Final Report

Nathan A. Lutes

Mentors: Justin S. Jones, Ryan A. Kent

Materials Engineering Branch, Code 541

June – August 2019

Table of Contents

Abstract

As mankind continues its exploration into the heavens, the possible discovery of life elsewhere in the universe is coming closer to fruition. However, life exists in many diverse and subtle forms and correctly identifying life on worlds other than Earth can be tedious and difficult. As such, an efficient answer to this problem is desperately needed to prevent progress from being hindered. An effective predictive model, coupled to future exploration instruments and trained to detect life based on pattern recognition, is the proposed solution in this study. This model will be constructed using state of the art machine learning (ML) methods, a discipline that has recently gained much traction in various fields for its versatility and dependable robustness, especially in image classification and segmentation. The chosen ML algorithm is a modification of a robust framework referred to as a convolutional neural network (CNN). Typical CNN's are widely considered the benchmark in image classification; however, for this project the algorithm needs to be slightly altered. The resulting architecture, commonly referred to as a fully convolutional neural network (FCNN), differs from a traditional CNN only in the construction of the last few 'layers' and the ultimate output. A CNN responds to an image with a set of numbers, indicating its confidence level that a certain class exists within the image. An FCNN returns a mask of classifications on the pixel level, denoted by different shadings. When applied as an overlay, these classifications physically segment the image into the different classes. In this manner, an FCNN can show exactly where each object is in the image. For this project, a data-set comprised of 10,000 x 10,000 pixel images of various samples was collected. The images were taken using a Keyence VHX 5000 digital microscope. The FCNN was constructed and will be trained using the Matlab coding language. The 'ground truth' images – the references used to train the model – will be created from images containing biotic material. This will be done using a special application developed by MathWorks. Once the 'ground truth' images are created, the data-set can be split into training and testing sets. The model will then be trained on the training set until an acceptable accuracy is achieved. Finally, the model will be tested on the testing set to gauge its performance on an image base it has never seen before. The model will be accepted or further refined based on its performance on the test set.

Executive Summary

My activities for this internship consisted of various duties performed on the Textural Pattern Recognition (TPR) project, the "xCT in Space" proposal and building an external housing for a 3-axis, StepCraft milling machine. From this work, I created two deliverables: an abstract detailing my work on the TPR project (included within this report) and a poster expanding upon this abstract and also briefly detailing my other activities. The purpose of this executive summary is to provide a comprehensive explanation and evaluation of my work performed for the benefit of all parties involved. The summary will be structured based on contribution size and therefore will begin with the StepCraft external housing and will end with a detailed synopsis of current progress on the TPR project.

With my colleague Scott Santoro, I partially completed the external housing for the StepCraft 3-axis CNC machine. Progress was made until approximately halfway into the assembly where it was discontinued due to growing priority on other projects. Unfortunately, a number of errors were made along the way that required the disassembly and reassembly of parts and as such minor defects such as scuffs or small tears can be observed in some parts. Personally, I blame the difficulty in assembly on vague instructions and unusual fitting of some parts. The major steps remaining are to finish the two side doors, complete the front door and add these to the main assembly. Skills developed or refined include Recommendations for anyone (including myself) participating in future projects of this sort are to completely read the instructions before attempting assembly and to pay extra attention to the assembly details, especially for similar looking parts.

The other side project this summer was the creation of an animation for the "xCT in Space" proposal. This animation demonstrated how a 6-axis robotic arm might be used to enable CT scans on other planets or in suitable spacecraft and was used during a short promotional video accompanying the proposal. It was created using SolidWorks by making an assembly using an Epson robotic arm, a Martian rock and a small, simple xCT frame model. Individual images of the SolidWorks assembly were also used in a written proposal for the project. Skills developed or refined include basic CAD skills and animation creation and development in SolidWorks. I also gained a cursory understanding of the xCT process and how it might be extended to extraterrestrial environments. This project was interesting and provided me the opportunity to perform a task that I had previously never done.

The majority of my time this summer was spent developing machine learning applications and supporting software for the TPR project. I was responsible for overseeing and facilitating the dataset creation, developing software that could pre-process the data and creating model architecture for the machine learning procedure. The first problem-solving approach involved developing a convolutional neural network capable of receiving an image and making a prediction of whether the image contained life based on its confidence. Programs developed for this approach were coded in the Python coding language. Later, the strategy changed to a semantic segmentation approach – where the algorithm was expected to not only identify life but accurately predict its location spatially within the image. Matlab was used for this alternative scheme and luckily it contained built in applications that greatly accelerated the process. Overall, four programs were developed: three for the first approach and one for the second.

