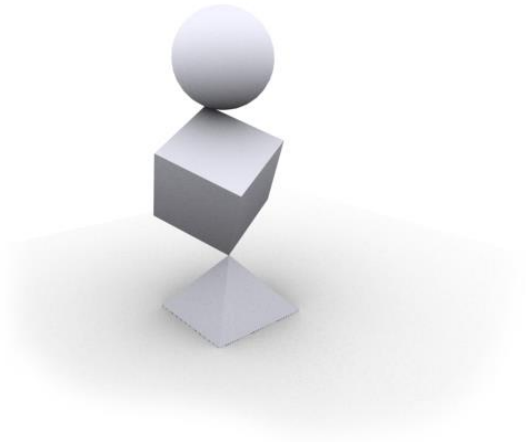


Technical Design Document

Wenzheng Huang

Version 0.3

Position Title	Name	Signature
Programmer	Wenzheng Huang	

Document Revisions Table

Version	Description	Requestor	Date
0.1	*Initial Document*	*Corey Clark*	*04/02*
0.2	*Add Theory Section*	*Corey Clark*	*04/09*
0.3	*Detailed writeup on Cascade Classifier, showing process and math used in technique. Show analysis of classifiers working on different backgrounds (i.e. different Street Fighter 5 levels).*	*Corey Clark*	*04/23*

Table of Contents

Table of Figures 5
Project and Motivation 6
End Product 6
Deliverables 7
System Requirements 8
Target System 8
Minimum 8
Recommended 8
Development System 8
Resource Budget 9
CPU Execution Time Estimate 9
Memory Utilization Estimate 9
Assets Budgets Estimate 9
Art Assets Budgets Estimate 9
Technology Sources 10
Components 11
Component Descriptions and Risk Assessments 11
Engine: 11
Screen Captor: 11
Command Sequence Analyzer: 11
Command Analyzer: 11
Classifier: 11
Input Injector: 11
Software Configuration Management 12
Naming Convention 12
Category/File Formats 12
Coding Standards 12
Tutorials 13
How A Cascade Classifier Works 13
Feature Types for Matching 13
Haar-like Features 13

Local Binary Patterns 14
Finding the Critical Features 15
Config Build Environment 15
Expected File Structure 16
Train a Classifier for Detecting a Pose 17
Prepare Positive Samples 17
Prepare Negative Samples 18
Start Training Process 18
Test and Run 19
Result 19
Reference 21

Table of Figures

Figure 1 - Component Structures 11
Figure 2 - Features for Testing at Each Stage 13
Figure 3 - Summed-area Tables 14
Figure 4 - Local Binary Patterns 15
Figure 5 - File Structure of the System 16
Figure 6 - Test Results in *The Grid* 19
Figure 7 - Test Results in *Hillside Plaza* 20
Figure 8 - Test Results in *Bustling Side Street* 20

Instruction: Generate an automatic Table of Figures. Correct for any bookmark errors and confirm that the CNTRL-click navigates properly.

Project and Motivation

Machine Learning has become more and more popular among game researchers, but researchers rely heavily on developer API, memory hacking or reverse engineering to gain access to game status. Blizzard Entertainment released the StarCraft II API in 2017 that provided full access to game input/output. Many games do not allow such access to source or development/research APIs, and reverse engineering or memory-hacking the game is strictly prohibited by the end user license agreements. This limited access prevents researchers from access and gaining information from games that is required for training data generation as well as standard data science methodologies. This poster introduces an approach that allows researchers to get client data by analyzing prerecorded game footage in either video or image formats using computer vision techniques provided by OpenCV[1]. This system extracts the timeline of the user inputs required to generate character poses in replay video footages.

End Product

The analyzer outputs a time and user input for each classifier detection during playback of game footage.

Deliverables

Format and checklist of all deliverables such as .cpp, .dll, libraries, samples, .exe, etc.

