Final Report: Video Game Recognition from Screenshots

Nikhil Devraj devrajn@umich.edu
Victor Hao vhao@umich.edu
Nolan Kataoka nkataoka@umich.edu
Nicholas Kroetsch kroetsch@umich.edu

Abstract

We apply contemporary computer vision techniques in the effort to classify video games from their screenshots in order to allow Twitch streamers to automatically change their stream information as they switch games. We collect our own dataset of Twitch and YouTube game screenshots, with 60000 examples in 12 classes. We then compare the Squeezenet architecture pretrained on ImageNet with our own CNN architecture. We then analyze the results of these networks, show that our own architecture performs better than a pretrained network for this task, and apply this network to a fully functional Twitch app prototype.

1. Introduction

The classification of video games from screenshots is an increasingly relevant problem in the field of computer vision and social media. The ability to automatically detect and classify which games a video is about or a streamer is playing can be useful for the platforms that host these content creators. For example, when Twitch streamers stream different games in succession, they may forget to manually update their stream information. Failure to update stream information results in miscategorization of streams in user searches, which can lead to loss of potential viewership for content creators and the streaming platforms themselves. This problem is solvable using contemporary computer vision techniques since many games have very distinct features that allow human viewers to recognize them instantly, and a CNN is able to be trained on such features to classify screenshots taken from gameplay video.

We approach this task using a dataset of screenshots we have collected across a selection of games hosted on YouTube and Twitch. Our approach utilizes these images as inputs to a both pretrained SqueezeNet model that inputs into an SVM to categorize the screenshot and a baseline CNN classifier. While both models achieve statistically significant levels of accuracy above random, our baseline CNN classifier performs far better than the SqueezeNet classifier with an accuracy of 95%. We then use our trained CNN model inside a Twitch app that is capable of automatically classifying the game a Twitch streamer is playing and changing stream information as the streamer switches games. By analyzing multiple screenshots over time, our Twitch app is able to reach accuracy levels of greater than 99%.

2. Contributions

The inspiration for this project was original, and as a result there hasn't been much related work on this problem. We thought it best for us to use an established model as a starting point to see what kind of performance a pre-trained model can achieve, and compare its performance with a very basic CNN that we designed. The pre-trained model that we decided to use was SqueezeNet, which was originally developed by researchers at DeepScale, University of California Berkeley, and Stanford University and trained on the ImageNet data set.

We lifted the basic framework of the neural network training code from starter code given in a University of Michigan EECS 442 Computer Vision homework. The final code used to run both our CNN and SqueezeNet are heavily modified to work for our particular problem and dataset. The code for our Twitch application is original and utilizes the Twitch APIs to grab stream image information which is used to make predictions.

We collected game recordings from videos on YouTube and VODs from Twitch all coming from a variety of content creators and players. The script used to split the videos into the frames which made up the dataset was written as an original work.

3. Data

The data used for this project consists of 5000 224x224x3 frames of in-game screenshots from 12 popular video games, for a total of 60000 screenshots in the dataset. This data was collected using a script that downloads YouTube videos of gameplay. We then manually trimmed the clips to remove any streamer intro/outros and out-of-game menu screens. The script then extracts one frame every 5s of footage. These frames are resized to 224x224x3 and saved in our dataset folders.

The full list of games used is as follows:

- League of Legends
- Counter-Strike: Global Offensive
- Minecraft
- Fortnite
- Grand Theft Auto V
- FIFA ’20
- Hearthstone
- Super Smash Bros. Ultimate
- Rocket League
- Team Fortress 2
- Hollow Knight
- Slenderman

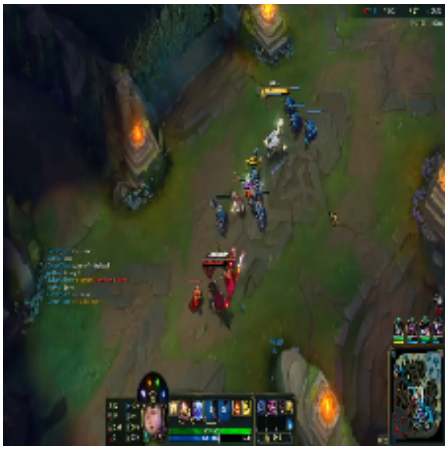


Figure 1: An example dataset image from League of Legends.

