Using Digital Image Processing Techniques to Train a Neural Network

Andrew L. Hanshaw, Joel Q. Quanbeck, Genzo Namikawa, Kwame T. Ampofo

***Abstract*—This project aims to train a neural network to play a simple video game using purely visual input. This project will use digital image processing to identify various visual aspects of the game.**

***Index Terms*—Digital image processing, High performance computing, Neural networks.**

I. Introduction

NEURAL networks can be trained to perform many different tasks. In many implementations, this training directly accesses internal variables of the program the neural network is being trained with. Generally, these variables are not able to be monitored directly unless the original source code is available, limiting the use of neural networks to open-source programs. This project aims to train a neural network to perform a task using visual cues only, in real time, by using digital image processing techniques to analyze images and extract useful information. The approach of treating the game as a "black box" means this technique could be expanded to just about any game given the ability to simulate user inputs and the ability to take screenshots and process them sufficiently quickly.

The program the neural network will be trained on is the dinosaur game built into Google Chrome, Fig. 1. The simplicity of this game, both in controls and in visuals, should allow for relatively easy training of the neural network and identification of the visual elements of the game. The SDSU "Roaring Thunder" computing cluster [1] will be used to run the game and train the neural network. Running trials one at a time would take prohibitively long. Parallelizing this task is necessary to quickly create and simulate each generation so the network can evolve as quickly as possible.

This project is not the first to attempt to train a neural network to play the Chrome Dinosaur game, However, these previous attempts recreated the game to directly provide the neural network with variables such as game speed and distance from obstacles. Other projects have slowed the game down and simplified the (already simple) graphics to save on computing time [CITE]. This project aims to extrapolate these variables by visually identifying and tracking the obstacles and background before feeding them into the neural network. This process more closely emulates how a human would play the game.

In Section II the necessary background information used when creating this project is described. Subsections include software overview, project challenges, and high performance computing. Conclusions for this paper are presented in Section III.

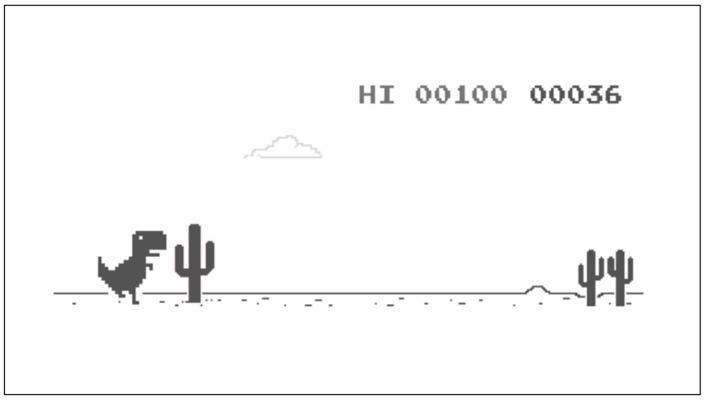


Fig. 1. The Google Chrome dinosaur game.

II. Background Information

A. Software Overview

For this project, Tensorflow was selected to create the neural network. Its ubiquitous use in the field makes for easy support for any issues that arise during the development of the project. OpenCV was selected to run the image processing algorithms on the collected images due to its widespread use and similarity to MATLAB. Finally, Xnest was used to interface with the Chrome browser and produce screenshots of the game at a high rate.

Since the dinosaur game is built into the Chrome browser, which is based on the open-source Chromium browser, the source code for the dinosaur game is available to the public. As such, the sprites used in the game, Fig. 2 could be easily extracted and used in the image processing software algorithm to classify which sprites are on the screen at a given time [2]. The most robust and fastest method of classifying these images is to run a template matching algorithm on them using each sprite as a template. The Roaring Thunder computing cluster uses the Slurm scheduler to schedule jobs for each node, so familiarity with that software is necessary to ensure the program is correctly distributed on the computing nodes. OpenMPI will be used to parallelize the program, as it is compatible with OpenCV and Tensorflow. The project was primarily coded in C or C++ as the method of parallel processing being taught, OpenMP, is only available in those languages and Fortran. Other supplementary languages may

Fig. 2. The spritesheet used in the Chrome dinosaur game

be used, although most lower-level optimizations and features will be written in C or C++.

