山口大学教育学部研究論叢　第 75 巻　235 頁～ 244 頁　2026 年 1 月

筋電位でのジェスチャー操作におけるKerasとPyTorchの性能比較

伊藤　正剛*・北本　卓也**

Performance Comparison of Keras and PyTorch in Gesture Manipulation Using Myoelectric Potentials

ITO Masataka*, KITAMOTO Takuya**

（Received September 26, 2025）

これまで、筆者らは表面筋電位（以下、筋電位）を用いたジェスチャーによる手の動作識別やカーソル操作に焦点を当て研究を行ってきた。最初の研究から2024年11月まで一貫して深層学習用ライブラリ Keras を用いた[1]-[3]。Keras は直感的に少ないコードでプログラミングでき、素早いプロトタイピングが可能である。しかし、筆者のコンピュータで動作識別には Keras で遅延があり、さらなるスムーズさが求められた。そのため、他の深層学習用ライブラリである PyTorch を導入する。また、カーソル操作におけるライブラリ間での性能比較を客観的に行うため、簡単なゲームを作成する。ゲームのスコアの推移により、深層学習用ライブラリ切り替えにおける性能を比較する。

１．はじめに

ニューラルネットワークの発展について、McCulloch と Pitts のニューロンモデル（1943）、Rosemblatt によって発表されたパーセプトロン（1958）、Widrow と Hoff による学習規則（1960）、Rumelhart らの誤差逆伝播法（1986）のように生体の仕組みを数学的に表すという試みが古くからなされてきた[4]。大量のパラメータに関する積和演算は計算コストが高いため2000年代中頃、積和演算に有利な GPU のニューラルネットワークへの応用に関心が向けられた。GPU、畳み込みニューラルネットワーク、CUDA などのハードウェアやソフトウェアの進歩により2012年、ILSVRC（ImageNet Large Scale Visual Recognition Challenge）コンテストの ImageNet データセットにおいて、AlexNet と名付けられたモデル（チーム名 SuperVison）が画像の分類において圧倒的な成績を残した[5]。それ以降、深層学習（ディープラーニング）が広く着目、重要視された。その過程で、ニューラルネットワークをより簡単で効率的に実装できる様々な深層学習用ライブラリが作られた[6], [7]。言語を問わず代表的なものを列挙すると TensorFlow、Keras、PyTorch、MXNet、DeepLearning4j、Flux.jl などがある。また、海外製だけでなく Chainer という日本製のライブラリが知られている。他にも、ライブラリは多数存在するが、開発の縮小や停止により現在積極的に使われてないものもある。

本研究では、上記のライブラリで Keras と今回新しく導入する PyTorch を用いて、筋電位によるリアルタイムでの手の動作識別を行う。手の動作識別については、Abdelaziz らが CNN と LSTM を組み合わせ、手のジャスチャーデータセット（UC2018 DualMyo datasets、EMG36 datasets）で訓練・検証精度ともに97% 以上の高い精度を達成している[8]。また、Ponomarchuk らが、複数のチャンネルで様々な機械学習手法を用い、動作識別の精度を調査した[9]。筆者らもこれと類似した研究を行っている[2]。しかし、研究[2]以降では機械学習手法を深層学習のみで行っている。

本研究では、リアルタイム動作識別を行ったあと、その信号に応じてカーソルやゲームを動かしている。ここで、動作識別≒カーソル操作（方向）≒ゲーム操作（方向）である。（また、カーソル操作は動作識別を目で確認できるフィードバックシステムとしても働く）カーソル操作は深層学習の計算（順伝播）とカーソル操作自体

＊ 放送大学教養学部　　＊＊ 山口大学教育学部　〒753-8513　山口市吉田1677-1　kitamoto@yamaguchi-u.ac.jp

の2つの代表的な処理オーバーヘッドが存在する。そのため、より精度が高く、遅延の小さいシステム実現への課題があった。そこで、処理オーバーヘッドの1つである深層学習の計算速度を改善するために別の深層学習用ライブラリを新しく導入することを考える。

