Bizzy Moore
Professor Bridgeman
Professor Dumitriu
CS Independent Study
Spring 2020

Harmful Algal Bloom Training Program - User Manual

Overview

There are three main tasks that are involved in the project for classifying Harmful Algal Blooms in the Finger Lakes through machine learning. The components include acquiring images, often through extracting frames from videos, training the machine learning model, and classifying the images using the trained model. Therefore, this project is broken up into three separate programs: the VLC media player, the training program, and the classifier program.

The Harmful Algal Bloom Training Program is a program that is designed to train, test, and evaluate a machine learning model based on images of the Finger Lakes that are imported into the model. Once the model is trained, tested, and evaluated, it is able to determine whether a given image is clear, turid, or contains blue-green algae (bga), which signifies that a Harmful Algal Bloom (HAB) was present. The specific kind of model that is trained in this program is a Convolutional Neural Network (CNN). A CNN was selected to be trained for this project because CNNs are well-known for their ability to process and classify images. Therefore, since the data that we are importing into this model are images, a CNN was a logical choice. Upon completion of running the program, the trained classifier model is saved to a h5 file. This file saves the architecture of the trained CNN model, and allows us to load it into another program. This gives us the ability to use this model to predict the classification of new images.

Necessary Software

In order to run this program, the software, Python, needs to be installed on your computer. This is because Python is the language that was used to write this machine learning program. Python's software was selected because it has many open-source libraries for machine learning. Therefore, it was able to easily support what we wanted to do for this project. If Python is not already installed on your computer, then it can be downloaded here: https://www.python.org/downloads/.

Acquiring Data

The HABs training program takes in an image set and trains the CNN to recognize patterns in this data set in order to classify the images into three categories: bga, clear, or turbid. Therefore, in order for this program to properly run, it needs to be given a group of images that are saved in the jpg format.

If you need to obtain your images by extracting them from videos and need some guidance, then see the separate document, "Extracting Images Using VLC - User Manual."

Setup

Summary:

- You need a directory containing:
    - the training program file (*HABs_Train.py*)
    - a folder for the images used for training, which contains 3 subfolders named:
        - *bga*
        - *clear*
        - *turbid*
- Advanced Option: you can specify a name to save the classifier model file as, instead of using the default name that is hard-coded into the program (*HABs_CNN_Model_FINAL.h5*)

Detailed Explanation:

In order to make sure that the program runs correctly, you need to make sure that everything is set up and organized in the right way. This means that you need to confirm that all of the files and images that you will be using to run the program are under the same base folder in your directory. If your directory is not set up properly, then the program will not be able to find everything that it needs when it is running, and it will crash with errors.

The first step is to create a new folder somewhere in your directories on your computer that will act as the "base" folder for the project. This folder can be placed in your documents folder, on your desktop, or in any other folder that you would like. It does not matter where the folder is stored. You just need to make sure that you know the directory path of the folder because this path will be used for us to find the folder and access the programs inside of it when we run the program. For the purpose of this explanation I am going to refer to this base folder as *HABsProject*. You can name this base folder anything you would like. If I placed *HABsProject* in my Documents folder on my computer, then the directory path for *HABsProject* on a Mac would be /Users/BizzyMoore/Documents/HABsProject. For a Windows user this would be C:\Users\BizzyMoore\Documents\HABsProject.

After the base folder is set up, the next step is to make sure that the necessary file, *HABs_Train.py,* is in this base folder. *HABs_Train.py* is the program that you will be running.

The last step is to make sure that the base folder, *HABsProject*, contains all of the images that you want to classify. All of the pictures need to go into a seperate folder under *HABsProject*. These pictures will be the images that you accumulated in the "Acquiring Data" section above. They will be the pictures that are imported into the program. Therefore, I named this base folder for the images, *ImportedPics*. You can name this folder anything you would like.

Inside of this *ImportedPics* folder, you need to have three more folders. These folders need to be named *bga, clear,* and *turbid.* All of the images need to be sorted correctly into their respective folder before the program runs. Therefore, all of the bga pictures need to be in the folder labeled *bga,* all of the clear pictures need to be in the folder labeled *clear,* and all of the turbid pictures need to be in the folder labeled *turbid.* This is because when the program runs, it will label each picture with the name of the folder that they are in. Labels are important in the training process because when the model is looking

at the characteristics of a certain image, it needs to know which classification group the image belongs to. This allows the model to make connections between images that belong in the same group. If the images in the imported data set are not sorted beforehand, then the model will have no way of making connections between images and the different classification groups.

