CIS 431 Final Report

Team-Image

Je Min Son, Jim Lee, Yanhua Luo

**Initial Proposal**

Our team, Team-Image, are targeting to do our project related with image processing. The reason we chose to do this field is because one of the fields that parallel programming is commonly applied to is image processing which sounded very interesting for us.

Digital images are built based on pixels which each pixel has its own specific values and they are completely independent. If we focus on only processing a single image, there is no point to apply the parallel programming to solve the problem. However, if we try to process a video, then parallel computing becomes extremely useful in performance. In the video processing, the video is represented by the structure units, frames or shots which can be understand as an image. The purpose to be explored in this project is to detect a static image/object from a video, and determine whether the given image belongs to the video. Additionally, outputting where the image appears (in time) in the video.

More detailed reason for us to choose this topic is image/object detection is an important area in computer vision which is common used nowadays. Moreover, there are many hilarious screenshots from some videos or movies which would be convenient for people to find out where the screenshot is from by using our program. Furthermore, our ideal product can be detecting specific objects for specific purposes. For example, we can have camera in front of the feeding machine for pets like dogs or cats. Whenever, they are hungry and come up to the machine, the camera detects their face and provide them their foods.

To accomplish our goals, we have divided our project into two approaches. The primary approach is to identify whether a given screenshot image belongs to the given video, or find out which video is the given images belongs to. The secondary approach is object detection which tells the program whether this is a specific object appeared in the video. In this step, we will use machine learning knowledge to create a model to train it with data and let it predicts the output. However, it will not implement the moving object tracking feature in this project.

We are expecting some challenges that we might run into. First, it is necessary for us to get familiar with any library of C++ that manipulates the video. Our whole project is based on video splitting and searching, so we need to figure out how to use any kinds of vision library like OpenCV. Second, any algorithm which can help us to detect images should be learned and comes into use. The magnitude of videos is huge, so it is crucial to find and apply an efficient algorithm for completing the project. Last but not the least, we should find out all the possibilities to apply the parallel computing methods we learned in class to the project because it will not only boost the efficiency of our project, but also put what we learn into practice.

**Introduction**

With the development of artificial intelligence, the technique for processing video nowaday is fancy and amazing, such as frame filtering, object detection, and so on. Among of them, object detection is really an extensive application. For example, CCTV needs powerful object detection methods to detect any potential threats on the street, campus or a supermarket. Another example can be, when motor corporations who are spending money to advertise their products in a movies will want to track how long their cars showed up in the movies. To measure this part to check if the movies are advertising their products as well as expected, they will want to know about object detection. Therefore, it is worth working on a object detecting application. Once doing some relative researches online, many different resources was found, which was all about how to implement an application to detect an object in a video. There are a lot of ready-made machine learning library to accomplish it pretty well. However, we notice that few of them mention how to optimize the code by parallel computing in order to speed up the application. Video processing is actually multiple frames manipulating, which should be well optimized by parallel computing approach. Therefore, we are going to apply what we’ve learned in parallel computing course for implementing a well paralleled object detector. Furthermore, we will be focusing on OpenCV which is computer vision library that is very useful and powerful tool for image detection.

**Goals**

Our ultimate application should be a modern, fast and intellect object detector that fulfills the desired perfect expectation listing below:

1. There are bunch of objects provided to be selected. That is to say, our application should provide some built-in objects for users to detect in a video.
2. A easy-accessing tool for user to train a new object models. With all specific steps that users have done, a new mature models should be generated to put into the application.
3. The application should support digital container format as much as possible, including avi, mp4, rmvb, mkv, etc. in order to get rid of the risk that some videos cannot be opened.
4. The detector should run accurately. That means the detecting frame should pop up as long as the targeted object shows up in the video and go along with the object. Once the targeted object in the video disappears, the detecting frame should also vanish as well.
5. The application should provides multiple options: (1) play and pause the video; (2) calculate and record the total detecting time; (3) play the video of a specific timestamp (i.e when the targeted object appears).
6. Provide some supporting tools: (1) a video cutting tool that splits the video into frames; (2) model viewing tool that visualizes all object detecting models, letting users have an idea of how models look like and what potential objects can be detected through the object.
7. Supporting multiple object detection. That is , the application should allow users to detect different objects in the same video at the same time, using the distinct color of frames to capture the objects.
8. Supporting real time detection. Therefore, object detector should not only apply for detecting objects in a video, but also apply for objects detection in real time camera. For instance, the object can be utilized on a security camera device, making the alarms go off if a stranger appears in the cctv.
9. A user-friendly UI should be provided to have users get access to any functionality easily if possible.