In order to collect 5000 screenshots for each game, around 7 hours of footage was necessary per game. These videos were trimmed to remove any content creator intro/end sequences as well as any out-of-game menus or screens. The videos used were specifically chosen to not have any streamer overlays, facecams, or game modifications in order to ensure that these would not affect training. However, a possible side effect of this is that streamer overlays may cause the classifier to produce incorrect output.

Some potential issues that may occur due to the limitations of our dataset are:

- Updates to a game that change the UI or textures of a game could potentially cause it to be misclassified.
- Some games have many maps and characters that each look very distinct from one another, and if a very unique map/gameplay feature is not included in the dataset it could skew the results.
- Some games have multiple game modes with different UIs which could cause different features to be recognized than those in the training dataset.
- Many streamers have stream overlays, facecams, or game modifications. These were included in the training set, so images including these could be incorrectly classified.
- A major limitation of the dataset is the limited number of games that we included. There are thousands of popular games on the market today, and our dataset only contains 12 different games to act as a proof of concept of the model. A production version of this concept would need to be trained on thousands of different classes.

4. Method

Our method used to classify video game screenshots consists of preprocessing the image, then using that as an input to a pretrained SqueezeNet model. We then use the outputs of the SqueezeNet as inputs to a SVM classifier which outputs its prediction of which game the screenshot is.

4.1. SqueezeNet

The pretrained SqueezeNet model that we used was included with Pytorch along with the SVM used for classification based on SqueezeNet outputs.

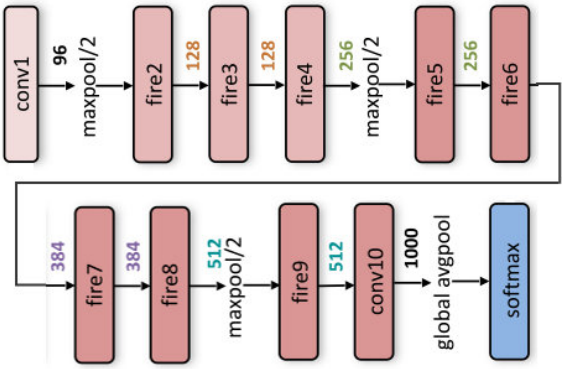


Figure 2: A portion of SqueezeNet’s architecture.

This model takes a 224x224x3 input and outputs a vector of 1000 classes. We then use this as an input into a SVM with an output dimension of 12 (the number of our games). So to train the SVM, we evaluated Squeezenet on our training set and use those outputs as training inputs for the SVM, which was evaluated on the labels originally generated. Evaluation worked the same way, where Squeezenet was first evaluated and then SVM evaluation was run.

4.2. CNN

We also tested our SqueezeNet-based classifier against a baseline CNN implementation.

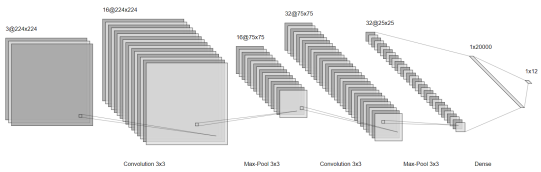


Figure 3: The CNN architecture.

The CNN used is composed of two 3x3 convolutional layers each followed by a 3x3 maxpool layer, with two fully connected layers at the end. The final linear layer's output is of size 12, which is the number of classes (1 for each game). ReLU is used as the activation function, and is applied to the outputs of each convolutional layer before maxpooling. Activation maps are obtained by observing the output of the final convolutional layer during model evaluation and averaging each of the channels to form a single heatmap.

The hyperparameters used for training the CNN were: batch_size=64, learning_rate=1e-3, weight_decay=1e-5, and num_epochs=10.