B. Project Challenges

One immediate challenge discovered when researching this project was running the game on the computing cluster. As the cluster computer is a Unix system, running a fully-functional Chromium browser would be difficult, if not impossible. Finally, as the game runs in real-time at sixty frames per second, analyzing each frame may prove challenging. One potential solution to this problem would be to only process a reduced number of frames per second (thirty or even as low as twenty-four frames per second may be acceptable). With a reduced framerate, the object detection algorithm will have more time to process each frame without altering the original code.

C. High Performance Computing

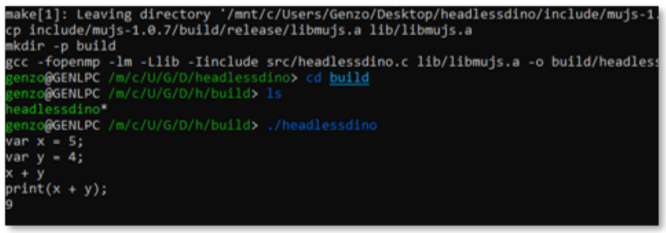
Another important aspect to this project is the parallelization of the finished program. Neural networks are inherently multi-threaded, which allows for them to scale well when provided more resources. However, this scalability does not come without cost. Some overhead work must be done to ensure the program is parallelized efficiently. Familiarity with general program paralellization methods learned in EE592/CS592 (High Performance Computing) is required to ensure the program is properly parallelized. Familiarty with the OpenMP syntax will allow for these parallelization methods to be utilized properly.

III. Current Progress

A. Game Interface

The challenges associated with running the game were attempted in many ways with differing levels of success. The goals in implementation are to reach a comfortable thirty frames per second including the time required for image extraction, image processing time, neural network processing time, and input management. The methods that were attempted each had their respective issues during development. The final, working, implementation of the game interface is also described here.

The first implementation of the game interface was through a lightweight implementation of a JavaScript interpreter, Fig. 3, that was implemented in C. The implementation was successful, and the JavaScript ran as intended, although, there were issues with principle and implementation scope. The issue on principle is the question if the implementation has access to the internal variables associated with the project; we aim to obtain this information through the use of image processing while treating the game as a black box. The other



```
make[1]: Leaving directory '/mnt/c/Users/Genzo/Desktop/headlessdino/include/mujs-1.
cp include/mujs-1.0.7/build/release/libmujs.a lib/libmujs.a
mkdir -p build
gcc -fopenmp -lm -Llib -Iinclude src/headlessdino.c lib/libmujs.a -o build/headless
genzo@GENLPC /m/c/U/G/D/headlessdino> cd build
genzo@GENLPC /m/c/U/G/D/h/build> ls
headlessdino*
genzo@GENLPC /m/c/U/G/D/h/build> ./headlessdino
var x = 5;
var y = 4;
x + y
print(x + y);
9
```

Fig. 3. Implementation of the game interface using MuJs showing a working JavaScript interpreter.