Regarding the first approach, a short summary of each program and the work still needed to make this approach feasible is as follows. The first two programs developed, located in the *finished* folder of the *python programs* directory, are used for dataset creation and pre-processing. The first program, SplitnClass_tool_GUI_OFF.py, is used to slice the original dataset images into smaller sub-images and classify these sub-images by storing them into the appropriate folders, created in the directory of the user's choice. The second program, Train_Test_Creator.py, is used to manually divide these sub-images into training and testing sets, this is an important step in the machine learning process. The last program in the

finished directory, TPR_ML_v1.0.4.py, is where the model is built, trained and tested. It will need access to the directories where the sets of training and testing images are located. It completes the last steps of the pre-processing before facilitating the training and testing of the model. It is important to note that because of the sudden change in project direction, this program was never totally completed and the current version would need some minor adjustments before it would be ready for deployment, especially since it was originally designed to accept the smaller sub-image slices produced by the aforementioned programs. A more detailed rundown of these changes will be included in the program description section.

The second approach was accomplished using Matlab and took advantage of a pre-built MathWorks application named Image Labeler. Using this app, the images could be properly labeled and the label and image data exported directly from the app into the Matlab environment. Because of this, it was only necessary to construct a program that would build, train and test a model. This program does all of the image pre-processing (resizing, train and test set split, normalization, etc.) model training and model evaluation locally. This program was fully developed and implemented before the end of the internship with results discussed in the following paragraph.

Unfortunately, the results obtained from the testing session at the end of the internship were widely inconclusive. Problems encountered were short training times (the computer was still needed for labeling) and a relatively small dataset. The models created seemed to have problems with a bias towards biotic material as it would often classify the entire image as biotic. It is difficult to tell what implications this has on the project as a whole but more work tuning the model should be done to get a better idea of the maximum potential that machine learning has on solving the problem posed by this project. Efforts should also be made to broaden the dataset as much as possible. A larger, more encompassing dataset will have more potential for overall success. I plan on continuing this project when I return to Missouri S&T so there will be more to report in the future.

Program Description

The intent of this section is to break down each program's functionality in more detail than what was provided in the executive summary. The Python programs will be explained first, in the order they were developed, followed by the Matlab program. The step-by-step user guides will be provided in the next section.

As a general note, the libraries needed to run each program are listed at the top of the program, accompanied by an "import" statement.

SplitnClass_tool_GUI_OFF.py

This program was created with the intent of streamlining the process of creating the dataset from the microscope images. It takes advantage of one of Python's general user interface packages to create a GUI in which the parent image and the current slice are displayed on the screen with a button menu used to classify each slice. The program first asks the user to specify the file location containing the parent images and the file location where it will store the classified slices. It then gives you the option to set the slice size with the default being 1000x1000 (note that the slice dimensions will always be square). Afterwards, the program loops through every image contained in the parent directory. Each image is resized to be compatible with the chosen slice dimensions and then systematically sliced left to right, top to bottom. For each slice, clicking a button will cause that slice to be stored in the appropriate folder (bearing the same name as the button). This is important because the program that trains the machine learning model uses the names of the folders to decide the appropriate label for a given image. After the slice is stored, it will then be replaced with the next slice in the parent image. This continues until there are no more images left in the parent directory. You can manually leave the program at any time by clicking the "close" button. It is important to note that this program also creates a temporary text file bearing the names of the images that it has already sliced. This "record" is stored in the same directory where the image slice subfolders are created. Upon running the program using a storage directory that contains a record file, the program will skip any file names within the record to avoid duplicates. However, a parent image is only considered "complete" when every slice has been classified. This means that its filename gets added to the record only after the program switches to the next parent image. As such, it is highly not recommended to exit the program until completing all of the slices for the current parent image; exiting in the middle of a parent image will cause duplicate slices the next time the program is run. Please also note that switching the storage directory between program executions (switching to a different directory that does not contain a record file) will cause the program to start with the first image found in the parent directory. The record file can be simply moved to the new storage directory to alleviate this symptom however this practice is not recommended.

