CSCI 5980/8980: Spatial Enabled Artificial Intelligence

Assignment 3 & 4 (10 points each)

3rd Due Date: 2022/03/28 11:59:00 PM CST

4th Due Date: 2022/04/11 11:59:00 PM CST

1. Overview of the Assignment

This description file contains two assignments. Task 1 is for assignment 3, and task 2 is for assignment 4.

For assignment 3, you will work on the semantic segmentation task to detect road pixels from imagery. You will train a segmentation model called Fully Convolutional Network (FCN). For this assignment, you have the option to implement a UNet model from scratch, in addition to using the FCN from the torchvision package. Implementing the UNet model will give you a 50% bonus maximum (i.e., 5 pts towards the total score) if your implementation meets the grading criteria.

For assignment 4, you will work on object detection for detecting planes in imagery. You will adapt a pretrained Faster-RCNN model using the Plane dataset with transfer learning.

The goal is to get you familiar with the Pytorch packages (e.g., torchvision) and build/train deep neural networks. You can access all the data and starter code on Google Drive.

2. Programming Requirements and Environment Settings

You will use **Python 3.7** for coding.

We have listed all the required packages in the requirements.txt (on Google Drive).

```
opencv-python==4.5.*
torch==1.10.0
torchvision==0.11.1
geojson==2.5.0
setuptools==57.5.0
GDAL==2.4.0
pycocotools==2.0.4
```

To install all the packages, run

- pip install -r requirements.txt

Google provides $50 free credits for each student to use the Google Cloud Platform (GCP). A tutorial for using GCP is available on Google Drive. If you have not received the coupon redemption link by email, please email your TA at li002666@umn.edu. (Remember to **stop the instance** when not using it. Otherwise, you will get severely overcharged.)

If you have other resources for GPUs or prefer to use Google Colab, feel free to use them as long as you install the same versions of packages.

3. SpaceNet Road Data

This dataset is for Task 1 Road Segmentation, it can be downloaded from the Google Drive.

Under `task1_train`, there are two subfolders: PS-RGB and geojson_roads.

a. PS-RGB contains the road images in the geotiff format.

b. geojson_roads contains the ground truth vector data for corresponding images.

You should use 80% samples for training and 20% for validation. The data can be split randomly.

Under `task1_test`, there is one folder `PS-RGB`. We will use this data for grading. You need to generate predictions for images in this folder. Details about the output format is in Section 5.1.

4. Plane Dataset

This dataset is for Task 2 Plane Detection, it can be downloaded from the Google Drive.

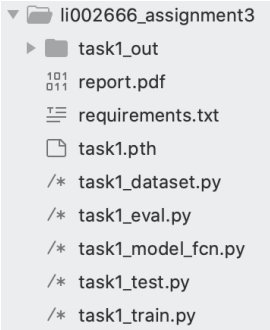
a. train_data: the folder contains all the training images
b. train_list.csv: a csv file where each row contains one axis-aligned bounding box for a plane.
c. test_data: the folder contains all the testing images

Under `train_data` folder, there are 80 images with ground-truth annotations. You should use 70 images for training and 10 images for validation. Images in `test_data` are used for grading.

5. Tasks

Assignment 3 Submission

You need to turn in a zip file, named as [your_login_id]_assignment3.zip (all lowercase), e.g., li002666_assignment3.zip, containing the following files.
The folder structure should be the same as the figure below.



a. [REQUIRED] Python scripts:
   - all the **python scripts** provided to you with the `TODO` part filled in

b. [REQUIRED] Model file**: task1.pth**

c. [REQUIRED] Output predictions:

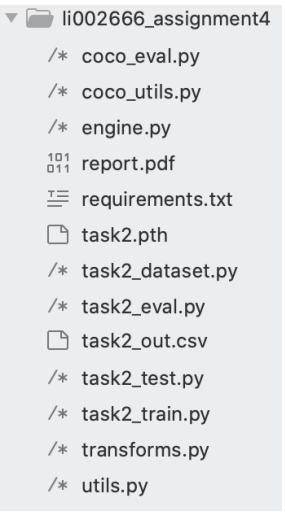
- **task1_out** (a folder contains all the output files)

d. [REQUIRED] Report file: **report.pdf**

e. [OPTIONAL] You can include other scripts to support your programs (e.g., callable functions).
f. [OPTIONAL] If you your code needs support from other additional packages, you can submit a requirements.txt file with all the packages.

