282 Scribing: Lecture 12

Jiayang Nie, Ophelia Wang

October 2022

1 Review: Skip Connections and ResNets

From the gradient perspective, residual connection blocks help address the issue of vanishing gradients. The key to residual block is in the gradient formula: $\frac{dH}{dx} = \frac{dF}{dx} + I$. Thus, the baseline gradient is 1 and can avoid gradient vanishing. The pesudo-code is in Figure 1. Through skip-connection, a deeper network yields better accuracy unlike what happens in vanilla convNet, as shown in Figure 1.

2 Speeding Up Training

- Use larger batch size, and linearly scale learning rate to speed up the training process.
- With larger batch size and possibly insufficient memory, use distributed data-parallel training by splitting one batch to different machines.
- To avoid over-fitting and obtain a better result in the test set, regularize by aggressive data augmentation.

3 Example-Difficulty of Samples in CNN

C-score is defined as

$$C_{p,n}(x, y) = E_D[P(f(x; D \setminus \{x, y\}) = y] \tag{1}$$

x, y are the design matrix representing pixels and the label respectively, and D are n i.i.d. samples from some population. We can think it as an analogy to leave-one-out validation. For example, as shown in Figure 2, the easier for the model to correctly identify an object, the higher the C-score would be. This is also can be thought of as consistency profile: the number of samples in the dataset with the same label that look like this image. For instance, the Game of Throne chair has a smaller consistency in compare to a regular chair in terms of identifying a chair. This gives us a proxy of how hard it is to identify one sample correctly.

In general, C-score increase as n increase, and C-score converge to 1 as n goes to infinity.

4 Depth-Complexity

Prediction depth is defined as the earliest layer in the model that classifies the sample correctly by KNN. Prediction depth is a good proxy for example difficulty. In general, examples with higher C-score (easier examples) have a small prediction depth and can be learnt in earlier epochs.

More formally, prediction depth is defined as:

$$argmin_{l \in L} f(x, \theta_l) = f(x, \theta_{>l}) \tag{2}$$

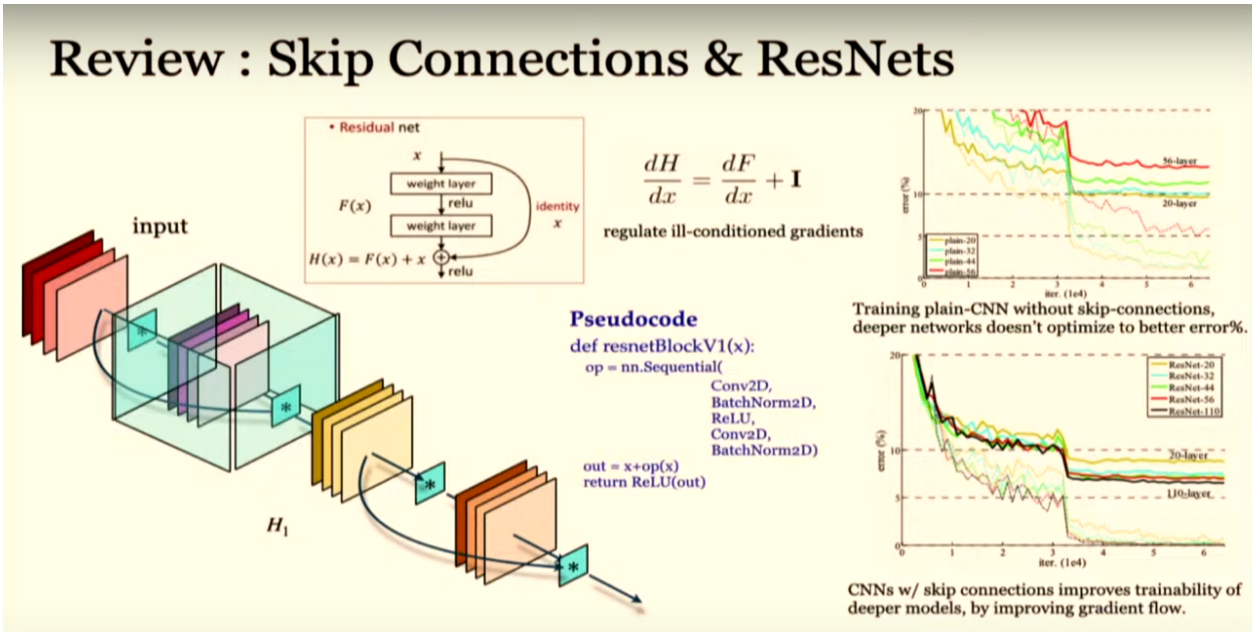

Figure 1: ResNet Layer Review [1]

