Embedded Face Detector System Capstone Project Report

Team #5: Pranav Puritipati, Dongyeong Kim, Prem Chandrasekhar

ENEE408M: Section 0101

Shuvra Bhattacharyya

05/09/2022

Table of Contents

Signature (Approval) of Each Team Member

Pranav Puritipati

- Created and implemented the strong classifier actor
- Summarized the Viola-Jones algorithm in the main README file
- Created the classifiers directory
- Worked on the naming conventions for both the weak classifier and strong classifier files
- Worked on the scripts directory

"I pledge on my honor that I have not given or received any unauthorized assistance on this assignment/examination."

Pranav Puritipati

Dongyeong Kim

- Explained Viola Jones training systems and its algorithm of AdaBoost.
- Created integrator for the integrated image
- Implemented utils based on skeleton of Prem
    - Weight updates
    - Threshold updates
    - WeakClassifiers
- Debugged program in classifiers

"I pledge on my honor that I have not given or received any unauthorized assistance on this assignment/examination."

Dongyeong Kim

Prem Chandrasekhar

- Created and maintained directory structure for Github repository
- Wrote Bash scripts for generating random features, downloading dataset, performing classification, training classifiers, and measuring accuracy of classifiers
- Designed, Implemented, and Debugged Welter Actors/Graph programs, Driver program, and Training/Adaboost program
- CMake Compilation for Graph, Driver, and Training projects
    - Allowed for debugging using gdb
    - Cross-compilation for Raspberry Pi
- Trained classifiers and measured performance
- Created test cases for classification using various classifiers and images, training, and accuracy metrics

"I pledge on my honor that I have not given or received any unauthorized assistance on this assignment/examination."

Prem Chandrasekhar

Executive Summary

The main goal of this project was to use what we learned about the Viola-Jones algorithm to develop our own embedded face detector system (EFDS) that identifies human faces in any image or image regions. To achieve this, our team implemented a features/classifiers configuration subdirectory defining each feature type, created the structure for the image subwindow, implemented the weak classifier actor, implemented the strong classifier actor using the Viola-Jones algorithm as a template, created the image integrate and file write actors, designed and implemented the EFDS graph, and wrote a test suite to test various images.

The main design that we considered for our project was the dataflow graph implementation for the EDFS graph, which we created to visualize how our driver program read in the different actors in our project. In our design, we used four different actor types to read in the classifiers and images, convert the images into 2D vectors, integrate the images, classify the images, and then write the results to the files.

Similarly, in each of our test suites, which uses the driver program, builds the executable by loading in the C and C++ compiler and debuggers, scanning the target dependencies from cmake build, and then creating the object and executable files. Then, the executable is run, which reads in the classifier file, goes into the scheduler, and then goes through the different classifier modes (configure, read, classify, and continue). Then, the results of the classification will be written to the results text file, and compared and validated to our expected-output text file. Additionally, the entire process of the test is documented in our diagnostics text file, and any errors that we expected in the test will be documented in the expected errors text file.

Main Body

Goals and Design Overview

The goal of this project was to design and implement a face detector system targeting the Raspberry Pi using the Viola-Jones algorithm. This involved designing Haar-like features to identify faces, creating a dataflow graph that allowed 24x24 grayscale images to be classified using a cascade of classifiers, and creating scripts to test and automate several steps in the process. Lastly, the entire application would be cross-compiled to run on a Raspberry Pi.

Originally, the goal was to be able to apply the face detection algorithm to an image of any size, and the features and image subwindow would be rescaled and translated across the image to detect faces. However, this is significantly more complicated than we anticipated, so we focused only on 24x24 images. Additionally, we intended to train a cascade of strong classifiers, and although our graph program can support a variable number of strong classifiers, we were only able to train one strong classifier due to long training times.