Assignment 4 Submission:

You need to turn in a zip file, named as [your_login_id]_assignment4.zip. Folder structure should be the same as the figure.



a. [REQUIRED] Python scripts:
   - all **the python scripts** provided to you with the `TODO` part filled in.

b. [REQUIRED] Model file: **task2.pth**

c. [REQUIRED] Output predictions: **task2_out.csv**

d. [REQUIRED] Report file: **report.pdf**

e. [OPTIONAL] You can include other scripts to support your programs (e.g., callable functions).
f. [OPTIONAL] If you your code needs support from other additional packages, you can submit a requirements.txt file with all the packages

The starter code is to help you get started with the tasks. You are free to change the code outside of the `TODO` blocks as you need.

5.1 Task1: Road Segmentation (5 pts)

5.1.1 Task description

In this task, you will train the Fully Connected Network[1] implemented in the torchvision package for road segmentation. Figure 1 shows a pair of sample input and ground-truth. With FCN, you need to predict the probability that a pixel is a road pixel or non-road pixel.

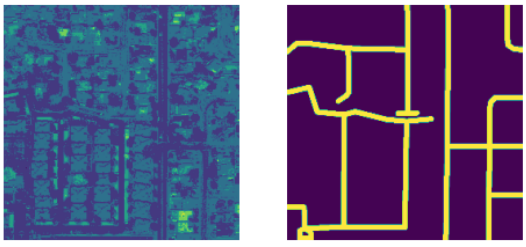


Figure 1: Sample input and ground-truth image (input image has been enhanced for better visualization)

[1] Long, Jonathan, Evan Shelhamer, and Trevor Darrell. "Fully convolutional networks for semantic segmentation." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3431-3440. 2015.

We have provided some starter code where you need to fill in the `TODO` blocks to complete the code.

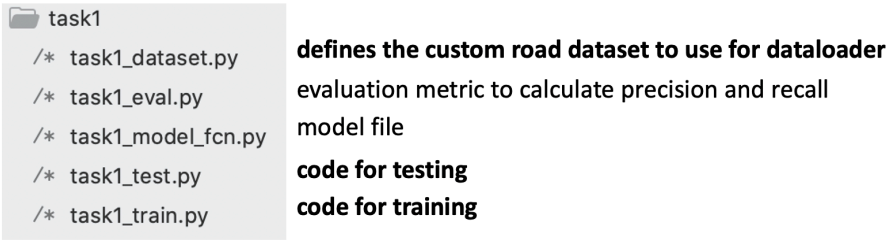


Figure 2: File Structure. Bolded files are the one you need to complete.

There are three files that you need to fill in: task1_dataset.py, task1_train.py and task1_test.py.

You also need to try 3-5 different line thickness options during the ground-truth mask generation and analyze the model output in report.pdf. We will describe in detail in Section 5.1.2.

5.1.2 Code

- task1_dataset.py

To prepare the data for training, you need to align the vector data with the geotiff image and produce the <image, mask> pairs, where mask is a binary array same size as the image, with road pixels labeled as 1 and non-road pixels labeled as 0 (See Figure 1).

In the vector data file, a road is represented as several line segments. The line geometry could be one of the two types: LineString and MultiLineString. The LineString geometry contains only one line segment and MultiLineString contains a list of line segments.

The coordinates of the line are represented as <lat, lng>. The geotiff file contains metadata about the geo-boundary of the image. You can use the osgeo package to obtain the geo-coordinates of the upper left and bottom right corner of the image. We provide the code for you to get the corner in image coordinate system in the calculate_mask() function. You need to complete this function and return a mask image as in Figure 1.

```
ds = gdal.Open(tif_path)
geoTransform = ds.GetGeoTransform()

minx = geoTransform[0]
maxy = geoTransform[3]
maxx = minx + geoTransform[1] * ds.RasterXSize
miny = maxy + geoTransform[5] * ds.RasterYSize
```

We have the steps to follow in the comment block.

```
# TODO
# 1. Parse vector_data dictionary and process each feature
# 2. Get the line geometry coordinates and convert from lat,lng to pixel coord system
# 3. Plot the line on mask_img with cv2.polylines() function.
# 4. Return the mask image
# Hint: the number of channels for mask_img should be 2
```

