DeepLogger: Extracting User Input Logs From 2D Gameplay Videos

Thanapong Intharah[1,2] **and Gabriel J. Brostow**[1]
[1]University College London, London, United Kingdom
[2]Khon Kaen University, Khon Kaen, Thailand
[t.intharah and g.brostow]@cs.ucl.ac.uk

ABSTRACT

Game and player analysis would be much easier if user interactions were electronically logged and shared with game researchers. Understandably, sniffing software is perceived as invasive and a risk to privacy. To collect player analytics from large populations, we look to the millions of users who already publicly share video of their game playing. Though labor-intensive, we found that someone with experience of playing a specific game can watch a screen-cast of someone else playing, and can then infer approximately what buttons and controls the player pressed, and when. We seek to automatically convert video into such game-play transcripts, or logs.

We approach the task of inferring user interaction logs from video as a machine learning challenge. Specifically, we propose a supervised learning framework to first train a neural network on videos, where real sniffer/instrumented software was collecting ground truth logs. Then, once our DeepLogger network is trained, it should ideally infer log-activities for each new input video, which features gameplay of that game. These user-interaction logs can serve as sensor data for gaming analytics, or as supervision for training of game-playing AI's. We evaluate the DeepLogger system for generating logs from two 2D games, Tetris [23] and Mega Man X [6], chosen to represent distinct game genres. Our system performs as well as human experts for the task of video-to-log transcription, and could allow game researchers to easily scale their data collection and analysis up to massive populations.

CCS Concepts

•Computing methodologies → Video summarization; Supervised learning by classification; •Human-centered computing → *Interaction design theory, concepts and paradigms;*

Author Keywords

Convolutional Neural Network; User Input Log; Learning From Gameplay Video.

INTRODUCTION

To capture a player's experience means watching them, probing them with different scenarios, and interviewing them to understand how they felt in the game and afterward. Game analytics and large scale game evaluations are also important, to supplement such careful analysis of individual players. But practical constraints influence the balance between these kinds of depth vs breadth analyses. For example, when studying how players experience different game levels, important insights about play-tuning and interfaces come from both small focus-groups, and global-scale cohorts of players. When available, just the log files themselves provide invaluable insights [33, 26, 25, 27]. Critically, the sampled population size for each study is partly a question of time and cost.

The focus of this paper is to make the millions of hours of recorded gameplay videos usable, for each game's analytics and game-user research. In short, we seek to automatically convert each video into a log-file transcript, as if the player had used sniffer software while playing a known game. Privacy concerns and OS security, especially on mobile devices, keep gamers from generating and sharing log files of their game play. But *millions* of players are comfortable sharing videos online. On YouTube, there are 2.3m and 7.8m search result videos for The Legend of Zelda: Breath of the Wild [10], and PlayerUnknown's Battlegrounds [11], respectively, after the games were released for only 12 months. Twitch.tv is a website expressly for live streaming videos of games, which has 2m unique players broadcasting every month. Until now, that rich data was hard to interpret.

We introduce DeepLogger, a Convolutional Neural Network (CNN), that is custom-built to generate player-computer interaction log-files from gameplay videos. An example of its intended use would be to collect information about a specific game, and how players do better/worse depending on whether they play using a keyboard, game controller, or a particular mobile phone model. A DeepLogger CNN would first be trained by a cooperative game-player. She would record video and key/button logs on a computer with a sniffer program installed. The trained DeepLogger could then be run on each of the $10^3 \ldots 10^6$ relevant gameplay videos of that game. The resulting log files could reveal trends and advantage-giving interfaces. Additionally, for interface-researchers who don't own the game's copyright or code, they can avoid the copyright infringement risks that sometimes go along with building emulators [9].

Figure 1. Classical titles: Tetris from Nintendo Entertainment System (NES) and Mega Man X from Super Nintendo Entertainment System (SNES) are used in our experiments. Cover art © Nintendo Co., Ltd.

The proposed network is evaluated on a spectrum of quantitative metrics, through different scenarios: training and testing on different players' videos, different levels, and different video encoders. We picked two classic games from two popular console systems: Tetris (NES) [23] and Mega Man X (SNES) [6] as our test cases, shown in Figure 1.

Both obvious and unexpected challenges are set out in the next section. These are challenges faced by both humans and baseline networks, as they try to convert a video into a log file. The rest of the paper works through our solutions, and evaluation criteria for validating the DeepLogger approach.

RELATED WORK

We briefly summarize two different branches that relate to our own work. First, we discuss projects where different gaming log-files served as the main input data to studies of user behavior and gameplay experiences. Second, we highlight machine learning projects that treat computer games as experimental platforms. Where feasible, these establish a connection between machine learning algorithms and computer game analytics research.

Understanding Game Users Through Log-files

A log-file records information as a stream of events or messages. Log files have been used extensively in game user research to extract important or hidden information of gamers' emotions or behavior. Here, we touch on some of the many research projects that utilize gaming log-files (with a variety of content) as initial sources of information to extract analytics and better understand player experiences.

Shuteet al. [26, 25, 27] analyze user interaction logs in their stealth assessment of digital games for education. Stealth assessment is an assessment made from the interaction data collected during gameplay sessions. The assessment is not noticeable by the users/learners. [8] uses customized game log data to generate video summarizations of what happened during gameplay. In [31], Tveit et al. proposed using a player's action log-files from Massive Multiplayer games to mine for user patterns of behavior. Zaman and MacKenzie [33] use user input logs and game logs to evaluate different input devices for touchscreen phones. Nacke et al. [21, 22] study the use of game session logs and game event logs to assess game player experience, and to better understand the gameplay itself. Smith and Nayar [29] successfully model user's play style from the user's raw controller inputs using Latent Dirichlet Allocation (LDA).

While these works use log information as a core input component, we are unaware of any practical tool that allows researchers to retrieve such information from publicly shared gameplay videos. To the best of our knowledge, the closest research which aims to extract game information log from gameplay video is Marczak et al.work [18]. This work extracts gameplay metrics such as health bar and in-game items by simple hand-coded image processing techniques. We, however, propose a system to allow researchers to automatically extract raw control input information from existing videos. This should speed up our research community's ability to work with very large datasets, without conducting extremely laborious and costly data collection processes.

