CMPUT 328 Fall 2023 Assignment 4

Object Detection and Semantic Segmentation

Worth 10% of the total weight

Overview

In this assignment, you are going to do object detection and semantic segmentation on the MNIST Double Digits RGB (MNISTDD-RGB) dataset. This dataset contains RGB images of size 64×64 where each has two digits from 0 to 9 placed randomly inside it as shown in Figure 1. Your task is to figure out which digits are contained in each image and where are they located followed by their precise pixel-wise segmentation masks as in Figure 2.

Quiz

There will be an in-lab quiz worth 2% of the total weight and the remaining 8% is split evenly between the detection and segmentation tasks. The quiz will be about detection and semantic segmentation with deep learning in general and will require you to be familiar with the corresponding lecture material. There will also be questions about the details of this assignment for which you will need to familiarize yourself with the contents of this document and the provided template code.

Dataset

The MNISTDD-RGB dataset is divided into 3 subsets: train, validation and test containing 55K, 5K and 10K samples respectively. A sample consists of:

- Image: A $64 \times 64\ x\ 3$ image that has been vectorized to a 12288-dimensional vector.
- Labels: A 2-dimensional vector that has two numbers in the range [0, 9] which are the two digits in the image.
  - These two numbers are always in ascending order. For example, if digits 7 and 5 are in an image, then this two-vector will be [5, 7] and not [7, 5].
- Bounding boxes: A 2×4 matrix which contains two bounding boxes that mark the locations of the two digits in the image.
  - The first row contains the location of the first digit in labels and the second row contain the location of the second one.
  - Each row of the matrix has 4 numbers which represent $[y\_min,\ x\_min,\ y\_max,\ x\_max]$ in this exact order, where:
    - y_min = row of the top left corner
    - x_min = column of the top left corner
    - y_max = row of the bottom right corner
    - x_max = column of the bottom right corner
  - It is always the case that $x\_max - x\_min = y\_max - y\_min = 28$. This means that each bounding box has a size of 28×28 no matter how large or small the digit inside that box is.
- Segmentation Mask: A 64×64 image with pixel values in the range [0, 10], where 10 represents the background.

Each set comprises 4 *npy* files which can each be read using *numpy.load* to obtain the corresponding matrix stored as a *numpy.ndarray*.

Following are detailed descriptions of the 4 files, where *{subset}* denotes the name of the subset (train, valid or test) and N is the number of samples:

- *{subset}_X.npy*: 2D matrix with dimension $[N, 12288]$ and containing the vectorized images. Each row is a single vectorized RGB image of size 64 x 64 x 3
- *{subset}_Y.npy:* 2D matrix with dimension $[N, 2]$ and containing the labels. Note that the labels are always in ascending order in each row.
- *{subset}_bboxes.npy:* 3D matrix with dimension $[N, 2, 4]$ and containing the bounding boxes. For more information, see the description of bounding boxes above.
- *{subset}_seg.npy:* 2D matrix with dimension $[N, 4096]$ and containing the pixel-wise labels. Each row is a single vectorized segmentation mask of size 64 x 64. The digit on top in the source image is also on top in the mask so the other one is occluded by it, i.e., any pixel occupied by both digits will have the label of the top one.

For example, following are the dimensions of the *numpy.ndarray* in 3 files of the train set:

- *train_X.npy*: $[55000, 12288]$
- *train_Y.npy*: $[55000, 2]$
- *train_bboxes.npy*: $[55000, 2, 4]$
- *train_seg.npy*: $[55000, 4096]$

Task

You are provided with the train and validation sets that can be downloaded from e-class in a zip file named *MNISTDD_RGB_train_valid.zip*.

This contains 8 files: *train_X.npy*, *train_Y.npy*, *train_bboxes.npy*, *train_seg.npy*, *valid_X.npy*, *valid_Y.npy*, *valid_bboxes.npy*, *valid_seg.npy*.

The test set is not released to better represent real-world conditions where test data is not available before the trained model is deployed.

You are also provided three python files: *A4_main.py*, *A4_utils.py* and *A4_submission.py*. The latter has a single function *detect_and_segment* that you need to complete. It takes a single matrix containing all test (or validation) images as input and returns three *numpy* arrays *pred_class, pred_bboxes, pred_seg* respectively containing the classification labels, detection bounding boxes and segmentation masks in the same format as described above.

To reiterate, the two digits in each row of *pred_class* must be in ascending order and the two rows in *pred_bboxes* corresponding to that image must match this ordering too.

Similarly, the digit that is on top in the source image should also be on top in *pred_seg* so the other one is occluded by it, i.e., any pixel occupied by both digits will have the label of the top one.

A4_main.py can be run to evaluate the performance of your method in terms of the classification accuracy (% of digits classified correctly), Intersection over Union (IOU) of the detected boxes with the ground truth boxes and segmentation accuracy (% of pixels with correct labels) of the masks.

It also contains code to visualize MNISTDD-RGB images with both ground truth and predicted bounding boxes and labels drawn on them which might help you figure out how to convert the *npy* files into a format suitable for use with existing frameworks on the Internet.

A4_utils.py contains helper code for visualization.

You are free to add any other functions or classes you need including those in additional files imported from *A4_submission.py*. Just make sure to submit all files needed to run your code including any trained checkpoints.

You can use any machine learning or image processing method to solve this problem. One of the main objectives of this assignment is for you to learn how to adapt existing code on the Internet to solve a new problem so there is no restriction or guideline as to what algorithms or libraries you can or should use. You are also not restricted to using *pytorch* for this assignment.