Running the Program

Summary:

- The command line syntax to run the program is below:

  python HABs_Train.py --dataset *folderName*

- The 1 parameter is:
  - -- dataset *folderName* - the name of the folder holding the images used for training
- Advanced Option: the parameter below can be used to specify the name to save the classifier model file as, instead of using the default name
  - --model *modelName*

Detailed Explanation:

You will be running the program through your computer's terminal and command line. You can access the terminal by going to your computer's application folder and searching for "terminal". It should pop up as an option to select.

Once you have the terminal open, the first step is to make sure that you are in the correct current directory. This means that you want to make sure that the computer is looking in the folder that contains all of the files that you will be using. In our case, HABsProject is the folder that contains everything that we need to run the program. Therefore, we need to set the current directory to the path name of *HABsProject*.

For a Mac, the command for setting the current directory in the terminal is "cd". In order to set the current directory, you need to type "cd" followed by a space and the path name of the folder that you want to access. Therefore, for our example on a Mac we would type:

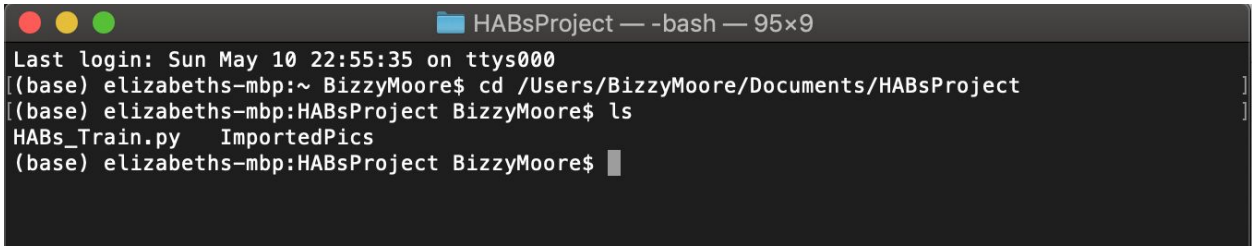
cd /Users/BizzyMoore/Documents/HABsProject

If you are using a Windows operating system, then the command for setting the current directory is below:

C:\Users\BizzyMoore\Documents\HABsProject

Press enter once you have finished typing either of these lines.

Next, you can use a specific command to view the files and folders within your current directory. This allows you to double check to make sure that all of the necessary things that are discussed in the section above are included in this directory. The picture below portrays these terminal commands and their output. For a Mac, you can use the list command, "ls", to do this. For Windows, the command for this is "dir". Below is an example of completing the above steps in a Mac terminal. I do not have access to a Windows terminal to portray these actions.



Since all of the necessary parts are in the *HABsProject* directory, then it is time to run the program. You will need to know two things in order to run the program:

1. The name of the program
2. The name of the folder that holds the images that will be used for training

In our example, the name of the program is *HABs_Train.py*, and the name of the folder that holds the images that will be used for training is *ImportedPics.*

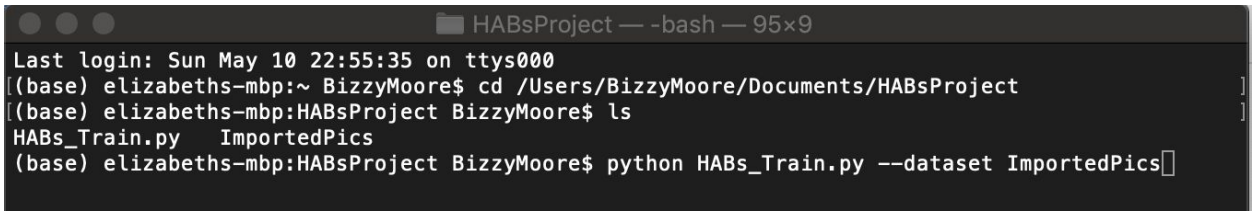
The format of running a Python program in the command line is the following:

python programName.py --command1 argument1

The command that we are going to use is --dataset. After the command is entered, make a space and then type the respective argument after it. Therefore, for our example we would run the program by typing:

python HABs_Train.py --dataset ImportedPics

The picture below depicts the steps for running the program.