Train_Test_Creator.py

This utility of this program is to simply take the data created from the above program and split it into training and testing sets. It is a relatively simple, easy to use script with only one action being necessary from the user. When the program is called, the user simply navigates to the storage folder containing the label name subfolders. The program creates two new folders within the storage folder titled train and test and splits the original data appropriately into these new directories. The program is setup to do an 80/20 split so have 80% of the data will be copied into the train folder and the remaining 20% will be filed into the test folder (this can be changed within the program by redefining the values of the variable 'train_ratio'). Note that this program makes copies of the original data so keep in mind the extra storage constraints. The idea behind this decision was so the original data would not be tampered with in case a backup is needed.

TPR_ML_v1.0.4

This is the final version of the program for building, training and testing the machine learning model. It is a fully functional code however there are some changes that need to be made to certain variables listed in the beginning of the program. The variables that need to be changed are: train_path, test_path, results_path, sliceSize, num_classes and datasetSize. The train_path and test_path variables need to be replaced with the file path to the train and test directories created by the Train_Test_Creator.py program. The results_path variable needs to contain the file path to where the user is expecting the test results. The sliceSize variable doesn't need to be changed unless the size of the image slices is different from 1000. The num_classes variable can be left at its default of 2 unless there are additional classes that need to be accounted for. Finally, datasetSize needs to be changed to reflect the size of the data being used. If you are unsure what the size of the dataset is, you can run the sections of the code containing the flow_from_directory commands for both the training and test set and then add together the two numbers that Python outputs. Once the appropriate changes to the above variables have been made, the code can be run. The code loads in a pre-trained model from the Keras library and then accesses the training data based on the file path it was provided in the train_path variable. It then trains on this data for a default of 100 epochs or iterations. This can be changed by the user but it is not recommended unless the user has prior knowledge of machine learning. Once the model is done training, it is then evaluated on the testing set and the program outputs the percentage of correct classifications. The program also creates an excel file containing each particular image slice name, what the correct label was and what the algorithm labeled this slice as. This file is stored in the path denoted by the results_path variable. Please note that the documentation for much of the commands used in this program can be found by google searching 'Keras documentation'.

PredModelV1_0_2.m

This is the program written in Matlab to perform the semantic segmentation. The program is split into multiple sections that can each be run individually by clicking inside the section (a section is selected when its background is highlighted) and then clicking 'run section' in the top ribbon. This is advantageous because some of the sections are optional and may not be within your best interest to run for a given situation. The intent of this design was to provide an improved sense of flexibility for the user. This program uses a pre-built network from the Matlab deep learning toolbox and refines that network based on the project dataset. It provides functionality to load in a previously saved network in the case that additional training is needed at a later time. Furthermore, this program does everything needed to build, train and test the model. However, the user will need to manually export the data from the image labeler app (more information on this in the step-by-step user guide). Note that the training and testing split in this program is a 70/30 split (70% used to train and 30% used to test). This program provides multiple ways of analyzing the test results of the model. First, the user can specify a specific image from the test set to analyze, second the user can loop through all of the test images and third, the user can calculate a set of numerical metrics to get a good sense of the performance of the network. Note, however, that since the output of the network is a segmentation of the image, numerical results can be somewhat misleading and it is generally good practice for the user to compare the actual segmentation to the 'ground truth' or correct answer as provided by the dataset.

Step-by-step User guides

This section will walk any potential users through the use of the provided programs, highlighting any and all needed or recommended actions and providing foresight into how to achieve optimal results. It is recommended to skim through all included steps before attempting to use the programs.

Also, here is a list of the Python libraries you will need to have in your environment to run the programs:

glob, os, random, shutil, sys, tkinter, numpy, skimage, PIL, pandas, keras

You can install all of these by using the pip installer (google search 'how to use pip python' for more help).