Using Machine Learning in Computer Game Research

Several attempts to probe Artificial Intelligence (AI) agents in the context of computer games played by humans are made recently through advancements in Machine Learning (ML) research, and through newly available platforms. To allow the AI and ML research communities to validate their algorithms, a variety of games has been used. Each presents its own challenges, with platform specifically chosen or custom-made by these researchers to more easily plug in their AI and ML algorithms as required. We highlight several of these well known platforms.

OpenAI Gym [5] and ALE [3] provide environments for computer agents to play Atari 2600 games and many flash games. VizDoom [16] and DeepMind Lab [2] provide API's which allow AI and ML agents to learn to play classical first person shooter (FPS) games, e.g. Doom and Quake III respectively. Recently, RLE [4] was proposed to be an environment for SNES games, and Project Malmo [15] is an environment for the game Minecraft.

These encourage ML research to develop new agents, e.g. through Reinforcement Learning. Some examples are [12] for Tetris and [20] for Mario; Deep Reinforcement Learning agents: [17, 7, 24] for Doom, [19] for Atari games, and NeuralKart [13] for Mario Kart 64; and Supervised Learning agent: TensorKart [14] for Mario Kart 64.

DeepLogger is a tool intended to complement these works, by allowing researchers to train their Machine Learning algorithms on human gameplay videos, in addition or in contrast to having these agents learn only by self-play.

BASELINES AND CHALLENGES THEY FACE

We start by illustrating the performance of human gamers when asked to estimate what buttons were pressed during the middle frame of a short video clip of someone else's gameplay.

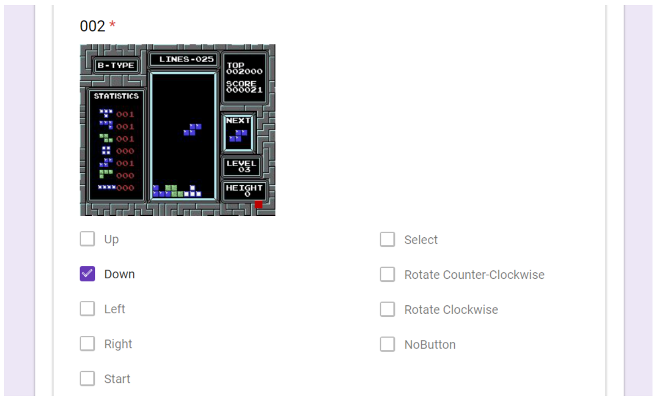
Tetris	Human	DeepLogger
Single-label Accuracy	0.8400±0.00	0.7885
Multi-Label Accuracy	0.8400±0.00	0.7911
F1-score (Example-based)	0.8644±0.00	0.8882
F1-score (Label-based)	0.7794±0.02	0.5691
Mega Man X	Human	DeepLogger
Single-label Accuracy	0.2250±0.18	0.5356
Multi-Label Accuracy	0.4420±0.20	0.7060
F1-score (Example-based)	0.5936±0.19	0.8325
F1-score (Label-based)	0.4722±0.18	0.5363

Table 1. The performance of both human experts and the proposed DeepLogger system, on estimating a gamer's controller inputs (the log) from gameplay videos only: Tetris and Mega Man X. The criteria (rows) are explained in the text, but higher accuracies and F1 scores are better. Humans are better with Tetris videos, while DeepLogger does better with Mega Man, possibly due to the UI complexity.

We then discuss the challenges faced by both a human and the automated systems, as we attempt to generate interaction logs from videos.

Baseline: Human Performance

In this section, we analyze how people fare, when estimating user input logs from gameplay videos. The small user study was conducted online, with 8 gamers recruited to participate. The gamers were first screened, selecting only those gamers who had experience playing Tetris and Mega Man X. In questionnaires, the gamers had to answer 50 questions for each of the two games. Each question displays a short video clip (Figure 2), randomly extracted from our gameplay videos. We ask the user to select all the check-boxes for game-control buttons that they think were pressed in the short video clip.



002 *

Up	Select
[✓ Down]	Rotate Counter-Clockwise
Left	Rotate Clockwise
Right	NoButton
Start	

Figure 2. An example video clip of Tetris gameplay, with check-boxes for a user to indicate what buttons they think were pressed at the middle frame. While this is a demanding and time-consuming task, users were fairly successful when "transcribing" Tetris.

Table 1 summarizes our human subjects' and previews DeepLogger's performance on the task of predicting input logs from video. We can see that the system performs better than human experts on a harder game like Mega Man X where there are many possible button-combinations, but it performs worse than human experts on Tetris, a game that is visually easier to decipher, with fewer possible button-combinations.

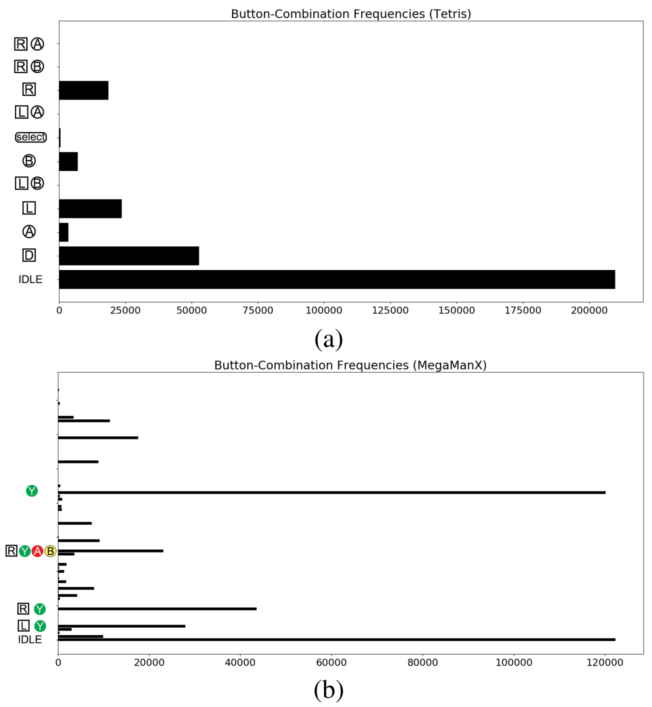


Figure 3. Button-combination Frequencies for Tetris (a) and Mega Man X (b). For Tetris, the dominant input is Idle (nothing pressed), followed by three main buttons: Down, Left, and Right. For Mega Man X, the two main classes are Idle and Shoot, followed by the "Right and Shoot" combination.

