Computer vision in parsing AI digestible data from Human-centric UIs

Tobias Dean
Earlham College
Richmond, Indiana
tadean21@earlham.edu

ABSTRACT

There is no denying that AI, Artificial intelligence, is a versatile technology with countless applications, but this versatility is far from its full potential. One of the greatest limitations to modern AI is its need for carefully formatted data, both to train and make decisions.

This document covers my research and work on developing an algorithm to parse data from the Tetris UI, user interface, into AI digestible data. This project aimed to give insight into techniques for automatically extracting data from human digestible UIs into one consistent format for AI use.

The python algorithm produced, Packing Coordinator, uses a combination of ImageMagick, scikit-image, and Keras to produce an abstract representation of the current game state. This algorithm uses images of the game UI as input allowing UIs built with only humans in mind to be read by AI. Although further work and testing is needed for stronger conclusions, initial results seem promising as the algorithm successfuly captured the game state from screen shots.

KEYWORDS

Machine Learning, AI, Tetris, Convolutional Neural Networks, General Intelligence, Computer Vision

1 INTRODUCTION

One of the long-standing goals of research on Machine Learning and Artificial Intelligence, in general, is to develop AI that displays general intelligence, which allows for the AI to fulfill multiple roles [2]. Ideally, such an AI would also be versatile enough to complete tasks using pre-existing interfaces designed for human use. This would be an obvious advantage as the developers looking to use an AI with general intelligence would not always have the luxury of changing the user interface or AI to allow the latter to interpret the former easily [2].

Therefore, designing an AI that can gather information directly from the computer's screen would greatly increase the versatility of that AI. With the ability to gather information more independently and adapt to small changes the AI would be one step closer to operating at a similar level to a human.

Naturally such a large step can not be taken in one proverbial stride, so it must be broken into smaller steps. The first step I have chosen to take with my research is to develop a program to convert one type of UI into one universal abstract representation. Fortunately, Tetris, and video games as a whole, have long served as a testing ground for machine learning and AI systems, and there is no shortage of foundational research on the topic.

For that reason this research project focuses on developing an algorithm to create an abstract representation of the current game state of any Tetris implementation. Using a convolutional neural network and a verity of open source tools this algorithm could gather all the relevant data from a screenshot of the game. This data would then be formatted into an abstract representation of the game more easily understood by an AI. This would, assuming no deviations from stranded game rules, allow an AI built for playing Tetris to operate across different implementations of Tetris with out the otherwise necessary changes to its code.

A way to parse data about Tetris from the screen into a form that an AI can use will, in turn, give insight into how to use similar methods to parse AI usable data for other tasks from any screen. This research project has the express goal of answering the question of whether using neural Networks and machine learning would be an effective method of parsing the pixels from a screen into data an AI can effectively work with

2 BACKGROUND AND RELATED WORK

2.1 Why Tetris?

There are a number of reasons Tetris stands out as an appealing challenge for this avenue of research. The first and most simple reason is that Tetris is an old, well-known, and simple game. For this reason, Tetris has a long history of use in testing machine learning. The article *The game of Tetris in machine learning* [1] by Algorta and Şimşek details how Tetris has been used as a testing ground for machine learning AI as early as 1996. This same article also covers how much success different approaches have had over time. The earliest attempt discussed simply used large-scale feature-based dynamic programming and tracked the number of holes and the height of the highest column. Later attempts would go on to include hand-crafted agents, genetic algorithms, and approximate modified policy iteration [1].

Furthermore, the article details how, over time, certain practices, such as using the scoring system of Tetris's original implementation over its more complicated successors, have become the standard for testing AI within Tetris. It is important to give some thought to which implementation of Tetris is used in testing. There is a myriad of small but significant changes possible between even standard implementations. For instance, some versions award additional points for clearing several rows in one move. More significantly, implementations vary on whether or not certain moves are allowed. Sliding a Tetromino sideways into a gap under a placed Tetromino is one such inconsistently valid move.[1].

Differences between implementations of the game must be considered when comparing AI performances, as having one AI having

access to more possible moves or getting more points for completing the same number of rows can make it look unfairly superior to an AI tested in an implementation featuring neither of those features.

Naturally, such a long-used benchmark has a wellspring of resources and foundational research tied to it, such as *The game of Tetris in machine learning* [1] and *Using dual eye-tracking to unveil coordination and expertise in collaborative Tetris* [6], making it all the more appealing to use. Tetris is, however, not the only long-standing benchmark, and its veterancy as a testing ground is not the end of its merits. Tetris also offers a simple yet sufficiently complex challenge that features a random element that will prevent the AI from simply learning an optimal series of keys to press without being able to adapt this learned method to a task where even one detail is different from the training set.