SplitnClass_tool_GUI_OFF.py

1. Open the python environment (IDLE or Spyder depending on your version)
2. Open the program, hit the F5 key on your keyboard
3. A message box will appear telling you to select the folder containing the images, click ok
4. A file browser will appear, navigate to the folder containing your dataset
5. A new message box will appear telling you to select the directory for subfolder creation, click ok
6. Another file browser will open, allowing you to select the directory you want to create the label folders inside of. It is recommended that you have an empty folder created specifically for this purpose. Select your folder and click ok.
7. The next window that appears will allow you to set the slice size. If you are fine with the default of 1000x1000 pixels, simply click no. To set a new size, type the size into the window and click ok. Please only type integers into the box.
8. Finally, three windows should open: one displaying the parent image, one containing the GUI and one containing the current slice to be classified. Choose the appropriate label for the current slice.
9. Once you click a button, the program should automatically progress to the next slice. Once all the slices have been labelled, the program should progress to the next image. For best practice, it is recommended to have patience and not sort too rapidly as this could outrun the program. Once the program has successfully sliced every image in the parent folder it will automatically exit.
10. If the user should need to exit the program before progressing through every image, it is highly recommended to do so only after the beginning of the next image. Failing to do so will result in the creation of duplicate slices for a specific image. To exit the program, simply press the close button.
11. Upon running the program for the next session, it is important to be consistent with choosing the same folder for slice storage as this is how the program knows which images have already been sliced.
12. RECAP: Make sure all of the images you want to slice are in the same parent folder. Make sure you are consistent with selecting the folder for slice storage. Be patient with the classification speed as you could outrun the program. Exit the program only after labelling all of the slices for the current parent image.

Train_Test_Creator.py

1. Open the python environment (IDLE or Spyder depending on your version)
2. Open the program, hit the F5 key on your keyboard
3. A message box will open prompting you to select the folder containing the slice subdirectories, click ok
4. A file navigator will open, select the file you used as your slice storage file during the SplitnClass program.

5. That's it
6. You should now have two additional folders in your chosen directories named train and test. These folders should contain biotic and abiotic folders having the appropriate amount of images for an 80/20 split

TPR_ML_v1.0.4

1. Open the python environment (IDLE or Spyder depending on your version)
2. Open the program
3. Scroll to the third section from the top. You will need to make some changes to the variables here
    a. Type the path to the train folder in your sliced image directory in between the quotations in the train_path variable.
    b. Do the same for the test folder in the test_path variable
    c. Type the path to the folder where you want your test results stored in the results_path
    d. Change the sliceSize variable to reflect the size of your image slices (if necessary)
    e. Set the number of classes in num_classes (if necessary)
    f. Set your dataset size in datasetSize
4. Press F5 on your keyboard
5. You should see the program training the network in the output console. Note the default number of epochs (or iterations through the training set) is 100.
6. Once the training is finished, the program will automatically test the model as well. You should see a percentage correct output in the console. The program will output a more comprehensive results evaluation in the form of an excel spreadsheet to the folder that you specified in results_path.
7. There are other parameters and variables that can be changed within the script but this is not recommended for anyone who does not have prior knowledge of machine learning and Keras documentation.

PredModelV1_0_2.m

List of toolboxes needed to run this script: Parallel computing (optional), deep learning toolbox, VGG16 (type vgg16 in the command window and follow the link), computer vision toolbox

1. Open Matlab. Open the program into the environment
2. Open the image labeler application (you can navigate to it by clicking the drop down arrow in the 'apps' ribbon)
3. Load the session containing all of the data and their respective labels into the labeler application. Click export, click 'to workspace', it should want to name it 'gTruth', click ok.
4. You should now have a groundtruth object named 'gTruth' in the matlab workspace
5. Click in the first section to highlight it. If you want to load an existing model, change the value of check to 'true' and then press ctrl + enter or click run section in the ribbon. Otherwise, skip this section and move to the next one
6. Run the next section. You can know if a section has run if new variables have populated in the workspace. Move on to the next section
7. Run the sections in order to execute the necessary operations for the program. A section labeled optional isn't necessary to train the model but might provide useful information.
8. You should eventually come to a section title 'Create and Train the FCN network'. If you loaded a previous network, this section should further train that network on the data in the environment. If you didn't load a previous network, this section will create and train a new network.

9. Once the network has finished training, the following three sections provide different ways to test the network.
    a. The first section allows you to test the network on a particular image in the testing set. Just change the value of picNum to the appropriate POSITION of the desired image in the testing set index. For instance, if I wanted to test the network on the first image in the test set, I would set picNum to 1.
    b. The second section allows you to loop through every image in the test set and compare the network results with the ground truth
    c. The last section calculates several numerical metrics based on the performance of the model. Be careful though, these metrics can sometimes be misleading and should only be considered as a guideline to the model's performance.
10. The next section gives you the option to save the network. If you want to save multiple networks, simply change the name of the network in the first argument of the save command.
11. The sections following this section are function definitions that are called at other points within the program. These need not be run explicitly but can be altered by someone with the appropriate knowledge if need be.