Challenge: Class Imbalance

Class imbalance happens when some controls or button-combinations are used more often than others. See button-combination statistics for Tetris and Mega Man X in Figure 3. It is quite common for game logs to have substantial imbalance. For instance, in a side-scrolling game such as Mega Man X, the character progresses by moving steadily right, so other direction-controls are pressed less often.

Substantial class imbalance, as found here, severely impacts the training of machine learning systems, and colors the performance metrics. For example, if one button combination occurs 90% of the time, and the training optimization focuses on subset accuracy (no partial credit for imperfect key combinations), then regardless of input, the network will simply always predict that one combination.

In the Architecture Section, we give the details of training our CNN while accounting for imbalanced data, and set out the performance metrics used throughout this paper.

Challenge: Multiple Control Buttons Per Record

For each time step, gamers can and tend to press multiple control keys at once. This leads to many possible unique button-combinations. For NES, eight control buttons can generate 8^2 combinations. For SNES, twelve control buttons can generate 12^2 combinations, though "select" and "start" are rarely pressed, leaving 10^2 most of the time.

Notation: Q is the number of input buttons. x is the input of our classifier, so in practice, a short gameplay video clip. Instead of a single output y, Y is the classifier's output vector, representing the state of all the controller buttons. Y' is the ground truth label associated with x.

Our CNN would normally be trained using a standard multi-class loss function H,

$$H_{Y'}(Y) := -\sum_{q}^{Q} Y'_q log Y^*_q, \tag{1}$$

that compares the ground truth output vector Y' to the softmax output tensor of the network Y^*. Y^* is

$$\text{softmax}(Y_q) := \frac{e_q^Y}{\sum_q^Q e^{Y_q}}. \tag{2}$$

Training our CNN by minimizing the normal multi-class loss (1) will not work here for two reasons. First, if each button is a disjoint class, then the loss will encourage the CNN to treat each button as mutually exclusive of others, discouraging chords. Second, if we consider each button-combination as a class, the network will not be able to predict classes that weren't present in the training set.

The Losses section in our Architecture description details how our multi-label loss and multi-label-multi-choice loss are designed to cope with these problems.

Challenge: Many-to-One

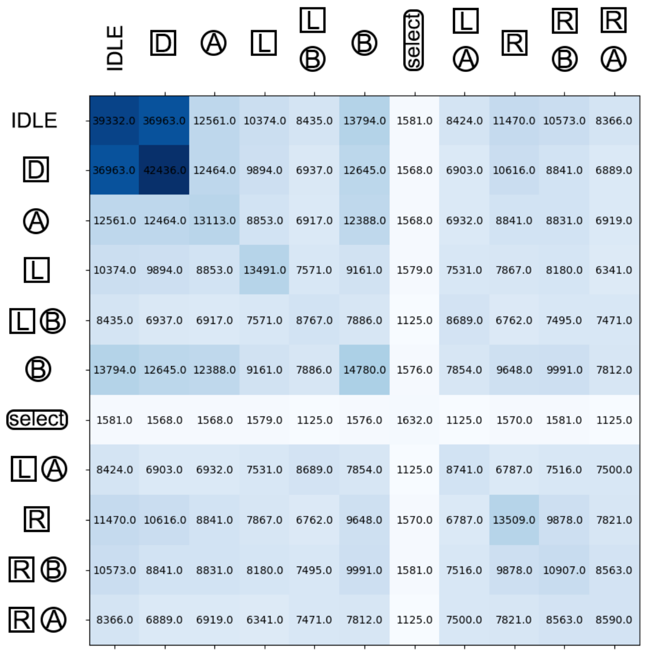
This is a problem that we did not anticipate, and it may surprise readers who have not analyzed interaction logs. In most games, there are times when pressing two different buttons (or button-combinations) produces the same in-game result. We refer to these situations as functionally equivalent, or as many-to-one situations. Figure 4 depicts the confusion statistics of our two sample games.

This challenge is the obstacle that most affects human performance on the video-to-log task. We consider a multi-button-combination as a single class label, for simplicity.

Many-to-one happens when many classes generate one output. For example, when the game is unresponsive, no matter which keys are pressed, the subsequent frames are the same. To account for video examples with more than one associated label, we propose the multi-label-multi-choice loss, to tackle the problem. The result is analyzed in **Result: Many-to-One** section.

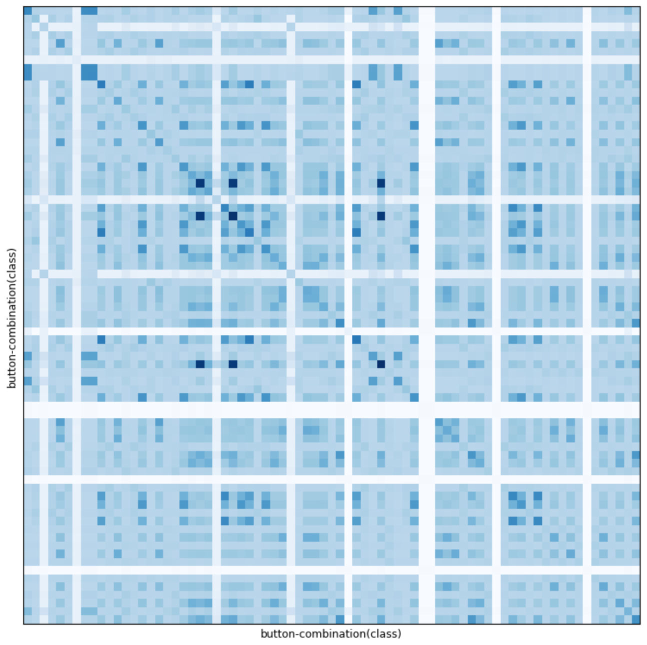
ARCHITECTURE

Convolutional Neural Networks (CNN's) have proved successful in a wide range of applications, especially on computer vision tasks such as image classification. We devise a new CNN architecture for transcribing user input logs from videos of gameplay, by combining existing CNN components and carefully choosing appropriate training procedures to tackle gameplay video challenges. Moreover, a new loss function, multi-label-multi-choice loss, is proposed to tackle the many-to-one challenges, described in the **Challenge: Many-to-One**