You need to change the line thickness option for ground-truth segmentation mask generation. The default value is set to be 30, and you can change this number to observe the changes in the output road probability map.

```
def caclulate_mask(self,tif_path, geojson_path, line_thickness = 30, color = (1,1,1)):
```

The second block you need to complete is the __getitem__ function. You should prepare the input and output data for the model (i.e., the image and the ground-truth segmentation mask).

```
  def __getitem__(self, idx):


      # TODO
      # 1. Get image and vector data path according to the idx
      # 2. Read image and calculate the mask
      # 3. Apply transformations
      # 4. Return image and mask



      return img, mask
```

- task1_train.py

In this file, we perform the actual training of the model. We have provided the data_loader, model, optimizer, and loss function. One thing that has been left out is the weight update.

```
for epoch in range(epochs):
    model.train()
    for ii, (data, target) in enumerate(train_loader):

        # TODO
        # 1. get data from train_loader
        # 2. set all tensor gradients to be 0.
        # 3. feed the input to model and get prediction
        # 4. calculate the loss of ground-truth (GT) and prediction
        # 5. back propagation
```

In the missing part, you should iterate through all the samples and update the model weights through back propagation. In the lecture we have shared the CIFAR10 tutorial and transfer learning tutorial. You can refer to the tutorial code and understand how the model optimization is played out.

- task1_test.py

Once your model is trained, you can apply it on previously unseen images. In this file, you will generate the output prediction for the test images. You should save the output probability map in gray-scale image to the output folder

```python
for img_path in img_path_list:
    img = cv2.imread(os.path.join(test_dir,"PS-RGB",img_path))

    # TODO
    # 1. load test image
    # 2. convert test image into tensor
    # 3. feed input to the model and get output
    # 4. save the output probability map to the output folder

    # Hint: need to prepare the image into 4-d tensor for model input
    # Hint: need to resize the image to the approporate size
    # Hint: the direct outputs of the model are logits, you need to convert them to probability
    # Hint: output range need to be rescaled to [0,255] to save with cv2.imwrite()
```

5.1.3 Output format

For each image in the test set, the output should be a 1-channel **(gray-scale)** probability map image of the same shape. The color intensity represents the probability that the pixels is a road pixel. Since the probability is in range [0,1] while color intensity is in range [0,255], so you need to rescale the probability to the range [0,255] when generating the output probability map.

Naming: Use the same name as input file name, save in png format, and change the extension to `.png`. For example, if the input image is `img.tif`, then the output image should be `img.png`.

You need to put all the output files into a folder called `task1_out`.

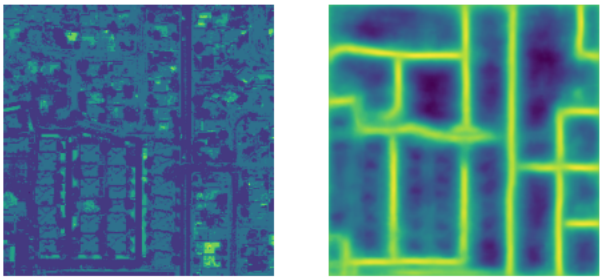


Figure 3. Sample input and probability map prediction (input image has been enhanced for visualization)

5.1.4 Execution commands

Training commands:

```
usage: task1_train.py [-h] [--input_dir INPUT_DIR] [--epochs EPOCHS]
                      [--model_dir MODEL_DIR] [--is_unet]

optional arguments:
  -h, --help            show this help message and exit
  --input_dir INPUT_DIR
                        the training folder that contain the geotiff and road
                        vector data.
  --epochs EPOCHS       the number of epochs to train the model.
  --model_dir MODEL_DIR
                        the output file path to save the trained model
                        weights.
  --is_unet             use this param if you implemented UNet.
```

Prediction commands:

```
usage: task1_test.py [-h] [--input_dir INPUT_DIR] [--model_file MODEL_FILE]
                     [--output_dir OUTPUT_DIR] [--is_unet]

optional arguments:
  -h, --help            show this help message and exit
  --input_dir INPUT_DIR
                        the testing folder that contain the geotiff files.
  --model_file MODEL_FILE
                        path to the model file generated after training
                        process.
  --output_dir OUTPUT_DIR
                        the output folder to save all the output probability
                        maps
  --is_unet             use this param if you implemented UNet.
```

5.1.5 Grading

a. Your code should be executable for both training (2pt) and testing (2pt)
b. With confidence threshold 50%, the segmentation prediction on the test set should have an average precision > 40% , and average recall > 40%. Evaluation code is available at task1_eval.py (4pts)
c. In the report, analyze the impact of different line thickness when preparing the GT mask. (2pt)
d. If you implemented the UNet model and the code satisfies the above criteria, you will get the bonus points for this assignment. For the report, you should compare the differences between FCN and UNet.