5 Dense Prediction with Convolutions: Object Localization

In sparse prediction, the task is to predict the class of an image. In contrast, the task of object localization is to not only classify the image to a class but also detects which area of the object belonging to the class is in the image.

5.1 Problem Setup & Measurement Metrics

Previously we have $D = \{x_i, y_i\}$ where x_i represents the image and y_i represents the label. Now we have $D = \{x_i, y_i\}$ where x_i represents the image and y_i is a vector of (x_i, y_i, w_i, h_i) where (x_i, y_i) is the top left corner of the bounding box of the object, w_i represents the width and h_i represents the height. A common metrics for measuring performance is Intersection over Union(IoU). Assume we have a ground-truth box that locates the object, and the model predicts another box, IoU is the ratio of the intersection area(I) over the union area(U) of the ground-truth box and the predicted box. Usually, we declare that the model predicts correctly if IoU ≥ 0.5 and predicted class is right. Figure 3 gives a concrete example of what IoU is: the red area is the ground truth box, and purple area is the predicted box. IoU in this case is the area of the intersection of red and purple boxes over the area of the union of red and purple boxes.

5.2 Naive Approach

The most straightforward approach for solving the problem is to first train the classifier with cross-entropy loss, and then train a regression model on top of the convNets to learn the location of the box with Gaussian log-likelihood or MSE.

5.3 Sliding Windows Approach

A better approach is to classify every patch in the image. E.g. stretching and dividing an image into multiple sliding windows. Then then we could output the box with the highest class probability. In Figure 4, the original image of a cat is stretched vertically and horizontally so that we can find the optimal box that covers the cat object in this image. The key reason for stretching is to not limit the box to a fixed size box. For instance, for this example, the middle red box from the bottom image from Figure 4 is the best

Figure 2: C-Score Example [1]

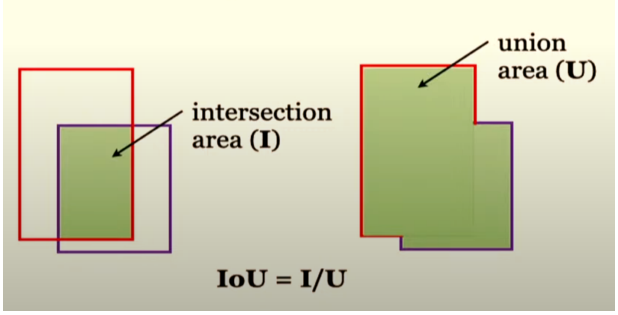

Figure 3: IOU Example [1]

