THE HONG KONG UNIVERSITY OF SCIENCE & TECHNOLOGY
Department of Computer Science and Engineering
COMP4211: Machine Learning
Fall 2020 Project
Due time and date: 11:59 pm, Dec 22 (Tue), 2020.

In this project, you will perform turbulence detection based on multispectral images. Each group can pick either project 1 or project 2.

1 Project 1

We categorize the turbulence according to its intensity as "moderate" (MOD) or "severe" (SEV). For each reported turbulence occurrence, you are given the corresponding multispectral images (of size 224×224) with the following 4 bands: (i) band 8: which detects water vapor; (ii) band 12: ozone; (iii) bands 13: infrared; and (iv) band 14: another wavelength in the infrared spectrum. You are also given some multispectral images that do not contain turbulence (class NIL).

1.1 Tasks.

You are required to build a convolutional neural network (using python and pytorch) for the classification of multi-spectral images into the SEV/MOD/NIL classes. The data set proj1_data can be downloaded from the link: https://hkustconnect-my.sharepoint.com/:u:/g/personal/jliude_connect_ust_hk/EepKnZR7IGNKr9Gob1XrltkBj0n5_9JyMDBBe0ZMig-r0g?e=b0Dj9P. The images are numpy arrays stored in pickle format, and can be read with the provided MyDataset class. Optionally, you can perform additional pre-processing on the data.

1. First, build a baseline CNN model (baseline1) using

   - the cross-entropy loss; and
   - basic data augmentation strategies including random cropping, flipping, color jittering and rotation.

   Note that the CNNs covered in the tutorials are for color images (with the three R/G/B bands), while the multispectral images here have four bands.

2. Next, you have to build another baseline model (baseline2), with the same architecture as baseline1, but with the LDAM loss in [1]. Note that you are expected to read the paper [1] in order to understand what it is and how it works.

   - For a sample (x, y), the LDAM loss is defined as

   $$L_{\text{LDAM}}((x,y);f) = -\log \frac{e^{z_y - \Delta_y}}{e^{z_y - \Delta_y} + \sum_{j \neq y} e^{z_j}}, \quad \text{where} \quad \Delta_j = \frac{C}{n_j^{1/4}} \text{ for } j \in 1, \ldots, k.$$

   Here, f is the CNN model, k is the number of classes, C is a hyper-parameter to be tuned by using the validation set, z_j is the jth class output from the model $f(x)$, and n_j is the number of samples in the jth class.

3. Finally, by improving upon baseline1 or baseline2, you have to build a final model by implementing additional techniques of your choice. You may also simply turn baseline1 or baseline2 as your final model, but your grade will be partially based on the accuracy of this final CNN model on a test set (which is hidden from you).

For performance evaluation, we will use the precision, recall, and F1-value evaluated for each of the three classes (NIL/MOD/SEV) separately. These are defined as:

$$\begin{aligned} \text{Precision} &= \text{TP}/(\text{TP}+\text{FP}), \\ \text{Recall} &= \text{TP}/(\text{TP}+\text{FN}), \\ \text{F}_1 &= 2(\text{Precision} \cdot \text{Recall})/(\text{Precision} + \text{Recall}). \end{aligned}$$

Here, TP is the number of true positives (i.e., ground truth: positive; prediction: positive) for the class of interest, FP is the number of false positives (i.e., ground truth: negative; prediction: positive), and FN is the number of false negatives (i.e., ground truth: positive; prediction: negative). For example, when considering the SEV class, FN includes those samples whose ground-truths are SEV while the predictions are either NIL or MOD.

In your report, you have to provide detailed descriptions and codes on

- data preprocessing (if any);
- baseline1
  - including model specification, tuning procedure of hyperparameters, training and validation set performance (precision, recall, and F1) results on the three classes (NIL/MOD/SEV).
- baseline2
  - including tuning procedure of hyperparameters, training and validation set performance (precision, recall, and F1) results on the three classes.
- final model
  - including model specification, tuning procedure of hyperparameters, training and validation set performance (precision, recall, and F1) results on the three classes.
  - Note that if you have used techniques not covered in the lecture notes (e.g., other convolutional neural network constructs or machine learning models), you have to describe these techniques clearly.