4.3. Twitch App

We then created our own app for Twitch streamers that can automatically take screenshots, run our trained model on the screenshots, and then use the output to update the current stream information on the Twitch stream. The application works as a locally hosted web server built on Python (Flask) with an HTML/CSS3/Javascript browser interface.

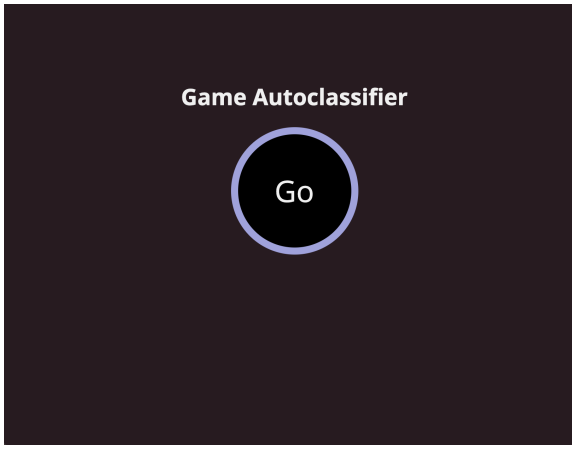


Figure 4: A screenshot of the web interface for our autoclassifier app.

The app runs as follows:

1. Authenticate with Twitch (OAuth)
2. While the classifier is running, once per second:
   (a) Take a screenshot
   (b) Analyze the game in the screenshot using the classifier
   (c) Based on the past M screenshots, find the game g to which the highest fraction of recent screenshots belong γ
   (d) If $\gamma >$ a confidence threshold γ^*, update the Twitch stream information

By analyzing multiple screenshots over time rather, the application can significantly reduce error from the classifier's single-image accuracy rate.

Given that there are N classes (games), the trained classifier has a single-image accuracy rate of α_s, the past M images are considered when guessing the current game being played, and a confidence threshold of γ^*, we can calculate the error rate ζ_{app} of the application. Specifically, we want to know how likely it is that, if a game exceeds the threshold γ^*, it is the wrong game.

Let the single-image error rate of the classifier be $\zeta_s = 1 - \alpha_s$ and $d = \gamma^* M$ for convenience.

Assuming that each of the incorrect games has the same chance of being chosen:

The probability an image is incorrectly classified as a specific game is:

$$\frac{\zeta_s}{N-1}$$

The probability a particular incorrect game is chosen exactly d times out of M is:

$$\binom{M}{d}\left(\frac{\zeta_s}{N-1}\right)^d\left(1-\frac{\zeta_s}{N-l}\right)^{(M-d)}$$

Neglecting the chances that multiple games exceed the $\gamma^* M$ threshold, the probability that a particular incorrect game is chosen at least d times out of M is:

$$\frac{\zeta_{app}}{N-1} = \sum_{k=d}^{M}\binom{M}{k}\left(\frac{\zeta_s}{N-1}\right)^k\left(1-\frac{\zeta_s}{N-l}\right)^{(M-k)}$$

Using a program or spreadsheet, the above equation can be plotted to determine the effect of M and d on ζ_{app}, given our classifier's accuracy.

There is a clear trade-off between d and the speed with which the application recognizes the current game being played and updates the stream accordingly. If the application takes screenshots at a rate of 1 image per second, d = 3 means the earliest the application could detect a game change is 3 seconds after the game has been changed.

With M=10 and d=3, $\zeta_{app} = 1.2e-4$. This is an extremely low error rate, so there is no need to increase d and M. Since streams often last on the order of hours, we believe even a 10 second delay (d=10) would be acceptable to streamers. In that case, M=40 and d=10 results in

$\zeta_{app} = 3.1e - 14$. Additionally, even if the app misidentifies the game, it would likely fix its mistake soon after.

Earlier, we assumed that each of the incorrect games had the same chance of being chosen by the classifier. This is a bad assumption given that some games have similar visual styles (from being built in the same game engine to even sharing assets). This can be remedied by increasing $zeta_s$ by a safety factor.

