Image Binary Classification

6.111 — Fall 2015 — Project Proposal

Juan Huertas and Andrés Salgado-Bierman

November 13, 2015

1 Overview

Image recognition is a broadly used practice in industries ranging from video games to manufacturing. However most detection cannot be done in real time at high reliability on hardware. Current implementations use feature algorithms that are not as effective as traditional software approaches, or require large amounts of hardware to generate CNN models. Software on the other hand will never reach the specificity of creating efficient timing in a digital circuit. Our recognition system will be able to generate a feature representation of the textures from a low resolution picture of a face to classify and display objects in real time. For our final project we want to design and implement our own facial recognition system in hardware using efficient feature representations as well as some hardware adjusted software approaches to object recognition via linear classifiers. The basic output would be a yes/no response to whether there is a face in the video frame. A stretch goal is to create a display that identifies where the face is located in the frame.

This proposal will first cover the overall design of the project. The rationale behind certain specific design decisions will be included the section on implementation along with the specifics of the hardware approach. A project timeline and gantt chart of key milestones will be presented. Lastly there will be a discussion of how the design will be tested during development and the resources required for the project.

Briefly on the motivation for the project, Juan has had the idea of implementing machine learning algorithms in software for some time and in fact took this course to learn more about the timing related to digital circuits. Andrés is a course 2 student with an interest in automation and robotics. He has been exposed to machine learning and computer vision projects in the labs he has worked in and has been waiting for the opportunity to work on such a project himself.

2 Design

The core of our project is successfully implementing the binary classifier in hardware; to be able to answer the yes/no question is there a face in this window of pixels. Achieving this classification in hardware is complex, but a binary output is not that interesting. Stretch goals include running many instances of this classification in parallel on multiple windows within a picture. This approach would allow detection of where a face is in the image, and a visual display could be created to highlight where the face stands in the picture. While facial identification is what we have chosen to focus on, we aim to develop a system that can perform a wide range of classifications.

This paragraph will briefly discuss the machine learning approach for face recognition and then delve into the major components required to implement the classifier on the FPGA. How can an a computer recognize a face? The computer receives a photo made of many pixels that have different level of brightness (this project will work with black and white photographs). The texture of a face as compared to other objects is quite unique. The computer interprets texture in this case as how the brightness of different pixels compares to the brightness of pixels next to them. There are four major components to the project that can be seen in the block diagram below (Figure 1). Preprocessing crops the image to a manageable size and stores brightness comparisons. Feature extraction evaluates these brightness comparisons in different regions of the photo. Each feature is just an abstract representation of texture for some part of the photo. Classification takes the features and casts a vote for each one whether the image is part of a face or not. AdaBoost takes a weighted sum of the votes to make the final yes/no decision. Candidate window generation and VGA display are stretch blocks of the project and will be discussed in greater detail at the end of the implementation section. The 6.111 labkit will be used for this project.