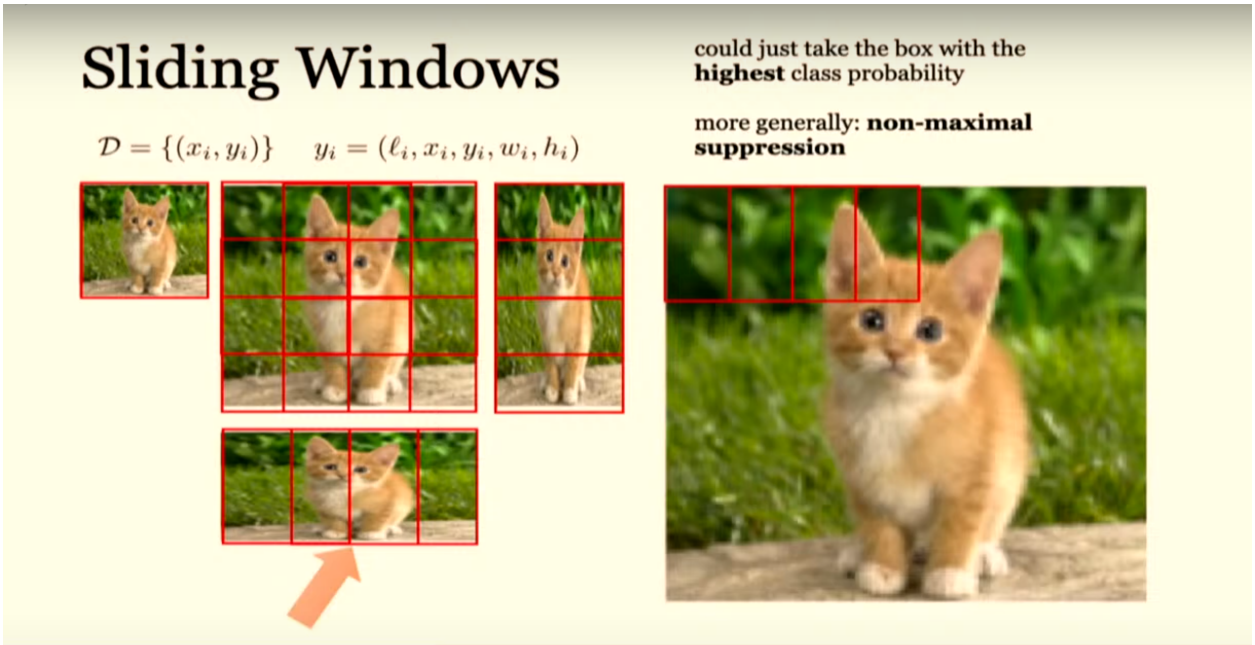

Figure 4: Sliding Windows [1]

bounding box. However, if we do not stretch it, then there does not exist a bounding box that is as good as this one in the original image. [Answer to audience's question]: In this example of find the cat, we should use tall and thin rectangles as the bounding boxes. Once we figured out which patch is the best patch, we can get the corresponding the corresponding bounding box in the original image by undoing projection. Then we can return the patch or the bounding box with the highest probability being a cat. In the case of localization, once we have all the predictions from the bounding boxes, the next question is that which one should we pick. In the case of localization when we know there is only one object in the image, then we can pick the one which has the highest probability. However if we are doing multi-object localization, then there are other algorithms such as non-maximal suppression. We basically say that we have some threshold and look particularly at a neighborhoods and pick objects corresponding to one particular class. [Answer to audience's question]: We look at different scales of the image and then run sliding window over that.

5.4 A Practical Approach - OverFeat

Combining the idea of regression and sliding window together, the approach of OverFeat provides a little "correction" to sliding window by adding small adjustments to the vector defining the bounding box. [3] We do the sliding window trick, but instead of predicting just the class, we also predict a bunch of coordinates, which can be think of as little corrections to the bounding box. First we can do a pre-train with the classifier, and then train the regression head on top of classification features. By passing over different regions at different scales, we can take an average of all the boxes as the final answer. Doing sliding window is expensive. Hence, the more practical way is to implementing convolutional layers to recuse calculations across windows.

Figure 5 shows an example adapted from the original paper of Overfeat. In combine of all the classification and regression heads from the sliding windows, the model is able to find the area in the image that it is most confident that it is a bear.

One of the downsides of using this approach is the increased computation complexity (36 windows = 36x the compute cost). To solve this, one reuse the calculations as shown in Figure 6. Fully connected layers are size-1 filter convolutional layers in disguise. Therefore, instead of using fully connected layers for classification task, we use convolutional layers. The benefit here is that when we scale the image, each convolutional layer will give more than 1 output, and we can reuse each of them for other windows. Assume