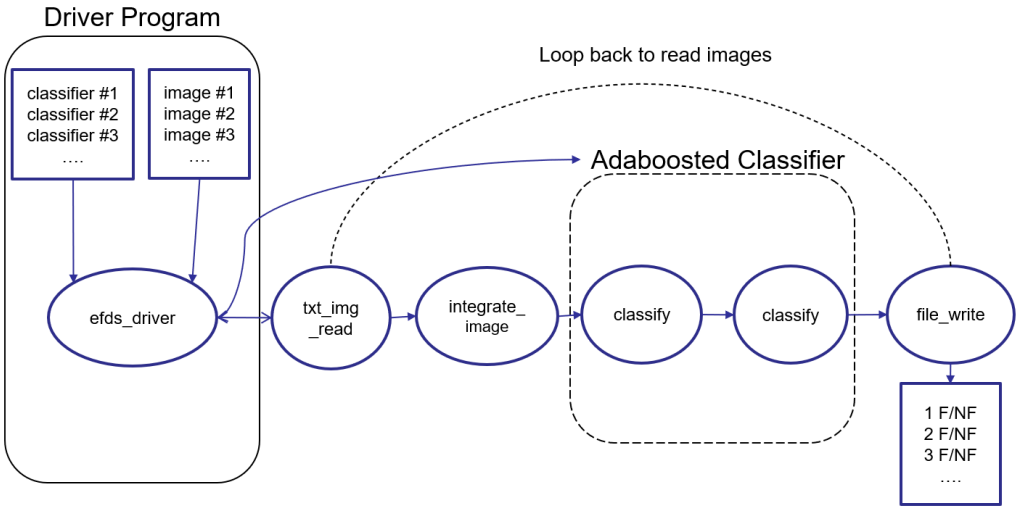

Figure 1. Dataflow for Embedded Face Detector System Graph Implementation

As illustrated in Figure 1, the driver program reads in the classifiers. The dataflow graph contains four types of actors: txt_img_read, integrate_image, classify, and file_write. In each iteration, the graph reads in a 24x24 grayscale image as a text file, converts it to a 2D vector, integrates the image, passes the image through the classifier cascade, and writes the result to a file.

The classify actor represents a "strong classifier" which holds several weighted weak classifiers. The mode design of this actor is shown in Figure 2. It can hold an arbitrary number of weak classifiers. After the classification is made, the result is written to the image subwindow and passed to the next actor. If the received image subwindow is already rejected, the classifier does not perform any computations and just passes along the image subwindow.

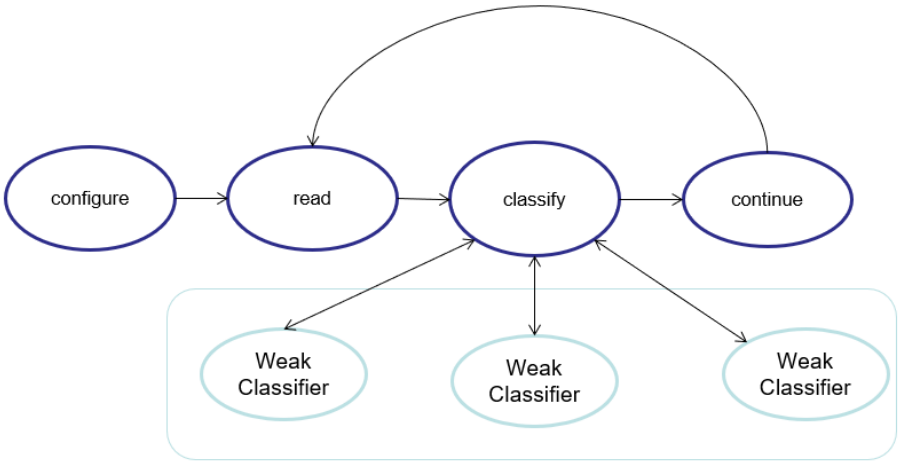

Figure 2. Design of the Classify Actor Modes. The read classify mode relies on the WeakClassifier class to perform the classification

One of the key functions of Viola-Jones algorithm is its classifier. Once the classifier classifies its feature value based on all input training integral images and updated threshold, it can update the threshold and its weight. As illustrated in Figure 3, the point is that with the updated value, the updated weight can be used as a strong classifier with error values.

4. Update the weights:

$$w_{t+1,i} = w_{t,i}\beta_t^{1-e_i}$$

where $e_i = 0$ if example x_i is classified correctly, $e_i = 1$ otherwise, and $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$.

- The final strong classifier is:

$$h(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{otherwise} \end{cases}$$

where $\alpha_t = \log \frac{1}{\beta_t}$

Figure 3. Weight update and its usage as a strong classifier.

The basic challenges to this are 1) implementing the classification design that allows for a variable number of classify actors, 2) training classifiers through the AdaBoost process described above, and 3) collecting performance metrics about the accuracy of said classifiers.