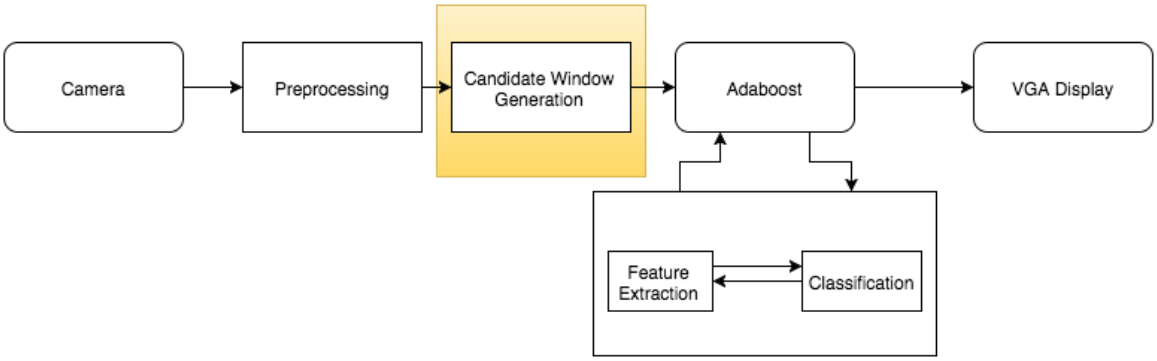

Figure 1: System Overview

2.1 Algorithm Selection

Several algorithms are required for object recognition, our proposed implementation opts for minimizing complexity of the computer vision algorithms implemented while achieving high fidelity results in three key areas: candidate window generation (Rosenfeld), feature kernel(Local Binary Patterns), and classifier (AdaBoost+SVM). Local Binary Patterns are broadly used in face detection algorithms, the kernel is used as a way to take color or chrominance data from images and translate it to texture data, the algorithm is not computationally intensive requiring no divisions or multiplications to be carried out. The computation of the classifiers have been abstracted out into software such that only the final computation is carried out in hardware using precomputed, prestored coefficients to calculate the classifier boundary output. It is this precalculation of coefficients that makes our system robust to classify a wide variety of objects. Example images are used to train the coefficients. To classify a different object the coefficients can be trained with different example images. We are placing the Rosenfeld algorithm on our stretch goals for the project pending the first test of classifier accuracy which may demand some tweaking of or feature representation or classifier generation, the Rosenfeld algorithm is used to label and identify connected components.

2.2 Implementation Decisions

As the model for our feature representation would ultimately dictate the accuracy and sophistication of detection possible,many of our considerations will have to be experimentally derived, however

several design tradeoffs that will affect the overall performance of our detection have been identified herein. There are 256 different possible eight bit local binary pattern values, each of which represent a dimension along which our features are defined, to achieve a sparse representation features include information from 80 pixels in an 8x10 block, this limits the feature size to 256 variables each 8 bits. Knowing our lower bound on the feature size we decided to implement detection on a 128px by 150px candidate window giving us a total of 240 features to classify, this is done to be able to keep running totals of classification values so that at most 256, 8 bit numbers are computed at once. The effective tradeoff we are making is losing some invariance to size (we will not be able to detect objects that are so far away they correspond to a single 8x10 block) and minimum camera range for detection (we will not be able to detect objects that are so close that they do not fit into a single 128x150 candidate window). To reduce memory constraints we have decided to calculate an 8 bit local binary pattern from the chrominance data only, allowing us to use the preexisting zbt architecture to store 4 local binary pattern values per block.

3 Implementation

3.1 Image Preprocessing

The first step for this project's implementation is to take the input from the NTSC camera and preprocess the data to compare pixel brightness. To make the number of calculations more manageable for the classification algorithms the image is cropped to 150x128 pixels, a size which is convenient for later steps. The NTSC camera outputs luminance and crominance values for each pixel. The preprocessing block will take the 8 bit luminance values for the brightness of each pixel. The preprocessing block will also calculate the linear binary pattern (LBP) for each pixel (Figure 2). The LBP is our measure of how the brightness of one pixel compares to the brightness of the pixels around it. In a 3x3 grid the middle pixel will be surrounded by 8 other pixels. The LBP just compares this middle pixel to the other pixels around it. The LBP is an 8 bit number where each bit represents whether one of those 8 surrounding pixels is brighter or darker than the middle pixel. 0 means it is lighter and 1 means it is darker. The 8 bit LBP is stored in ZBT memory for each pixel in the 150x128 window. A benefit of the LBP is that as a relative brightness measure it should be robust in different lighting conditions.

3.2 Feature Extraction

The frequency of different LBP values is unique in different areas of the photo, allowing for unique features to be extracted. The 150x128 photo is divided into a 15x16 grid of 10x8 pixel boxes. The LBP values from each of these boxes will be used to create a feature. A feature is a 256 value histogram where each value is a tally of how many of the 80 pixels in the box share a LBP value. Each tally is stored as an 8 bit number making for a 2048 bit feature made of 256 8 bit tallies. The output of the feature extraction block is one 2048 bit feature (Figure 3). The block performs the above calculations in series 240 times for each different feature.