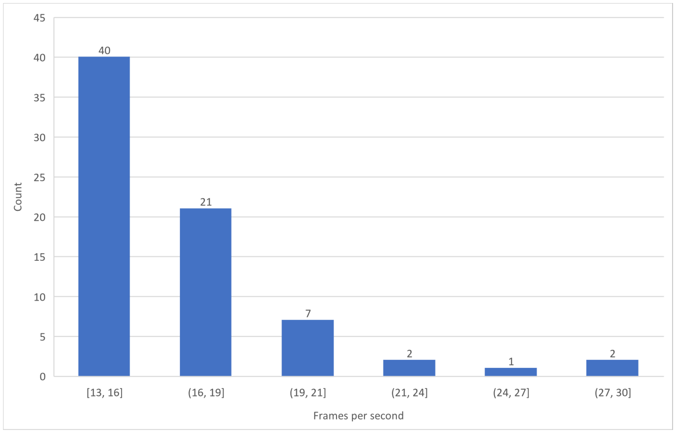
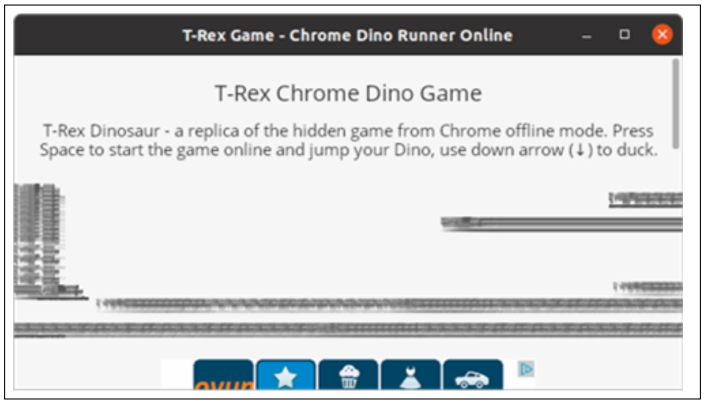


Fig. 4. Distribution of timing discrepancy in C Sharp for the implementation using Selenium.

issue with this implementation is the design scope. Although the use of a barebones JavaScript interpreter is a highly flexible approach to the design of the game interface, it also creates a scope issue as each function evoked by the game will have to be implemented in order to function. These functions included the rendering functions that are associated with browsers that were not implemented in this barebones JavaScript interpreter.

The second implementation of the game interface was through the use of Selenium, a headless browser. This headless browser is a fully featured browser that was intended for automated unit testing of websites. The implementation of this game interface attempt was successful, but with very poor performance. Capturing each image from the game took from 200ms to 500ms to complete, thus rendering the interface useless within reason. On top of this, the timing library used was not precise, and a large variation in screenshot time was measured, Fig. 3. Attempts at offloading this performance hit to different threads were attempted, though it was found that Selenium is not thread-safe, and each thread was blocked by the others. This effectively made the entire multi-threaded system asynchronous and concurrent while still being just as slow as the single-threaded version.

The third implementation was through the use of Ultralight, a lightweight browser meant for the development of JavaScript


T-Rex Game - Chrome Dino Runner Online

T-Rex Chrome Dino Game

T-Rex Dinosaur - a replica of the hidden game from Chrome offline mode. Press Space to start the game online and jump your Dino, use down arrow (↓) to duck.

Fig. 5. Implementation demo of a game interface using Ultralight.

games that supported hardware acceleration and boasted a large performance benefit compared to other chromium-based or electron-based implementations. This implementation appeared to be great, however due to the infantile state of Ultralight, many usefuly functions were not yet implemented. This caused many visual bugs and issues, Fig. 5, related to the rendering of the game which made it unfeasible to use this browser.

The final and working implementation of the game interface was developed on a system running Ubuntu; Xnest has been used to launch the game inside an X server, Fig. 6, of the ideal dimensions to run the game (150 x 600 pixels). Google Chrome is launched within the X server and can be manipulated using the C library Xlib. Keypresses are emulated using Xlib to navigate to chrome://dino where the game can be found in Google Chome. F11 is pressed (virtually) to enter full-screen mode. Up and down arrow keypresses are used to control the game. Each pixel can be inspected and stored in a byte array, avoiding unnecessary file reading/writing. The byte array can then be passed to the digital image processing functions.

