University of Massachusetts Boston, Department of Computer Science

Term Project Phase II

Apply Deep Convolutional Neural Network to Real World Application (Crater)

CS480/CS697 Big Data Analytics

Hamidreza Mohebbi, Wei Ding

1. Educational Goals

- Train a Deep Convolutional Neural Network model on Crater data.
- Use the image pyramid and sliding window techniques to parse the image and detect craters.

2. Project Description

Deep Convolutional Neural Networks (CNN) have recently achieved the state of the art performance on challenging computer vision tasks like ImageNet challenge. Unlike the conventional classification techniques based on feature engineering and classification, CNNs learn both features and classification model during training. The github repository of the project is this link:
https://github.com/mohebbihr/cs697_term_project_phaseI/

The main objective of project is to develop a CNN on crater data and then apply the trained model on a big planetary image to detect the craters.

The following shows the general steps of this phase of the project, you can find more information about each step on the rest of this document.

a) Prepare the dataset.
b) Develop a deep neural network model.
c) Use pyramid image and slicing window techniques to detect crates on planetary images.
d) Compare and analyses the results of phase I with phase II and write the report.

3. Prepare the dataset.
   Before developing the neural network, we need to obtain enough samples and divide the crater and non-crater images into train, validation and test sets. In this phase, we need to use both original images and also the normalized images from phase I. So, the first step is to copy the original *images* folder and *normalized_images* folder into a new folder. The next step is to develop a function that shuffles crater and non-crater images randomly divide them into 70% train, 15% validation and 15% test sets. Please follow below instructions for this part of the assignment:
   I. Create a directory named *phaseII_images* under *images* directory of the dataset to store the results of this step. Please create two sub-directories named *crater* and *non-crater* in *phaseII_images* directory.
   II. Copy all the crater and non-crater images from *images/tile3_24*, *images/tile3_25* and *images/normalized_images* into *phaseII_images* folder. If you used more than one method for pre-processing the images on phase I, you can copy all the images into the images folder for phase II.
   III. Please write a python function named *load_crater_data_phaseII_wrapper* and add it to *crater_loader.py* script from phase I. This function is similar to the *load_crater_data_wrapper* function that you developed for the phase I of the project. The only difference is that this function divide the dataset into training, validation and test sets (70%, 15% and 15%).

4. Develop and train the model
   The objective of this part of the project is to develop a deep convolutional neural network based on the *conv.py* script that you can find it on sample code. Your network must have at least 6 layers and you may use *network3.py* from sample code in this part too. Please follow the instructions below for this part of the project:

   I. Please read and understand the *conv.py* script (inside *src* directory of sample neural network).
   II. Please develop your own deep model based on the *conv.py* model and saved it in a file named *crater_deep_network.py*. Your need to use convolutional layers in your implementation. You may develop several architecture and use different activation function to receive bonus points.
   III. Write a python script named *run_deep_experiment.py* which its functionality is to load the dataset, train the model and test it on validation and test sets. Please, train your model for at least 20 epochs and then report the accuracy, detection, false and quality rates as a plot

like the phase I of the project. You may use *plot_error* function in *conv.py* file as an inspiration to write required functions for extracting these rates. You may use *plot_filters* function to plot the filters of your network to receive bonus points.

5. Image Pyramids and Scanning Window
   Utilizing an image pyramid allows us to find objects in images at different scales of an image. And when combined with a sliding window we can find objects in images in various locations.

   At the bottom of the pyramid we have the original image at its original size (in terms of width and height). And at each subsequent layer, the image is resized (subsampled) and optionally smoothed (usually via Gaussian blurring).

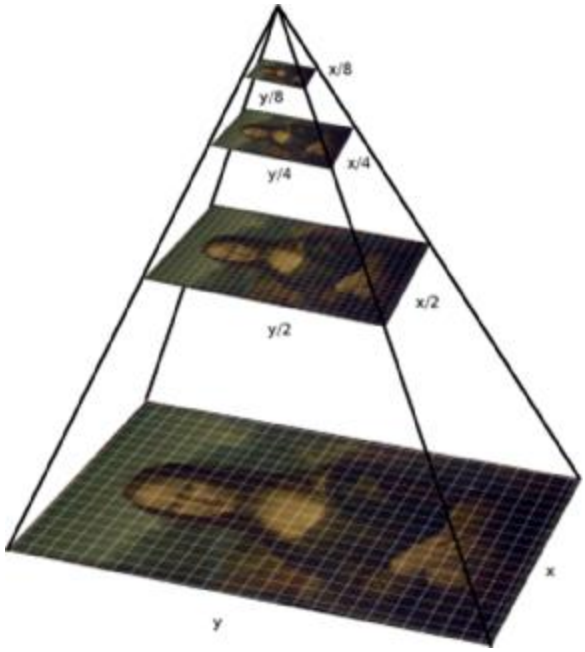

Figure 1 An example of an image pyramid. At each layer of the pyramid the image is downsized and (optionally) smoothed

The *pyramid_gaussain* function from scikit-image library can be used to get the pyramids of an image. The following code shows how to use this function. You can read more about this function on this link: http://scikit-image.org/docs/dev/api/skimage.transform.html#pyramid-gaussian

```
from skimage.transform import pyramid_gaussian
import argparse
import cv2

for (i, resized) in enumerate(pyramid_gaussian(image, downscale=2)):
        # if the image is too small, break from the loop
        if resized.shape[0] < 24 or resized.shape[1] < 24:
                break

        # show the resized image
        cv2.imshow("Layer {}".format(i + 1), resized)
        cv2.waitKey(0)
```

In the context of computer vision (and as the name suggests), a sliding window is rectangular region of fixed width and height that "slides" across an image. For each of these windows, we would normally take the window region and apply an image classifier to determine if the window has an object that interests us in this case, a crater. Below code shows this concept.

```
from skimage.transform import pyramid_gaussian
import argparse
import cv2

# loop over the image pyramid
for resized in pyramid(image, scale=1.5):
        # loop over the sliding window for each layer of the pyramid
        for (x, y, window) in sliding_window(resized, stepSize=32, windowSize=(winW,
winH)):
                # if the window does not meet our desired window size, ignore it
                if window.shape[0] != winH or window.shape[1] != winW:
                        continue

                # THIS IS WHERE YOU WOULD PROCESS YOUR WINDOW,
SUCH AS APPLYING A
                # MACHINE LEARNING CLASSIFIER TO CLASSIFY THE
CONTENTS OF THE
                # WINDOW

                # since we do not have a classifier, we'll just draw the window
                clone = resized.copy()
                cv2.rectangle(clone, (x, y), (x + winW, y + winH), (0, 255, 0), 2)
                cv2.imshow("Window", clone)
                cv2.waitKey(1)
                time.sleep(0.025)
```

Please follow these instruction:

I. Please install the scikit-image library using `pip install scikit-image` command.
II. Please use the provided code samples and write a python script named *crater_slice_winodw.py* to go through *tile3_24.pgm* and *tile3_25.pgm* images and detect craters on them.
III. Use your designed neural network on phase I and phase II to detect craters on the image.
IV. Please draw a green circle around a true positive window and a red circle around false positive window. Please save the detected images as *detected_tile3_24.jpg* and *detected_tile_25.jpg* and put it on your report. You may use imwrite method to save the image.

```
cv2.imwrite('detected_tile3_24.jpg',img)
```

6. Evaluation

Please evaluate and analysis the results of the previous step and answer the following questions on your report.

a. Please compare the results of phase I neural network with the phase II model and analysis their differences.
b. Please determine which of the craters was difficult to detect by all of the neural networks and suggest ways to detect them.