For those reasons and a moderate personal familiarity with the game, Tetris was settled upon as the testing environment for this research project.

2.2 How humans play Tetris

Fundamentally, a human and an AI see and understand images very differently, but the two processes share a lot of proverbial DNA. Understanding how a human visually parses data from the Tetris UI sheds light on how an algorithm could do the same. *Using dual eye-tracking to unveil coordination and expertise in collaborative Tetris*[6], by Jermann et al., provided a break down of player eye movement and UI components that received the most attention. The player's unplaced tetromino and the contour created by the stack of placed tetrominoes were the most tracked features.

Beyond knowing what features to track, simply knowing how to proficiently play Tetris gives its own insights. Jonas Neubauer, a seven time championship winner, shared his strategies in an interview.[8] More importantly than the advice its self, these winning strategies can tell us what information is factored into deciding an optimal move.

With this knowledge of how a human views the UI, and what information is prioritized, we can begin to form our abstract UI design. From this insight into the most critical information for a optimal move, we can reduce the UI to the most important data. The current Tetromino, the dimensions of the board, and the location of placed all placed tetrominoes. Thus, using the same UI and strategies as a human, an AI could quickly determine and execute its next move.

2.3 AI implementations

The development stage of this project focused entirely on the Packager Algorithm over the hypothetical AI it would work with. However, hypothetical or not the Packager Algorithm's design is inescapably shaped by the AI it works in tandem with. As previously mentioned, Tetris is a time honored benchmark, and there are already many well-documented AI developed for the game. Thus, reviewing the existing AI agents created for the task is a natural step in the research stage.

Keeping the goal of versatility in mind, it is worth studying the AI entered into the general game-playing competition featured in *General video game playing* [2]. The AIs built for that competition were centered around versatility and working with no prior experience of the task.

One of the strategies used by previous attempts was the implementation of evolutionary algorithms [4]. Looking at their report on this study, we can see several interesting choices the authors made, which may inform our own choices. This implementation chooses the best move by assessing how desirable potential subsequent boards are based on several sub-ratings, the weights of which are determined by the genetic algorithm. Additionally, this implementation chooses to use a heuristic approach when rating boards in order to act more quickly. A valid concern as tetrominoes continue to fall from the moment they enter the board. The board quality is assessed based on 12 metrics, but the authors also acknowledge that using more complicated calculations to rate how desirable a board is can have advantages and disadvantages. Using more criteria to judge a board led to better performance but made that AI more niche for the exact environment it was trained in [4].

Although this AI is fundamentally different from the hypothetical AI this project would create, there are still lessons to learn from this article. Much like with human players, Knowing what data this AI needs informs the what data is captured by the abstract representation.

Another team's work that is rather interesting and, more importantly, relevant to this project is documented in the paper *Tetris Artificial Intelligence* [10]. This team built a Tetris AI that, like last example, saves time by choosing not to calculate every possible game state a piece can create. What is more of note is their decision to make three kinds of AI, each using different strategies. The "Greedy AI" uses the most intuitive strategy, prioritizing clearing rows over all else.

A bit more sophisticated was the "Tetris AI", which focused on setting up the board for simultaneous 4-row clear, also known as scoring a Tetris. It did this by waiting until it has an I piece and four-row hole to fill. Although both of these AI cleared about the same number of rows, the "Tetris AI" achieved higher scores on average. This increased success came down to the extra points awarded for clearing clearing multiple rows in one move.

The last and most successful AI created by this team was the "Two Wide Combo AI", which specialized in setting up combos over any one scoring move. This proved to be the most effective of the three human strategies adapted for use by the teams AI. It also proved how important data on the incoming tetrominoes is to an AI's decision making process. Naturally, this means the data of the next n tetrominoes provided by most implementation's preview window is a high priority for the abstract.

2.4 Suitable working environments

An important aspect of this project was choosing a suitable environment to work in, both for testing and code development. During research far more focus was given to the former of those two due to the comparative lack of difference between available text editors. Of course, that doesn't mean that the latter was a complete afterthought.

It was decided very early into the development stage that the code writing and testing will be conducted in two text editors. The

bulk of the work with the images and their parsing into game state data was done in Jupiter notebook. This was due to its greater support for visualizations and quick testing of specific cells of code. Once code was finished in Jupiter is was then exported as normal Python code to Atom where the rest of the code was being written.