Realistic Constraints

The main constraint on the implementation of this project was that the face detection application needed to be implemented using Welter and FIFO buffers. This required the program to be divided into several actors that performed an individual task when invoked. This made the

The programming languages used for this project were limited to C++ and Bash. External libraries like OpenCV could not be used.

<Testing requirements, which impose minimum constraints that must be satisfied for design validation.>

The intended hardware platform for the system is the Raspberry Pi, despite being developed on the Linux server. This requires the project to be cross-compiled so the executable can be run on the Raspberry Pi. Additionally, since the Raspberry Pi is a relatively low-resource device, the face detector algorithm must be leaner and more efficient than some image algorithms, such as Deep Learning models.

Time was also a constraint. We were given a limited amount of time to develop, test, and measure our system. One interesting concern was that weak classifier training required the classification system to be functional. However, once we had

A constraint more specific to our group is the number of people.

Engineering Standards

Discuss how appropriate standards have been incorporated and guided design choices in the project. Examples of relevant standards to discuss here are the use of C++ and Bash as standard programming languages that are involved in the project.

In our project requirements, the training module was to be implemented as a C++ program, the EFDS and parts of the training were to be implemented using Welter-C++, the scripts subdirectory was to be implemented using Bash, and the overall project would also include text files and cmake code.

C++ is one of the standard programming languages used throughout our project. For each actor, we had a header file, which listed the libraries and other header files we wanted to include, define directives, and function headers to be implemented in the .cpp files, and the use of both public and private members. One C++ standard that we incorporated and guided our design was

keeping the class variables private and the functions public so that only the functions can be accessed by other actors. Additionally, we attempted to enforce a common style for all of the C++ programs we wrote for this project.

With regard to Welter-C++ specifically, the define directives are used to indicate the modes of the actor. Additionally, the enable and invoke functions are used to implement the actor firings, the reset function is used to return the actor to its original state, and the connect function is used to connect an actor to its input and output edges.

Since the classification system was designed using Welter actors and graphs, this required us to create each element to switch between modes. With specific regard to the graphs, the simple scheduler was used, which is responsible for enabling each of the actors before they are fired.

Bash was also another standard programming language we used throughout our project. For each script, we included *#!/usr/bin/env bash* as the first line in our scripts to use the default version of bash, and then we wrote the linux commands we wanted to run sequentially to perform the actions we wanted.

For our classifiers directory, we chose our file formats for the weak and strong classifiers so that the files properly describe each classifier type. Additionally, we also used CMake for our test suites as a way to generate build tools for cross-compilation on the Raspberry Pi.

Alternative Designs and Design Choices

Highlight the Pareto designs (design points) that you implemented for your design exploration. Provide the real-time performance and image analysis accuracy for these designs, and plot these values on a performance versus accuracy plot for your set of Pareto designs.

Tradeoff: Performance vs Ease of Implementation

For collecting the feature values when training,

Faster training (by limiting training data) vs Better features/more adaptable

Test complexity vs Strict Assumptions about Inputs

wc_0.txt -> wc_1024.txt

In the threshold update, there were two options, one was to update its values from large size to small size instead of increasing its values in small size 0.1 or 1. And the other option is sort the feature values and regard its values as a single threshold value. There are some pros and cons for two solutions. The first one has less runtime, but it needs more scripts and loops, so when the program has errors, the debugging time takes longer. And the second option has a longer runtime, but the sorting function is already in the standard C++ library, and it only has one loop, so it is easier to implement, and debug. We observed the feature values can vary with thousand values or hundreds values, so the first option is normally better, but for the time conflict, we chose the second option for threshold update.

Technical Analysis for System and Subsystems

Document your system design and what you learned from it — e.g., what, if any, improvements did you discover for the design as you developed, experimented with, and analyzed it; how efficient is the design from a design complexity point of view (i.e., in terms of having an implementation that is easy to understand and maintain)?