	IDLE	$\boxed{D}$	Ⓐ	$\boxed{L}$	$\boxed{L}$ Ⓑ	Ⓑ	select	$\boxed{L}$ Ⓐ	$\boxed{R}$	$\boxed{R}$ Ⓑ	$\boxed{R}$ Ⓐ
IDLE	39332.0	36963.0	12561.0	10374.0	8435.0	13794.0	1581.0	8424.0	11470.0	10573.0	8366.0
$\boxed{D}$	36963.0	42436.0	12464.0	9894.0	6937.0	12645.0	1568.0	6903.0	10616.0	8841.0	6889.0
Ⓐ	12561.0	12464.0	13113.0	8853.0	6917.0	12388.0	1568.0	6932.0	8841.0	8831.0	6919.0
$\boxed{L}$	10374.0	9894.0	8853.0	13491.0	7571.0	9161.0	1579.0	7531.0	7867.0	8180.0	6341.0
$\boxed{L}$ Ⓑ	8435.0	6937.0	6917.0	7571.0	8767.0	7886.0	1125.0	8689.0	6762.0	7495.0	7471.0
Ⓑ	13794.0	12645.0	12388.0	9161.0	7886.0	14780.0	1576.0	7854.0	9648.0	9991.0	7812.0
select	1581.0	1568.0	1568.0	1579.0	1125.0	1576.0	1632.0	1125.0	1570.0	1581.0	1125.0
$\boxed{L}$ Ⓐ	8424.0	6903.0	6932.0	7531.0	8689.0	7854.0	1125.0	8741.0	6787.0	7516.0	7500.0
$\boxed{R}$	11470.0	10616.0	8841.0	7867.0	6762.0	9648.0	1570.0	6787.0	13509.0	9878.0	7821.0
$\boxed{R}$ Ⓑ	10573.0	8841.0	8831.0	8180.0	7495.0	9991.0	1581.0	7516.0	9878.0	10907.0	8563.0
$\boxed{R}$ Ⓐ	8366.0	6889.0	6919.0	6341.0	7471.0	7812.0	1125.0	7500.0	7821.0	8563.0	8590.0

(a)



(b)

Figure 4. Functionally equivalent button-combinations for Tetris, shown in (a), and Mega Man X shown in (b), are visualized through confusion matrices. Each numerical entry indicates how many times the button-combination (of that row) produces the same visual output as another button-combination (column).

section. The network is implemented in Tensorflow [1]. The dataset and the codes can be found at the project page.[1] It is worth to note here that all the network setting: the number of convolution and fully connected layers as well as dropout ratio and learning rate were achieved empirically through a number of experiments which are omitted in this paper.

The Network

Our DeepLogger network is composed of five 3D convolutional layers, each of which has rectified linear units (ReLU) as activation functions. The network then has five fully connected layers with dropout between layers. Table 2 demonstrates the diagram of our proposed CNN network.

Layers	Kernel Dimensions	Number of kernels
Input	input dimensions = D×1×W×H	
Conv1	3×5×5	24
Conv2	3×5×5	36
Conv3	3×5×5	48
Conv4	3×3×3	64
Conv5	3×3×3	64
Dense6	output dimensions = 1164	
Dropout	keep = 0.8	
Dense7	output dimensions = 100	
Dropout	keep = 0.8	
Dense8	output dimensions = 50	
Dropout	keep = 0.8	
Dense9	output dimensions = 30	
Dropout	keep = 0.8	
Output	output dimensions = C	

Table 2. Diagram of the proposed DeepLogger Network where D is the number of frames per clip, which are 21 frames and 11 frames for Tetris and Mega Man X respectively. W×H are the image dimensions of the gameplay videos, which are the default screen dimensions of NES, 256×224 for Tetris, and SNES, 586×448 for Mega Man X. C is the number of buttons, with 8 buttons for Tetris and 12 buttons for Mega Man X.

3D convolution layers

In this work, we deem a video as a stack of temporal 2D images (frames). Before passing each frame to the network, the RGB frame is transformed into a grayscale image. While 2D convolution layers look at a whole temporal block at a time which does not consider relations between consecutive frames and the frames order, 3D convolution layers look at smaller temporal blocks. In Table 2, "Conv" are 3D convolution layers. The temporal dimension of each layer is set to 3 which means that the network is restricted to consider temporal relations of every three consecutive frames of the given temporal block.

Training

Since user input log data is extremely imbalanced, as shown in Figure 3, we train the network by over-sampling all other classes. The effect is that in each mini-batch, the non-majority classes (button-combinations) have comparable numbers of samples to the majority class.

We chose ±5 and ±10 consecutive frames per short clip, as one data point for Mega Man X and Tetris respectively. The network is trained with the mini-batch scheme using batch size 16 for 50 epochs. We use 1E-5 as our fixed learning rate for both games.

We compare and discuss the imbalanced training procedure with the normal training procedure on both games in the Experiments section.

Losses

As discussed in **Challenge: Multiple Control Buttons Per Record**, each input button can be pressed at the same time and pressing one button is independent of pressing another button. We frame this problem as the multi-label problem where each class can occur independently. To train a multi-label NN classifier, sigmoid cross entropy loss is used:

$$H_{Y'}(Y) := -\sum Y'_q log(\sigma(Y_q)), \tag{3}$$

where $\sigma(\dagger) = \frac{1}{1+e^{\dagger}}$ is the result of applying the sigmoid function to the output of the network.

However, an extremely challenging characteristic of gameplay video is the many-to-one issues are not yet addressed; and from our inspection of the training data, these situations happen a lot, as shown in figure 4.

To tackle the issues, we modify the sigmoid cross entropy loss to consider multiple label choices. When there are more than one class (button-combination) which generates the same visual output as another class, functionally equivalent class, the loss should not penalize those classes even if they are not the ground truth label.

To make our network aware of that, we generate a new ground truth by replaying the emulator with the ground truth input logs and checking which button-combination is functionally equivalent to the ground truth label. By doing that, the new ground truth label for each example become a set of labels. We call it multi-choice labels.