2 Project 2

Each satellite image may contain several turbulence regions, the number of which is unknown. As in project 1, the turbulence level can be MOD or SEV. In this project, you have to identify all turbulence regions by specifying the

1. bounding box: a 4-dimensional vector (r, c, h, w) representing the top-left corner (r, c), height h and width w of the rectangle; and
2. level (SEV/MOD)

of each turbulence region. An example is shown in Figure 1.

r	c	h	w	class label
180	2	139	189	SEV
37	6	85	128	SEV
354	112	62	100	SEV
3	146	59	95	MOD
209	418	66	223	MOD

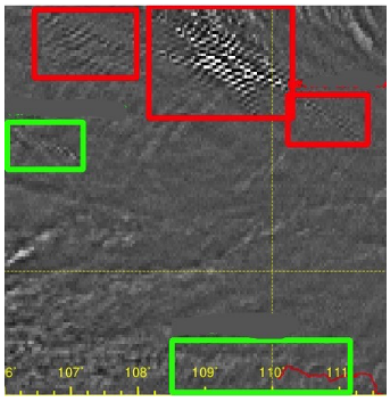

Figure 1: Left: Each row is a turbulence, with its bounding box and level. Right: Visualization of the bounding boxes (green is MOD and red is SEV).

You have to implement a simplified version of the Fast R-CNN [2]. Its major steps are:

1. Using a CNN, process the image to produce a convolutional feature map;

2. For each object proposal, a region of interest (RoI) pooling layer extracts a fixed-length feature vector from the feature map;

3. Each feature vector is fed into a sequence of fully connected layers that finally branch into two sibling output layers:

   (a) one sibling output layer produces softmax probability estimates over the two classes (SEV/MOD) plus a background class (NIL);

   (b) another sibling output layer outputs four real-valued numbers (encoding refined bounding-box positions) for each of the two classes (MOD/SEV).

An illustration is shown in Figure 2.

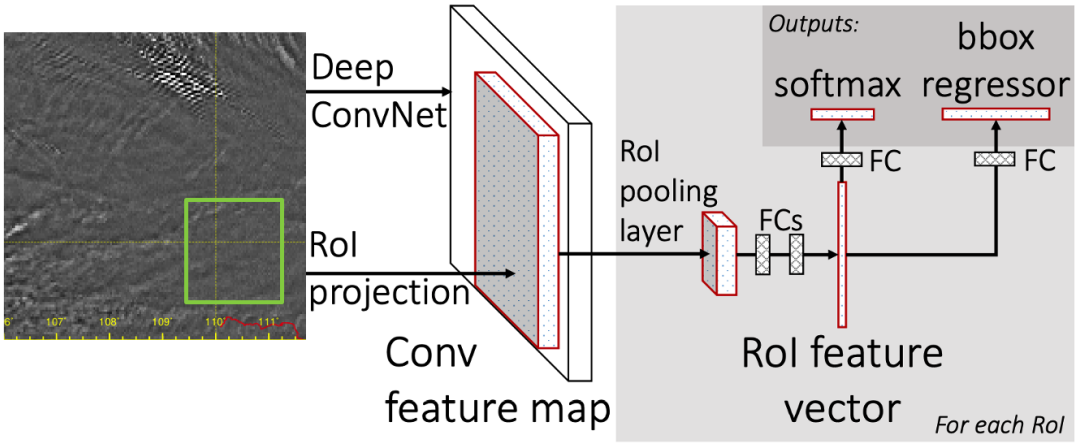

Figure 2: Fast R-CNN architecture.

2.1 Tasks.

First, you have to understand the Fast R-CNN model in [2] (figure 2), and then implement it in pytorch and python. The following are the processing in the fast R-CNN that you have to implement:

1. Define a CNN to obtain the conv feature map

   - Use model S in section 4.1 of [2], in which the feature extractor is obtained by modifying the AlexNet [4].

2. Obtain the conv feature map from the image

   - This is described in section 2.1 of [2]. Use $H = 13, W = 13, h = 130, w = 130$.