You have the option to either train two separate models for detection and segmentation or a single model that does both. If you choose to take the latter route, you might want to look into instance segmentation which combines object detection with aspects of semantic segmentation.

The only limitation is that it must be able to run on the test set in **< 1000 seconds** on Colab Tesla T4 GPU, i.e., it must be able to process at least **10 images / second**.

Following are a couple more practical constraints related to submission and evaluation:

- E-class has an upload limit of 400 MB so your models must be small enough to fit in this size.
- Your model must be able to run on Colab GPU
    - Available GPU memory on Colab can vary from session to session so make sure to run your model several times before submitting to ensure that it is not so close to the memory limit that it fails to run during evaluation.

- Memory needed for training is often significantly more than testing and it is only the latter that matters for evaluation.

Any submission not satisfying any of these criteria will get **no marks** for this assignment.

Marking

Your marks will be determined by the following three criteria:

- Classification Accuracy
- Detection IOU
- Segmentation Accuracy

For each criterion, your marks will scale linearly from **70% to 98%.** Submissions with accuracy, IOU, or segmentation accuracy <= 70% and >= 98% will respectively get no marks and full marks for that criterion.

In order as to give equal weightage to the detection and segmentation tasks, your overall marks will be computed as:

$$((classification + IOU) / 2 + segmentation) / 2$$

To reiterate the test time restriction, any submission that runs slower than 10 images / second on Colab Tesla T4 GPU will get no marks.

What to Submit

You need to submit a single zip file containing the modified *A4_submission.py* along with any other files or checkpoints needed to run it on the test set. Testing will be done by running *A4_main.py* and it is your responsibility to ensure that your code can be imported and run without errors.

You should not include training code in your submission. If you do and your model starts training during evaluation, it will again get no marks for this assignment.

Due Date

The due date is **Thursday, November 2 at 11:59 pm**. The assignment is to be submitted online on e-class.

For late submissions, there will be a **10% penalty per day** for the first 4 days after which no submissions will be accepted, i.e., submissions will not be accepted after November 6, 11:59 pm.

Collaboration Policy

This must be your own work. Do not share or look at the code of other people (whether they are inside or outside the class). Do not copy code from the Internet though you can look for ideas to write your own code. You can talk to others that are in the class about solution ideas but not in so much detail that you are verbally sharing or hearing about or seeing the code. You must cite whom you talked to or what websites you consulted to write your code in the comments of your programs.

This policy also applies to the submitted checkpoints (if any) so make sure to train your own model even if you use one of the standardized architectures (e.g., in *torch.hub*) that is shared with other submissions. We will be able to determine from the contents of the submitted checkpoints if they have been plagiarized.

All submissions involved in detected cases of plagiarism will receive no marks for this assignment irrespective of who copied from whom.

Submission Checklist

All submissions will be used in an automated evaluation system and any failure to run, for any reason, will be heavily penalized and might lead to the submission getting a zero.

- Make sure that your code runs without errors on Colab, whether GPU is enabled or disabled.
- Remove any notebook specific lines (e.g., mounting Google drive) or any other code that can produce an error when it is imported.
- Do not submit *A4_main.py*, *A4_utils.py* or any of the *npy* files.
- Do not rename *A4_submission.py* to anything else (including changing any part of it to upper or lowercase).
- Training and validation files will not be available during testing so make sure to remove any code that loads these.
- Set paths for loading weights (if any) relative to *A4_submission.py* in your submitted zip file.
- Make sure that the testing batch size is small enough to not run out of memory on Colab.
- All included files and folders must be in the root of the zip file (instead of being contained inside another folder).
- Submit a zip file instead of any other archive format (e.g., rar, tar or tar.gz)
- E-class has a maximum upload size limit of **400 MB** so your submission must be small enough to fit in this size.
    - Google Drive (or any cloud storage) links for any part of your submission will not be accepted.

Resources

Following are links to some popular Pytorch-based object detection and semantic segmentation repositories on github though there is no guarantee that any of these models will run on Colab or provide sufficient performance. It is your responsibility, if you choose to use any of these, to adapt them to work within the constraints of this assignment.

Many of these are also included within torchvision.

- Detectron2
- Open MMLab Detection Toolbox
- YOLO v5 / v8
- Swin Transformer Object Detection
- DETR
- pytorch-retinanet

- Deeplab v3
- UNet
- Open MMLab Semantic Segmentation Toolbox
- Swin Transformer Semantic Segmentation

Paper / code collections:

- Awesome Object Detection
- Object Detection with Deep Learning
- Awesome Semantic Segmentation
- SemanticSegmentation_DL

Tutorials:

- SSD: Single Shot MultiBox Detector | A PyTorch Tutorial to Object Detection
- Torchvision Object Detection Finetuning Tutorial
- PyTorch object detection with pre-trained networks
- Training an object detector from scratch in PyTorch
- Train your own object detector with Faster-RCNN & PyTorch
- Custom Object Detection using PyTorch Faster RCNN

Figure 1: Visualization of the 256 images generated by the same GAN used for generating the images in MNISTDD-RGB. The two digits in an image may partially or completely overlap each other, though complete overlap only happens in a small fraction of images. The digits may also be of the same class.

Figure 2: Visualization of segmentation masks for the images from Figure 1.

Following colors are used here to show the various digits:

0: red 1: green 2: blue 3: magenta 4: cyan 5: yellow 6: purple 7: forest_green 8: orange 9: white

Note that the value of each pixel in the mask is same as the digit it corresponds to. Background pixels have a value of 10 and are shown here in black.