The modified loss is designed to consider each ground truth label as a set. It the looks for the best label from those label set, and computes multi-label loss against that label. We name this loss multi-label-multi-choice loss. The multi-label-multi-choice loss is mathematically defined as

$$H_{\mathbf{Y'}}(Y) := \min_{Y' \in \mathbf{Y'}} (-\sum Y'_q log(\sigma(Y_q))), \tag{4}$$

where $\mathbf{Y'_i}$ is a multi-choice ground truth label of example i.

EXPERIMENTS AND RESULTS

In this section, we discuss experiments which were used to validate different components of the network, as well as the whole system's performance. Table 3 demonstrates the key figures from the CNN experiments. DeepLogger is our proposed network; DeepLogger2D is the modified version of our proposed network to validate performance of using 3D filters in the convolutional layers; the VGGNet [28] is a baseline CNN due to its broad acceptance in the computer vision community.

Data

We collected the datasets of both Tetris and Mega Man X using the BizHawk Emulator version 2.1.1. Gamers for each

[1] http://visual.cs.ucl.ac.uk/pubs/DeepLogger/

Tetris	DeepLogger (-balanced)	DeepLogger2D (-balanced)	VGGNet [28] (-balanced)
Single-label Accuracy	**0.7885** (0.7735)	0.5712 (0.6558)	0.6386 (0.6558)
Multi-Label Accuracy	**0.7911** (0.7735)	0.5734 (0.6558)	0.6298 (0.6553)
F1-score (Example-based)	**0.8882** (0.8754)	0.7675 (0.7921)	0.7841 (0.7921)
F1-score (Label-based)	**0.5691** (0.1865)	0.0874 (0.0000)	0.03907 (0.0000)
Mega Man X	DeepLogger (-balanced)	DeepLogger2D (-balanced)	VGGNet [28] (-balanced)
Subset Accuracy	0.5356 (**0.6014**)	0.4126 (0.5088)	0.2730 (0.4480)
Multi-Label Accuracy	0.7060 (**0.7459**)	0.6135 (0.6901)	0.5151 (0.6400)
F1-score (Example-based)	0.8325 (**0.8587**)	0.7740 (0.8278)	0.7179 (0.7970)
F1-score (Label-based)	**0.5363** (0.4352)	0.3578 (0.3149)	0.2866 (0.2616)

Table 3. Performance of 3 CNN's: DeepLogger, DeepLogger2D, and VGG on Tetris and Mega Man X. Figures in parentheses indicate performance of the networks when they were trained without over-sampling the non-majority classes.

game were recruited to play the game multiple times. For Mega Man X, we use one playthrough recoded from one gamer as our training data and another recoded playthrough from the same gamer as the test data. For Tetris, the network was trained on twenty gameplay recordings of one level from one gamer; and the test data comprises of gameplay recordings of three different levels and two additional gamers. BizHawk records user control inputs at each time step for each gameplay, in a log-file which can be played back later. We use this user input log, Y_i and the associated screenshot image, im_i, to construct the training data for our model. For our training data, one data point is a clip of 2n+1 frames, $x_i = \{im_{i-n}, im_{i-n+1}, ..., im_i, ..., im_{i+n-1}, im_{i+n}\}$, where $n = 5$ for Mega Man X and $n = 10$ for Tetris.

The data is split into 80% training set, 5% for the validation set, and 15% test set, which amounts to 316,848 training examples for Tetris and 436,203 training examples for Mega Man X.

Performance Metrics

We benchmark the performance of our system over a range of different metrics [34]. All metrics and their characteristics

Single-label Accuracy Subset Accuracy is a performance metric that captures the fraction of perfectly correct predictions. This performance metric only counts as correct the predictions where Y is identical to the ground truth for all buttons/controls. This metric is similar to the accuracy of the single-label problem.

Multi-label Accuracy evaluates the fraction of correctly classified labels in the multi-label setting. It returns 1 if the predicted set of input buttons is identical to the ground truth, similar to the subset accuracy above, but it returns non-zero numbers if the predicted result is partly correct, so if some of the input buttons match. This metric allows us to analyze more fine-grained performance measurements of the classifiers, because it gives partial credit.

Example-based F1-Score measures the harmonic mean of Precision and Recall, where it first evaluates the performance of each example separately, and then returns the average value across the test set. Precision and Recall are complimentary performance measurements to Accuracy, because they take into account class imbalance, via false positive and false negative statistics.

Label-based F1-Score computes an F1-score by first evaluating the performance of each class label input button separately, and then returning the average value across all class labels, where every class label is given equal weight (Macro average).

Although the Example-based F1-Score is good for being sensitive to imbalanced data, it can be misled by very large numbers of test examples. Label-based F1-Score, instead, focuses on measuring the performance of predicting individual class labels for each input button.

Result: Overall Results

Table 3 presents the performance of the DeepLogger network on Tetris and Mega Man X. It is clear that DeepLogger has better performance across all four criteria, at least as compared to the 2D alternate version of DeepLogger, and against VGGNet [28], which is a standard architecture used in thousands of computer vision tasks. We touch on the numerical results below, with respect to the different challenges identified at the outset.

Figure 5 and Figure 6 demonstrate detailed analysis of how the network performs on per button and per class (button-combination) predictions. For Tetris, the per button accuracy chart (a) suggests that the class *rotate block clockwise* and class *rotate block counter-clockwise* are harder for the network to distinguish than the classes which translate the block. This leads to poor performance on the button-combination prediction, where the classes have both direction and rotation buttons being pressed. For Mega Man X, the per button accuracy chart (b) indicates that the network always made mistakes at predicting the button X because this button does not map to any action, Also, the network got relatively low scores on the button L because L and R are sliding the inventory window left and right respectively, and there is no visual distinction between the two.