Using a safety factor of 3, tripling ζ_s from 0.05 to 0.15, using M=10 and d=3 again, ζ_{app} only increases to 3.1e-3.

There are additional considerations that could affect the application's performance, such as the presence of non-game screenshots (e.g. when the streamer switches to their desktop) and game screens that weren't included in our dataset (e.g. title screens and some menus). A future consideration could be to output the softmax of the neural network's output and set a certainty threshold such that a game update will not occur unless the certainty is above a given point, even if a classification is repeated multiple times with low certainty.

5. Experiments

In order to determine the performance of our networks, we ran them both on a randomly chosen test subset of our collected dataset (that does not overlap with our training or validation sets). The accuracy results are summarized below:

Network	Accuracy
CNN	95.31%
SqueezeNet	50.23%

Table 1: Network Test Accuracy

As seen by the above results, our baseline CNN classifier performed much better for this task than the pretrained SqueezeNet model with SVM. While both had accuracy significantly above the threshold for statistical significance, the CNN performed almost twice as well as the SqueezeNet classifier. We believe this large difference in performance is due to the fact that the pretrained SqueezeNet is trained using ImageNet and not games. Therefore, it likely has difficulty picking up on the features that are needed to recognize games (like UI), and instead focuses on features like the objects that appear on screen (people, vehicles, etc.). These features can not necessarily be used to categorize games since many games contain multiple classes of objects and the same kinds of objects as other games (for example, Counter-Strike and Grand Theft Auto both contain buildings, people, and weapons). However, since the CNN was trained from scratch using only our dataset, the features that it has learned to detect are specific to classifying video games, like game HUDs and text.

5.1. Pretrained SqueezeNet with SVM

Since the results of the pretrained SqueezeNet with SVM are disappointing (50%), we will focus mainly on the results of the successful CNN classifier. However, the failure of pretrained Squeezenet to classify these screenshots accurately is an interesting lesson in how CNNs need to be trained to detect the features that are relevant to the problem at hand instead of arbitrary pretrained classes that are irrelevant to the classification task that the network is being used for.

5.2. Baseline CNN

Training of our CNN was very successful, with 10 epochs taking around 20 minutes to train on Google Colab.

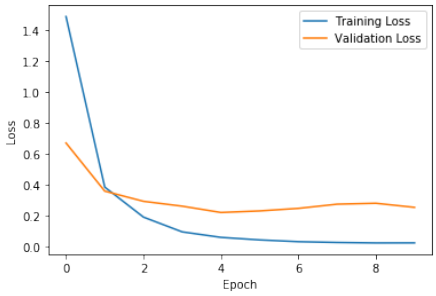


Figure 5: Training loss of CNN across epochs.

While the overall accuracy of the CNN across all classes was 95%, across the different classes the accuracy varies significantly. While it performs very well (99.3% across 64 batches) at classifying League of Legends, it struggles with Rocket League (85.3%) and Slenderman (84.4%).

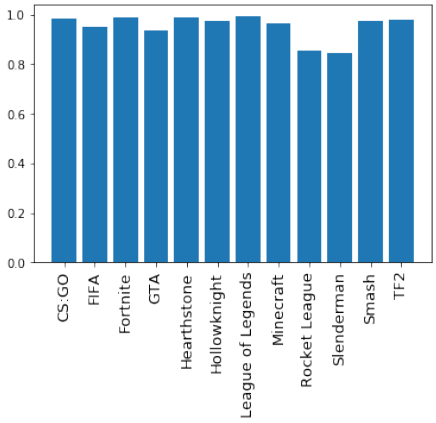


Figure 6: Accuracy of CNN across all games.

One possible reason for this discrepancy are differences in the consistency of the UI. League of Legends and Hearthstone have a very consistent UI, while Slenderman has no

real UI and Grand Theft Auto changes its UI depending on the situation (cutscenes, first person view, etc.). An interesting low performer that this difference does not explain, however, is Rocket League. Rocket League has a very distinct style and consistent UI appearance, yet the network performs poorly at categorizing it. Some possible reasons for this are that its UI could look similar to another game in screenshots, or that there are a large number of outliers (menus, loading screens, etc.) in the data.

