SFU

Practices in Visual Computing II (Spring 2023)
Assignment 1
Object Detection with SSD

Total points: 100 + 5 points

Due: Thursday, 2nd February, 23:55 pm

Overview

In this assignment, you will be implementing SSD (Single Shot MultiBox Detector), a type of object detection network. Similar to YOLO, SSD divides the image into grids and outputs bounding boxes in each cell. However, SSD is more complex as it divides grids at different scales and uses default bounding boxes as anchors.

It is worth noting that the SSD you will be implementing in this assignment is a simplified version. If there are any discrepancies between the instructions provided in this assignment and the SSD paper, please follow the instructions provided in the assignment. Additionally, you can learn more about the task by reading the **SSD paper**.

1 Introduction (0 Points)

The main difference between YOLO and SSD is the default bounding boxes. In YOLO, we divide the image into 5x5 grid cells. Each cell has its confidence, or probabilities for each class (P(cat), P(dog), P(person), P(background)). For SSD, each cell will provide some default bounding boxes. In this assignment, each cell will provide four default bounding boxes as follows:

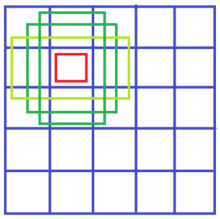
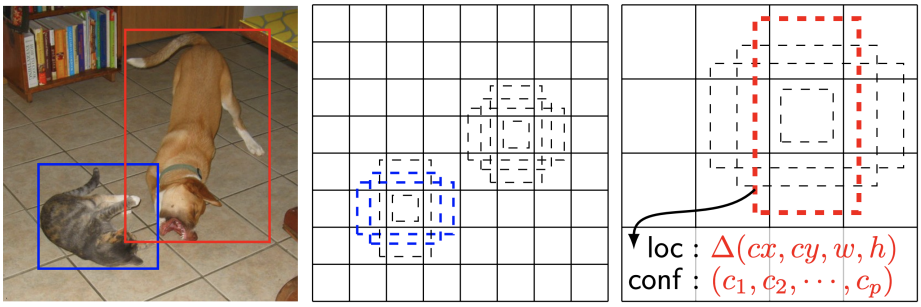


Figure 1: Default bounding boxes of cell [1,1]

As you see, if for YOLO, we have at most 25 bounding boxes, then for SSD, we will have $25 \times 4 = 100$ bounding boxes. Furthermore, we will introduce grids for multiple scales. In YOLO, we only have one 5x5 grid in the output layer, but for SSD in this assignment, we will have four output branches: 10x10, 5x5, 3x3 and 1x1, a total of 135 cells and 540 default bounding boxes.



(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map

We aim to match **ground truth bounding boxes** to some of our pre-determined **default bounding boxes**. This will ensure that the default bounding boxes are accurate estimates of the ground truth bounding box. We can then use these default bounding boxes as "anchors" to predict the relative positions and sizes of objects in the image. That is, we do not need to predict the absolute width and height as in YOLO;

we can just predict the relative positions and sizes with respect to the default bounding box, as shown in the above figure, $\Delta(cx, cy, w, h)$. In all, the main challenge in this assignment (SSD) compared to the tutorial (YOLO) is how to generate the default bounding boxes and how to assign ground truth objects to those default bounding boxes.

2 Workspace Initialization (0 Points)

Download the dataset from **here** and extract it in the `data/` directory.

3 Generating default bounding boxes (10 Points)

As mentioned, in this assignment you are going to work on $(10 \times 10 + 5 \times 5 + 3 \times 3 + 1 \times 1) \times 4 = 540$ default bounding boxes. The first step is to generate them.

You need to complete the function `default_box_generator` in `dataset.py`. The function takes a series of parameters and eventually outputs a 540×8 array, storing 540 bounding boxes. The last dimension 8 means the 8 attributes of each default bounding box:
`[x_center, y_center, box_width, box_height, x_min, y_min, x_max, y_max]`

Note that here, all values are absolute positions and sizes. For example, if a default bounding box has a center (x=0.3, y=0.4), width=0.1 and height=0.2, the attributes you need to store are:
`[0.3, 0.4, 0.1, 0.2, 0.25, 0.3, 0.35, 0.5]`

You need to generate 4 default bounding boxes for each grid cell in each grid (10, 5, 3, 1), using the provided scales large_scale and small_scale to determine the sizes of the default bounding boxes. There may be bounding boxes exceeding the image boundary, therefore you may clip them so that the bounding boxes stay inside the image.

For example, consider filling the default boxes for the first cell. The size of the grid is 10 since layers[0]=10. The scale of the three large boxes is lsize = large_scale[0] = 0.2. The scale of the one small box is ssize = small_scale[0] = 0.1. The first cell is (0,0) in a 10x10 cell grid.