Table 1 - Table of Deliverables

Asset	Description
Engine.dll	Visualization function library
Classifier.hpp	Cascade classifier functionality
Classifier.cpp	Cascade classifier functionality
CommandClassifier.hpp	Analyze an image and find the pose that matches the image best
CommandClassifier.cpp	Analyze an image and find the pose that matches the image best
CommandSequence.hpp	Analyze an image sequence and output the input timeline
CommandSequence.cpp	Analyze an image sequence and output the input timeline
CV.hpp	OpenCV utilities
CV.cpp	OpenCV utilities
CommandInjector.hpp	Input injecting functionality
CommandInjector.cpp	Input injecting functionality
Game.hpp	Main workflow
Game.cpp	Main workflow
GameConfig.hpp	Configuration handling
GameConfig.cpp	Configuration handling
main.cpp	Application entry point
cascade_HK.xml	Cascade classifier for Hard Kick
cascade_MP.xml	Cascade classifier for Medium Punch
commands.xml	Parameter setting for cascade classifiers
cvTest.xml	Parameter setting for source samples
CV.exe	Built application executable

System Requirements

Target System

Minimum

Processor: Intel(R) Core(TM) i7-6820HK CPU @ 2.70GHz
Memory: 2GB
Video Card: NVIDIA GeForce GTX 980M
Operating System: Windows 10
Software: DirectX 11
Peripheral: Monitor

Recommended

Processor: Intel(R) Core(TM) i7-6820HK CPU @ 2.70GHz
Memory: 8GB
Video Card: NVIDIA GeForce GTX 980
Operating System: Windows 10
Software: DirectX 11
Peripheral: Monitor

Development System

Processor: Intel(R) Core(TM) i7-6820HK CPU @ 2.70GHz
Memory: 32GB
Video Card: NVIDIA GeForce GTX 980M
Operating System: Windows 10
Software: DirectX 11
Peripheral: Monitor

Resource Budget

CPU Execution Time Estimate

15ms per classifier per image in release build

Memory Utilization Estimate

300MB

Assets Budgets Estimate

Maximum value for visible polygons: 100

Art Assets Budgets Estimate

*Maximum image size: 1280 * 720*

Technology Sources

Table 2 - Table of Technology Sources

Name	***Description***
Visual Studio 2015	*Provide coding environment*
DirectX 11	*Provide visualization methods*
OpenCV library	*Provide cascade classifier functionalities*
Adobe After Effects CC	*Used to process training samples*
Adobe Media Encoder CC	*Used to serialize video into image sequence*
Windows 10 Game DVR	*Used to capture game footages*