Despite GitHub announcing the sunsetting of Atom in 2022 it has remained a reliable text editor for code development and remote editing. Due to a personal familiarity and this continuing reliability, the work less reliant on visualizations was conducted with Atom. This also allowed me to remotely access and work on the collage's server for student projects. Many of the libraries featured in this server were essential to the project.

What seemed more important to give careful consideration to was the specific details of the environment in which testing was conducted. Although the base game of Tetris seemed an obvious choice, using a standard game for testing raised some concerns. When covering aspects of testing in Tetris that have become standard, *The game of Tetris in machine learning* [1] points out how the need for faster testing led to many researchers using non-standard implementations. This allows for changes like reducing the game size to 10 by 10 from 20 by 10. A change that shortens the game length and, in turn, speeds up AI training.

Although my concern was not with training an AI to play Tetris, an implementation of Tetris that could be modified on the fly still had it's advantages. Most obviously this would facilitate any future work with AI for the same reasons it helped earlier researchers, but the advantages extend beyond that. The option to quickly make changes to implementation's UI would open up more options for testing the versatility of the Packaging Coordinator algorithm's functions.

Conveniently, the general game-playing competition has led to the development of an especially qualified environment. The companion article to *General video game playing* [2], *Towards a Video Game Description Language* [3] goes into depth about the need for a programming language designed to be easy to interpret for humans and AI alike. VGDL was written with features that make games implemented in it conducive to quick modifications and easy interactions with AI made for general game playing. For these reasons, an implementation of Tetris written in VGDL was a goal of early development.

There are a number of articles written subsequent to *Towards a Video Game Description Language* [3] that describe the author's implementations of VGDL. One such article, *XML-Based Video Game Description Language* [7], written by Quiñones and Fernádez-Leiva, describes a more recent implementation of VGDL, known as XVGDL, implemented using XML that the authors published to github. Naturally, the article also explains how to use this implementation and compares it to other versions of VGDL.

Just as naturally, XVGDL is not the only available implementation of VGDL, and another article *A Video Game Description Language for Model-based or Interactive Learning* [9] details an implementation of VGDL in PyGame. This other implementation, PyVGDL, shows how an implementation of VGDL tailored to more specific needs is far from hard to find.

The complication to all of this was the time investment to find, or write, a implementation of Tetris using VGDL. Although the advantages are obvious, the core of this project centers around parsing standard implementations made for human use. As a result a VGDL implementation remained a useful, but not essential, tool for development and testing.

3 DESIGN AND IMPLEMENTATION

The scope of this project ended up changing quite a bit over the corse of development. In large part this was a side effect of the gradual process of bringing the proposed project into reality and realizing more and more how ambitious the the research proposal truly was. However, by the end of the process the design had successfully evolved from a nebulous idea to a actionable plan.

The final design settled on near the end of development uses three methods to gather Data from the game UI. The files utilizing these three methods are called by one organizational program, the Packaging Coordinator. This program ensures that data is passed between supporting programs, Packaging Coordinator and its supporting programs will be expanded upon in the next section.

3.1 The Packaging Coordinator algorithm

The Packaging Coordinator as mentioned previously serve to call upon and pass data between the algorithms responsible for the three methods of data extraction.

After sending and receiving the necessary data from each of these programs, Packaging Coordinator combines that information into an abstract representation of the game state. This abstract representation is stored as an object with the gathered data as attributes. It also features a matrix the size of the game space with functions to place new Tetrominoes and automatically clear rows.

After creating the abstract representation, Packaging Coordinator will then continue to update its data each time a new Tetrominoes are placed. In addition to placing the new Tetromino on the board Packaging Coordinator also adds the predictions for each new Tetromino in next.

3.2 Tetromino identification

Perhaps the most important of the supporting algorithms called by Packaging Coordinator is Identification. Identification is built around implementing a Convolutional Neural Network with Keras used to identify the class of a Tetromino image.

The file it's self has two behaviors depending on if it is called directly or merely imported and used for predictions. Calling it directly trains the network for 15 epochs, and then trains the best performing model for a further 15 epochs. The idea behind this is that spending half the training time building upon the best results will lead to a better performance then simply training the model for 30 consecutive epochs.

After the model has trained and saved the best results, as measured by validation accuracy, identify can then be called with *identify.getPred()*. This takes a images file path as an argument and returns the predicted class using the saved model. First this is called on every Tetromino on the board, but after initial abstract is formed it is only called upon the Tetromino that enters next.

This method gives any AI using the abstract the knowledge of the next n Tetrominoes needed to form any strategy beyond stabs in the dark. The further ahead the AI can previews Tetrominoes, the more effectively it can calculate the optimal sequence of moves.

This information is saved in the abstract as an array of the letters corresponding to incoming Tetrominoes class.