今まで、深層学習用ライブラリに、バックエンドがTensorFlow の Keras を用いてきた。Keras と PyTorch を比較した2017年と2025年のベンチマークでは PyTorch が Keras より推論が速くリアルタイム推論に向くことが示唆されている[7], [10]。また、CNN を使った画像分類の分野ではあるが、Keras、PyTorch、MXnet によって推論の精度が異なることを示している[11]。

以上から、学習時、推論時において Keras より PyTorch が有利であることが示唆された。したがって、今回、使用してきたライブラリ Keras に新しく PyTorch を追加する。次に、簡単なゲームを作成し、そのスコア推移で客観的な性能を比較する。

2．方法

2.1　実験参加者

実験には健常な2名が参加した。筆者（著者A）と別の著者（著者B）である。なお、本研究は放送大学研究倫理委員会の倫理審査の承認を得たものである（通知番号2023-73）。実験参加者は、事前に説明文書と同意書を読み自由意思のもと同意を得た上で実験を行った。

2.2　ARV 計測機器、使用コンピュータ、使用言語

ARV（整流平滑化）計測機器として、スポーツセンシング社製（以下、SS 社、[2]）の DSP ワイヤレス筋電センサ湿式タイプ（SS-EMGW-HM）を用いた。小型軽量でリアルタイムに ARV を無線で送信する機能があり使いやすい[2]。ここで、ARV とは筋電位の絶対値を求め平滑化（このセンサでは移動平均）したものである。体表にセンサパッドを貼り、筋電位を計測するため痛みを感じることはない。電極貼り付け位置は著者Aの腕の写真とともに図１に提示した。

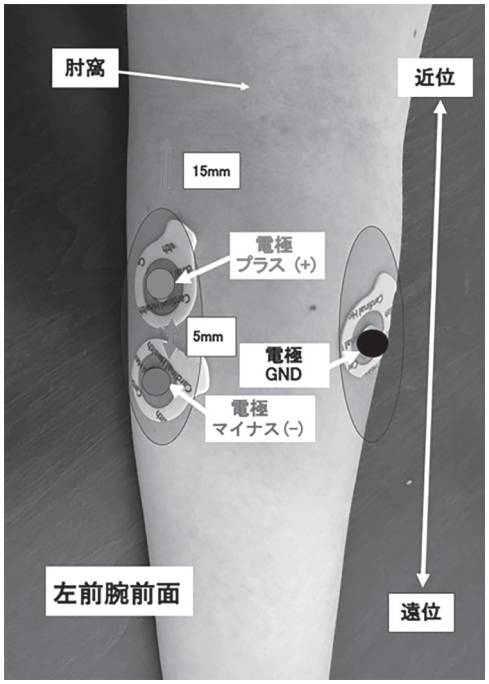


図１　センサ取り付け位置（橈側手根屈筋の位置）

電極貼り付け位置については、参考文献[3]に記載のとおり、筋肉の大きさや位置に個人差があるため毎回全く同じ位置に貼り付けることは困難である。しかし、本研究では同じ位置に貼り付けないでも極端に位置が外れていなければ、正しく動作識別が行われることが実験的に分かっている。したがって、橈側手根屈筋の部分であれば図１のような電極貼り付け位置を提示し、それに基づき取り付けても、動作識別に大きく影響が出ない。また、1つのセンサ1ch だけで実験を行った。これは単純で頑健なシステムの作成及び低コスト化を狙ったためである。また、センサを複数から1つにすることにより見逃していた特徴なども調べることもできる。

次に、使用コンピュータスペックを表１に示す。最初の研究[1]からコンピュータは変わっていない。2025年8月の時点で Intel の第14世代 CPU が販売されている。表に示した第8世代の CPU でも本研究独自の特徴量[1]では通用する。

表1　実験コンピュータスペック

OS	Windows 11 Pro 64bit
CPU	Intel Core i7-8565U (1.80GHz-4.60GHz 4コア/8スレッド)
RAM	16GB LPDDR3 2133MHz

