R., 2018. *Mask R-CNN.* [Online]
Available at: https://arxiv.org/abs/1703.06870

Ioffe, S. & Szegedy, C., 2015. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.* [Online]
Available at: https://arxiv.org/abs/1502.03167v3

Jaderberg, M., Simonyan, K., Zisserman, A. & Kavukcuoglu, K., 2016. *Spatial Transformer Networks.* [Online]
Available at: https://arxiv.org/abs/1506.02025

Lowe, D. G., 1999. *Object recognition from local scale-invariant features.* s.l., ICCV, p. 1150–1157.

Ren, S., He, K., Girshick, R. & Sun, J., 2016. *Faster R-CNN: Towards Real-Time Object.* [Online]
Available at: https://arxiv.org/abs/1506.01497

Ronneberger, O., Fischer, P. & Brox, T., 2015. *U-Net: Convolutional Networks for Biomedical Image Segmentation.* [Online]
Available at: https://arxiv.org/abs/1505.04597