3.3 Grid scan

The second near equally critical algorithm used is Grid Scan, which returns the dimensional data needed to create the abstract and locate key UI features. This data includes the number of rows and columns in the game board and the location of the board and next window.

This also find the size of a single grid space, from which the expected location of incoming tetrominoes can be calculated relative to the game board. This same information is also used to divide the next window in the event that it previews more then one tetromino.

The initial scan of the game board for its dimensions and current state is done using the tools from the ImageMagick library. The main work for this algorithm is using canny edge detection in determining where grid spaces begin and other features begin and end. Once edges are isolated they are then measured with the scikit-image library to determine which edges belong to the desired UI features. The edge measuring features in scikit-image are also used to get a bounding box for these features. This bounding box is used to save UI feature locations on in the abstract representation, essential information for updating the abstract as the data presented by the UI changes during gameplay.

3.4 Text extraction

Lastly, text sweep is called on the entire uncropped image. This algorithm uses *pytesseract* to extract a string of all the text in the UI. This text is then searched for keywords that correspond to desired information. The cleaned data is then saved with this keyword to the abstract representation's dictionary for miscellaneous data.

This is primarily done with the goal of giving the AI all the information a human player would have from their view of the game. This infrmation may be more relevant to any AIs decision making process then one would initially expect as the score is not the only text information in the UI. Many implementations of Tetris display the game's current difficulty level, a modifier for how fast the tetrominoes fall, as text in the UI.

3.5 Purpose made AI

During the initial research and project proposal I had the ambition to create an AI for the specific purpose of testing the abstract representation. The idea being that an AI built with that input in mind would be far better for testing then adapting an existing AI. It quickly became apparent that these advantages could not justify the labor required, and the AI was made a secondary goal. This then became a project extension for future work, but the planing put into warrants its own subsection.

The purpose made AI, referred to by the working name of Process, would be used to test how effectively the abstract representation captures all important Data. In the most simple testing configuration planed, Packaging Coordinator would pass each abstract representation produced to the AI. Many things would be measured, but the main focus would be on the AI's ability to quickly make and execute optimal moves from the data given.

Similar tests could be done with a modified existing AI, but the difficulties of working with unfamiliar code may outweigh the potential time saved. Furthermore, any extensive modifications to this unfamiliar code to conduct other test would likely be difficult.

Despite all of this, a test AI for the project, be it new or modified, was always a lower priority. Working around an entirely hypothetical AI brought its own difficulties, but the crux of research question remained the priority. The actual transformation of the pixels on the screen into an abstract representation of the game occurs entirely within the Packaging Coordinator algorithm. Thus, any plans for AI testing remains a high priority objective in the possible future work for this project.

3.6 Language to be used

The programming langue Python was used for all of the original code written for this project. This was due to my existing familiarity with the language and the abundance of open source python libraries. These libraries were invaluable, often featuring useful functions that would have taken extensive time to write from scratch.

The idea of using a more specialized languages was considered, but ultimately dismissed. The idea behind this consideration was that the benefits of using a programming langue well suited for each task could outweigh cost in time and effort. However, As research transitioned into actual developments it became clear that this would not be necessary or practical. The popularity of the Python programming language had led to a abundance of open source Python libraries. Seeing as I was not limited by using only Python, learning another language for the project did not seem like a reasonable use of the limited time available.

During research, one idea considered was the potential benefits of creating a simple Tetris implementation that could easily be modified for testing. Although such a implementation was ruled out for the time being, as mentioned in section 2.4, it remains a possibility for future work. Of the two video game description languages researched, PyVGDL is the most likely candidate over XVGDL. Like before this choice came down to a greater personal familiarity with Python than the alternative, in this case XML.

Looking at the comparison of the two considered languages provided by *XML-Based Video Game Description Language* [7], I feel confident this is the right choice, There are some features supported by XVGDL that PyVGDL lacks, but none of theme are useful enough to justify the greater time commitment.

3.7 Data sets to be used

All of the data sets used in this project will unfortunately have to be created on my own do to the specific nature of the problem. I did briefly check several repository's of data sets for Neural Networks but none had anything remotely close to what was required. As a result a large part of the early work for this project was dedicated to building several small data sets from images available on the internet and screenshots of Tetris games.

The Tetromino Identification Network was trained on a data set made of images of Tetrominoes, size 100 by 100, classified as either I, J, L, O, S, T, or Z. This data set started with 10 images in each class and periodically grew as the project continued. This data set

was then further expanded with data augmentation by rotating each image to all possible in game orientations. Although Keras offered similar tools for data augmentation the rotation function was insufficient for this task. The need to ensure images were only rotated to the orientations possible for their class necessitated doing this with a batch file of ImageMagick commands.