3. Construct the RoI pooling layer with the two sibling output layers

   - For each RoI, the first sibling output layer outputs a discrete probability distribution $\boldsymbol{p} = (p_0, p_1, p_2)$ over the 3 categories (class 0 for NIL, class 1 for MOD, class 2 for SEV). Compute $\boldsymbol{p}$ by a softmax over the 3 outputs of a fully connected layer.
   - The second sibling layer outputs the bounding box regression offsets $\boldsymbol{g}^k = (g_x^k, g_y^k, g_h^k, g_w^k)$ for class $k = 1, 2$, which are obtained from the bbox regressor in Figure 2, by the following procedure.
     Let $\boldsymbol{b}^k = (b_x^k, b_y^k, b_h^k, b_w^k)$ be the output for class k ($k = 1, 2$) from the FC layer just before the bbox regressor in Figure 2. We obtain the bounding box

$$\hat{\boldsymbol{b}}^k = (\hat{b}_x^k, \hat{b}_y^k, \hat{b}_w^k, \hat{b}_h^k) \tag{1}$$

as:

$$\begin{aligned}\hat{b}_x^k &= b_w^k \cdot d_x(\boldsymbol{b}^k) + b_x^k,\\ \hat{b}_y^k &= b_h^k \cdot d_y(\boldsymbol{b}^k) + b_y^k,\\ \hat{b}_w^k &= b_w^k \cdot \exp(d_w(\boldsymbol{b}^k)),\\ \hat{b}_h^k &= b_h^k \cdot \exp(d_h(\boldsymbol{b}^k)).\end{aligned}$$

Here, $d_x(\boldsymbol{b}^k) = \boldsymbol{w}_x^T \phi(\boldsymbol{b}^k)$, where $\boldsymbol{w}_x$ is a parameter vector,and $\phi(\boldsymbol{b}^k)$ is the pool_5 feature of the RoI pooling layer. Similarly, $d_y(\boldsymbol{b}^k) = \boldsymbol{w}_y^T \phi(\boldsymbol{b}^k), d_h(\boldsymbol{b}^k) = \boldsymbol{w}_h^T \phi(\boldsymbol{b}^k)$, and $d_w(\boldsymbol{b}^k) = \boldsymbol{w}_w^T \phi(\boldsymbol{b}^k)$.
Let $\boldsymbol{g}^k = (g_x^k, g_y^k, g_h^k, g_w^k)$ be the output of class k from the second sibling layer, where

$$\begin{aligned}g_x^k &= (\hat{b}_x^k - b_x^k)/b_w^k,\\ g_y^k &= (\hat{b}_y^k - b_y^k)/b_h^k,\\ g_w^k &= \log(\hat{b}_w^k/b_w^k),\\ g_h^k &= \log(\hat{b}_h^k/b_h^k).\end{aligned}$$

Essentially, g_x^k (resp. g_y^k) specifies a translation in the row (resp. column) dimension, and g_h^k (resp. g_w^k) a log-space height (resp. width) shift. Parameter $\boldsymbol{w}_x$ is learned by optimizing the following regularized least squares objective:

$$\boldsymbol{w}_x = \arg\min_{\hat{\boldsymbol{w}}_x} \sum_{k=1}^{2} \left(g_x^k - \hat{\boldsymbol{w}}_x{}^T \phi(\boldsymbol{b}^k) \right)^2 + \lambda \|\hat{\boldsymbol{w}}_x\|^2, \tag{2}$$

where λ is a hyperparameter. Similarly, $\boldsymbol{w}_y, \boldsymbol{w}_w$ and $\boldsymbol{w}_h$ are obtained from equations analogous to (2) (sharing the same λ). After learning, the predicted bounding box is $\hat{\boldsymbol{b}}^k$ in (1) for class k. For more details, please see Appendix C in [5].

- Try different sizes for these two sibling layers, get best ones according to the performance on validation set.

4. Train the network

   - Each training sample (a RoI) is labeled with a class u and a bounding box v.
   - In each iteration, construct mini-batch by sampling some RoIs as described under the section "Mini-batch sampling" in [2].
   - Perform SGD on the loss, which is a multi-task loss L on each labeled RoI to jointly train for classification and bounding-box regression, as described in the section "multi-task loss" in [2]. Set $\lambda = 1$.