Components

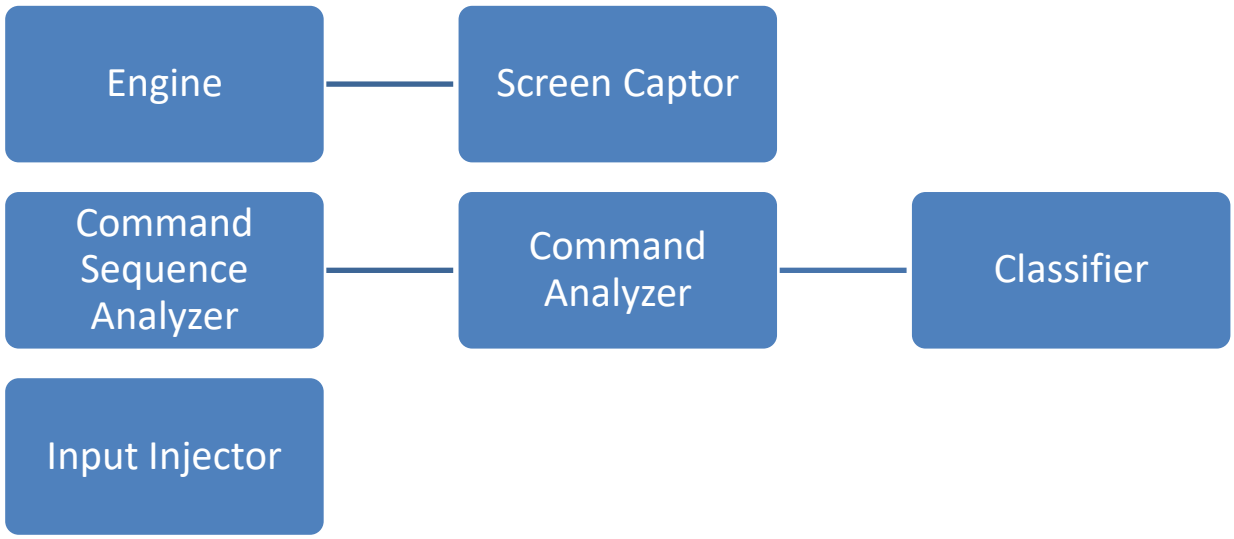


Figure 1 - Component Structures

Component Descriptions and Risk Assessments

Engine:

Description: *Provide basic game workflow and visualization.*

Screen Captor:

Description: *Capture the screen for testing classifiers in real time.*

Command Sequence Analyzer:

Description: *Analyze image sequence and give a timeline of input.*

Command Analyzer:

Description: *Analyze a single image with multiple classifiers and find the best classification result.*

Classifier:

Description: *Analyze a single image and give the best match.*

Input Injector:

Description: *Use results from Command Sequence Analyzer to reproduce input in the game in real time.*

Software Configuration Management

Naming Convention

Category/File Formats

Table 3 - Table of Naming Conventions

Category	***File Formats***
Code file	*[ClassName].cpp*
Cascade classifier data	*cascade_[ActionName].xml*
Test image sequence	*[testname]_[index].png*

Coding Standards

All code adheres as strictly as possible to the UnrealScript coding standards. This includes naming conventions and class structures.

Tutorials

How A Cascade Classifier Works

A cascade classifier[2] is a machine learning module that contains multiple weak classifiers in stages. Each weak classifier uses some critical image features to test the window and reject any false result. As the stage goes higher, the classifier conducts tests with more detailed features. The cascade classifier returns a positive result when the test window passes all stages.

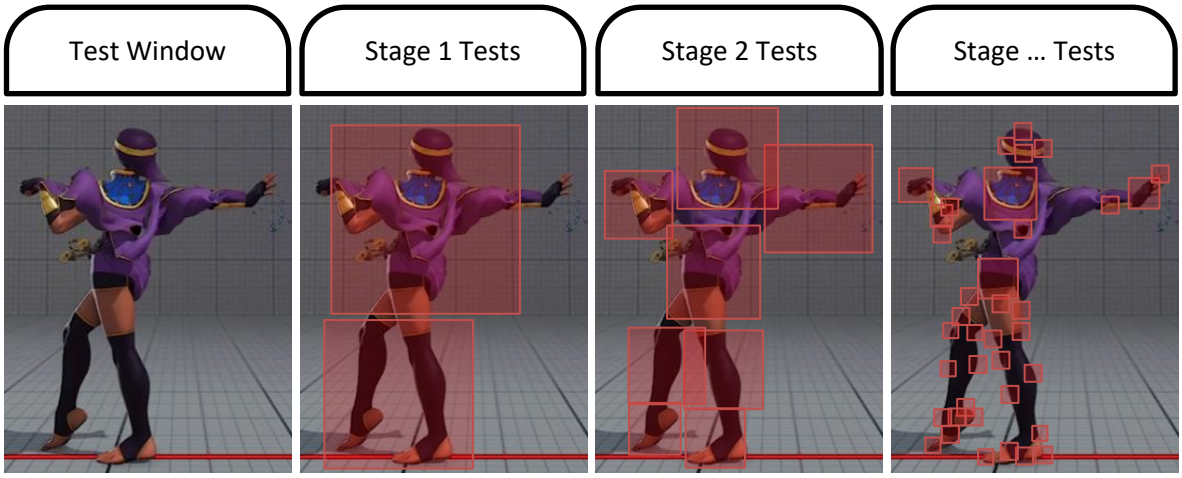


Figure 2 - Features for Testing at Each Stage

Feature Types for Matching

There are two feature types can be used when testing a window: Haar-like features and Local Binary Patterns[3]. In short, Haar-like features is more accurate than Local Binary Patterns but consumes much more time to evaluate.