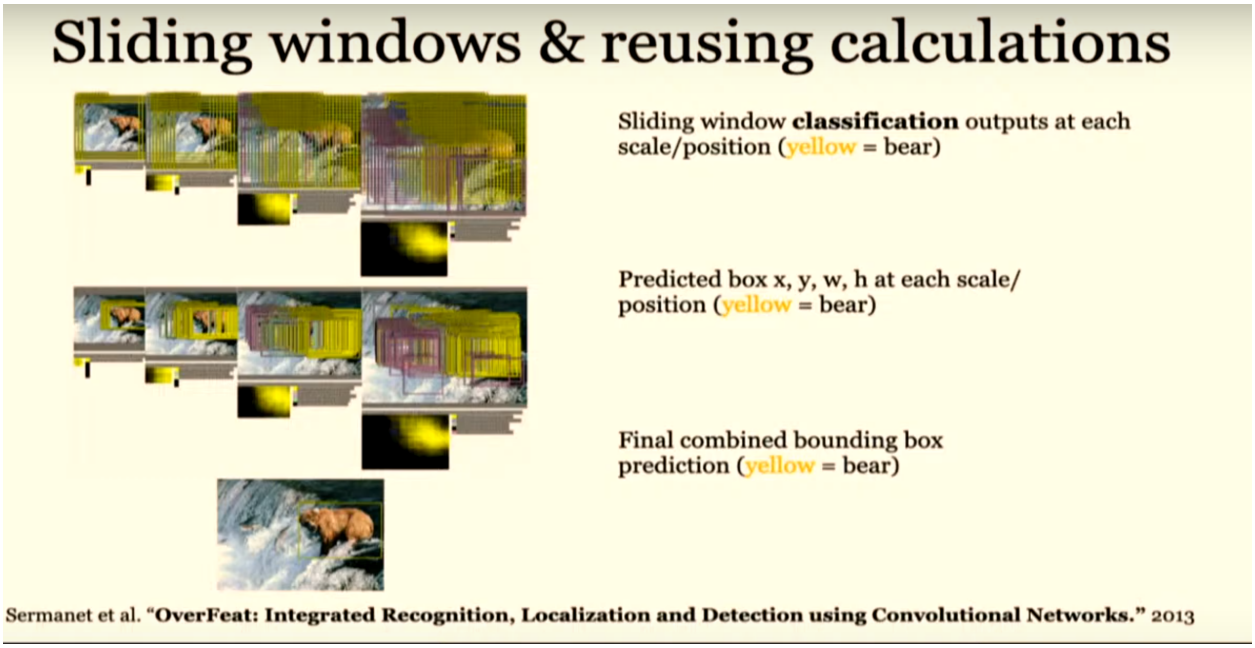

Figure 5: Overfeat Example [3]

we were running classifier on 14 x 14 image, by upsampling the image to be 16 x 16, the output dimension changes from 1 x 1 to 2 x 2. In this way, the computation time can be reduced to be about the same as convNets without sliding windows.

In summary, the building block is the convolutional network that outputs class and bounding box coordinates. Instead of looking at the combinatorial version of the problem, we look at different patches by upsampling and using a sliding window technique where we can still predict the probability of the class and the bounding box. To implement the sliding window effectively, what we do is instead of implementing the classifier with fully connected layers, we implement it with convolutions. Implementing the sliding window as just another convolution, with 1 by 1 convolutions for the classifier or regressor at the end to save on computation.

6 Object Detection

Then let's move on to object detection. Now the problem set up is slightly different. It is similar to the localization problem, but there is a added level of complexity where before we looking at the image and we have to predict details of one object. A more realistic setting is you want to identify all the objects in the image. The number of objects may be different for different images.

6.1 Dense-Prediction: Generating Multiple Outputs

One solution is: instead of making prediction for one class, we make predictions for all classes and predicting bounding boxes correspondingly. Each window can be a different object. Instead of selecting the window with the highest probability, just output an object in each window above some threshold.

6.2 Case Study

In Figure 7, this is a case study of the algorithm YOLO [2]. This algorithm provides a different take at the sliding window trick that we have seen in the lecture. The algorithm proposes that we only look at the image only once. Take this image and convert this into seven by seven grids in this example. For each of the grid, we predict what the bounding box is, what the class label is and additionally the confidence in our

