

# ディープラーニングによるゲームエンジン解析

PM コース 矢吹研究室 1442045 川辺明俊

## 1. 研究の背景

現在のゲーム業界ではどのゲームエンジンを使用しているかは、基本的に公開されていない。オープンソースのゲームエンジンであっても製作者側に使用ゲームエンジン情報を公開する義務はなく、一般人からはどのエンジンを使用しているのかは、知るすべがない。もしゲームエンジンの種類や数を詳しく知ろうとしても、そのようなデータは存在しない。

そこでディープラーニングを使用して、すでにゲームエンジンが特定されているゲームの画像情報を学習させることによって、どのゲームエンジンを使用しているか特定できるのではと考えた。

ディープラーニング（深層学習）とはニューラルネットワークの一技術であり、ニューラルネットワークはニューロンと呼ばれる脳神経を模した単位を連結されたネットワーク状のグラフである。

ディープラーニングの最も得意な分野は、画像データや波形データのような記号にできないデータの中のパターン認識だ。入力層から画像を入力し、段階的に学習していく。一般的によく使われているニューラルネットワークの構造は、各層をそれぞれすべて繋いでしまうパーセプトロン型だが、画像認識の場合、特殊なつなぎ型をすると比較的うまくいくことが知られている。それを畳み込ネットワーク（コンボリュショナルニューラルネットワーク）と言う [1]。

今回使用するディープラーニングのフレームワークは TensorFlow と呼ばれるものだ。

TensorFlow とは、Google が開発しオープンソースとして公開した人工知能のソフトウェアライブラリである [2]。主にできることは下記の通りである。

- 顔認識
- 画像検索
- リアルタイム翻訳

ディープラーニングのフレームワークとして TensorFlow を使用とする理由は、Qiita（プログラマの技術情報共有サービス）での記事の数が Chainer や Caffe, Theano に比べて圧倒的に多かったからである。

## 2. 研究の目的

ディープラーニングを使用して人間では特定することのできないゲームエンジン情報を特定する。

## 3. プロジェクトマネジメントとの関連

実際に使われているゲームエンジンの情報を解析することによって、新たにゲーム開発プロジェクトを開始する際に種類や数のデータが指標になりえる。

## 4. 研究の方法

下記に研究の方法を記す。

1. CryENGINE, FOX ENGINE, Frostbite, PhyreEngine, Unity, UNREAL ENGINE の 6 つのゲームエンジンの画像を JPEG 方式で集める。
2. 各ゲームエンジンの画像をトレーニング用に 1000 枚ずつ計 6000 枚を集める
3. 各ゲームエンジンの画像をテスト用に 200 枚ずつ計 1200 枚を集める

4. 収集したゲームエンジンの画像を読み込ませるためにテキストファイルでラベル付けをする。
5. OpenCV を使用して画像データを読み込んだ際に  $28 \times 28$  にリサイズする。
6. TensorFlow を使用してトレーニング用に 6000 枚の画像を学習させて、テスト用に 1200 枚の画像データを読み込ませてテストデータに対する精度を表示させる。

## 5. 現在の進捗状況

TensorFlow のインストールを終えた後に、まずディープラーニングを試したデータは MNIST という手書きの数字の画像が入っているデータセットだ。このデータセットの中には 0~9 までの手書きの数字、縦 28px × 横 28px = 計 784px の訓練用データ 55,000 件、テスト用データ 10,000 件、検証データ 5,000 件の計 70,000 件が入っている。

TensorFlow を使用したディープラーニングの MNIST データの解析の流れは、まず訓練用データ

55,000 件の訓練用データを学習させモデルを作成し、作成されたモデルを用いて、テスト用データ 10,000 件から予測された 0~9 のいずれかの値が算出され、その予測された値と実際の値を比較し、識別率が 0.9179 と出力された。

ゲームの画像を動画からコマ送りで上記で述べた枚数の画像を集めた。

TensorFlow の decodecsv を用いて、CSV ファイルでタグ付けされた JPEG 形式の画像を解析し、画素の情報を 3 次元行列（横、縦、色面）として表現したものが得られた。