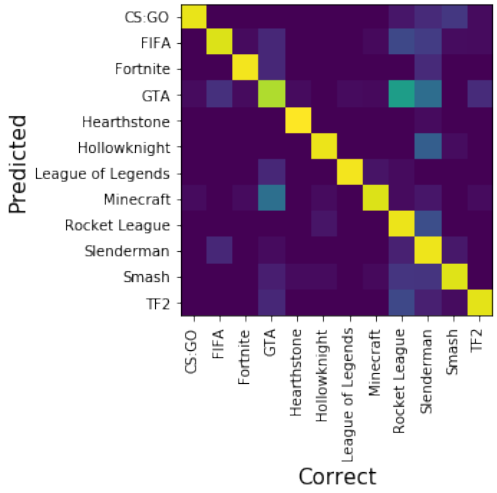
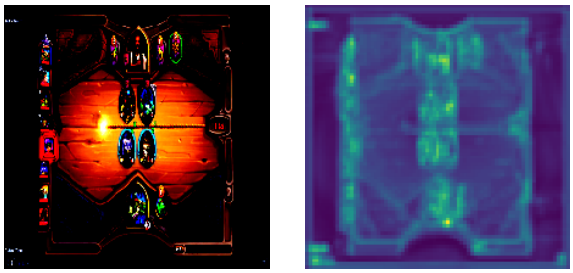


Figure 7: Heatmap of predicted vs. correct class.

It can be seen from this heatmap that while our classifier generally performs well on every class, there are a few incorrect predictions that are more common than others. The most frequent incorrectly prediction is Rocket League being classified as Grand Theft Auto. Other common mispredictions include Grand Theft Auto being classified as Minecraft, and Slenderman being classified as Grand Theft Auto or Hollowknight.

5.3. Class Activation Maps for CNN

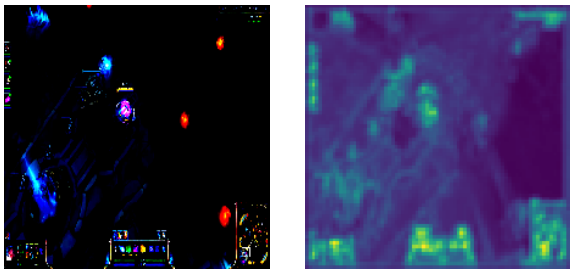
In order to better understand how our CNN classifier is identifying features in our game screenshots to base its decisions off of, we decided to generate class activation maps for the CNN. From these we can see that our CNN primarily activates on UI features and other elements such as characters that are consistent across many screenshots of a game.



(a) Input (b) Activation Map

Figure 8: Activation Map for Hearthstone

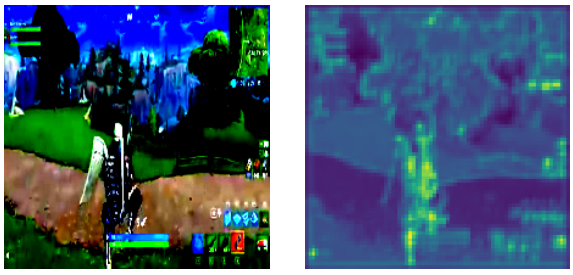
The above images are an example of a correctly identified image from Hearthstone. In the activation map, it can be clearly seen that the CNN is detecting the cards, the Hero in the lower and upper center, as well as the move history UI on the left side of the screen. The outline of the board itself is also clearly visible in the activation map.



(a) Input (b) Activation Map

Figure 9: Activation Map for League of Legends

Similarly to Hearthstone, the primary features highlighted on the activation map for League of Legends are UI features common across the game, namely the HUD and minimap at the bottom of the screen. However, it can also be seen that the player's character in the middle of the screen is clearly highlighted as well.