Result: Class Imbalance

We trained three different CNN architectures with/without over-sampling the non-majority classes, to balance the number of examples from each class of button-combinations. The results are listed in Table 3 where training with the over-sampling scheme is shown first, and results after training without the over-sampling scheme are shown in parentheses. From the table, we see that for Mega Man X, training with the over-sampling scheme can cause a slight performance drop on three metrics: single-label accuracy, multi-label accuracy, and

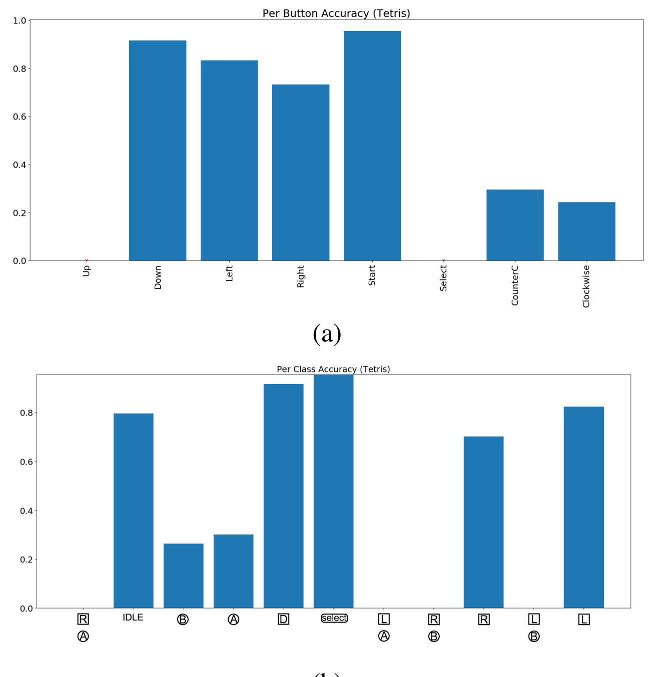


Figure 5. Detailed analysis of our DeepLogger network's performance on Tetris for per button and per class prediction. (a) shows the performance of per button prediction and (b) shows the performance of per class prediction. In both cases, rotation is the hardest to recognize. Small red X's on the per button accuracy chart indicate there is no ground truth label for the buttons (Up and Select).

example-based F1-score. However, the over-sampling scheme always improves the label-based F1-score.

Result: 2D VS 3D Filters

This experiment demonstrates the benefit of using 3D filters for predicting an action from a sequence of images. The performance of two identical network where one uses 3D convolutional layers (DeepLogger) and another that uses 2D convolutional layers (DeepLogger2D) are shown in Table 3. Please note that, VGGnet [28] also uses 2D convolutional layers.

For both games, we can clearly see that for predicting user inputs from gameplay video, 3D convolutions significantly improve the performance of the CNN's across all metrics.

Result: Many-to-One

Figure 7 shows the improvement in score when training with multi-label-multi-choice loss, compared to training with ordinary sigmoid cross entropy loss.

To generate a multi-choice label from a normal single-choice input log, we rely here (only) on an emulator. We replay the recorded user input logs through the emulator, where at each subsequent time step, we try all possible button-combinations, looking for ones that generate the same visual output as the original single-label. These functionally equivalent (at least in the short-term) extra combinations are added to the list of multi-choice labels, supplementing the original ground truth.

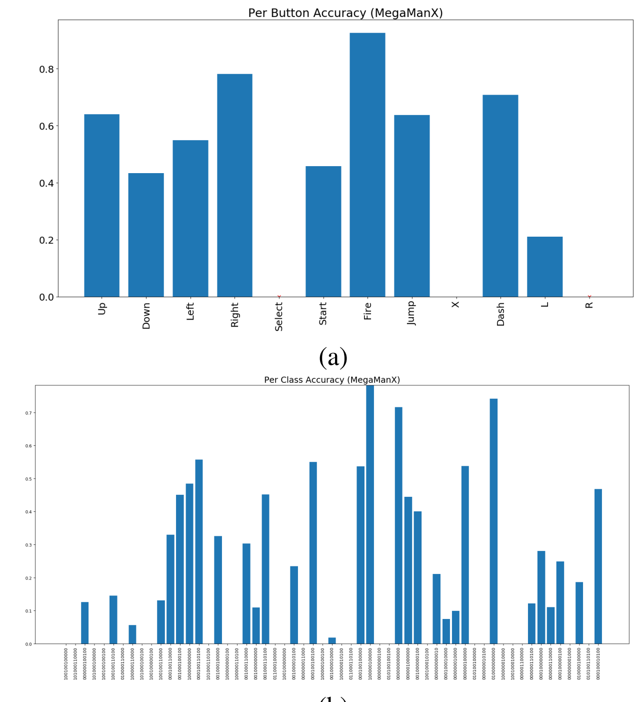


Figure 6. Detailed analysis of our DeepLogger network's performance on Mega Man X for per button and per class prediction. (a) shows the performance of per button prediction and (b) shows the performance of per class prediction. Binary labels in (b) along the horizontal axis represent *pressing (1)* and *not pressing (0)* that game controller button and The Binary codes preserve the button-order from (a). Red X's on the per button accuracy chart indicate there is no ground truth label for the buttons (R and Select).

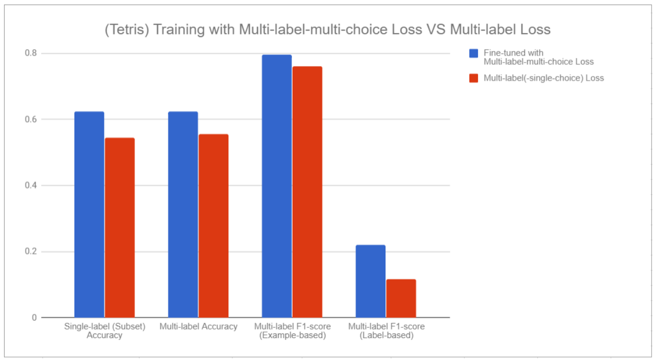


Figure 7. Comparing loss functions. This bar-chart compares two DeepLogger networks using four performance metrics. For the red bars, the network was trained with normal multi-label loss, and for the blue bars, the network was first trained with normal multi-label loss on 90% of the training data, and then fine-tuned with multi-label-multi-choice loss on the remaining 10% of the training data. This bar-chart is generated from the Tetris gameplay dataset. Under each measure, fine-tuning with multi-label-multi-choice loss yielded better scores.

This process takes substantial time per gameplay log-file, so figure 7 only reports the performance when we fine-tune the network with multi-choice-multi-label loss on 10% of the training data. This process is trivially parallelizable if needed.