5.2 Task2: Plane Detection (5 pts)

5.2.1 Task description

In this task, you will build a plane detection model by fine-tuning Faster R-CNN. The Faster R-CNN model is implemented in the Pytorch torchvision package. The model can be created with the code below:

torchvision.models.detection.fasterrcnn_resnet50_fpn(pretrained = True)

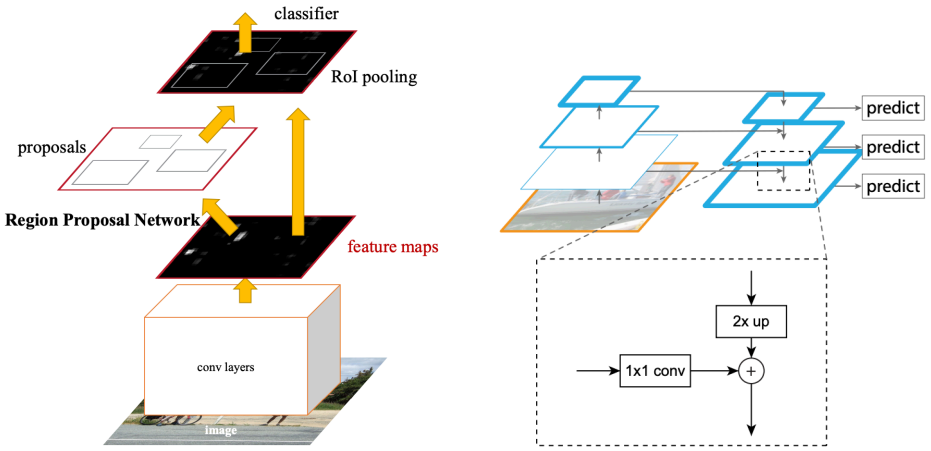


Figure 4. *Left*: Framework for Faster R-CNN. *Right*: Framework for Feature Pyramid Network (FPN)

You need to fine-tune it with the Plane dataset and predict the bounding boxes of each plane. You should breakdown the provided training dataset into training and validation sets randomly with 70 images for training and 10 images for validation.

We have provided some starter code where you need to fill in the `TODO` blocks to complete the code.

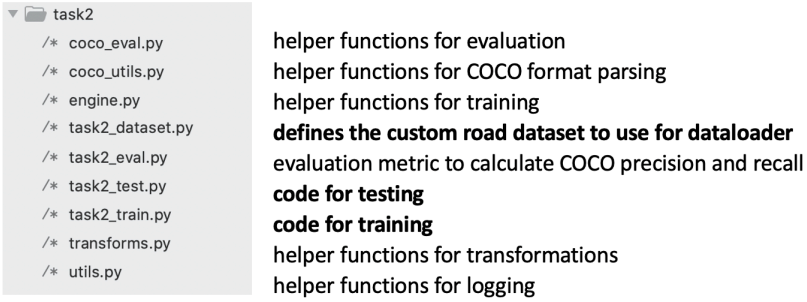


Figure 5: File Structure. Bolded files are the one you need to complete.

There are three files that you should fill in: task2_dataset.py, task2_train.py and task2_test.py.

You should compare the following settings and provide the analysis in the report. You should plot the bounding box regression loss curve for training and validation set for the following experiments:

(1) SGD optimizer with different learning rates

(2) Compare Adam and SGD optimizer

(3) Compare training with and without data augmentation (e.g. random flipping, random resizing, cropping, etc).

For the submission, you only need to submit your model trained with the best configuration that you observed. You can optionally experiment with different learning rate schedulers and different augmentations to improve your model performance.