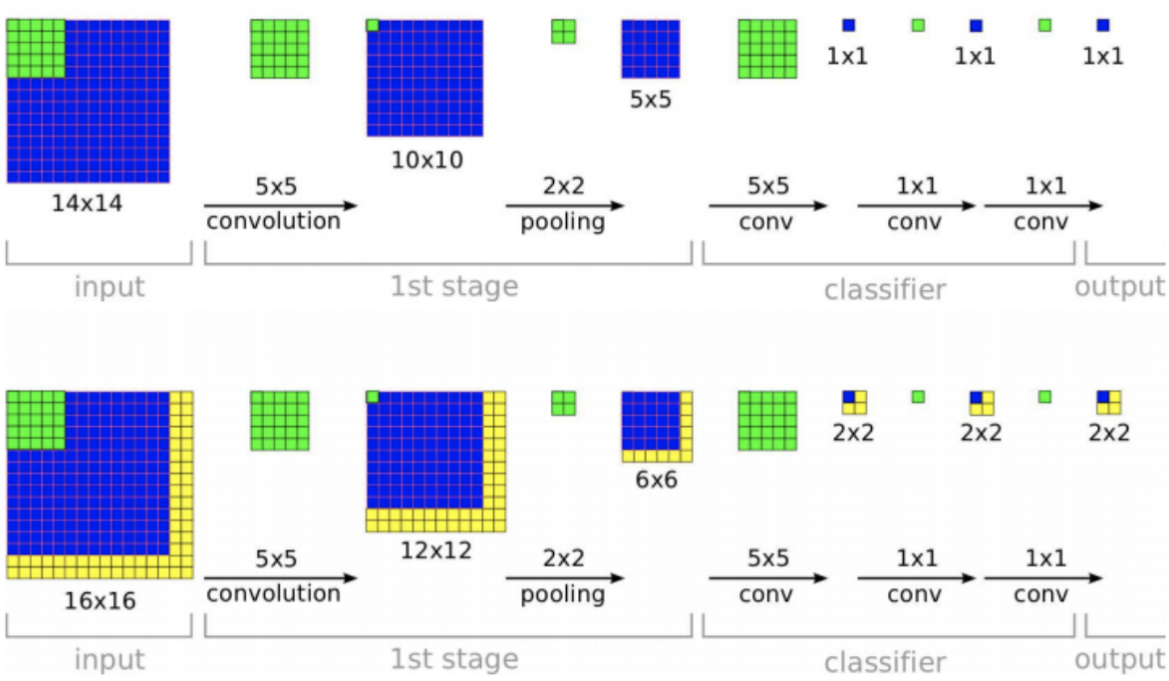

Figure 6: Convolutional Layers

own predictions such as IoU. For class label, we can use probability as a proxy. For the bounding box, we don't have any proxy of the confidence of the model.

6.3 CNNs + Region Proposal Networks

There is a different class of algorithms. Originally we pre-define which part of the image we should look at. Do we really need to do this before-hand or can my network learn to even predict where I should be looking at. The region proposal networks achieve that. The algorithm takes in an image as an input, then extract about 2 thousands region proposals. For each region proposal, the algorithm computes CNN features and then classify each region. This is a smarter sliding window technique. Instead of running sliding window on the region proposal on the input image, we first do the heavy lifting where we run the image through a coordinate for some layers. We get some activations and then run region proposal on top of these activations. How should we train the region of interest proposals? It's a very similar design to what we saw before such as OverFeat and Yolo, but now we predict if any object is present around that location. To build efficient detectors, we can use feature pyramids. We look at the image at different resolutions. We can think of it as am image pyramid where you have the lowest resolution version at the top and the highest resolution at the bottom. You can try model at each of the resolution and then generate predictions from each of these features and then pull them in some way such as looking at the max of those two to get your prediction. The key idea is to aggregate information across multiple scales.

6.4 Compute Efficient Detection

There are other different architectures people have run automated search for finding how we should connect these. Compared to mast RCNN and YOLO, the cost of training the Efficient Det models really pushes the boundary.