Haar-like Features

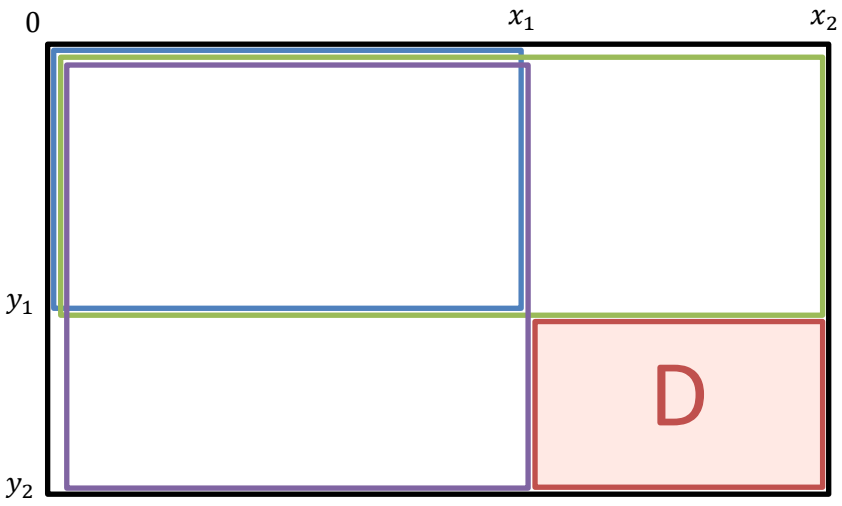
Haar-like features(HAAR) compare the sum of pixels (usually the intensities of the pixels) of adjacent rectangle regions. To compute Haar-like features fast, it uses summed-area tables that each cell at (x, y) stores the sum of pixels in the region from $(0,0)$ to (x, y):

$$I(x,y) = \sum_{\substack{x' \le x \\ y' \le y}} i(x', y')$$

where $i(x', y')$ is the value of the pixel at (x', y').

Then the sum of pixels in a rectangle region D from (x_1, y_1) to (x_2, y_2) equals:

$$I(D) = \sum_{\substack{x_1 < x' \le x_2 \\ y_1 < y' \le y_2}} i(x', y') = I(x_2, y_2) + I(x_1, y_1) - I(x_1, y_2) - I(x_2, y_1)$$



$$I(D) = I(Black\ Area) + I(Blue\ Area) - I(Green\ Area) - I(Purple\ Area)$$

Figure 3 - Summed-area Tables

Local Binary Patterns

Local Binary Patterns(LBP) are feature vectors that are calculated with the process below:

1. Divide the window into cells
2. For each pixel in a cell, compare its value to neighbor pixels clockwise or counterclockwise
3. Return 1 If a neighbor pixel is greater than the center pixel, or 0 if not. Collect all 8 values and compose into a binary number.
4. Compute a histogram of the frequency of each binary number in the cell.
5. Concatenate histograms of all cells.