5.2.2. Code

- task2_dataset.py

The __getitem__ function should return the paired image and target. Steps are outlined in the comment

```
def __getitem__(self, idx):

    #TODO:
    # 1. load images and bounding boxes
    # 2. prepare bboxes into [xmin, ymin, xmax, ymax] format
    # 3. convert everything into torch tensor
    # 4. preprare target_dict with the following keys: bboxes, labels, image_id, area, is_crowd
    # 5. return img, target pair




    target = {}
    target["boxes"] = boxes
    target["labels"] = labels
    target["image_id"] = image_id
    target["area"] = area
    target["iscrowd"] = iscrowd

    if self.transforms is not None:
        img, target = self.transforms(img, target)

    return img, target
```

- task2_train.py

This is the part to train the model. The training procedure for one epoch is implemented in the helper functions. You can call the function to perform weight update.

```
for epoch in range(num_epochs):

    # TODO:
    # train for each epoch and update learning rate
    # save model for each epoch
    # Hint: you can use train_one_epoch() implemented for you
```

- task2_test.py

This is the part to test the model on new files. The output should be one csv file contains all the bounding boxes in all the images. The format of the csv file is shown in Figure 6.

```
with open(output_path, 'w') as out_f:

    line_idx = 0
    for img_path in img_path_list:

        # 1. read image from img_path and conver to tensor
        # 2. feed input tensor to model and get bbox with output[0]['bboxes']
        # 3. write to csv file
        # Hint: You can plot the output box on the image to visualize/verify the result
```

5.2.3 Execution commands

Training commands:

```
usage: task2_train.py [-h] [--input_dir INPUT_DIR] [--model_dir MODEL_DIR]
                      [--epochs EPOCHS]

optional arguments:
  -h, --help            show this help message and exit
  --input_dir INPUT_DIR
                        the input directory that contains train_data folder
                        and train_list.csv
  --model_dir MODEL_DIR
                        the output file path to save the trained model
                        weights.
  --epochs EPOCHS       the number of epochs to train the model.
```

Predicting commands:

```
usage: task2_test.py [-h] [--input_dir INPUT_DIR] [--model_file MODEL_FILE]
                     [--output_file OUTPUT_FILE]

optional arguments:
  -h, --help            show this help message and exit
  --input_dir INPUT_DIR
                        the input directory that contains test_data folder
  --model_file MODEL_FILE
                        the model (weights) generated during the training
                        process.
  --output_file OUTPUT_FILE
                        the output csv file that has the same format as
                        train_list.csv
```

5.2.4 Output format:

Prediction format:

The prediction results for all the test images should be written in one csv file: task2_out.csv. Each line should contain:

id: a unique number. You can simply use the line number as id

img name: test image name

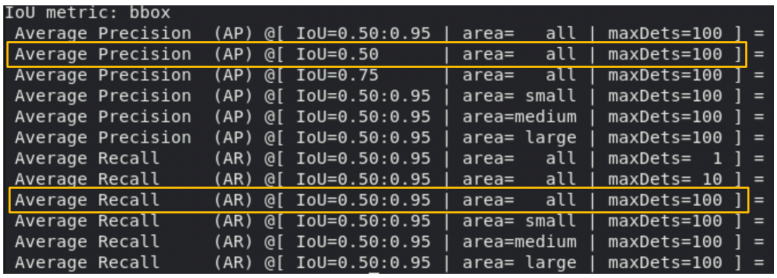
bbox: the bounding box information in the same format as train.csv

```
12,4f833867-273e-4d73-8bc3-cb2d9ceb54ef.jpg,"[(2052, 1772), (2135, 1772), (2135, 1834), (2052, 1834), (2052, 1772)]"
```

Figure 6: One example line in task2_out.csv

5.2.5 Grading

a. Your code should be executable for both training (2pt) and testing (2pt)
b. The prediction on test set should have AP@IOU 0.5 > 82%, and AR@IOU=0.5:0.95 > 60%. The numbers can be generated with task2_eval.py as in the Figure below (4pts)
c. Your report should contain the three comparisons mentioned above. (2pt)



Appendix:

If you would like to implement the Task1 model from scratch, here is the **UNet[2] with ResNet18[3] backbone** model for your reference.

UNet is a deep neural network initially proposed for medical image segmentation, and it is now widely used in various segmentation tasks such as object segmentation and text detection. The main contribution of UNet is its skip connections between low-level features and high-level features.