最後に、使用するプログラミング言語及び深層学習用ライブラリを述べる。今回は、これまでの研究[1]-[3]に引き続いて Python3.11を使用した。使用した深層学習用ライブラリとバージョンを表２に示す。なお、2025年7月23日での最新バージョンは、Python 3.13.5、Keras 3.10.0、PyTorch 2.7.1である。

表2　使用ライブラリ

TensorFlow	2.16.1
Keras	3.3.3
PyTorch	2.5.1

2.3　動作（ジェスチャー）ごとの教師ラベル

参考文献[2]から動作の図を引用した。図２のように各動作に対して握りは0、反りは1、動作なしは2と教師ラベルを設定する。本研究では、この教師ラベルを前提とする。握ってすぐに力を抜くなど動作は瞬間的である。全ての実験は著者Ａと著者Ｂで行った。著者Ａは右手でマウスを操作しながら実験を行っていたため左腕での測定となった。統一するため著者Ｂも左腕での測定とした。

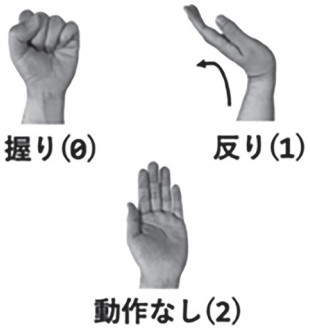


図2　動作（ジェスチャー）と教師ラベル

2.4　深層学習

以下に、今回の深層学習の流れを、数式を用いて説明する。また、Keras と PyTorch のハイパーパラメータについても表に示す。

$\boldsymbol{x}$ を、D 次元列ベクトルとする。握り、反り、動作なしのデータ数はそれぞれ4100個とする。このデータから1動作5行20列、全体の3動作15行20列の ARV データ（事前学習データ）を得る。これらの詳しいデータ取得方法は[3], [19]を参照されたい。

各動作（握り、反り）1サンプル分を具体的に表すと、

$$\boldsymbol{x}_{sample} = (x_{r-d}, x_{r-d+1}, \cdots x_r, \cdots x_{r+d-1})^T \quad (1)$$

$\boldsymbol{x}_{sample} \in \boldsymbol{P}^D$ となり、$\boldsymbol{P} = [0, 8.3]$ である。r はピークを表すインデックスである。ピーク前10個、ピーク1個、ピーク後ろ9個のデータを取るため、$d = 10$、$D = 2d = 20$ である。動作なしはフラットなデータなため、ピークと関係なく別の方法でデータを取得する。これらを、動作ごとに5サンプル集めて結合し事前学習データを作成すると、

$$\boldsymbol{X}_{train} = \begin{pmatrix} \boldsymbol{X}_{train1} \\ \boldsymbol{X}_{train2} \\ \boldsymbol{X}_{train3} \end{pmatrix}, \quad \boldsymbol{X}_{train} \in \boldsymbol{P}^{N \times D} \quad (2)$$

となる。$N = 15$ である。$X_{train1} \in P^{l \times D}$ は握りデータ、$X_{train2} \in P^{l \times D}$ は反りのデータ、$X_{train3} \in P^{l \times D}$ は動作なしのデータである。各動作サンプル数 $l = 5$ とする。

X_{train} から取り出した、特徴ベクトルをニューラルネットワークに入力し、

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + b_j^{(1)},\ j = 1, \ldots, H_1 \quad (3)$$

となる。上付き(1)は1層目（第1隠れ層）を表しており、隠れ層ニューロン数、$H_1 = 1200$ である。

次に、

$$z_j = h(a_j) \quad (4)$$

非線形活性化関数 h により変換される。今回の場合、ReLU であり、

$$h(a) = \max(a, 0) \quad (5)$$

と定義される。

2層目（第2隠れ層）では、

$$a_k = \sum_{j=1}^{H_1} w_{kj}^{(2)} z_j + b_k^{(2)},\ k = 1, \ldots, H_2 \quad (6)$$