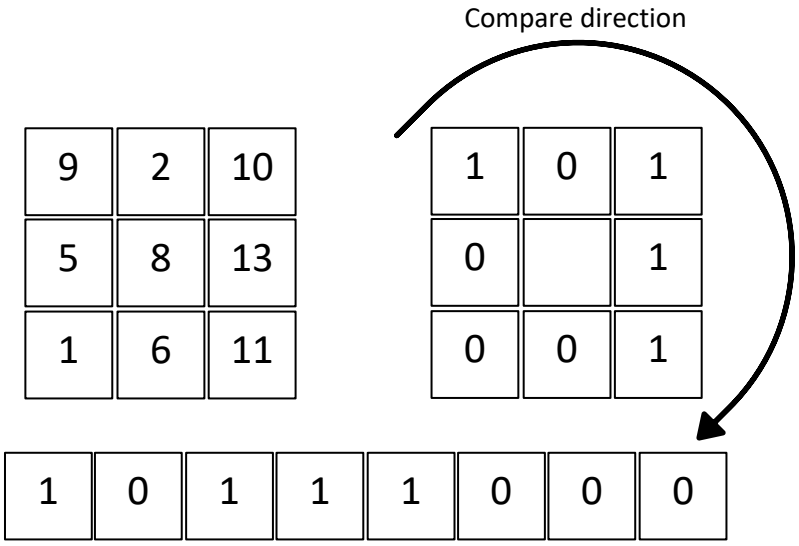


Figure 4 - Local Binary Patterns

Finding the Critical Features

Since the number of features in a window is far larger than the that of pixels, it is too expensive to test the window with all features. It is important only to pick some of the features that best separate between two classes so that the classifier can be both accurate and efficient. To achieve it, a variant of AdaBoost[4] algorithm was used in the training process. The algorithm is as follows:

1. Collect a large number of sample images $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$, where $y_i = \begin{cases} 1, & if\ x_i\ is\ positive\ sample \\ 0, & otherwise \end{cases}$
2. Initialize weights $w_{1,i} = \begin{cases} 1/2l, if\ x_i\ is\ positive\ sample \\ 1/2m, otherwise \end{cases}$, where l is the number of positive samples, m is the number of negative samples
3. For $t = 1,2, \dots, T$:
    1. Normalize the weights, $w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$
    2. For each feature j, train a classifier h_j using only this feature and evaluate the error regarding to w_i: $e_j = \sum_i w_i \left| h_j(x_i) - y_i \right|$
    3. Choose the classifier h_t with the lowest error e_t
    4. Update the weights $w_{t+1,i} = \begin{cases} w_{t,i}, if\ x_i\ is\ classified\ incorrectly \\ w_{t,i} \frac{e_t}{1-e_t}, otherwise \end{cases}$
4. Form the cascade classifier as $h(x) = \begin{cases} 1, if \sum_{t=1}^{T} a_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} a_t \\ 0, otherwise \end{cases}$, where $a_t = \log \frac{1-e_t}{e_t}$

Config Build Environment

1. Download and build/decompress [OpenCV Library 3.3.0]
2. Add OPENCV_DIR into environment variables with value ***[your opencv root folder]*** (Example: ***C:\Program Files\opencv***)

3. Add ***[your opencv root folder]\build\x64\vc14\bin*** into environment variable PATH (Example: ***C:\Program Files\opencv\build\x64\vc14\bin***)
4. Download and install Visual Studio 2015 with C++ and DirectX 11 package