Further steps are required to run the system on the cluster. The SDSU HPC cluster support team were contacted to get the Xnest software installed on the Roaring Thunder server. Xnest is a window manager that makes it possible to run multiple programs in the X system window environment. Xnest is a standard part of the X server that can be used to control the display server and simulate keyboard input as well. Mr. Chad Julius from the support team was very instrumental in getting Xnest installed. Upon several deliberations and going through the workflow, the conclusion was that Xnest will be better for our implementation comparing it to the VNC server solution which is an alternative that the SDSTATE cluster services provide. This is because Xnest is a better solution for Linux to Linux connectivity whiles VNC is generally a good solution for cross platform connectivity [3]. The figure below describes a sample script in Xnest which can be used to change the dimensions of the image

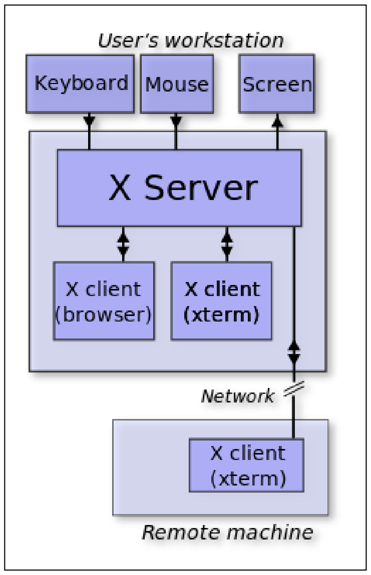
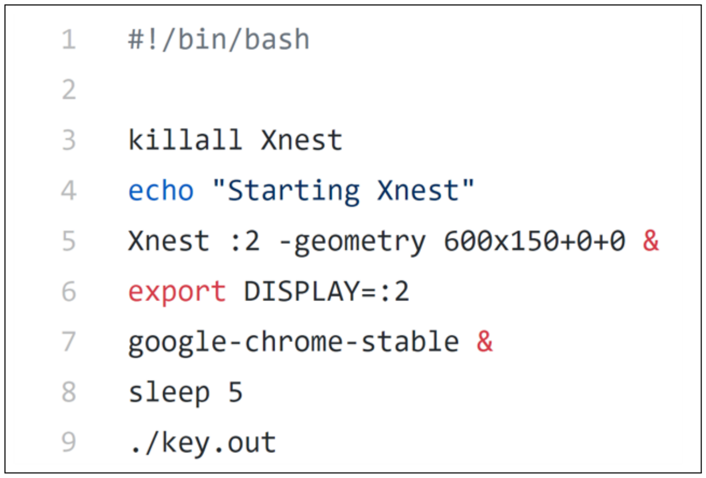


Fig. 6. X Window Framework Illustration.

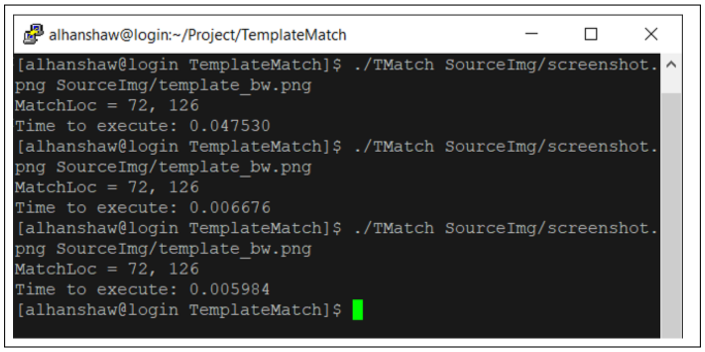

```
#!/bin/bash

killall Xnest
echo "Starting Xnest"
Xnest :2 -geometry 600x150+0+0 &
export DISPLAY=:2
google-chrome-stable &
sleep 5
./key.out
```

Fig. 7. Script for Changing the Dimensions of an Image in Xnest.