となる。隠れ層ニューロン数、$H_2 = 1200$ である。a_k は1層目と同様に、ReLU によって変換され、

$$z_k = h(a_k) \quad (7)$$

となる。

3層目（出力層）では、

$$a_o = \sum_{k=1}^{H_2} w_{ok}^{(3)} z_k + b_o^{(3)},\ o = 1, \ldots, C \quad (8)$$

となる。3動作分類のため、$C = 3$ である。その後、softmax 関数により変換され、

$$y_o = \frac{\exp(a_o)}{\sum_{o'=1}^{C} \exp(a_{o'})} \quad (9)$$

となる。

ここで、教師ラベルの集合 $\{\boldsymbol{t}_o\}$ が与えられたとき、誤算関数に、クロスエントロピー誤差を用い、

$$E = -\sum_{o=1}^{C} t_o \ln y_o \quad (10)$$

を計算する。

誤差関数の微分は、

$$\frac{\partial E}{\partial a_o} = y_o - t_o \quad (11)$$

となる。これを、

$$\delta_o^{(3)} = y_o - t_o \quad (12)$$

とする。上付き(3)は3層目（出力層）を表す。

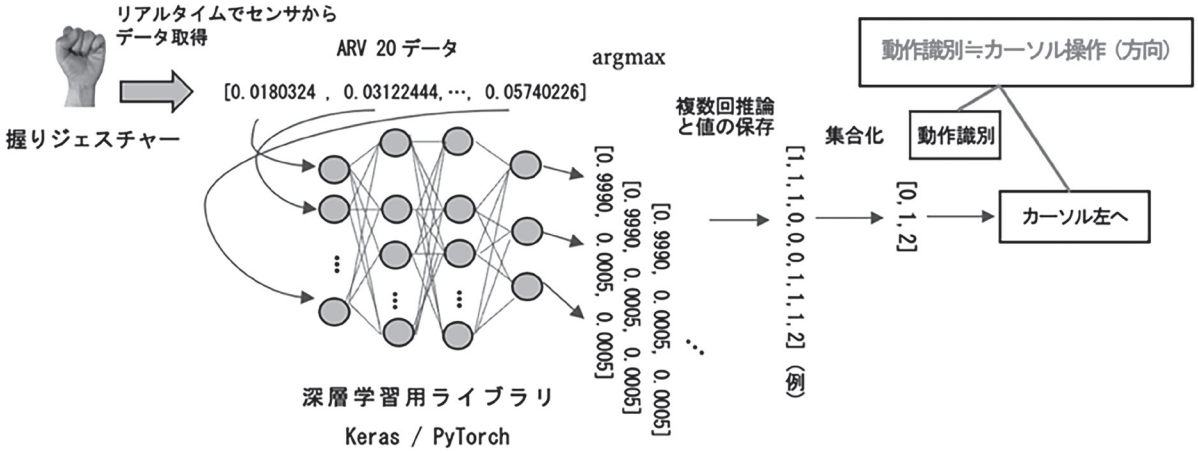


図３　握りのジェスチャーでの動作識別およびカーソル操作概略

また、誤差 E の3層目の重みとバイアスの微分は、

$$\frac{\partial E}{\partial w_{ok}^{(3)}} = \delta_o^{(3)} z_k,\ \frac{\partial E}{\partial b_o^{(3)}} = \delta_O^{(3)} \tag{13}$$

である。また、逆伝播の公式（導出は[4]を参照）は、

$$\delta_k^{(2)} = h'(a_k) \sum_{o=1}^{C} w_{ok}^{(3)} \delta_o^{(3)} \tag{14}$$

である。ReLU の微分は、

$$h'(a) = \begin{cases} 1 & (a > 0) \\ 0 & (a \leq 0) \end{cases} \tag{15}$$

であり、$a=0$で0とする。
式(14)を使い誤差 E の2層目の重みとバイアスの微分を求め、

$$\frac{\partial E}{\partial w_{kj}^{(2)}} = \delta_k^{(2)} z_j\ ,\ \frac{\partial E}{\partial b_k^{(2)}} = \delta_k^{(2)} \tag{16}$$