TensorFlow をインポートする際に画像の縮小の作業を請け負ってくれる OpenCV というライブラリを同時にインポートすることができなかった。その解決策として Anaconda2 と Anaconda3 を使って TensorFlow と OpenCV を同時に動かすことができた。

## 6. 今後の計画

今の段階で研究の方法に記してある各ゲームエンジンのディープラーニングはできてはいないから、まずはその部分を完了させる。たとえディープラーニングできるところまで行けたとしても、現在は CPU で画像の処理まで行っているので何日もの時間を奪われてしまうだろう。

そこで今後 CPU で処理するのではなく、GPU での処理が必ず必要となっていくだろうからグラフィックボードの用意が必須となっていく。

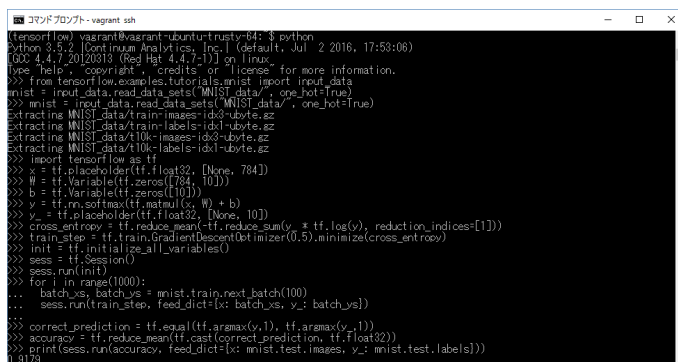
今後この実験ではゲームエンジンのトレーニング用データを学習したあと、テスト用データとの識別率が算出される。しかし現在の進捗状況に沿ってディープラーニングを行ったとしても、学習させるデータの数が少ないため、高く満足のいく識別率は算出されないだろう。

これらのことを踏まえて、以下を今後の計画とする。

1. 今の画像データ総数の 10 倍の訓練用データ 60000 枚、テスト用データ 12000 枚を集める。
2. 集めたデータを TensorFlow を使用してディープラーニングする。
3. 識別率が満足のいく数値が算出されれば、ゲームエンジンの情報が出ていないゲームの画像も解析させてデータをまとめる。

## 参考文献

- [1] 三宅陽一郎, 森川幸人. 絵でわかる人工知能. SB Creative, 第 1 版, 2016.
- [2] Tensorflow. Wikipedia. <https://ja.wikipedia.org/wiki/TensorFlow>(参照 2016-12-14).



```
tensorflow vagrant@vagrant-ubuntu-trusty-64: $ python
Python 3.5.2 [Continuum Analytics, Inc.] (default, Jul 2 2016, 17:53:06)
Type 'help', 'copyright', 'credits' or 'license()' for more information.
>>> from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets('./mnist', one_hot=True)
>>> mnist = input_data.read_data_sets('./mnist', one_hot=True)
>>> extract_mnist_data(train_images=mnist.train.images, train_labels=mnist.train.labels, test_images=mnist.test.images, test_labels=mnist.test.labels)
>>> import tensorflow as tf
>>> w = tf.Variable(tf.zeros([784, 10]))
>>> b = tf.Variable(tf.zeros([10]))
>>> y = tf.nn.softmax(tf.matmul(x, w) + b)
>>> cross_entropy = tf.reduce_mean(-tf.reduce_sum(y * tf.log(y), reduction_indices=[1]))
>>> train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)
>>> init = tf.initialize_all_variables()
>>> sess = tf.Session()
>>> sess.run(init)
>>> for i in range(1000):
...     batch_xs, batch_ys = mnist.train.next_batch(100)
...     sess.run(train_step, feed_dict={x: batch_xs, y: batch_ys})
>>> correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y,1))
>>> accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
>>> print(sess.run(accuracy, feed_dict={x: mnist.test.images, y: mnist.test.labels}))
0.9179
```

図 1 MNIST をディープラーニングで解析した時のコードと結果