For each cell;
Generate a box with width and height `[ssize,ssize]`.
Generate a box with width and height `[lsize,lsize]`.
Generate a box with width and height `[lsize*sqrt(2),lsize/sqrt(2)]`.
Generate a box with width and height `[lsize/sqrt(2),lsize*sqrt(2)]`.

All the four above boxes are centered at the center of the first cell (0.5/10, 0.5/10) The four boxes you get for the first cell is:

```
[0.05, 0.05, 0.1, 0.1, 0, 0, 0.1, 0.1]
[0.05, 0.05, 0.2, 0.2, -0.05, -0.05, 0.15, 0.15]
[0.05, 0.05, 0.28, 0.14, -0.09, -0.02, 0.19, 0.12]
[0.05, 0.05, 0.14, 0.28, -0.02, -0.09, 0.12, 0.19]
```

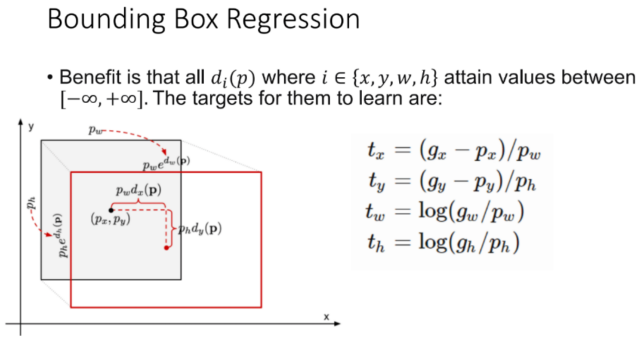
The four boxes after clipping (optional) is:

```
[0.05, 0.05, 0.1, 0.1, 0, 0, 0.1, 0.1]
[0.05, 0.05, 0.2, 0.2, 0, 0, 0.15, 0.15]
[0.05, 0.05, 0.28, 0.14, 0, 0, 0.19, 0.12]
[0.05, 0.05, 0.14, 0.28, 0, 0, 0.12, 0.19]
```

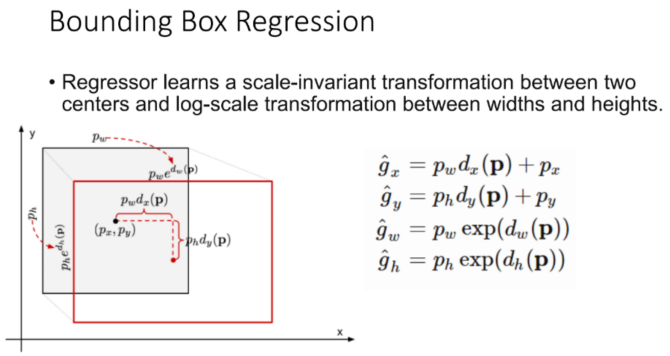
You need to create boxes for all cells in a similar manner. You can modify the box sizes in `large_scale` and `small_scale` to see if you get better results when training the network.

4 Assigning ground truth objects to default bounding boxes (15 Points)

This part is done in the dataloader in `dataset.py`. You need to read the image and ground truth bounding boxes, and then return the resized and transposed image, the ground truth probabilities `ann_confidence`, and the ground truth bounding boxes `ann_box`. `ann_confidence` is 540x4, since we have 540 default boxes and 4 classes (cat, dog, person, background). `ann_confidence` should be 540 one-hot vectors. `ann_box` is 540x4, since we have 4 attributes for a bounding box:
`[relative_center_x, relative_center_y, relative_width, relative_height]`
The attributes are all relative to the default bounding box as follows:



Suppose we have a default box `[px,py,pw,ph]`, which are exactly the first four attributes of our generated default bounding boxes. Also, we have a ground truth object bounding box `[gx,gy,gw,gh]`. The 4 attributes we need for `ann_box` are `[tx,ty,tw,th]`, the relative positions and sizes of the ground truth bounding box with respect to the default bounding box. If you need to recover the ground truth bounding box, or to show the predicted bounding box of your network, you can do an inverse process as follows, where `[dx,dy,dw,dh]` is the predicted relative attributes from your network.



Then, how to determine if a default bounding box is carrying an object? Different from YOLO, this time you need to assign one object to multiple default boxes. We will say a default box is carrying an object, if the ground truth bounding box of this object has an IOU greater than a threshold (0.5 in this assignment) with the default box. There may be cases where the ground truth box does not have sufficient overlap with any of the default boxes, in that case, we assign the object to the default box that has the largest IOU with the object bounding box, to make sure at least one default bounding box is used for each object.

You need to implement a function match to process each bounding box and update the entries of ann_confidence and ann_box. You only need to fill in the bounding box attributes for boxes that carry objects. For empty boxes, you can ignore them since they are not used when training the network.

