
# THE SECRET OF MAGIC - VR GAME

AMIT SHUSTER
MICHAL GUTTMANN

# TABLE OF CONTENTS

# INTRODUCTION

We developed a VR game that uses a convolutional neural network (CNN) to identify commands from the user, drawn using the controller. The game was inspired by Harry Potter and the Philosopher's Stone (2001).

The main goal of our project was to involve a Deep Learning mechanism in the VR development environment. After a lot of research, we decided to build a CNN by ourselves and use it in our game.

Our Game simulates several waves in which the player encounters different numbers of zombies that are trying to reach him and defeat him. The Players' goal is to defeat the zombies by using magic spells before they defeat him. To activate the magics, the player chooses a 'spell' and draws the glyph with his controller. Then, the game preprocesses the image and sends it into a pre-trained CNN model that identifies, with 90.196% accuracy, the glyph. The game gets from the model the classification and activates the chosen magic in the pointed direction.

# SYSTEM

We developed a game that runs on PC and is played with VR gear, targeted specifically for the HTC Vive.

The game was developed using Unity game engine, scripted with C# in Visual Studio.

Microsoft Visual Studio is an integrated development environment (IDE) from Microsoft. It is used to develop computer programs, as well as websites, web apps, web services and mobile apps. Visual Studio uses Microsoft software development platforms.

Unity is a cross-platform game engine developed by Unity Technologies.
Unity gives users the ability to create games in both 2D and 3D. The engine offers a primary scripting API in C#, for both Unity editor in the form of plugins and games themselves, as well as drag and drop functionality.

The HTC Vive is a virtual reality headset developed by HTC and Valve Corporation. The headset uses 'room scale' tracking technology, allowing the user to move in 3D spaces and use motion-tracked handheld controllers to interact with the environment.

Equipment required:

1. HTC-Vive Headset
2. HTC-Vive controller



Relevant links:

https://unity3d.com/

https://www.vive.com/us/product/vive-virtual-reality-system/

## GAME OVERVIEW

The game occurs in one scene, this scene is a 360° arena where the player encounters enemies from any direction. The game includes waves of undead enemies, in which the player needs to kill the enemies. To do so, the player should use his magic spells, which can be triggered by drawing one of these glyphs:

Each glyph will trigger a different spell, which will kill the undead enemy on hit. The enemy waves start from easy difficulty to hard. On the first wave there is a standing undead enemy, follow by two standing undead enemies, on the third wave the zombies start walking toward the player, when the enemies reach a defined radius, they will start attacking. On the fourth wave the sun sets and the game continues in dark environment.


The higher the wave the more undead enemies that spawn, from more directions, forcing the player to search for enemies all around him. For each enemy the player kills, he will receive score points. When the game is over the high score will be updated if the player achieved more points than the current score. If an enemy was able to get to the player, the game will be over, and all the current undead enemies will be destroyed.

# CNN DEVELOPMENT PROCESS

The main goal of our project was to involve a Deep Learning mechanism in the VR development environment. We decided to use a CNN (Convolutional Neural Network) to identify the players' gestures and translate them to magics that will help the player to win the game. The use of the CNN in the game is through a function that identifies when the player draws a gesture with his controller in the 3D environment, converts the 3D image to a 2D image by projecting them on the players' view plane, and calls a python script that returns the desired magic of the player.

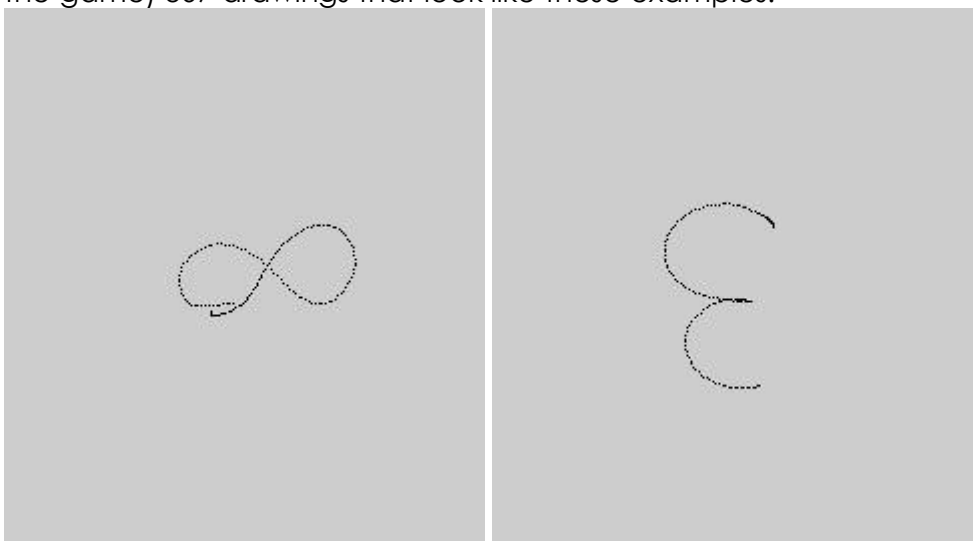
**CNN Development:**
At the beginning we wanted to use a trained model from an online open source. After investigating a few, including implementing then and running tests, we chose to build a CNN by ourselves. We had two main candidates, which were built to identify hundreds of drawings of symbols, signs and objects (while we wanted to identify 8 symbols). We got medium results – around 70% accuracy and therefore started to implement out CNN.