Tetris	Base	Diff 1 level	Diff 2 levels	Diff Gamers	Diff Encoder
Single-label Accuracy	0.7885	0.7251	0.7988	0.7228	0.7633
Multi-Label Accuracy	0.7911	0.7269	0.8017	0.7235	0.7649
F1-score Example-based	0.8882	0.8489	0.8870	0.8441	0.8743
F1-score Label-based	0.5691	0.4529	0.5504	0.4137	0.6690

Table 4. Generalization evaluation. *Base* is the performance of the system when testing on gameplay video from the same level, gamer, and encoder. *Diff 1 level* shows the performance of the system when testing with gameplay videos from a different level than the training data, by 1 level, *Diff 2 levels* tests the same thing, but where the difference is 2 levels. *Diff Gamers* level shows the performance of the system when testing with gameplay videos from different gamers. *Diff Encoder* level shows the performance of the system when testing gameplay videos downloaded from video-sharing site YouTube, where compression and frame-rate changes can occur.

Result: Generalization

Table 4 shows experiments where the network is validated on further challenging tasks: training on one person's gameplay videos and then testing on another person's, training on one level and then testing on different levels, and training with locally collected video, and then testing on different videos that were encoded by YouTube and downloaded back again, as if scraped.

Result: CNN Embedding

This secondary benefit of our system has some interesting opportunities. Figure 8 visualizes what is learned by the network. We can see that the network learns to ignore or at least tolerate the background of the screenshots, and focuses on the main character action.

While the estimated logs are the main focus of this paper, the neural network's ability to embed "similar" frames near each other in feature space is valuable for other kinds of analytics. For example, a researcher could search, across all recorded gameplay of one or many users, for situations where the game takes a certain turn, or the player performs a certain chain of actions. These would correspond to points in the CNN's abstract feature space, but could be found more reliably than, say, using the absolute time passed from the start of a level - different players move at different speeds. Also, the same game played across different devices may have different logs, but the appearance part of interesting in-game situations will be comparatively consistent.

DISCUSSION AND CONCLUSION

We have shown that it is possible to extract UI log information from gameplay videos. While the accuracy of our system still has room for improvement, compared with the 100% one would get from using sniffer software, our system could be applied to the millions of users' videos that are generated each month. Preserving users' privacy while collecting broad-scale usage statistics has a value that is presently hard to measure. Additionally, the CNN-embedding gives researchers new opportunities to compare gameplay across game levels and across players. This is similar to the opportunity DeepLogger may present for AI researchers who want to train game-playing AI's, without relying purely on self-play.

Our proposed network, along with the learning algorithm that finds functionally equivalent user commands, outperforms the baseline networks in Table 3. Presently, it performs on par with what humans can do when focusing their attention, as shown in Table 1. It seems from the Mega Man X experiments that human performance is not a ceiling on accuracy or F1 scores (people are slightly worse on Mega Man X), but further machine learning developments are needed to improve on those metrics.

The current network can only predict user input logs for a game where training data is available. In further work, it would be attractive to learn from many games, to try to generalize across games, increasing the pool of available training data.

Due to we frame the problem as a classification problem, the system is limited to games with discrete inputs. Further work on transcribing games with continuous inputs is worth exploring because modern games tend to use continuous inputs such as mouse trajectory, gyroscope, or gestures.

Last but not least, in future work it would be interesting to extend the network to work with multi-player games. For 2-player games which each of their two controllers is fixed to the certain part of the screen, such as Super Mario Kart [30], or fixed to a certain character, such as Contra [32], the network might need only a little tweak to work. However, the more challenging research topic is to transcribe the games of which the controllers are not fixed to a certain aspect.

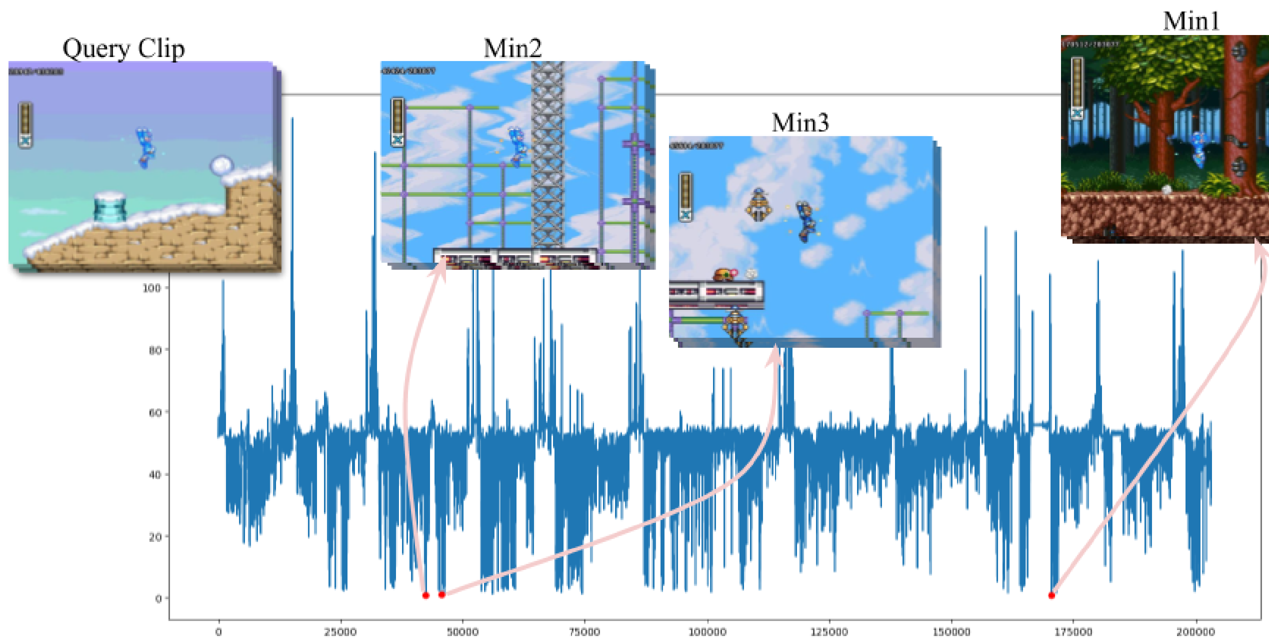


Figure 8. The graph visualizes Euclidean distances between embedding features, produced by the "Dense9" layer of the network, for the query clip and other clips from different videos. Min1, Min2, and Min3 clips are examples of the clips which have the smallest distances to the query clip. The Y-axis represents distance and X-axis represents time steps in the video of the retrieved gameplay.

