Advanced Topics in Machine Learning
Assignment 2

University of Bern

Due date: April 27th, 2021

1 Introduction

In this assignment you need to upload a zip file to ILIAS which includes: 1) A Jupyter Notebook file Assignment2.ipynb completed with code and answers and 2) a Jupyter Notebook exported to HTML (File / Export Notebook as / HTML). The zip file name must be FirstName LastName.zip. If your implementation requires auxiliary functions, you must implement that function inside the corresponding .py file. Please state your name at the beginning of the notebook.

1.1 Notes on code and submission quality

In addition to answering the different questions, you are also expected to provide well written submissions. Here are some recommendations to take into consideration.

- Please answer the question in the same order as in the assignment and use the same question numbers;
- Don't answer the questions in the code comments. Use the text cells in your notebook;
- Remove clutter such as unused code lines instead of turning them into comments;
- Make sure the right execution order of the notebook cells is from top to bottom. A TA should be able to reproduce your results by simply clicking "Run All" without having to guess which cells should be executed first.

Poorly written submissions might result in points deduction.

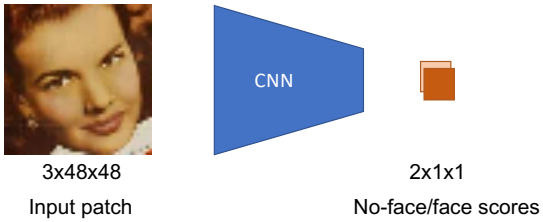

Figure 1: Scheme for training a binary face/no-face classifier. We train on image patches of fixed size and get a single score output for that patch.

2 Face detection problem

In this assignment you will train a face detection model, that given a picture will return pixel coordinates of face occurrences. You are given a dataset of small images of faces and non-faces (background, small part of faces). The images have resolution 48x48 pixels. You will train a binary classification model and use it to detect faces on the entire images.

Tasks:

1. **[25 pt.] Train a convolutional neural network for binary classification.**

   You will implement and train a convolutional neural network for binary classification of image patches of faces/non-faces (see Fig 1). Folders **train** and **val** contain test and validation image patches of faces and backgrounds (non-faces).

   - (5 pt.) Create Dataset and DataLoader objects for provided training and validation data (folders *train* and *val*). Visualize a few images from both classes.
   - (10 pt.) Implement a binary convolutional model according to the definition below:
     - Convolutional layer, filter size 3x3, stride 1, 32 channels
     - Max Pooling layer, input 3x3, stride 2, ceil_mode=True
     - Activation function LeakyReLU, slope 0.2
     - Convolutional layer, filter size 3x3, stride 1, 64 channels
     - Max Pooling layer, input 3x3, stride 2
     - Activation function LeakyReLU, slope 0.2
     - Convolutional layer, filter size 3x3, stride 1, 64 channels
     - Max Pooling layer, input 2x2, stride 2
     - Activation function LeakyReLU, slope 0.2
     - Convolutional layer, filter size 2x2, stride 1, 128 channels
     - Activation function LeakyReLU, slope 0.2

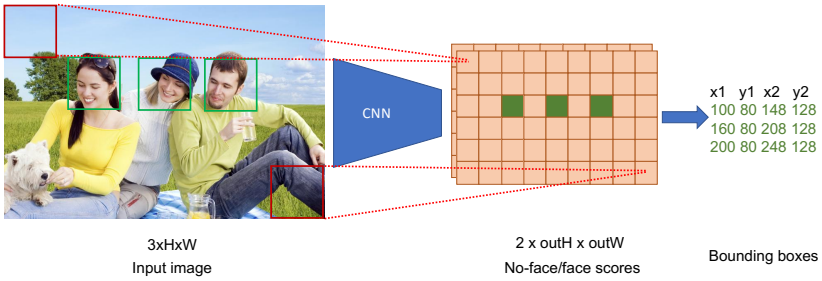

Figure 2: Since our network is *fully* convolutional, we can feed it with bigger images and obtain face scores for different locations of image patches all at once. Spatial output corresponds to face probability at different locations of the input image. "Green" outputs show high probability of face and correspond to bounding boxes (coordinates of image patch) that contain face

  - Convolutional layer, filter size 3x3, stride 1, 256 channels
  - Activation function LeakyReLU, slope 0.2
  - Convolutional layer, filter size 1x1, stride 1, 2 (output) channels
- (10 pt.) Write the training code and train the network you implemented
  - Train for 10 epochs with batch size of 64
  - Optimize cross entropy loss
  - Use Adam optimizer with learning rate 3e-4

  In your report include plots for training and validation loss and accuracy.