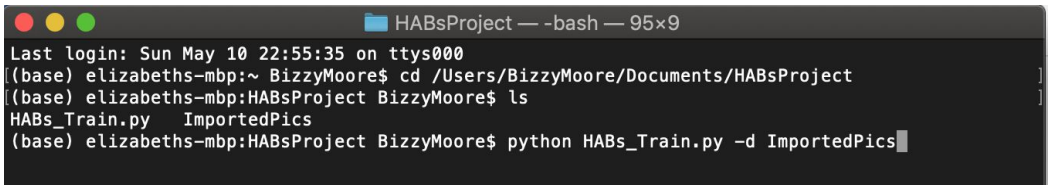
There is also a shortcut for the dataset command:

- --dataset can be shortened to -d

Therefore, we can also run our current program with the following command:

python HABs_Train.py -d ImportedPics

The picture below illustrates this.



As mentioned in the setup section, there is an advanced option that allows you to specify a different name for the classifier model file, rather than use the default one (*HABs_CNN_Model_FINAL.h5*). The command for this is:

--model *modelName*

The shortcut for this command is:

-m *modelName*

Evaluating the Results

Summary:

- The report outputs the following information in the terminal:
  - Classification Report
  - Confusion Matrix

Detailed Explanation:

By the time the program, *HABs_Train.py*, finishes running, the CNN model has been trained, tested, and evaluated. We want to be able to examine and observe the model's results from these steps. Classification reports and confusion matrices allow us to do this. They provide us with information that gives us a sense of how accurate the model was with its predictions of the images in the test set. We want to be able to evaluate the model and see how accurate its predictions are. The classification report and the confusion matrix are outputted into the terminal by the program. An example of this is shown below. The overall accuracy of the model's prediction is listed at the top above the classification report and the confusion matrix.

```
[INFO] results:
accuracy = 88.89%

Classification Report Trial:
              precision    recall  f1-score   support

         bga       1.00      1.00      1.00         1
       clear       0.88      0.93      0.90        15
      turbid       0.90      0.82      0.86        11

    accuracy                           0.89        27
   macro avg       0.92      0.92      0.92        27
weighted avg       0.89      0.89      0.89        27

Confusion Matrix Trial:
[[ 1  0  0]
 [ 0 14  1]
 [ 0  2  9]]

Saved model to disk.
```

A classification report "is used to measure the quality of predictions from a classification algorithm" [1]. The report provides information for three different classification metrics: precision, recall, and f1-score. These metrics "are calculated by using true and false positives, and true and false negatives" [1]. The positive and negative "are generic names for the predicted classes" [1]. Therefore, for our specific project, the positive and negative are represented by blue-green algae (bga), clear, and turbid. A true-negative is "when a case was negative and predicted negative" [1]. A true-positive is "when a case was positive and predicted positive" [1]. A false-negative is "when a case was positive but predicted

negative” [1]. Finally, a false-positive is “when a case was negative but predicted positive” [1]. Precision is “the ability of a classifier not to label an instance positive that is actually negative” [1]. Therefore, it is defined for each class “as the ratio of true positives to the sum of true and false positives” [1]. Recall is “the ability of a classifier to find all positive instances” [1]. Therefore, it is defined for each class “as the ratio of true positives to the sum of true positives and false negatives” [1]. Lastly, the f1-score “is a weighted harmonic mean of precision and recall such that the best score is 1.0 and the worst is 0.0” [1]. This means that f1-scores “are lower than accuracy measures as they embed precision and recall into their computation” [1]. Therefore, the f1-score should be “used to compare classifier models, not global accuracy” [1].

A confusion matrix “give[s] you a better idea of what your classification model is getting right and what types of errors it is making” [2]. For example, it “shows the ways in which your classification model is confused when it makes predictions” [2]. Each index represents a classification class. Index 0 represents “bga”, index 1 represents “clear”, and index 2 represents “turbid”. For each slot in the matrix, the row represents what the image actually is, and the column represents how the image was classified by the model. Therefore, if the model classified the images with 100% accuracy, the matrix would only have numbers down its right-diagonal. Zeros would be filled in everywhere else. This means that every bga image was classified as bga, every clear image was classified as clear, and every turbid image was classified as turbid. In the example above, the model had an accuracy of 88.89%. It mis-classified 3 images out of a total of 27 images that were in the test set. In the confusion matrix, you can see that there was one image where the image was actually clear, but the model classified it as being turbid. In addition, you can see that there were two images that were labeled as turbid, but were classified by the model as being clear. There was only one bga image in this test set and the model classified it correctly. Confusion matrices are very helpful because they give you insight into what specifically the model is getting wrong.

Works Cited

[1] https://muthu.co/understanding-the-classification-report-in-sklearn/

[2] https://machinelearningmastery.com/confusion-matrix-machine-learning/