REFERENCES

1. Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, and others. 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467* (2016).
2. Charles Beattie, Joel Z Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, and others. 2016. Deepmind lab. *arXiv preprint arXiv:1612.03801* (2016).
3. Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. 2015. The Arcade Learning Environment: An Evaluation Platform for General Agents. In *Proceedings of the 24th International Conference on Artificial Intelligence (IJCAI'15)*. AAAI Press, 4148–4152. `http://dl.acm.org/citation.cfm?id=2832747.2832830`
4. Nadav Bhonker, Shai Rozenberg, and Itay Hubara. 2016. Playing SNES in the Retro Learning Environment. *arXiv preprint arXiv:1611.02205* (2016).
5. Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. OpenAI Gym. *arXiv:1606.01540* (2016).
6. Capcom. 1993. *Mega Man X*. Game [SNES]. (17 December 1993).
7. Devendra Singh Chaplot and Guillaume Lample. 2017. Arnold: An Autonomous Agent to Play FPS Games.. In *AAAI*. 5085–5086.
8. Yun-Gyung Cheong, Arnav Jhala, Byung-Chull Bae, and Robert Michael Young. 2008. Automatically Generating Summary Visualizations from Game Logs.. In *AIIDE*. 167–172.
9. James Conley, Ed Andros, Priti Chinai, and Elise Lipkowitz. 2003. Use of a game over: Emulation and the video game industry, a white paper. *Nw. J. Tech. & Intell. Prop.* 2 (2003), 261.
10. Hidemaro Fujibayashi, Eiji Aonuma, Akihito Toda, and Satoru Takizawa. 2017. *The Legend of Zelda: Breath of the Wild*. Game [Nintendo Switch]. (3 March 2017).
11. Brendan Greene and Chang han Kim. 2017. *PlayerUnknown's Battlegrounds*. Game [Microsoft Windows]. (23 March 2017).
12. Alexander Groß, Jan Friedland, and Friedhelm Schwenker. 2008. Learning to play Tetris applying reinforcement learning methods.. In *ESANN*. 131–136.
13. Harrison Ho, Varun Ramesh, and Eduardo Torres Montano. 2017. NeuralKart: A Real-Time Mario Kart 64 AI. (2017).
14. Kevin Hughes. 2016. Tensorkart: self-driving mariokart with tensorflow. Blog. (28 December 2016). `https://kevinhughes.ca/blog/tensor-kart` Accessed April 13, 2018.
15. Matthew Johnson, Katja Hofmann, Tim Hutton, and David Bignell. 2016. The Malmo Platform for Artificial Intelligence Experimentation.. In *IJCAI*. 4246–4247.

16. Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski. 2016. Vizdoom: A doom-based ai research platform for visual reinforcement learning. In *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*. IEEE, 1–8.

17. Guillaume Lample and Devendra Singh Chaplot. 2017. Playing FPS Games with Deep Reinforcement Learning.. In *AAAI*. 2140–2146.

18. Raphaël Marczak, Jasper van Vught, Gareth Schott, and Lennart E. Nacke. 2012. Feedback-based Gameplay Metrics: Measuring Player Experience via Automatic Visual Analysis. In *Proceedings of The 8th Australasian Conference on Interactive Entertainment: Playing the System (IE '12)*. ACM, New York, NY, USA, Article 6, 10 pages. DOI: http://dx.doi.org/10.1145/2336727.2336733

19. Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

20. Shiwali Mohan and John E Laird. 2009. Learning to play Mario. *Tech. Rep. CCA-TR-2009-03* (2009).

21. Lennart Nacke, Craig Lindley, and Sophie Stellmach. 2008. Log whoâĂŹs playing: psychophysiological game analysis made easy through event logging. In *Fun and games*. Springer, 150–157.

22. Lennart Nacke, Jonas Schild, and Joerg Niesenhaus. 2010. Gameplay experience testing with playability and usability surveys–An experimental pilot study. In *Proceedings of the Fun and Games 2010 Workshop, NHTV Expertise Series*, Vol. 10.

23. Alexey Pajitnov and Vladimir Pokhilko. 1984. *Tetris*. Game [NES]. (6 June 1984).

24. Dino Ratcliffe, Sam Devlin, Udo Kruschwitz, and Luca Citi. 2017. Clyde: A deep reinforcement learning DOOM playing agent. (2017).

25. Val Shute. 2015. Stealth assessment in video games. (2015).

26. Valerie J Shute and Gregory R Moore. 2017. Consistency and validity in game-based stealth assessment. *Technology enhanced innovative assessment: Development, modeling, and scoring from an interdisciplinary perspective* (2017), 31–51.

27. Valerie J Shute, Matthew Ventura, and Diego Zapata-Rivera. 2013. Stealth assessment in digital games. (2013).

28. Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

29. Brian A. Smith and Shree K. Nayar. 2016. Mining Controller Inputs to Understand Gameplay. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology (UIST '16)*. ACM, New York, NY, USA, 157–168. DOI: http://dx.doi.org/10.1145/2984511.2984543

30. Tadashi Sugiyama and Hideki Konno. 1992. *Super Mario Kart*. Game [SNES]. (27 August 1992).

31. Amund Tveit and Gisle B Tveit. 2002. Game Usage Mining: Information Gathering for Knowledge Discovery in Massive Multiplayer Games.. In *International Conference on Internet Computing*. 636–642.

32. Shigeharu Umezaki and Shinji Kitamoto. 1987. *Contra*. Game [NES]. (20 Febuary 1987).

33. Loutfouz Zaman and I Scott MacKenzie. 2013. Evaluation of nano-stick, foam buttons, and other input methods for gameplay on touchscreen phones. In *International Conference on Multimedia and Human-Computer Interaction-MHCI*. 69–1.

34. Min-Ling Zhang and Zhi-Hua Zhou. 2014. A review on multi-label learning algorithms. *IEEE transactions on knowledge and data engineering* 26, 8 (2014), 1819–1837.