The inputs in the software used by Grid Scan was set based on observed results across images of the game boards in various stages of gameplay taken from a verity of Tetris implementations.

The Text Extraction Network selected will likely be trained in text recognition already but if a data set would be needed then many Data sets of text fonts are already publicly available. If for whatever reason a data set of text elements from Tetris implementation UIs is needed then one can be made from the same images used to train the Grid Scan network.

3.8 Game implementation to be used

Given the objective of the project, standard implementations of Tetris for ordinary human players were chosen over any research oriented options.

Due its personal familiarity and ease of accesses, the free implementation on Tetris.com was used for the majority of development. After success with this implementation other implementations were used to test versatility.

Another implementation to be considered is the free and open-source implementation of Tetris written by Kevin Chabowski in Python and uploaded to GitHub. This open source implementation would be the implementation used for testing the Process algorithm should that project extension be reached. An open source implementation will allow for a more in-depth look into the specifics of this implementation and for any necessary modifications to the code.

These two will not be the only implementations used in the project but in the interest of time the implementations used can be boiled down to being publicly available for free. Being freely available is a rather strong requirement due to a lack of budget outside of my own funding. If some implementation of Tetris this is not free to play is somehow uniquely suited for this project then an exception may be made but this seems highly unlikely.

3.9 Data parsing

The work done in the Packaging Coordinator algorithm will turn the pixels of the game window, passed to the algorithm as a screenshot cropped to the game window, will be parsed into an abstract representation of the game state. This would necessitate determining where the borders of certain game components begin and end, which is made easier by game design elements like most implementations, making all tetrominoes one solid colour unique to the particular tetromino type. Similarly, text boxes containing information like the score or the current difficulty level have straight or mostly straight lines acting as a clear stationary border to where that information starts and stops.

3.10 Convolutional neural network

From the research I have done, convolutional neural networks have stood out as the best method to parse the screen into an abstract representation of the game state. Although neural networks, in general, are good at pattern recognition, a convolutional neural network would be especially suited for working with images in the way we intend to.

A convolutional neural network primarily differs from a simple Artificial Neural Network in that its architecture is specifically built for working with features specific to images allowing it to be much more optimised for the task. This optimization is important as neural networks are inherently rather resource-intensive, and without it, the risk of overfitting the network goes up considerably.

The general gist of convolutional neural network architecture is that the input layer holds the pixel value for the image and feeds into a stack of convolutional layers. The titular convolutional layer takes a group of pixels and converts them into a single value, effectively reducing the size of the image from a computational standpoint. Next is a stack of pooling layers which serve to reduce the dimensionality of the representation cutting down the needed computational power even more. This reduced representation of the image is then passed to the stacks of fully connected layers which work in way as a normal Artificial Neural Network. [5]

Although the simplified architecture I just covered can be considered a complete convolutional neural network, it is more common and advisable to use more stacks of layers than just that for better results. For instance, it is common practice to have two convolutional layers before each pooling layer as well as splitting large convolutional layers up into many smaller sized convolutional layers. [5]

With this particular neural network architecture being so suited for efficiently working with the massive data set that images create, I am confident that a convolutional neural network will yield desirable results. Even if this research project finds that convolutional neural networks are not adequate for parsing data on the screen, I suspect that it is at least a step in the right direction, and a similar method or architecture could be derived from the lessons learned.

3.11 Testing parameters

During testing a collection of game screenshots paired with their corresponding moves were used to simulate a running game. These screenshots captured the moment a new tetromino entered the board and clears the first row, ensuring empty space on all sides.

In the interest of keeping the UI identical to the UI a human user would be interacting with, I chose to forgo the common strategy of decreasing game length by reducing the size of the game board. However, future work may utilize this in testing for adaptability. This would be done by changing the dimensions of the game board to see how well the agents can adapt to a new but similar UI.

For the most part, success in testing was measured by the accuracy of the abstract representation compared to the actual state of the board. Ideally, several different implementations of the game would be used in validation post-testing to ensure the model does not suffer from overfitting.

In regards to the potential extension to the project of testing its digestibility with a AI, I have the following rough plan. Due to having to dismiss one method of reducing training time, it is worth looking into the possibility of changing the game implementation to run much faster so the AI system can train at higher speeds.

Furthermore, once I have the data to back up such a decision, I intend to set an artificial cut-off point for training to stop training beyond the point of significant diminishing returns. Should this hypothetical AI becomes more then speculative, this plan will be fleshed out beyond this.