**Our CNN architecture:**
The network is built of:

- Two convolution layers
- 4 FC layers.
- ReLU & MaxPool activation

The input dimension is the dimension of each image – 3x32x32 and the output dimension is the digit prediction vector – 1x8. The number of parameters in the network is – 38,523,912.

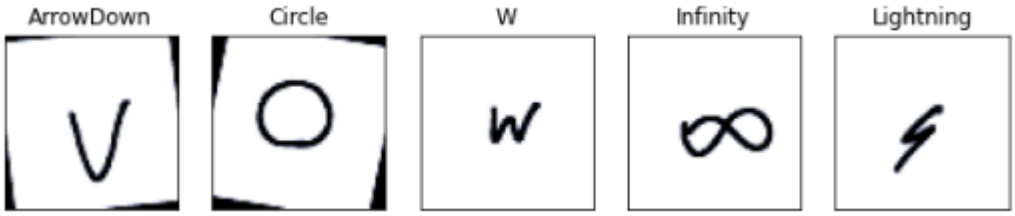
**The Classifier Training:**
In order to train our model, we recorded through our game in unity (as it is used in the game) 307 drawings that look like these examples:



Then, we started the training process that included few steps:
Preprocess the images:

- Load the recorded images and give them their label (from image name)
- Split into train and test set – 256 for the train and 51 for the test
- Strengthen the symbol by dilating the drawing
- Normalize the images
- Expand the train set by augmenting the 256 images with scaling and rotating – the train set grew to 512 images

Example of five train images after preprocessing:




Classifier Training:
To find the optimal hyper parameters we tested few parameters:

```
# Hyper Parameters
epochs = 25
batchSizeHList = [32, 64, 128]
learningRateHList = [1e-3, 1e-4, 1e-5]
```

and got the best results for these parameters:

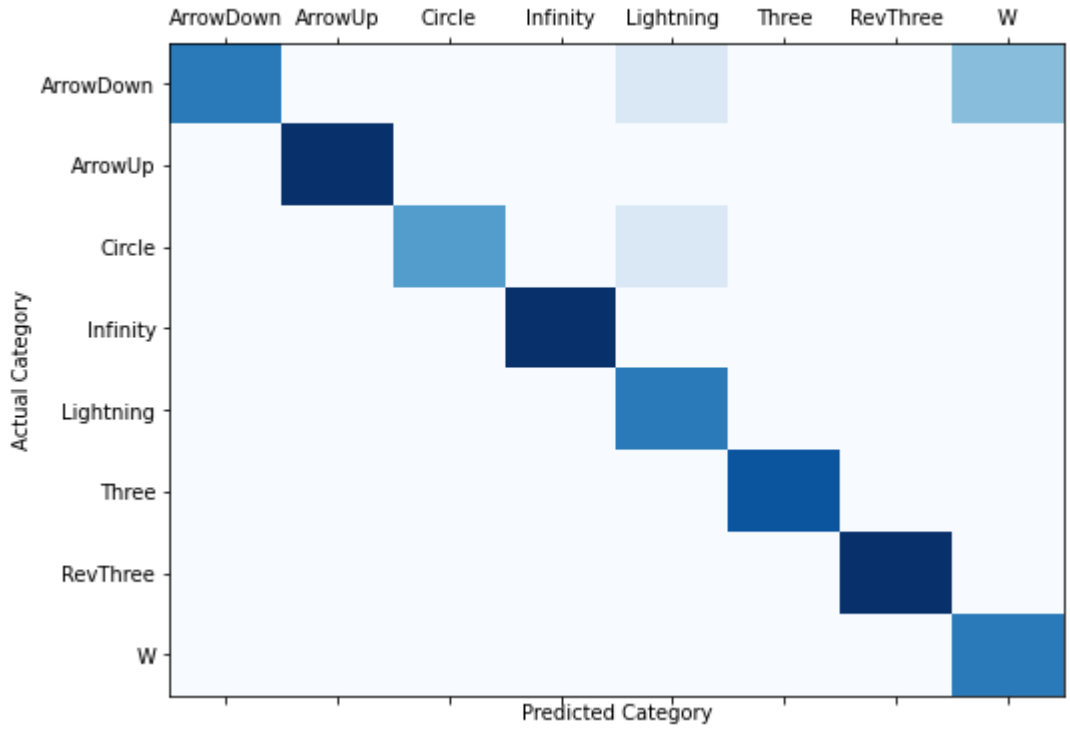
```
# best hyper parameters
batch_size = 32
learning_rate = 1e-4
epochs = 22
```