3.3 Candidate Classification

A feature can be viewed as a 256 element vector in the space of features and represents the texture of that part of the image. To obtain the feature's vote the cross product of the feature and a 256 element coefficient vector is taken. There are 240 different coefficient vectors for each of the different features. The coefficients are obtained in software with a matlab script that mimics the project's implementation on the labkit. The coefficients are then loaded in memory on the labkit as a lookup table. In matlab example photos will be fed to the program, and the coefficients will be trained with a support vector machine. The example photos are taken with the NTSC camera and transfered to

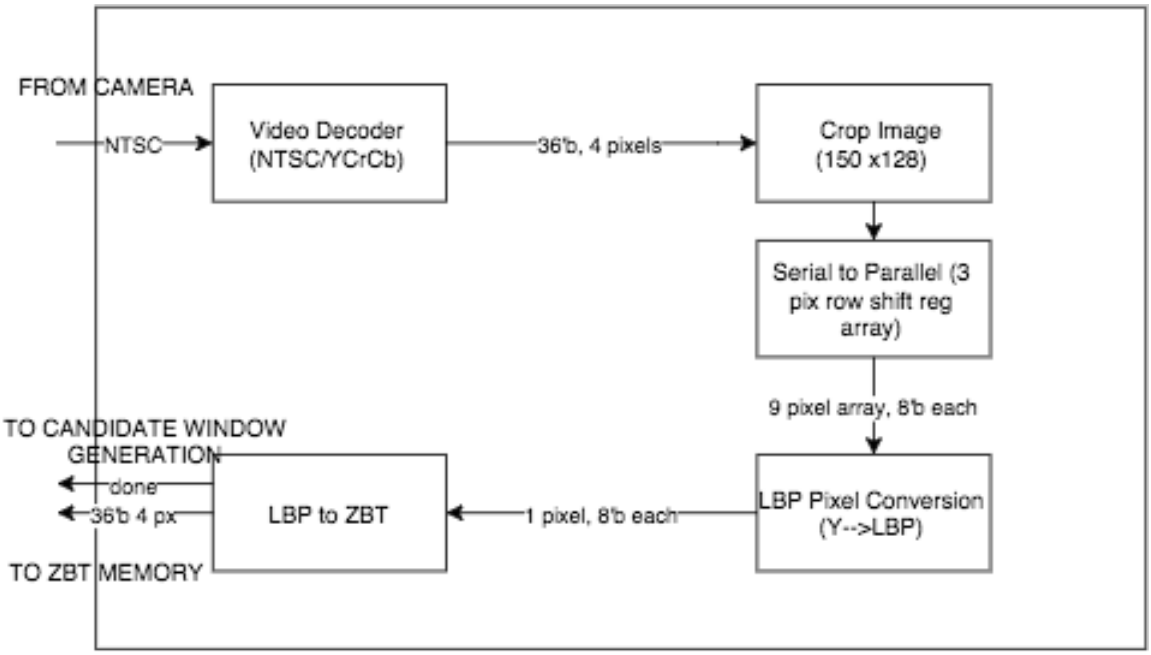

Figure 2: Preprocessing

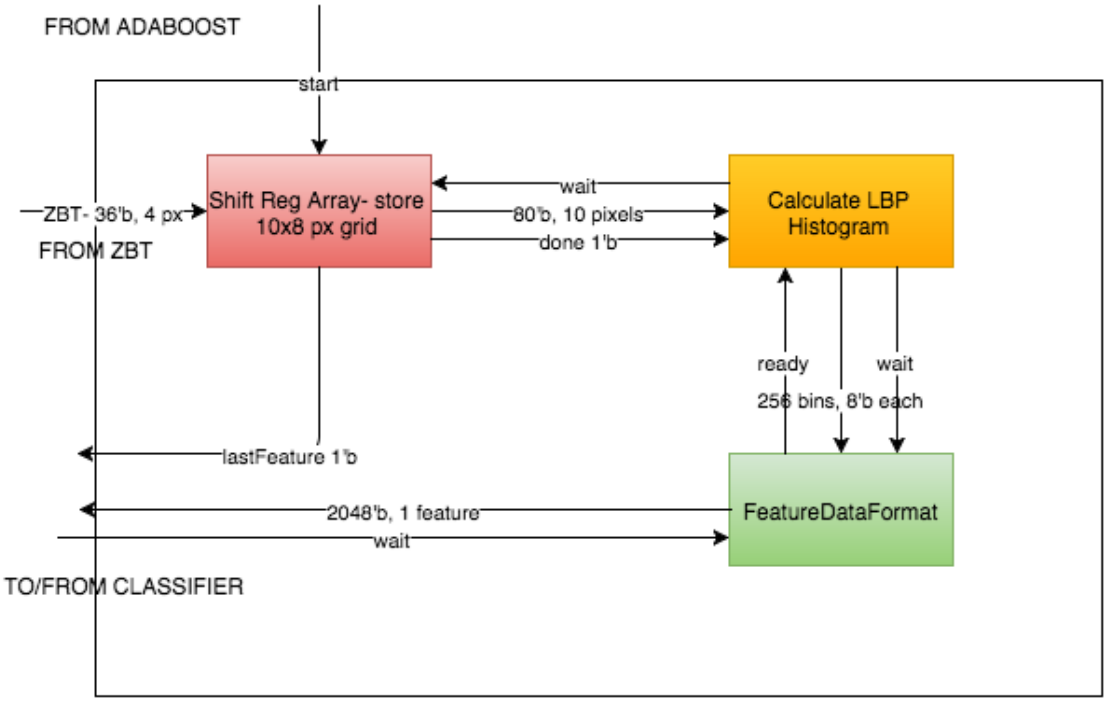

Figure 3: Feature Extraction

the computer from the labkit with existing verilog from the course. A python script will be written to interface with the existing verilog module. By taking the aforementioned crossproduct a 32 bit