The real concern for testing is the training for the Convolutional Neural Networks used in Identify. The largest concern here was finding a sufficient data set to train with, but this one could only be solved with more time to gather and label data. Something far easier to account for is the concern of carefully setting the parameters and structure. Here the plan is to make small changes one at a time and compare the plotted results to the accuracy and loss before the change. This strategy allowed for a slow convergence on the optimal structure and parameters with a clear indication of any progress being made.

3.12 Software architecture

To elaborate, the abstract representation of the game state is planed to be represented as a Matrix of board spaces which are filled or unfilled along with the next n Tetraminoes. Ideally, other data, such as the score and level, will be passed along as strings.

4 ANALYSIS PLAN

During the development of the project all goals, obstacles encountered, and successes or failures will be recorded in a document for work-in-progress notes. These notes will allow for a more detailed understanding of the over all process and what decisions helped or hurt development. These notes will primarily be used for writing the poster at the end of the project. All notes, Work-in-progress or otherwise, will have the date they were taken recorded in the note.

Once the Networks are being trained then Data recording becomes more meticulous. Any time a change is made to the layers of a Neural Network the specific code for its layers will be copied with a note of what changed in this iteration and why. Also recorded with these changes are the accuracy, loss, validation accuracy, validation loss, and number of epochs in training. This data may be recorded in graphs of the previous statistics over each epoch.

Similarly, a separate document of progress on image manipulation and which parameters and commands yielded the best results will also be kept. The most promising images from this process will be saved for latter reference. manipulated images will be saved with a name derived from the base file, command used, and parameters used. This will allow for ease of remembering which images were created with which effects. the explanation of this naming convention will also be documented in the image manipulation notes.

Any errors encountered with a cause more substantial than a spelling or syntax error will also have their Code and error message recorded in the general work notes for that day. Once a significant error is solved the solution will be recorded in that days work notes.

Once training and adjustments are no longer yielding significant improvements in loss and accuracy it will be time to look through the data and draw conclusions. The main goal is to find out which Network Layouts work best for each task and why. The effects changes have can be demonstrated by the graphs recorded turning development.

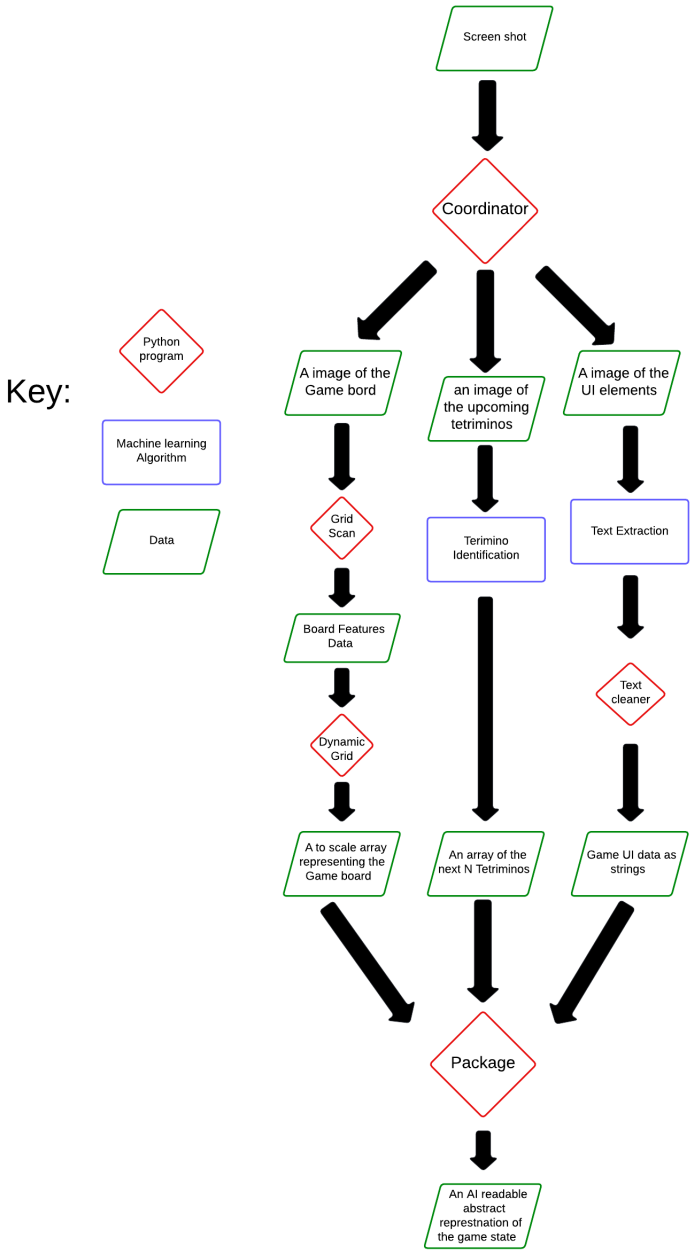

Figure 1: The software architecture for the Package Algorithm (this will be replaced with the final version of the diagram in the final version of the paper)