Other important things to do:

1. You should split the dataset into 90% training and 10% validation. You can use all images for the training of course, but you will not know whether your network has overfitted without a test or validation set. Your model may have very good performance on the training set, but have very poor performance on the testing set. Keep in mind that both training accuracy and testing accuracy are considered when grading your assignment.

2. Data augmentation. Usually the networks are trained on millions of images. It is uncommon to train an object detection network on only 6000 images. But we have to reduce the data size so you can finish training within 2 hours. Therefore you have a high chance of overfitting on the training set. You will need to augment the data to mitigate that.

5 The network (15 Points)

Please open `model.py` and implement the network on the next page. The network is the same as YOLO until you reach resolution 10x10. Be sure to include a bias term for your convolution layer.

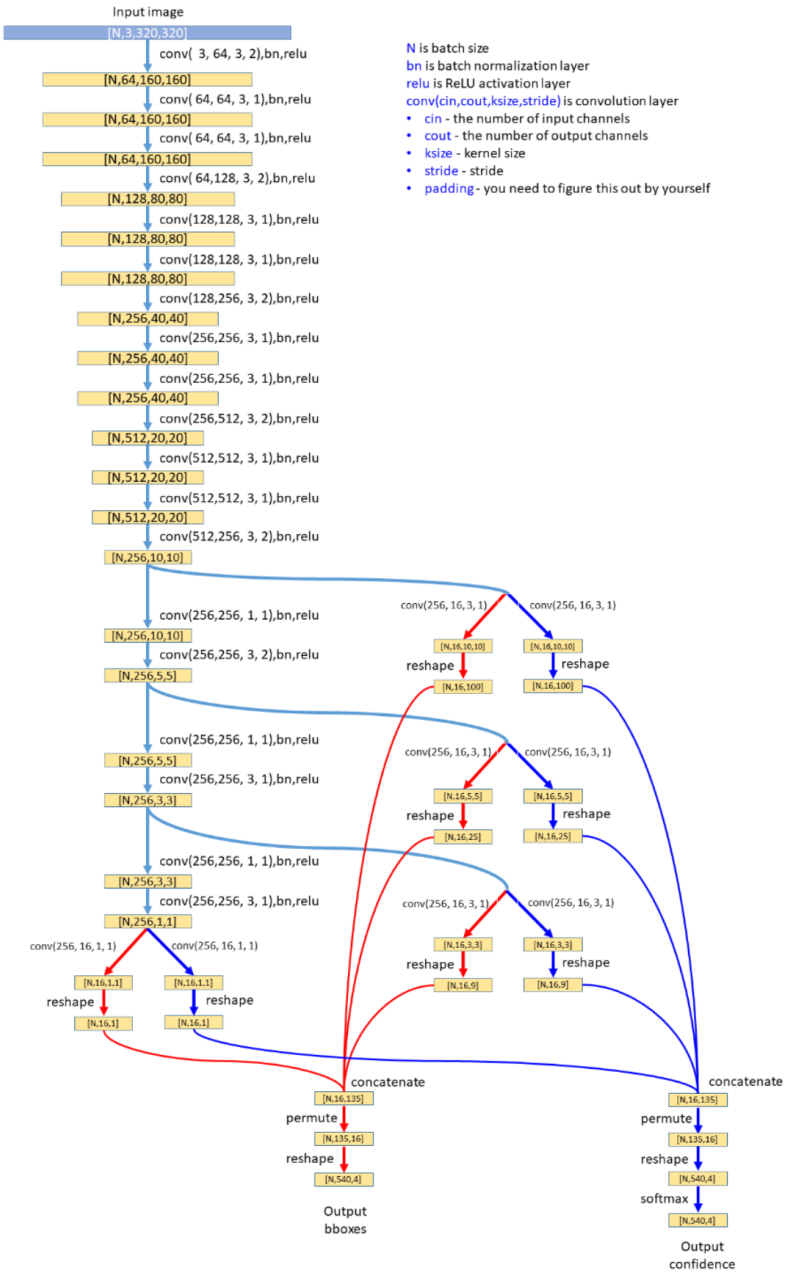


Figure 2: Model architecture

Pay close attention to how the four output branches are concatenated, especially their ordering. You need to make sure that each entry in your 540 output boxes actually corresponds to the entry in the 540 default boxes defined in the previous section. For example, if you define `ann_confidence[4]` and `ann_box[4]` to be a box in cell (0,1) of the 10x10 grid, your network output must have `confidence[4]` and `bboxes[4]`

correspond to the cell (0,1) of the 10x10 output branch. This part is very error-prone.