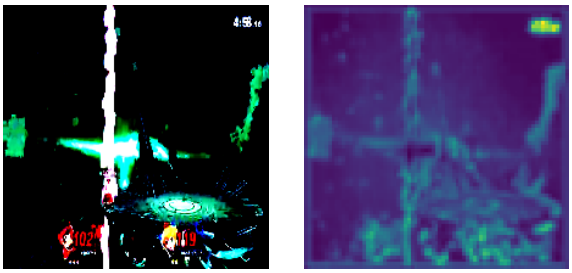


(a) Input (b) Activation Map

Figure 10: Activation Map for Fortnite

Fortnite is quite different from the other two games in terms of what is identified, in that while the UI is highlighted, the main feature that activates the CNN is actually the player character. This is likely due to the fact that Fortnite is a 3rd person game, so the player character appears consistently for most of the dataset.

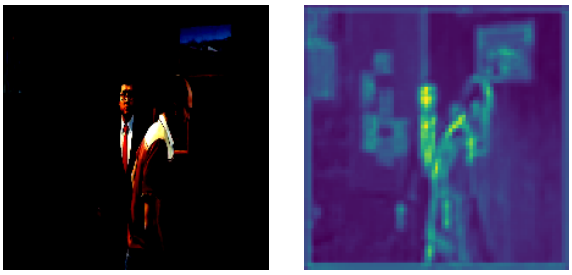
Another interesting thing that the activation maps for the CNN can do is give us some insight as to why some classifications were made incorrectly.



(a) Input (b) Activation Map

Figure 11: Activation Map for Smash

The above Smash screenshot was classified incorrectly as Hollow Knight. Interestingly, the only feature highlighted strongly in the activation map was the clock in the top right corner of the screen. This highlighted feature is similar to the UI feature typically present in the top left hand corner of Hollow Knight screenshots.



(a) Input (b) Activation Map

Figure 12: Activation Map for Grand Theft Auto

The above Grand Theft Auto screenshot was misclassified as being from Slenderman. This is a fairly common issue with some of the indoor dialogue cutscenes from Grand Theft Auto. The primary feature detected in the CNN activation map is the person in the middle of the screen, which is highlighted in the activation map very similarly to the Slenderman in Slenderman screenshots. The person in this screenshot is wearing a very similar suit to the Slenderman, and since this is a cutscene there is no HUD, so this mistaken classification is understandable.

6. Conclusions

We have shown that the task of video game classification is very much within the realm of practicality with currently available technologies. We were able to achieve a high accuracy when predicting between 12 classes of video games while using only a basic CNN trained on relatively weak hardware. On the other hand, SqueezeNet did not perform well for this task, likely due to it being pretrained on a different dataset and the SVM on top of it not being able to make up the difference.

Given the resources of a large company like Twitch, it would not be out of reach to train a larger CNN using more powerful hardware and a much larger dataset encompassing the library of games which are commonly streamed on Twitch. Additionally, we have also shown that even with a somewhat sub-optimal accuracy, it is possible to drastically reduce the false positive rate when classifying continuous streams or videos by only accepting a title update after a certain number of repeated same classifications. This makes automatic classification of gaming streams robust and shows that automated systems utilizing the techniques we outlined are deployable in the near future.

References

[1] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and ¡ 0.5MB model size. arXiv:1602.07360v4, 2016.

[2] Joshy. Playing Fortnite Battle Royale For 10 Hours Straight. Apr 2018. URL: https://www.youtube.com/watch?v=7nc5cjHC67M

[3] ViperSniper999. Fifa 20 basically the WHOLE 10 HOURS stream part 1. Sep 2019. URL: https://www.youtube.com/watch?v=F5K_DRK7TgE

[4] TidesofTime. PLAYING EU TODAY 8.1k mmr #1 world !leaderboards. Nov 2019. URL: https://www.twitch.tv/videos/509053546

[5] AbnormalPersona_. FIGHT ME !arena. Nov 2019. URL: https://www.twitch.tv/videos/509180302