fraction is obtained for each feature, with 16 bits for the numerator and 16 bits for the denominator. The candidate classification block sends a 32 bit classification to the AdaBoost block 240 times per image sending the classification of each feature (Figure 4).

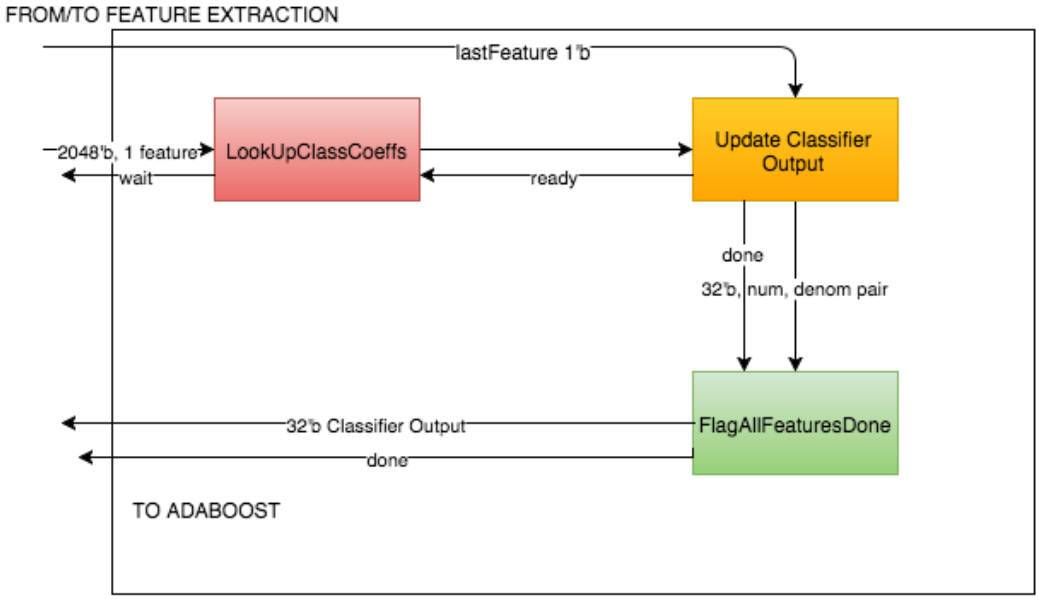

Figure 4: Candidate Classification

3.4 AdaBoost

AdaBoost functions in a similar way to the candidate classification (Figure 5). Each 32 bit value from candidate classification is multiplied by a 32 bit coefficient. The coefficients are representative of the importance of each feature to the overall classification, and are used to improve accuracy. The sum of each of these multiplications passed through an activation function that gives a binary answer to the question of whether or not there is a face in the photo. Coefficients are obtained and stored in the same manner as in the candidate coefficient block.