6 The loss function (15 Points)

The loss function is almost the same as YOLO.

$$L_{cls} = \frac{1}{\sum_i x_i^{obj}} \sum_i x_i^{obj} \text{crossentropy} \left(\text{conf}_i^{\text{pred}}, \text{conf}_i^{gt} \right) +$$
$$3 \cdot \frac{1}{\sum_i x_i^{noobj}} \sum_i x_i^{noobj} \text{ crossentropy} \left(\text{conf}_i^{\text{pred}}, \text{conf}_i^{gt} \right)$$
$$\text{L}_{box} = \frac{1}{\sum_i x_i^{obj}} \sum_i x_i^{obj} L1 \left(box_i^{pred}, box_i^{gt} \right)$$

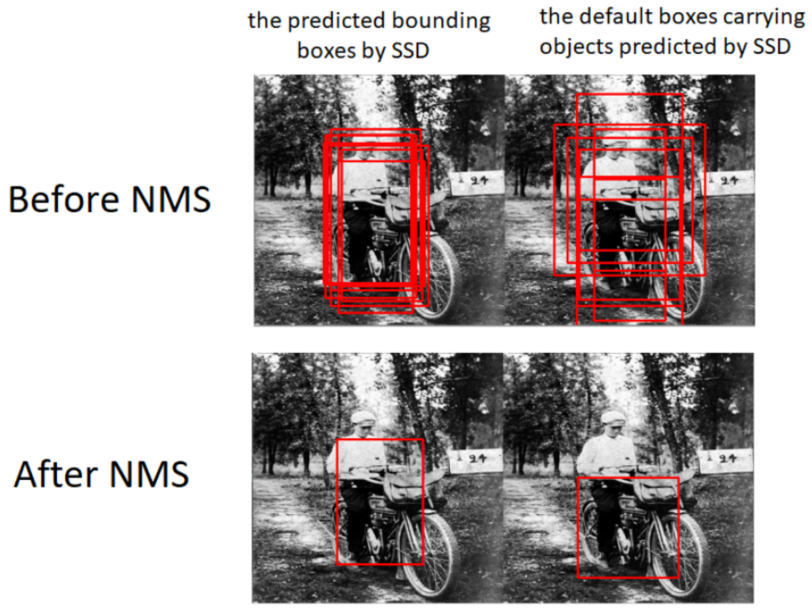
$\text{conf}_i^{\text{pred}}$: class probabilities, conf_i^{gt} : ground truth probabilities $\text{box}_i^{\text{pred}}$: predicted box attributes, box_i^{gt} : ground truth box attributes $x_i^{obj} = 1$ and $x_i^{noobj} = 0$, if box(i) carries an object otherwise, $x_i^{obj} = 0$ and $x_i^{noobj} = 1$

You need to figure out how you can get the indices of all boxes carrying objects, and use `confidence[indices]`, `box[indices]` to select those boxes. If your implementation is correct, after you get those indices, your code to compute loss should be no more than three lines.

Note that the average operation is done automatically by the built-in functions `F.cross_entropy` and `F.smooth_l1_loss`.

7 Non maximum suppression (10 Points)

This part is already covered in the lecture and in the tutorial sessions. Please refer to Ali's slides.
You need to complete the function in `utils.py`. An example of NMS is shown below:



A brief pipeline of NMS:

`Keep two lists: A = all predicted bounding boxes; B = [ ]`

1. `Select the bounding box in A with the highest probability in class cat, dog or person.`
2. `If that highest probability is greater than a threshold (threshold=0.5), proceed; otherwise, the NMS is done.`
3. `Denote the bounding box with the highest probability as x. Move x from A to B.`
4. `For all boxes in A, if a box has IOU greater than an overlap threshold (overlap=0.5) with x, remove that box from A.`
5. `Jump to 1.`

8 Evaluation (15 Points)

For this part, you need to plot a precision-recall curve and compute mAP using your validation set. Implement the `generate_mAP` function in `utils.py`.

9 (Optional) Albumentations (5 Points)

For this part, you will be using the **Albumentations** library to perform data augmentation. You then need to provide separate evaluation results for it.

10 Demo (20 Points)

During the demo session, first, we will check if you implemented the other parts. Then, we will ask you exactly four questions related to the assignment, each having 5 points.

1. We ask you to run your code on a set of examples provided by us. Your code needs to run without errors
2. After running, your code needs to produce the expected results. We will not be very tough on the numbers and metrics, we just visually inspect the results to see if your implementation is performing reasonably
3. Then we will ask you a question about a part of the code. You need to be able to clearly explain what's going on
4. Finally, we will ask a (fairly simple) question to test your knowledge and intuition about the assignment

Note that your codes should be zipped and submitted to Canvas before the deadline and you may not change them during the demo session day.