  Hint: Your network should have the output of size Bx2x1x1, *but the cross entropy loss requires input of size* Bx2. *You need to reshape the output before feeding it to the cross-entropy loss (do not do it in the forward function - it's important for the next step).*

2. **[45 pt.] Run your fully-convolutional model on full images**

   Since our model is fully-convolutional - and no layer needs a fixed size input - we can feed it with bigger images and get a spatial map of face scores for different image patches.

   For example, if we feed the model with image of size HxW, and we get the output of size outH x outW, the output at coordinates (0,0) corresponds to the top left corner patch of the image (0, 0, 48, 48) [coordinates: (x1,y1,x2,y2); where (x1,y1) - coordinates of top left corner of the patch and (x2,y2) - coordinates of bottom right corner of the patch]. The output at bottom right coordinates (outH-1, outW-1) corresponds to input image patch (W-1-48, H-1-48, W-1, H-1) [bottom right corner]. The other correspondences can be computed proportionally. See Fig. 2.

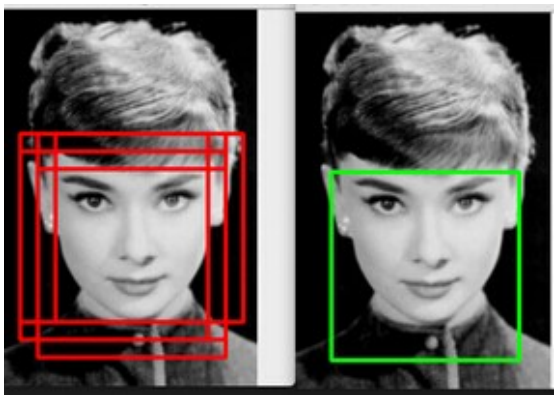

Figure 3: Running detection on overlapping patches will create overlapping correct face predictions (on the left). Non-maximum suppression algorithm merges the overlapping bounding boxes (on the right)

Our face detector can only detect faces of size 48x48. To detect bigger faces, we will first resize our images to smaller scales and then run the network on different scales of the same image.

Your task will be to implement the face detector using your trained model. You are given a skeleton of *FaceDetector* class.

- (25 pt.) Implement *detect_single_scale* method of *FaceDetector* class, that takes an input image and returns face coordinates using your trained model (details in the method comments)
- (10 pt.) Finish the implementation of *detect_multi_scale* method of *FaceDetector* class, that runs *detect_single_scale* on different sizes of the image.

  Note: Running the detection on overlapping patches will result in overlapping correct predictions of faces. To merge overlapping predictions, a non-maximum suppression algorithm is used (see Fig. 3) that can be activated with a parameter nms (already implemented)
- (10 pt.) There are 4 full size images in **full_images** folder and corresponding ground truth face coordinates (bounding boxes) in **full_boxes** folder. Read the images and bounding boxes using *read_images* function (already implemented).

  Initialize the detector with your trained model, default scales and default maximum resolution (1024) and a threshold=0.99. Run the multi-scale detection on the images and plot the detection results with *draw_boxes_gt* function.

  Manually find a threshold for which there are few False Positives (detected faces where there are no real faces), while keeping as much as possible True Positives (correctly detected faces) on these 4 images.

Write the selected threshold and plot the detection results for the 4 images using this threshold. Plot for both nms=False and nms=True.

3. **[30 pt.] Improve your model**

   Try to improve your model by tuning your architecture, hyper-parameters, regularization etc. Some examples of improvements: training parameters (e.g. optimizer, learning rate, scheduling, batch size), network architecture (e.g. number of layers, number of units, activation function, normalization layers), model regularization (e.g. data augmentation, dropout, weight decay, early stopping), test-time augmentation, etc.

   Train at least 5 changed configurations from scratch, describe the changes you made and report classification accuracy on the validation set from point 1.

   Using your best model, run the multi-scale detection on the 4 full images from point 2 and one image of your choice. Find a good threshold and plot the detection results.