Expected File Structure

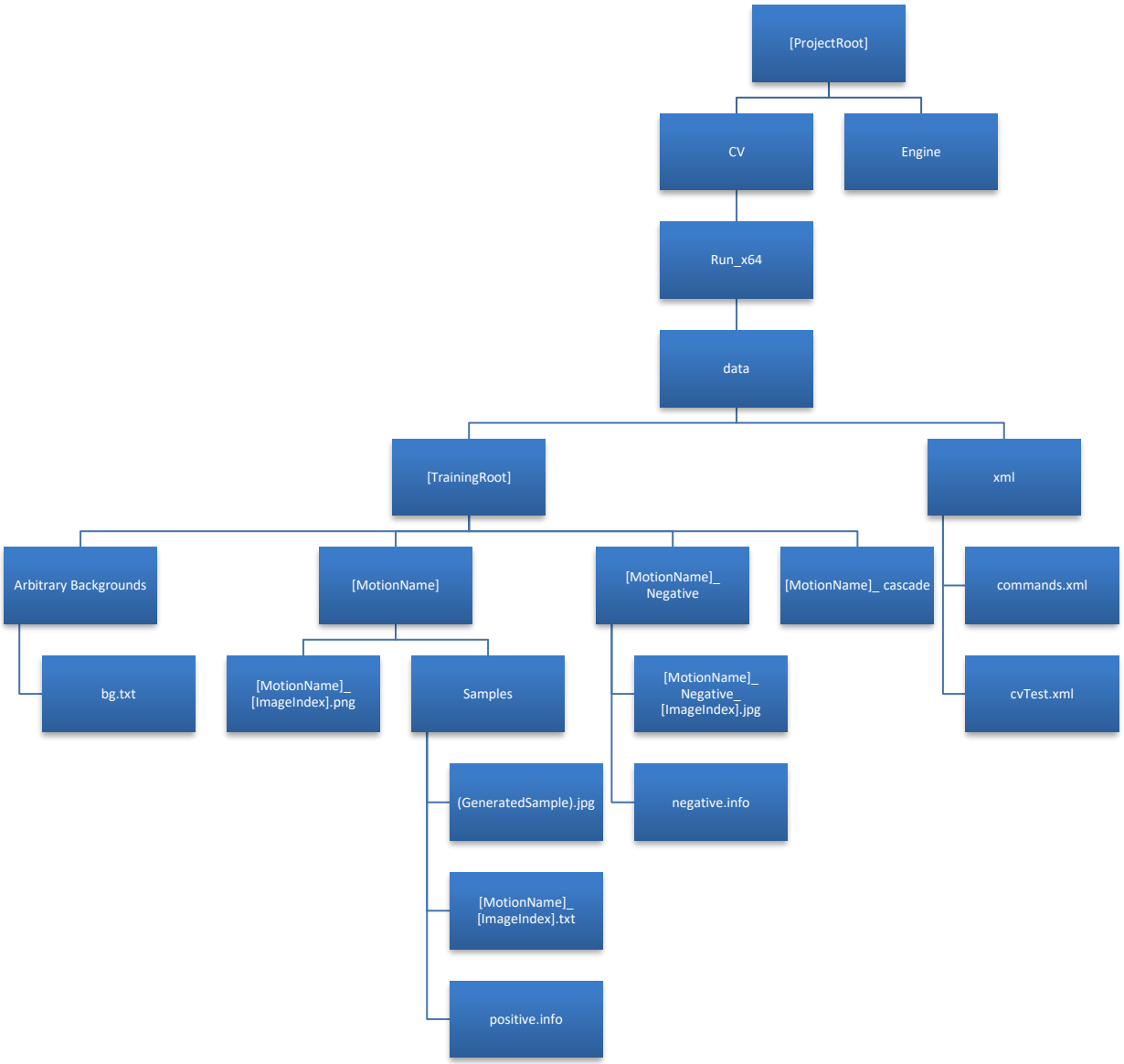


Figure 5 - File Structure of the System

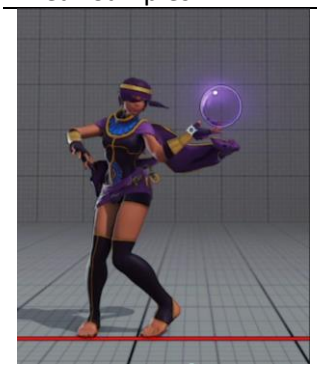
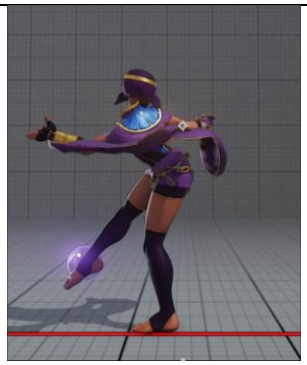
Train a Classifier for Detecting a Pose

This system uses cascade classifier trained with opencv_traincascade application[5]. Below are steps to start training.

Prepare Positive Samples

1. Capture game footage of target motion with Windows 10 Game DVR or Fraps
2. Cut the frames which best separate the pose with others with Adobe After Effects CC. (Example: for a kick action, it is safe only to segment the frames when the leg is completely stretched, not the frames when the character is transforming from the previous pose.)