5 RESULTS

When given a set of test game frames to simulate gameplay, the program is able to create and update a abstract representation of the game state. The game grid it's self meets the expectations excellently, it is able to match the dimensions of the game and added new tetrominoes as it is given placements. Similarly all the critical data is successfully stored and passed around with the abstract

object which even has a field for unexpected data that needs to be saved.

Furthermore the algorithm is able to locate and save the location of key UI features. Most important among these are the bounding boxes for the incoming tetromino and the window for the next tetromino. However, there is a significant issue of inconsistent accuracy from the Convolutional Neural Network used in Identify.

Although it achieved a modest but commendable accuracy percentage in the mid 70s during training, the accuracy is significantly worse in practice. This comes down to a oversight in the difference between the white background of the training dataset and the varied backgrounds found in games. This black background is addressed using the tools in ImageMagick, but this too is an imperfect solution. The resulting distortion, although insignificant to the human eye, is enough to greatly through off the model's accuracy.

5.1 Complications encountered

Right away one of the issues I encountered was the lack a data sets of Tetris game components due to the niche nature of the problem. Due to the limited number of Tetris game elements this was solved easily enough by makeing my own small data set from images found on the web. This did create the new issue of figuring out how to label the data set in a way the neural network could easily interpret.

Another complication encountered was that the simple designs of the 4 block geometric shapes used in Tetris made the neural network used extremely vulnerable to over fitting. A lot of the early work on the Identify Convolutional Neural Network was repeatedly scaling back the number of layers and parameters to account for the shallowness of the problem.

A small complication that came up was the inability to added libraries needed to the school's testing environment. This was resolved easily enough by waiting for assistance from faculty, but did add further delays.

A common but less direct complication to the work came not from the project its self, but the difficulties of my schedule. I had made a significant error and scheduled my classes as though it was an ordinary semester. As a result I found myself balancing large projects from my upper level Classes with the more important but less immediate deadlines of my capstone project. Given the chance to redo the semester, I would have taken a far less busy scheduled to allow more time for this project.

Perhaps the largest complication was simply underestimating the scope of the project and the labor required to achieve it. I often made the mistake of assuming that the existence of a library for a function I needed would cut all the time out of that aspect of the project. What I failed to account for was the time needed to learn these libraries through a combination of reading documentation and trial and error.

was the underestimation of how familiar I would need to be with the tools used in the project. Although open source tools like ImageMagick and Keras were essential pillars of this research project learning to use them effectively was quite the time sink.

If I could draw any through line across the majority of complications, it would be a lack of significant experience. Going into this project I was aware of how many of the goals were generally possible, but had yet to work with many of the methods needed. After research was concluded I knew that my goals were achievable and that the tools needed were available, but not lofty my goals truly were. To much planning was dedicated to the process of refining and improving the initial Packaging Coordinator through testing with AI or custom Tetris implementations.

6 FUTURE WORK

Although testing with a human agent instead of an AI for the decision making process was expedient it leaves room for improvement. Continuing this work I would like to test how AI digestible the abstract representation produced is with a AI disiged to chose the next move based on the game abstract. This same AI must also be capable of then relaying it's next move to the game and updating the abstract. Ideally this AI would also demonstrate greater versatility then others due to its ability to play based on an abstract which can be formed from a verity of different Tetris implementations.

Before any of that can be considered the unreliable Identification accuracy must be addressed; I have two ideas in mind to do this. The first is to find a means of cropping an image to the exact boarders of a tetromino, instead of a bounding box. This would remove the need to clean the background cutting down on the distortion in images given the the Convolutional Neural Network. More broadly a larger data set of tetrominoes would help the model become far more reliable and versatile. Unfortunately this will be a much more labor intensive process, but I believe it to be a necessity for any future work.

Naturally cut features would also be up for consideration as there removal was more owed to time then their irrelevance to the research question.

7 CONCLUSION

Having finally reached the end of this project I would call it a success. The abstract representation may not have been as versatile or thoroughly developed as I envisioned, but the proof of concept is there.