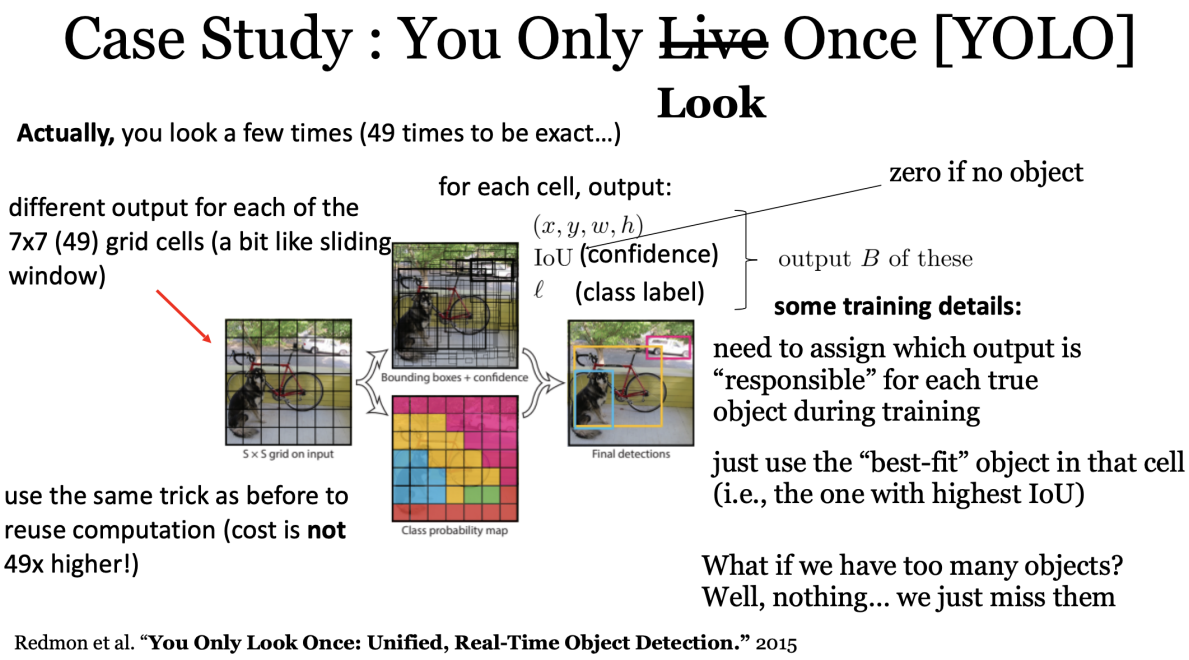

Figure 7: Case Study: You Only Look Once [2]

7 Semantic Segmentation

Previously we talked about how to locate one object inside an image. Now we would like to detect all objects in an image and label every single pixel with its object class.

7.1 Problem Setup

Assume there are K objects/classes inside an image, semantic segmentation is a K-class classification algorithm, predicting a label per pixel $y_i \in \{c_1, c_2, ..., c_K\}$. To make it computationally efficient, we can design a network a network architecture such as a fully convolutional network. We aim to have a set of operations that preserve the resolution at output. However, this is constrained by the fact that effective receptive field of convolution filters grows with depth. Only the early layers have the local view.

7.2 Conv. Operations: Down/UnSampling

Different convolutions operations can be used in the fully connected networks. There are normal convolutions, which reduce resolution with stride, padding, dilated convolutions, which increase receptive-field more rapidly, and transpose convolutions, which increase resolution with fractional stride. There are a lot of ways to design layers. One example is the Max Pooling, which remember the max element. In Max Unpooling, we set zeros at every other place and one only at the points which are used in the Max Pooling operations.

7.3 Bottleneck Architecture

As we get deeper in the network, we start to see a much bigger piece of the picture. Once we reach to low resolution, we can start up-sampling and turn low resolution feature vectors into high resolution per-pixel predictions. This is called a bottleneck architecture because we go from a high resolution and come to a low resolution with more depth and then go back to high resolution. Down and Up-sampling can be achieved with different kinds of convolution methods such as normal convolutions, dilated convolutions and transpose convolutions.

7.4 U-Net Architecture

When we down-sampling, we lose information, which cannot be recovered by up-sampling. The intuition behind U-Net architecture is to explicitly append filters from earlier down-sampling layer that preserve high-frequency details, concatnating them with filters along the channel dimension.

References

[1] Kumar Krishna Agrawal. "Guest Lecture: Kumar Krishna Agrawal". In: (2022). DOI: https://inst.eecs.berkeley.edu/~cs182/fa22/assets/slides/cs182lecture12vision.pdf.

[2] Joseph Redmon et al. "You Only Look Once: Unified, Real-Time Object Detection". In: (2015). DOI: http://arxiv.org/abs/1506.02640.

[3] Pierre Sermanet et al. "OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks". In: (2014). DOI: https://arxiv.org/abs/1312.6229.