Table 4 - Picking the Segment of Target Pose

Weak Samples		Good Samples
		
Her left arm may look like the idle pose	Good uniqueness, but it only lasts two frames	Last long enough (5 frames) and great uniqueness

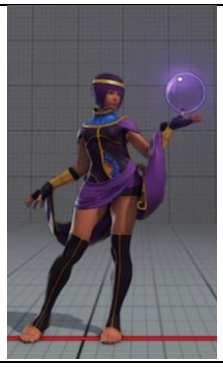
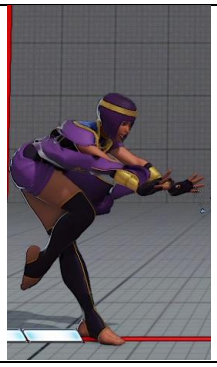
3. Note the delay from the first frame of target motion to the segment
4. Key the character from the background with Adobe After Effects
5. Output the footage into image sequence to ***[TrainingRoot]/[MotionName]/[MotionName]_[ImageIndex].jpg***
6. To multiply the number of positive samples and improve accuracy, use opencv_createsamples application to fill each image into arbitrary backgrounds:
    a. Collect thousands of random images at ***[TrainingRoot]/ArbitraryBackgrounds***
    b. Open Command Prompt at ***[TrainingRoot]/ArbitraryBackgrounds***
    c. Use Command Prompt command ***for %f in (*.jpg) do type "%f" >> bg.txt*** to collect file names info
    d. Open Windows PowerShell at ***[TrainingRoot]***
    e. For each image, call command ***opencv_createsamples -img [MotionName]/[MotionName]_[ImageIndex].jpg -bg ArbitraryBackgrounds/bg.txt -info [MotionName]/Samples/[MotionName]_[ImageIndex].txt -num 20 -maxxangle 0 -maxyangle 0 -maxzangle 0 -maxscale 1 -bgcolor 0 -bgthresh 1 -w [TrainingWindowWidth] -h [TrainingWindowHeight]***
7. Open Command Prompt at ***[TrainingRoot]/[MotionName]/Samples***

8. Use Command Prompt command ***for %f in (*.txt) do type "%f" >> positive.info*** to merge all info files into a single info file

Prepare Negative Samples

1. Capture game footage of all motion of the character except the target motion

Table 5 - Prepare Negative Samples

Target Positive Sample	Negative Samples			
				
	All other poses excluding the target poses			Opponent and UI Elements

2. Output the footage into image sequence with Adobe Media Encoder or Adobe After Effects to ***[TrainingRoot]/[MotionName]_Negative/[MotionName]_Negative_[ImageIndex].jpg***
3. Open Command Prompt at ***[TrainingRoot]/[MotionName]_Negative***
4. Use Command Prompt command ***dir /b "" > negative.info*** to collect file name into an info file

Start Training Process

1. Open Windows PowerShell at ***[TrainingRoot]***
2. Call ***opencv_createsamples -info [MotionName]/Samples/positive.info -num [NumberOfPositiveSampleImages] -w [TrainingWindowWidth] -h [TrainingWindowHeight] -vec [MotionName].vec*** to generate a vector file
3. Call ***opencv_traincascade -data [MotionName]_cascade -vec [MotionName].vec -bg [MotionName]_Negative/negative.info -numPos [NumberOfPositiveSamplesUsedPerStage] -numNeg [NumberOfNegativeSamplesUsedPerStage] -numStages [NumberOfStages] -w [TrainingWindowWidth] -h [TrainingWindowHeight] -numThreads [MaxNumberOfThreads] -featureType LBP -acceptanceRatioBreakValue 0.00001 -maxFalseAlarmRate 0.1*** to start training
4. Result cascade classifier is located at ***[TrainingRoot]/[MotionName]_cascade/cascade.xml***