Training process data:

```
Epoch: 1 | Loss: 0.0647 | Training accuracy: 28.516% | Test accuracy: 25.490% |Epoch Time: 8.36 secs
Epoch: 2 | Loss: 0.0553 | Training accuracy: 61.719% | Test accuracy: 54.902% |Epoch Time: 8.39 secs
Epoch: 3 | Loss: 0.0378 | Training accuracy: 68.750% | Test accuracy: 68.627% |Epoch Time: 8.21 secs
Epoch: 4 | Loss: 0.0236 | Training accuracy: 84.766% | Test accuracy: 80.392% |Epoch Time: 8.26 secs
Epoch: 5 | Loss: 0.0153 | Training accuracy: 92.188% | Test accuracy: 84.314% |Epoch Time: 8.26 secs
Epoch: 6 | Loss: 0.0089 | Training accuracy: 95.117% | Test accuracy: 84.314% |Epoch Time: 8.35 secs
Epoch: 7 | Loss: 0.0055 | Training accuracy: 94.531% | Test accuracy: 86.275% |Epoch Time: 8.25 secs
Epoch: 8 | Loss: 0.0044 | Training accuracy: 90.039% | Test accuracy: 80.392% |Epoch Time: 8.26 secs
Epoch: 9 | Loss: 0.0041 | Training accuracy: 93.359% | Test accuracy: 80.392% |Epoch Time: 8.29 secs
Epoch: 10 | Loss: 0.0037 | Training accuracy: 99.023% | Test accuracy: 86.275% |Epoch Time: 8.32 secs
Epoch: 11 | Loss: 0.0030 | Training accuracy: 97.266% | Test accuracy: 82.353% |Epoch Time: 8.30 secs
Epoch: 12 | Loss: 0.0016 | Training accuracy: 100.000% | Test accuracy: 92.157% |Epoch Time: 8.19 secs
Epoch: 13 | Loss: 0.0005 | Training accuracy: 100.000% | Test accuracy: 92.157% |Epoch Time: 8.25 secs
Epoch: 14 | Loss: 0.0003 | Training accuracy: 100.000% | Test accuracy: 92.157% |Epoch Time: 8.25 secs
Epoch: 15 | Loss: 0.0002 | Training accuracy: 100.000% | Test accuracy: 92.157% |Epoch Time: 8.22 secs
Epoch: 16 | Loss: 0.0002 | Training accuracy: 100.000% | Test accuracy: 90.196% |Epoch Time: 8.19 secs
Epoch: 17 | Loss: 0.0001 | Training accuracy: 100.000% | Test accuracy: 92.157% |Epoch Time: 8.24 secs
Epoch: 18 | Loss: 0.0001 | Training accuracy: 100.000% | Test accuracy: 90.196% |Epoch Time: 8.33 secs
Epoch: 19 | Loss: 0.0001 | Training accuracy: 100.000% | Test accuracy: 92.157% |Epoch Time: 8.29 secs
Epoch: 20 | Loss: 0.0001 | Training accuracy: 100.000% | Test accuracy: 90.196% |Epoch Time: 8.28 secs
Epoch: 21 | Loss: 0.0001 | Training accuracy: 100.000% | Test accuracy: 92.157% |Epoch Time: 8.26 secs
Epoch: 22 | Loss: 0.0001 | Training accuracy: 100.000% | Test accuracy: 90.196% |Epoch Time: 8.30 secs
==> Finished Training ...
```

To tune the model, we used the augmented train set we mentioned in the preprocess section. The final accuracy we got for the mentioned hyper parameters is – **90.196**% and here is a demonstration graph of our results:

```
test accuracy: 90.196%
```



# GAME DEVELOPMENT PROCESS

For the game development, we used Unity and SteamVR. We used multiple assets including: AirSig which we based our data collecting on, Realistic Effects Pack v4 for the spell's effects, and Heroic fantasy creatures full pack where the undead enemy is taken from.

Image samples collection for CNN Training:
To collect the samples, we implemented a system that draws particles following the controller, in which the player clicks and holds the trigger button to draw the desired glyph. Then, we take the collected 3d vectors and transform them from world coordinates to screen coordinates, save it in a jpg file and send it to the neural network to process. The neural network python process is run in a different thread making the game smooth without having to wait for the result, during this time, the player's controller will glow in purple and the player will see a thin laser beam which helps him aim the spells.

The main game components are:

- Effects manager – which will manage all the active spells, their instantiate, lifetime timer, and destruction.
- Gestures manager – collects the data (which is drawn by the player). Once new data is available, it creates an image by transferring the world to screen point, setting the relevant pixel on a smaller Texture2D, and encodes it to JPG. This manager is also the one that starts the python process on another thread.
- Enemy behavior – controls the enemy animator which change the animation according to the radius from the player, rotates the enemy to the players location, kill the enemy on collision with a spell, start a co-routine to kill the enemy after the death animation is done, and adds the score for the enemy kill.
- Game manager – controls the sun and light effects, switch between day and night, updates the wave text and spawn the enemies for the current wave, starts the next wave when the current one is finished and controls the game over behavior.

Main challenges:

- Transfer coordinates from world to screen
- connect python to unity – including running different process for the classification