5. Testing (detection)

   - The trained network takes as input an image and a list of object proposals (i.e., candidate RoIs) to score.
     - Instead of using selective sampling as in [2], we use a simple clustering approach to sample RoIs.
       (a) Divide the image into 10×10 initial regions.
       (b) Use a greedy algorithm to iteratively group regions together. The pseudocode is provided in Algorithm 1 of [3]. For simplicity, use random initialization instead of (Felzenszwalb and Huttenlocher (2004)) for initialization.
           i. Calculate the similarities between all neighboring regions
           ii. The two most similar regions are grouped together, and new similarities are calculated between the resulting region and its neighbors.
           iii. Repeat the grouping process until the whole image becomes a single region.
           iv. Extract object location boxes from all regions. After merging, each merged region may not be rectangular. In that case, return the smallest box bounding that region.
       (c) Measure the similarity between two image regions (r_i and r_j) by $s_{colour}(r_i, r_j)$ described under the section "Complementary Similarity Measures" in [3].

- For each candidate RoI r, the forward pass outputs a class posterior probability distribution and a set of predicted bounding-box offsets relative to r (each of the 2 classes gets its own bounding-box prediction).
- For each class k, assign p_k as the detection confidence for r (i.e., estimated probability $\Pr(\text{class} = k|r)$).
- Obtain the predicted label as: $\arg\max\{p_0, p_1, p_2\}$

For performance evaluation, we use IOU, which is defined as:

$$\text{IOU} = |\text{A} \cap \text{B}|/|\text{A} \cup \text{B}|,$$

where A is the bounding box you predicted, B is the ground-truth bounding box, $|\text{A} \cap \text{B}|$ is the area of the overlapping region, and $|\text{A} \cup \text{B}|$ is the combined area.

You have to perform the following:

1. Load the data
   - The input image data proj2_data can be downloaded from the link: https://hkustconnect-my.sharepoint.com/:u:/g/personal/jliude_connect_ust_hk/EXPrkxH2CodNhmFuiTsacEABInDZCsQ5BhMA_1TMDrcSEg?e=q3d5Dm. There are a total of 703 images. The image filename is of the format:

     HS_H08_YYYYMMDD_HHMinMin_latitude_longitude.npy

     where YYYY is the year, MM is the month, DD is the date, HH is the hour, and MinMin is the minutes. Images in the files are stored as numpy arrays.
   - Define the Dataset class for pytorch to load the dataset. Please refer the first section in https://pytorch.org/tutorials/intermediate/torchvision_tutorial.html for details.
   - The (rectangular) locations of the turbulence and levels are in files bbox_train.csv, bbox_val.csv and bbox_test.csv for the training set, validation set and test set, respectively.
2. Train the model by using the training set, and tune the sizes of the sibling layers and other hyperparameters by using the validation set.
3. Load the test set. Obtain and draw the predicted bounding boxes for the following images:

   HS_H08_20160313_0330_25.41_122.35.npy

   HS_H08_20171122_1600_32.56_133.1.npy

   HS_H08_20180312_1730_27.06_106.49.npy

   HS_H08_20190105_0430_32.541389_114.077778.npy

You can draw the boxes directly on the images as in Figure 1.

In your report, you have to provide detailed descriptions and codes on

- data preprocessing, including the implementation of Dataset class;
- the fast R-CNN model, including the implementation of conv feature map, the implementation of RoI pooling layer, tuning procedure of hyperparameters, performance (including IOU) results on the validation set.

3 Submission Guidelines

- Please zip your Python notebook (proj.ipynb) with all other necessary Python files (*.py) (the codes may be in several .py files) and *markdown* summary as your report, and the weights files (pure.tar, l2.tar, augment.tar) for all your models. Zip all the files as either [your student ID]_proj.zip or [your student ID]_proj.tar.gz.
- Please submit the project by uploading the compressed file to Canvas. Note that the submission should be legible, otherwise you may lose some points if the assignment is difficult to read. Plagiarism will lead to zero points.

References

[1] K. Cao, C. Wei, A. Gaidon, N. Arechiga, T. Ma, Learning imbalanced datasets with label-distribution-aware margin loss, Advances in Neural Information Processing Systems, 2019.

[2] R. Girshick, Fast R-CNN, International Conference on Computer Vision, 2015.

[3] J.R.R. Uijlings, K.E.A. van de Sande, T. Gevers, A.W.M. Smeulders, Selective search for object recognition, International Journal of Computer Vision, 2013.

[4] A. Krizhevsky, I. Sutskever, G.E. Hinton, ImageNet classification with deep convolutional neural networks, Advances in Neural Information Processing Systems, 2012.

[5] R. Girshick, J. Donahue, T. Darrell, J. Malik, Rich feature hierarchies for accurate object detection and semantic segmentation, arXiv:1311.2524, 2014.