Ideally, if playing the video while detecting objects, the speed of the video playing should be as fast as the video normally do. In another words, object detecting process should not have a negative effect on video playing. Additionally, if detecting objects without playing the video, the process should be as fast as possible, i.e processing a 1.5 hours movie in to less than 5 minutes, and output the record; showing how long the detecting objects appear in the video.

**Objectives**

Since our team did not had much experience of implementing machine learning base application, we set minimum objective which should be feasible and deliverable. Additionally, we set ideal objective which should well-address any users need in detail.

**Minimum Objectives:**

1. Have a basic knowledge of video editing library of C++ for choosing the best one to use for our project. Specifically, we choice to use OpenCV which is the popular and powerful library of programming functions mainly aimed at real-time computer vision that would be extremely helpful for our project, including but not limited to video reading, frame processing, video writing, etc.

*Time: Feb. 5- Feb. 12 -- Finished*

1. Learn the methods of object detecting. As far as we know, it is mainly an application of machine learning. We need to understand how different models work for object recognition and why we ultimately use one of them instead of the others. What are the pros and cons of each model (algorithm). Additionally, we might need to figure out whether there are different approaches besides machine learning model that could handle objects detecting function.

*Time: Feb.13 - Feb.19 -- Finished*

1. Base on all of the info that we collected, we implemented the most suitable application that meets minimum of our basic requirements and expectations: (1) An convenient object training tools. That is to say, an simple application which allows users to train an targeted object to be detected by doing the steps followed by the instructions. (2) A video playing mode that allows the object detecting function to run while playing the video. (3) A timer that records the appearing time of targeted objects in the video.

*Time: Feb. 20- March. 4 -- Finished*

1. Apply parallel computing methods that we’ve learned during this course to our project, making it to run significantly faster than sequential implement. We’ve covered ISPC, TBB, OpenMP, OpenMPI and OpenACC in class. The main idea of using these are to make any independent process to run paralleled as to reduce the total running time. No matter model training or video reading need implementing in parallel style.

*Time: March. 5 - March. 16 -- Finished*

**Ideal Objectives (which not only satisfy the condition listing above, but also):**

1. Implement more than three popular object detecting models, and providing multiple detecting modes to users so that they can choose which one they want to use. Each one may be more suitable for some kinds of objects then other modes.

*Not implemented yet.*

1. Design a UI for users to get access to any functionality easy, and including but not limited to popping up prompts and hints. Additionally, estimating the training model, speeding up or slowing down video playing, etc.

*Not implemented yet*.

1. Make it into mobile application which can detect the object directly from the camera in real-time.

*Not implemented yet*.

**What was attempted**

When we decided to do the image detection for our project, each one of us did lots of researches about it. Since we only had three members, it was easier to reach an agreement but it took us more time to do the researches. However, we tried few different things before we set into our final product. First, we started to learn about how does OpenCV and image detection are working, and one of the common example we could find easy was about face detection. Therefore, we attempted to do the face detection which came out well and gave us confidence about the project. After we finished learning how does the face detection works, we came up with two different options to detect the images. We tried both ways to figure out which might work better with our project, One way was capturing the object directly from the video and tracking how many frames the selected image showed inside the video. Other way was inputting image of the object that we want to detect, and do the image matching by each pixel to find out the same image.

Moreover, we attempted to divide the video into each single frame. We needed to use each frame to track the time, track the frame number, and detect which images were shown in which frames. For this attempt, we used car object for our practice object. After dividing the frames, we first attempted to train the car model. After we succeed it, we tried to train other objects. However, training each model took longer time then we expected. Additionally, since it was hard to train the models, we tried to do simper objects such as stop signs. At this point, we were pretty sure what we wanted to provide for our final product. Then, we tried two types of build-in OpenCV classifiers; HAAR cascade classifiers and LBP cascade classifiers were the ones. The results showed that HAAR cascade classifiers took longer train time but higher accuracy. On the other hand, LBP cascade classifiers took shorter train time but lower accuracy. This helped us to decide which classifiers to use for further attempt on our project.