となる。同様にして、

$$\delta_j^{(1)} = h'(a_j) \sum_{k=1}^{H_2} w_{kj}^{(2)} \delta_k^{(2)} \tag{17}$$

から、誤差 E の1層目の重みとバイアスの微分を求め、

$$\frac{\partial E}{\partial w_{ji}^{(1)}} = \delta_j^{(1)} x_i,\ \frac{\partial E}{\partial b_j^{(1)}} = \delta_j^{(1)} \tag{18}$$

となる。以上より、全ての層の重みとバイアスに関する微分を得る。

Keras と PyTorch のハイパーパラメータを表３、表４に示した。両者で異なるのは epoch 数だけである。epoch 数の決定については、同雑誌に投稿予定の別の論文[19]で詳しく述べた。

表３　Keras ハイパーパラメータ

隠れ層の数	隠れ層 ニューロン数	活性化関数	最適化手法
2	1200	隠れ層全てReLU	SGD

epoch	バッチサイズ	損失関数	Dropout
2000	8	Categorical Cross Entropy	隠れ層全て0.5

握りジェスチャーを例としたカーソル操作の概略を図３に示した。図３の集合化の補足を述べる。例えば、握りでは動作なし2に引き続き、複数の1、複数の0、複数の1、動作なし2の順に信号が現れる（図の例部分）。このシステムでは、握り信号は集合 {0, 1, 2} 、反り信号は {0, 1} 、動作なしは {2}　となる。詳しくは[3]を参照されたい。

表４　PyTorch ハイパーパラメータ

隠れ層の数	隠れ層 ニューロン数	活性化関数	最適化手法
2	1200	隠れ層全てReLU	SGD

epoch	バッチサイズ	損失関数	Dropout
10000	8	Categorical Cross Entropy	隠れ層全て0.5

2.5　Keras と PyTorch の概略と計算グラフ構築

Keras は、作者 François Chollet により初版として2015年3月27日に公開された。Keras は、簡単にネットワーク構造を構築することができ、素早い実験を行うことを可能としている。現在使用しているバージョン3.3.3は、2024年4月23日に公開された。初期では複数のバックエンドがサポートされていたが、一時期 TensorFlow のみであった。しかし、最近（2025年7月23日）では Keras3で TensorFlow、PyTorch、JAX のマルチバックエンドを再度サポートしている[12]。

一方、PyTorch は2016年10月に初版として公開された。初期では Facebook の人工知能研究グループ AI Research lab（FAIR）により開発された。NumPy に似

たコードを使い深層学習プログラムを作成することができる。

これらのライブラリは Define-and-Run や Define-by-Run といった計算グラフの構築の方法がある[13]。ここで、機械学習における計算グラフとは、有向非巡回グラフ（DAG; Directed Acyclic Graph）のことである。これは自動微分の研究で発展した[14]。Define-by-Run の仕組みは Autograd（自動微分 Python ライブラリ）で採用されていたが、深層学習用ライブラリとしては Chainer がさルネットワークはプログラムであり、Chainer はその実行履歴を管理している[16]。

図４にそれぞれの実行形式を示した。Define-and-Run は静的に計算グラフが定義される[17]。この方式は、畳み込みニューラルネットワークなどの構造が固定的なモデルでの最適化に効率的である[15]。また、ネットワークを使い回せるという利点がある。一方、Define-by-Run は実行初期にはグラフは定義されておらず、データが流れることにより計算グラフが動的に構築される。この動的な方式は、再帰的ニューラルネットワークなどのデータによりネットワーク構造が変わるモデルで効率的である[7], [17]。ただし、DeVito ら[18]によると、明示的に Define-by-Run と Define-and-Run に言及してないものの、推論において常に Define-by-Run が速いとは限らないことを示唆している。

```
import numpy as np
import tensorflow as tf

x = tf.placeholder(tf.float32, shape=(128, 28, 28, 1))
conv = tf.layers.conv2d(x, 12, 5, activation=tf.nn.relu)
flat = tf.layers.flatten(conv)
y = tf.layers.dense(flat, 40)                    グラフの定義(define)

with tf.Session() as sess:                       実行（run）
    sess.run(tf.global_variables_initializer())
    out = sess.run(y, feed_dict={x: np.random.randn(128, 28, 28, 1)})
                                                 データ
```