Figure 2 shows the outline of UNet. The main take-away from Figure 2 is to apply the skip connections on ResNet. Thus you do not need to follow the exact framework of UNet, but you need to think about where to insert the skip connections on ResNet.

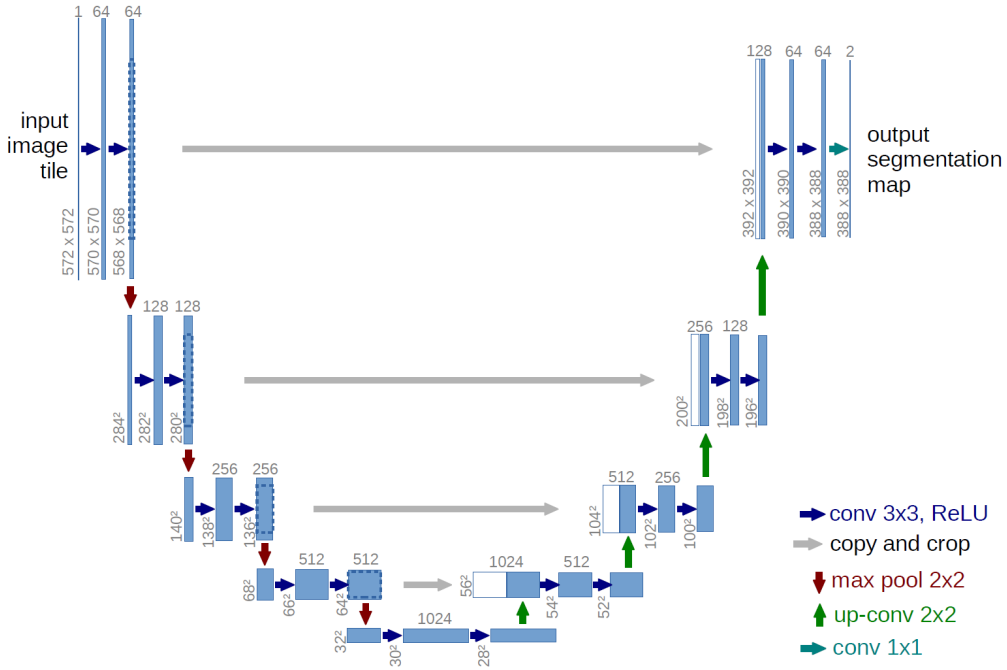


Figure 2: Illustration of UNet Architecture. The **gray arrows** are the skip connections that connects the low-level and high-level features. You **do not** need to follow exactly the design of the layers, especially the convolution layers (blue blocks), as you need to replace them by the ResNet layers. But you can refer to this figure in terms of where to add the skip connections.

---

[2] Ronneberger, Olaf, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation." In *International Conference on Medical image computing and computer-assisted intervention*, pp. 234-241. Springer, Cham, 2015.

[3] He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770-778. 2016.

ResNet is proposed to alleviate the vanishing/exploding gradient shift problem during the network training. The main contribution is the shortcut connections between layers. Figure 3 shows the building blocks of the residual learning implemented with shortcut connections. The idea is that, leaning the residual function F(x) instead of the desired full mapping F(x) + x should be easier for the network.

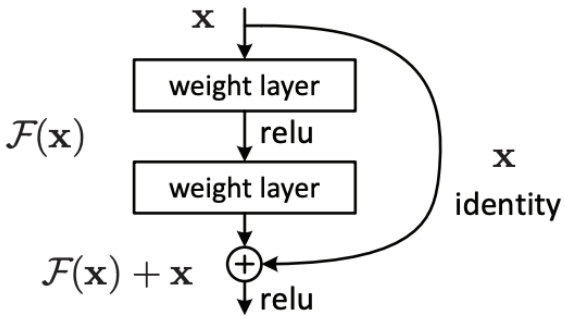


Figure 3. Building blocks for residual learning. Notice that there are shortcut connections between network layers.

You can implement the shallowest version of the Resnet, which is ResNet-18. Figure 4 shows the detailed design of ResNet-18. You should follow the layer configurations of the 18-layer version (the third column) and insert the skip connections as in the UNet.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 4. Architecture of ResNet. The brackets stand for the building blocks in Figure 3. In the example of [3x3, 64], 3x3 stands for the kernel size, and 64 means the number of kernels. The numbers next to the brackets stand for the number of times that these layers should be repeated.