After wrapping up the image detection parts, we were asked to do the parallel part. We tried to implement the parallelism that we learned during class such as OpenMP and TBB. However, we realized that it had encapsulated libraries like OpenMP and TBB already. Therefore, we had to find a different way to parallelize our code. Our attempt was using partitioning which was related with video parallelism and OpenMPI for the rest of the parts in the code. Moreover, we tried to use OpenACC to compare which parallel programming model works faster and efficiently, but this step wasn’t completed because we changed our plan to focus more on finishing up our project.

**Expected to accomplish, Final Description of What was Accomplished**

We had accomplished mostly everything we tried to implement. When we were working on face detection part, we made it available to capture the object first by letting the user to drag and select it, and used that captured image to detect the object from the rest of the video. This part was showing high accuracy with the person’s face since we were using the built-in face detection HAAR cascade classifiers. However, we wanted to detect something else, so we started training our model. After training few different models, we accomplished to build models such as cars and stop signs. As it was already mention above, these HAAR cascade classifiers were having high accuracy but took long time to train it. Therefore, we tried to use LBP cascade classifiers to make the training time shorter. When we were done with LBP cascade classifiers, we compared it with HAAR cascade classifiers, and then decided to use HAAR cascade classifiers since the result were showing higher accuracy even though it was slower.

Parallelizing the code was accomplished already in the OpenCV library, so we implemented the parts where it wasn’t parallelized already. One part we found is the part where video is divided into each frame. For this part, we used partitioning algorithm to parallelize the code more efficiently. Moreover, we used another method which was the TBB concurrent vector to divide the video.

**Observations**

We wanted to compare the results from different parallel computing methods. This part is showing the results from what we have observed from our application.

Unit: second

(#, #) => (number of positive source image, number of negative source image)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | (1,1) | (125, 125) | (2500, 2500) | (19000, 19000) |
| Sequential | 0.0054793 | 1.2059 | 20.678 | 176.075 |
| TBB | 0.0122974 | 0.3211 | 10.756 | 83.3774 |

Table 1: sequential vs. tbb::parallel\_for on reading the images

Unit: second

(#, #) => (number of positive source image, number of negative source image)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | (1,1) | (125, 125) | (2500, 2500) | (19000, 19000) |
| Sequential | 0.0036663 | 0.408031 | 19.710 | 170.183 |
| TBB | 0.0056658 | 0.093448 | 7.529 | 109.392 |

Table 2: sequential vs. tbb::task\_group for modifying the images

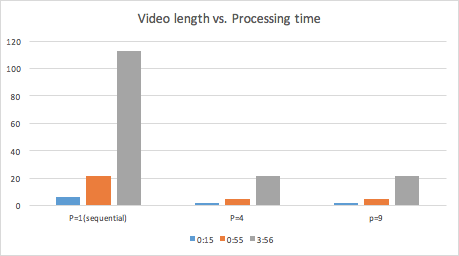
****

Table 3: sequential vs. OpenMPI for reading the video

**Final Description of What was Learned**

This project turns out to be an exciting learning experience for us, an experience that contains both positive and negative ways. We have learned a lot from this project; something exceeds what we have learned in the lectures during class. First off, while we just started the project, there was some misunderstanding and miscommunication problems in our team. In fact, we did not know what kind of image processing project we were suppose to do, and whether it was comparing two images to see the difference between them or was it going to be an images matching from a video. Moreover, we had to decide if it was going to be object tracking from the video or object to be detected. We spent quite a bit of time for just discussing the idea of this project. However, we were still even unclear with what the project requirements are after we started to implement our project coding. Initially, we decided to work on an application that can be useful to find the exact input images from a video frame; all team member spent a week or so of time on researching to find the possible ways to achieve this features with parallel computing involved. Our initial idea was to comparing pixel by pixel in RGB value between the input images to each frame image from a video. However, it was compute-intensive and seems to be not such a good idea. Therefore, we were thinking to use some algorithms to achieve our goals, such as comparing the ‘key-point’ of the images which means only comparing a certain section of the images like a 9x9 pixels on the four corners of the image and a 9x9 pixels on the center of the image. Since one second of video contains 24 frames, we thought it could reduce the compute-intensive problem by only comparing four or five frame images from one second of video. After we tried and learned the Histogram method, it helped us to build the features of histograms for each image, and allowed us to choose the image with the histogram closest to the input image histogram. This method improved the computing runtime, but it contained lacks of the robustness, matching fails on scaled, rotated, or discolored images. It turned out to be that this application could be useless because it was only able to do the matching from the images that were captured screenshot from the video.