Test and Run

1. Capture source footage from the game
2. Output the footage to PNG sequence with alpha channel
3. Open ***data/cvTest.xml*** and edit analyze parameters
4. Open ***data/commands.xml*** and edit classifier parameters
5. Build the project with build configuration ***OpenCV Release***
6. Run the project
7. The result will be output to ***Run_x64/resolvedResult.txt***

Result

We trained two classifiers: a classifier for Menat's Standing Medium Punch and a classifier for Menat's Standing Hard Kick, both with samples collected from *The Grid* Level. Three tests were run in three different levels: *The Grid*, *Hillside Plaza*, and *Bustling Side Street*:

Figure 6 - Test Results in *The Grid*

Table 6 - Test Results in *The Grid*

The Grid	**Time Format [minutes : seconds : milliseconds]**	
Source Timeline	Analyzed Results	Delay
[00:01:650] Medium_Punch	[00:01:650] Medium_Punch	0 ms
[00:02:850] Hard_Kick	[00:02:889] Hard_Kick	+39 ms
[00:05:617] Medium_Punch	[00:05:700] Medium_Punch	+83 ms
[00:06:217] Hard_Kick	[00:06:223] Hard_Kick	+6 ms
[00:08:433] Hard_Kick	[00:08:473] Hard_Kick	+40 ms
[00:09:883] Medium_Punch	[00:09:883] Medium_Punch	0 ms

Figure 7 - Test Results in *Hillside Plaza*

Table 7 - Test Results in *Hillside Plaza*

Hillside Plaza	**Time Format [minutes : seconds : milliseconds]**	
Source Timeline	Analyzed Results	Delay
[00:01:550] Medium_Punch	[00:01:650] Medium_Punch	+100 ms
[00:02:500] Hard_Kick	[00:02:506] Hard_Kick	+6 ms
[00:03:716] Medium_Punch	[00:03:683] Medium_Punch	-33 ms
[00:04:683] Hard_Kick	[00:04:690] Hard_Kick	+7 ms
[00:05:816] Medium_Punch	[00:05:849] Hard_Kick	+33 ms

Figure 8 - Test Results in *Bustling Side Street*

Table 8 - Test Results in *Bustling Side Street*

Bustling Side Street	**Time Format [minutes : seconds : milliseconds]**	
Source Timeline	Analyzed Results	Delay
[00:01:467] Medium_Punch	Missed	-
[00:02:533] Hard_Kick	Missed	-
[00:03:367] Hard_Kick	Missed	-
[00:04:017] Medium_Punch	Missed	-
[00:04:533] Medium_Punch	Missed	-
[00:05:067] Hard_Kick	Missed	-
[00:05:716] Hard_Kick	Missed	-
[00:07:167] Medium_Punch	[00:07:300] Medium_Punch	+133 ms

	[00:08:149] Medium_Punch (False Positive)	-
[00:08:567] Hard_Kick	Missed	-
[00:09:217] Medium_Punch	Missed	-

As the results shown above, with only training samples from *The Grid*, the classifiers were still working correctly in both *The Grid* and *Hillside Plaza,* but nearly useless in *Bustling Side Street*. It was because the lighting condition (light direction, ambient light color) in *Hillside Plaza* was similar to *The Grid*, while the lighting in *Bustling Side Street* had a significant difference. It is recommended to supply sample data at every level the classifiers would put in use.

Reference

[1] B. G., "The OpenCV Library," 2008.
[2] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," 2001, vol. 1, p. I-511-I-518.
[3] S. Liao, X. Zhu, Z. Lei, L. Zhang, and S. Z. Li, "Learning Multi-scale Block Local Binary Patterns for Face Recognition," in *Advances in Biometrics*, vol. 4642, S.-W. Lee and S. Z. Li, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 828–837.
[4] Y. Freund and R. E. Schapire, "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting," *J. Comput. Syst. Sci.*, vol. 55, no. 1, pp. 119–139, Aug. 1997.
[5] "OpenCV: Cascade Classifier Training." [Online]. Available: https://docs.opencv.org/3.3.0/dc/d88/tutorial_traincascade.html. [Accessed: 29-Apr-2018].