I regret that I created such a niche program when the goal was one that worked across many implementations, still the first step was taken. The initial results are promising and the largest issues have a solution in sight. I am confident that with further development the versatility of the method would only grow. The addition of cut features like text extraction and the tuning of parameters to work on more UI implementations would be a start to this process. However, further down the road I believe some of these methods can be applied to other UIs with some modification to parameters.

Beyond what this project contributed to the path towards Artificial General Intelligence, I myself learned many things. Perhaps chief among them was the true length of the road from the theoretical to the empirical. Along the way I made far slower progress then anticipated, but learned far more about each of these steps along the way.

From Keras I learned far more about data organization and Neural Network then I even thought I would need at the outset of this project. Similarly ImageMagick and Skim-Image both highlighted the merits of using existing libraries, and the difficulties of making

them work together. More then that I think I've learned a valuable lesson on the value of extensive research/planning and a good work flow. I may have made mistakes with the former, but I believe my growing understanding of the latter saved this project and my overcrowded semester. I hope I can take these lessons with me when I leave Earlham, at the vary least I know the last one will serve me no matter where I go.

ACKNOWLEDGMENTS

To David, for teaching me how to write a Project Proposal and helping to refine my work in CS388.

To Charlie Peck, for his extensive help and patience in CS488.

To Debanjali Banerjee, for helping me wrap my head around Convolutional Neural Network and machine learning in CS382

And to Doug Harms for his help with any questions I had during this project, regardless of their relevance to my classes with him.

REFERENCES

[1] Simon Algorta and Özgür Şimşek. 2019. The Game of Tetris in Machine Learning. (May 2019), 1–7. https://arxiv.org/abs/1905.01652 arXiv:1905.01652v2 [cs.LG].

[2] Marc Ebner-Graham Kendall Simon M. Lucas Risto Miikkulainen Tom Schaul Tommy Thompson Michael Mateas Mike Preuss Pieter Spronck John Levine, Clare Bates Congdon and Julian Togelius. 2013. General Video Game Playing. 6 (2013), 1–7. https://strathprints.strath.ac.uk/57217/ General video game playing. Artificial and Computational Intelligence in Games. Dagstuhl Follow-Ups, Volume 6, ISBN 978-3-939897-62-0.„; pp. 77–83 Dagstuhl Publishing Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Germany.

[3] Simon M. Lucas-Tom Schaul Tommy Thompson Julian Togelius Michael Mateas Mike Preuss Pieter Spronck Marc Ebner, John Levine and Julian Togelius. 2013. Towards a Video Game Description Language. 6 (2013), 1–16. https://strathprints.strath.ac.uk/45278/ Artificial and Computational Intelligence in Games. Dagstuhl Follow-Ups, Volume 6, ISBN 978-3-939897-62-0.; pp. 85–100 Dagstuhl Publishing Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Germany.

[4] Gabriella Kókai Niko Bhm and Stefan Mandl. 2005. An Evolutionary Approach to Tetris. (2005). https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=b0fe1ed14404db2eb1db6a777961440723d6e06f

[5] Keiron O'Shea and Ryan Nash. 2015. An Introduction to Convolutional Neural Networks. *arXiv preprint arXiv:1511.08458* (2015). https://arxiv.org/abs/1511.08458

[6] Marc-Antoine Nüssli Patrick Jermann and Weifeng Li. 2010. Using dual eye-tracking to unveil coordination and expertise in collaborative Tetris. (Sept. 2010), 1–9. https://www.scienceopen.com/hosted-document?doi=10.14236/ewic/HCI2010.7

[7] Jorge R. Quiñones and Antonio J. Fernández-Leiva. 2019. XML-Based Video Game Description Language. *IEEE Access* 8 (2019), 4679–4692. https://doi.org/10.1109/ACCESS.2019.2962969

[8] Jeff Ramos. 2019. Tetris tips from a seven-time world champion; How to begin mastering the classic puzzle game. *Polygon* (2019). https://www.polygon.com/guides/2019/2/22/18225349/tetris-strategy-tips-how-to-jonas-neubauer

[9] Tom Schaul. 2013. A Video Game Description Language for Model-based or Interactive Learning. 6 (2013), 1–8. https://strathprints.strath.ac.uk/45278/ Artificial and Computational Intelligence in Games. Dagstuhl Follow-Ups, 6 . Dagstuhl Publishing, Wadern, pp. 85-100. ISBN 9783939897620..

[10] Wei-Chiu Ma Tian-Li Yu Wei-Tze Tsai, Chi-Hsien Yen. 2012. Tetris Artificial Intelligence. *NTUEE, Taiwan* (2012). https://chihsienyen.github.io/pdf/Tetris_AI.pdf

Received 25 October 2024