After a brief talk with the professor, we finally committed what types of the project we would be working on which was the object detecting in the video. On the way of researching the image recognition or image detection classifier, we realized there were some open source computer visions and machine learning libraries that were designed for computational efficiency. One of the powerful ones we found was the OpenCV which could be used for strong focus on real-time applications. It was providing some built-in training classifier algorithms to help us to train any image detector classifier. The training cascade classifier (for the version of OpenCV we used) built with TBB was already running in parallel. Since it was what we needed for our project, we started to learn about OpenCV immediately. At first, we had some difficult time compiling the code using OpenCV. However, after we figured out the compiling problem, everything went smoothly and we learned how to use different build-in training classifier in OpenCV, and what the different require parameters do. Training time could be varied; it could take from minutes to days, it depends on the training classifier types, the number of training stages, as well as the number of training images. Additionally, in the process of learning the training classifier, we hit the wall for a million times. The longest hour of classier we trained took more than 10 hours, and it still got too many false positives, it could be the problem due to not enough number of positive images. Learning which classifier is the best fit for the particular types of detecting object, the number of stages that are not going to overfit the classifier, resolution, and size of the training image data, the ratio of positive images and negative images, and the number of training images would result in the best accuracy. With the useful built-in tool in OpenCV, splitting a video into each frame images with OpenMPI was straightforward than expected. Even though this functionality could also achieve in the standard audio library in C++.

However, after we built our project using the OpenCV library successfully, we realized that most of the built-in functions in OpenCV has constructed and implemented based on the OpenMP and TBB libraries. From the parallel.cpp that included in the OpenCV library, we found out that OpenCV indeed first uses TBB, then stripes, and then OpenMP. By that time, it was not too late for us to revise our project and start all over again. It was not enough time for us to learn to write our training classifier algorithm and rewrite our code without using the OpenCV. Therefore, we had to find a different way to parallelize our code to improve performance. The second part we learned from our project. We analyzed the coding structure, tired the best to make it less dependent variable as possible, to parallel the whole project. After reading the official documentation of OpenCV, we realized that the OpenCV contains parallel methods, such as the standard parallel\_for or the parallel pipeline in both of the TBB and OpenMP, and it uses the like the same way we learn from the class. Fortunately, we can parallelize in several sections, in both of the image modification and video spitting. In the Table1 and Table2 shown above, we can see the performance speeds up by approximately half of the running time. It does not seem much performance implementation but on slower CPUs and with a more substantial number of images need to process, 50% is a lot.

Another learning from this project was the parallel methods that OpenCV uses to build their library. Since OpenCV is already highly paralleled in many of its algorithms in either TBB, OpenMP, or even both. Due to this fact, we won’t always get much more from adding parallelism to these parts. We spent some time to look into the original code of the particular functions and learn how they implement them. For some of the functions, it would be possible for us to rewrite it on our own and tried to achieve the same functionality, but it seems unnecessary to do. Instead, we realized it would be beneficial to decouple the processing chain from the small IO operations first, then apply the parallel methods to the manipulations and transformation to perform on the source image, as well as the splitting and putting the video frames concurrently. In this case, it performs well on performance. In summary, the things/knowledge we learned are far more than what we expected. We learned not just how the parallel methods work, more profoundly, we learned how it applied in all kind of applications/project to improve their performances. Also, we were impressed on how parallel algorithms are so commonly used in all fields and how powerful it could be, especially in the machine learning area.

**Final Conclusion**

This project provides the opportunities for us to apply what we learn in class to real hands-on working experience. However, we are not entirely satisfied the output product we have made. There are much more implementations we can be done in all areas of this project, includes able to train a model with higher detection accuracy, able to train a model that can detect a more complicated object such as a dog, or a cat, if this project turns out to be a user application, it should be simpler and more convenient for users to use, etc. If we would have the chance to start over again on this project, we might not choose the OpenCV library. Since many built-in functions in OpenCV already paralleled, it may be not the best candidate for this project. We may learn and write our training classifier algorithm instead of using the OpenCV built-in train classifier algorithm. While we looked into the OpenCV’s traincascade algorithm, it contains mostly sequential code, but the grabbing of all features and calculating the corresponding weak classifiers can do in parallel. However, besides the challenges of learning a new library and realize it was already highly parallelized, we did have so much fun while working on this project and learned many valuable parallel methods/knowledge.

**Link to our Project**

<https://github.com/jemin6/CIS431_Image_Detector>

* More detailed informations are listed on the github wiki page. Please feel free to check it. https://github.com/jemin6/CIS431\_Image\_Detector/wiki