3.5 Candidate Window Generation and VGA Display

With regards to the complexity of implementing a linear classifier in hardware, candidate window generation and a more engaging display were left as potential add-on blocks towards the end of the project. By cropping the output of the NTSC camera the core blocks can detect whether or not there is a face in the cropped window of the larger photograph. A linear scan of the photo could be performed to generate multiple candidate windows. Then the core blocks could be instantiate multiple times and run in parallel for each of these windows. The candidate window generator would be responsible for the linear scan and generation of the multiple windows. This approach allows

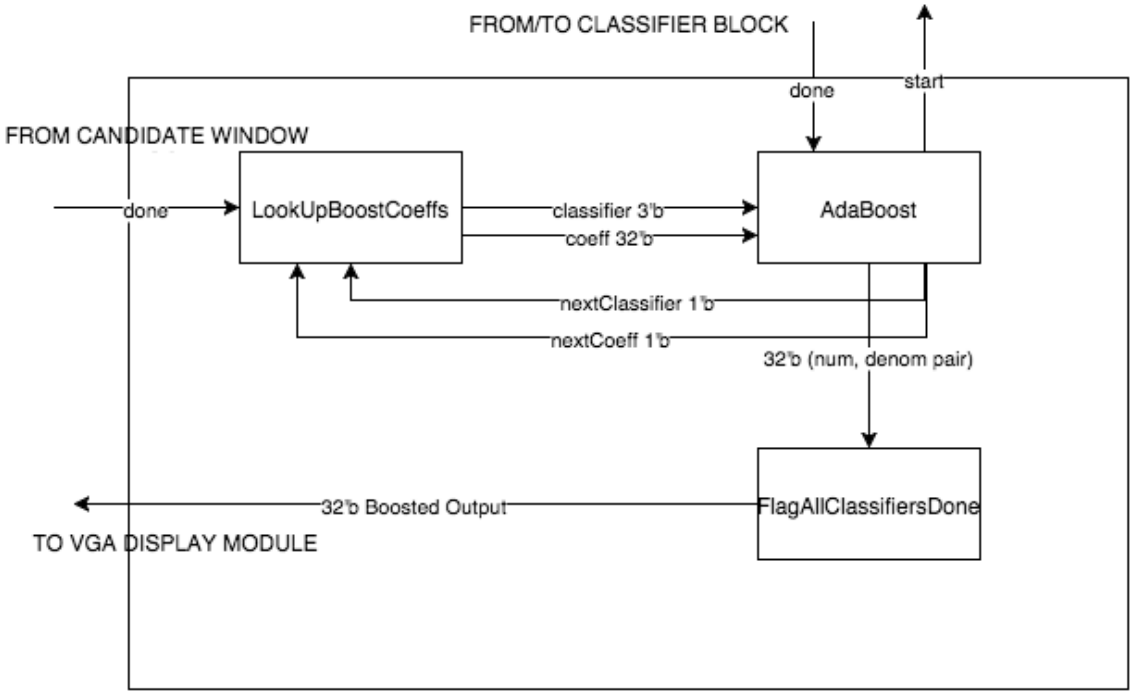

Figure 5: AdaBoost

for not only detection of a face in a window, but through the evaluation of multiple windows this approach allows for detection of where the face is located in the larger photograph. An interactive display that highlights the photo where a face is detected would be created in the VGA display block.

Much of the discussion throughout the implementation section has focused on the processing of a single image. However, the goal of the project is to perform this processing in real time for each frame of the video stream. The design of the linear classifier was optimized to be able to perform all the necessary calculations between frames of the NTSC camera. The rational behind these choices will be discussed in further detail in the following section.