B. *Image Processing*

An initial computer vision algorithm was implemented using MATLAB to explore the state-of-the-art of image processing. Familiarity with MATLAB allowed for a better understanding of the internals of the template matching algorithm. A simple single-object template matching algorithm was implemented, and the results were displayed visually for easy confirmation of results. Using MATLAB allowed for rapid prototyping, though the scripts created did not run quickly enough to be used in practice. After this process was complete, the program was translated to C++ and the OpenCV library. Results show that the speed at which the template matching algorithm were well within the margin required to run the game at real time, Fig. 8, though more complex tests will be done to ensure the speed is sufficient. Thresholding will be implemented to locate multiple objects within an image while also allowing for a state in the case that no matching template is found.



```
alhanshaw@login:~/Project/TemplateMatch
[alhanshaw@login TemplateMatch]$ ./TMatch SourceImg/screenshot.
png SourceImg/template_bw.png
MatchLoc = 72, 126
Time to execute: 0.047530
[alhanshaw@login TemplateMatch]$ ./TMatch SourceImg/screenshot.
png SourceImg/template_bw.png
MatchLoc = 72, 126
Time to execute: 0.006676
[alhanshaw@login TemplateMatch]$ ./TMatch SourceImg/screenshot.
png SourceImg/template_bw.png
MatchLoc = 72, 126
Time to execute: 0.005984
[alhanshaw@login TemplateMatch]$
```

Fig. 8. Preliminary OpenCV Template Matching Results

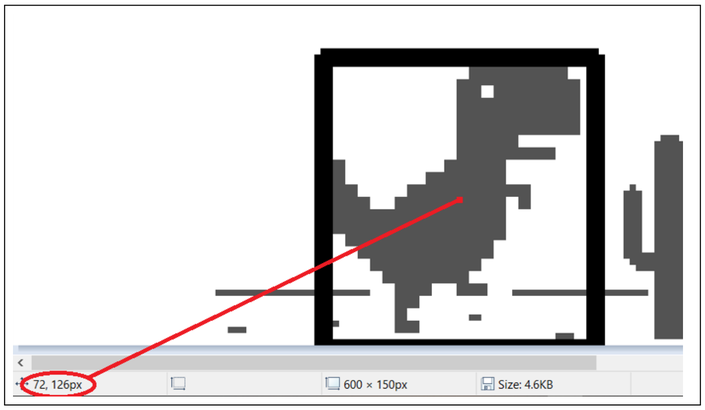


Fig. 9. Preliminary OpenCV Template Matching Results

The resulting algorithm outputs the location of the center pixel of where the template was found, Fig. 9, though this ordered pair can be shifted by half of the width or height of the template image to produce the leftmost edge or corner of the detected object. The locations of each object may need to be preprocessed by converting them into distance from the player rather than explicit image locations, though the neural network will most likely be able to make these calculations on it's own through training.

IV. Conclusions

Overall, performing this project has and continues to expose those working on it to many new software libraries and programming methods, both in terms of creating the program to play the dinosaur game and using it train a neural network, but also in parallelizing the program to run on the computing cluster. The challenges faced when interfacing with the different software proved to be an exercise in becoming familiar with reference material of each software. The expected outcome of the project will create an entertaining and impressive show of AI mastery over simple video games thanks to their easily scalable nature.

Acknowledgments

The authors would like to thank Dr. Tim Hansen and Dr. Songxin Tan of South Dakota State University's electrical engineering department for their assistance in the high performance computing and digital image processing fields through their undergraduate courses. The authors would also like to thank Code Bullet on YouTube, who has provided the group with meaningful and entertaining content on AI/neural network training [4].

References

[1] SDSU Information Technology. Research Computing. Accessed: Sep. 4, 2020. [Online]. Available: https://www.sdstate.edu/information-technology/research-computing
[2] wayou. T-Rex Runner Github. Accessed: Sep. 4, 2020. [Online]. Available: https://github.com/wayou/t-rex-runner
[3] TechnologyAdvice. Using Xnest. Accessed: Oct. 18, 2020. [Online]. Available: https://www.linuxtoday.com/blog/using-xnest.html
[4] Code Bullet. AI learns to play Google Chrome Dinosaur Game. Accessed: Sep. 4, 2020. [Online]. Available: https://www.youtube.com