The design is straightforward, and we spent some time understanding how the AdaBoost works in the Viola-Jones face detector, because the explanations were different according to the sources we found. But there were some potential improvement plans for Adaboost. The basic principle of image machine learning is to read an image, calculate the image, and use that data for updating some parameters for machine learning, but the problem is the input learning images are thousands and the cycle of training is one per each time, so we should wait for the next cycle. So, we thought of using parallel processing with two options. The one option is making 5 parallel processes, so the training process can be done for each haar feature at the same time, and the second is making parallel processes for whole training images, which means making thousands of parallel processes, so read and classify the images at one time. But the biggest problem is time conflict, and second, we are not familiar with the parallel process functions and it can work on a normal PC that we use in university, but we are not sure how the Raspberry Pi's performance works. The parallel processing is good for its performance usage and runtime optimization, but the performance of the platform had the problem.

Design Validation for System and Subsystems

Discuss the benchmark inputs that you have used to evaluate and compare the performance of candidate designs. Explain why this set of benchmark inputs is effective at evaluating your designs.

For evaluating the design of the subsystems,

Training/Testing Accuracy:

- False positive, False negative

$$F_{measure} = 2 \times \frac{precision \times recall}{precision + recall}, \quad precision = \frac{TP}{TP+FP}, \quad recall = \frac{TP}{TP+FN}$$

Test Plan

Elaborate on the organization and implementation of the test suite that you have used to validate the functionality of your designs.

Each of the cases in the test suite used a Bash script that gathered the appropriate directories and files for the classifiers and images, and then passed them to the EFDS graph driver to perform the classification.

The Bash scripts would convert images to txt files as needed,

For training

The functional check for training parts can be validated mostly with checking the error calculation. So, we focused on the fact that there were some points where the error value is minimized and there are largest alpha values that can be created.
The other sides are about the error with the input integrated images, and other parameters, like threshold, column, and rows are delivered in appropriate values and types.

Project Planning and Management

We used a Google Doc and GroupMe chat to keep track of the tasks that each of us was working on each week, as well as planning for tasks to do in the future. We used Github to keep track of changes in the project codes, and used the issues feature for reviewing changes as needed. Additionally, we held semi-weekly Zoom meetings to work together when needed and to keep each other informed about progress being made on the project.

Conclusions

- *Summary of design methodologies, key creative approaches and potential impact.*

Creative approach: Generating random features instead of exploring the entire feature space

Creative approach: Creating integrate image actor instead of performing integration in each actor classifier

Creative approach: Classify actor has single output FIFO that holds the image subwindow.

- *Review challenges faced and solutions.*

Theoretical challenge: It was difficult to fully comprehend the Viola-Jones algorithm at first, especially applied to larger images. Initially, we interpreted the features to only specify feature type, width, and height, and the position on the image subwindow would be determined by the image.

Implementation challenge: Implementing a variable number of actors.

Collaboration challenge: It was difficult to work on the same repository and following the same style.

Version control: There were a couple issues with merge conflicts, setting up the repository, etc.

- *Discuss lessons learned.*

C++:

- Learned more about stdlib (vectors, iterators, pairs), enums, classes, and file streams (reading in structures).

Git:

- Basic version control. Pulling, Committing, Pushing
- Gitignore

Bash:

-

GDB for debugging

Better communication and time management

References

[1] L. Shapiro, “Face Detection via AdaBoost,” *University of Washington Computer Science & Engineering*, 21-Feb-2017. [Online]. Available: https://courses.cs.washington.edu/courses/cse455/17wi/notes/FaceDetection17.pdf. [Accessed: 07-May-2022].

[2] P. Viola and M. J. Jones, Kluwer Academic Publishers, Cambridge, MA, tech., 2004.

[3] P. Viola and M. Jones, IEEE, Cambridge, MA, tech., 2001.

[4] R. Gupta, “The intuition behind facial detection: The viola-jones algorithm,” *Towards Data Science*, 12-Feb-2020. [Online]. Available: https://towardsdatascience.com/the-intuition-behind-facial-detection-the-viola-jones-algorithm-29d9106b6999. [Accessed: 05-May-2022].

[5] S. B. Lipman, J. Lajoie, and B. E. Moo, *C++ Primer*. Boston, MA: Addison-Wesley Professional, 2012.

[6] University of Maryland, College Park, Face Detection Using the Viola-Jones Algorithm, 2022.

[7] University of Maryland, College Park, Training Process for the Viola-Jones Algorithm, 2022.

[8] “Vocal Technologies,” *VOCAL Technologies*. [Online]. Available: https://www.vocal.com/video/face-detection-using-viola-jones-algorithm/ [Accessed: 05-May-2022].