4 Timeline

The first step of the project is to complete the image preprocessing modules, because the other modules are dependent on the preprocessing's output. At the same time a software implementation of the project will be built. This implementation will serve as a baseline to see what classifications the system is capable of performing. As well, implementing the project in software will provide invaluable lessons to the hardware implementation.

Once the preprocessing and software implementation is complete, images can be taken from the labkit to start training the coefficients. Feature extraction is the next important task, as classification is performed on the features. Work will start on the classifier at the same time and continue into the following week to allow for flexibility. AdaBoost has a similar implementation to the classifier and should be easy to start once the classifier has been worked on. The aim is to complete all major

parts of the project and start work on integration by Thanksgiving.

The break down of tasks between team members is detailed in the project gantt chart (Figure 6). Modules will be worked on separately until integration. No work is scheduled over Thanksgiving. Following the holiday a buffer period is left to allow for work on debugging and creating an engaging display time permitting. All work should be complete on December 3rd.

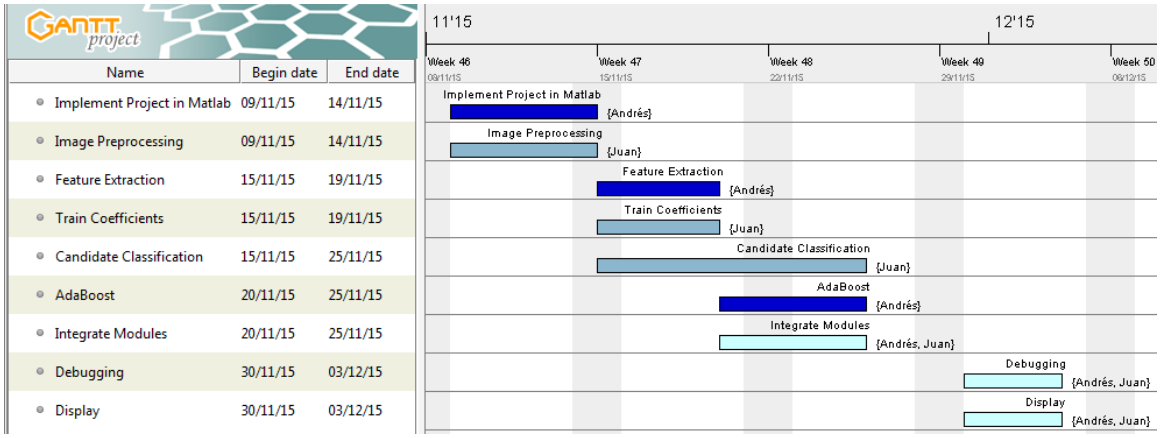

Figure 6: Project Gantt Chart

5 Testing

The first test of the planned implementation is in the software implementation. The planned Matlab implementation mirrors all the calculations that would be performed on the labkit. A successful software implementation will verify the approach for the hardware implementation.

Test benches will be written for individual modules within the larger project blocks as they are written. When the individual modules are integrated a test bench will be make to test each block's functioning as a whole. A test bench will also be written for the final integration of all modules.

To test the output of individual modules, as well as the final classification decisions of the system, output from the software implementation will serve as a baseline.

6 Resources

Few external resources will be required for the project. A NTSC camera has already been borrowed. Beyond the camera only the 6.111 labkit and VGA display will be used.

7 Conclusion

This is a project that we are excited about building. Andrés has been waiting for a chance to work with computer vision and Juan has been wanting to build such a system for some time. The complexity of the project is a good fit for 6.111. There are four major blocks that each function as a fsm with multiple modules operating within them. The planned implementation makes use of the

strengths and limitations of the labkit. Expensive one time calculations are performed in software and results are loaded into the FPGA. The digital circuitry of the labkit along with select algorithms are leveraged to have real-time classification on a 30 frames per second video stream. Finally, our real-time classifier can be used by students in future years who would like to have a computer vision aspect to their project.