（a）Define-and-Run（TensorFlow v1）

```
import torch
import torch.nn as nn
import torch.nn.functional as F

conv = nn.Conv2d(1, 12, 5)
linear = nn.Linear(12*24*24, 40)                 レイヤ定義

x = torch.randn(128, 1, 28, 28)                  データ

x = conv(x)                                      実行（run)+グラフ構築

x = F.relu(x)                                    実行（run)+グラフ構築

x = torch.flatten(x, 1)                          実行（run)+グラフ構築

y = linear(x)                                    実行（run)+グラフ構築
```

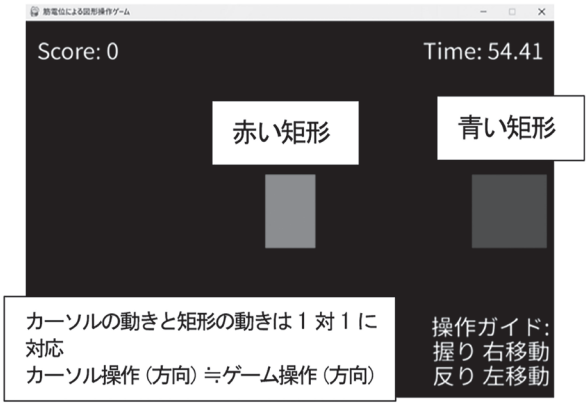
（b）Define-by-Run（PyTorch）

図４　(a) Define-and-Run と (b) Define-by-Run のコードでの比較、実行（run）は順伝播の実行

2.6　スコア推移による性能比較のためのゲーム作成

筋電位を用いたジェスチャーによる矩形の操作ゲームを2024年7月下旬ごろに開発した。これは、プログラムの改良やライブラリの変更による動作識別精度の変化を客観的に調べるためである。そこで PyGame 2.6.0（2025年7月23日最新バージョン）ライブラリを用い作成した。今回4つのゲームを作成した。1つ目のゲームは、ジェスチャーにより赤い矩形を水平方向に動かし、水平方向にランダムに表示された青い矩形に当てるゲームである。これを「HORIZONTAL Game」とした。ゲーム画面を図５に示す。(b) は注釈（白色部分）である。

（a）モード選択画面



（b）HORIZONTAL Game ゲーム画面

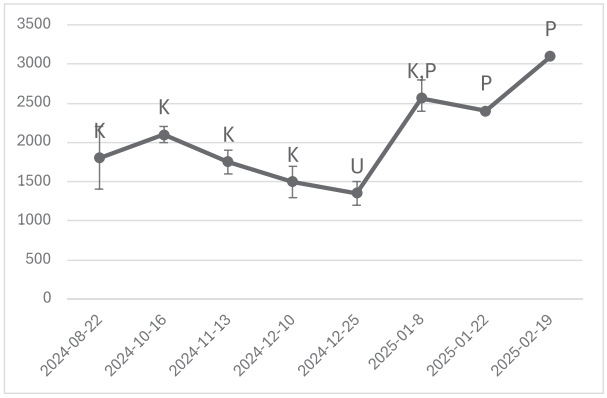
図５　ゲーム画面

3つの動作の握り、反り、動作なしで、水平方向と垂直方向への移動を同時には表現できない。つまり、「水平方向の移動と停止」あるいは「垂直方向の移動と停止」しか表現できない。そのため、2回手を握り水平方向と垂直方向の切り替えを行うプログラムを作成した。これは「2回握り」とした。これを行う2つ目のゲームを「DOUBLE-CLICK Game」とした。3つ目のゲームは、1つ目のゲームと2つ目のゲームおよび垂直方向への移動を統合させたものである。水平・垂直方向に赤い矩形を動かし、水平方向・垂直方向にランダムに表示される青い矩形に当てるゲームである。これ

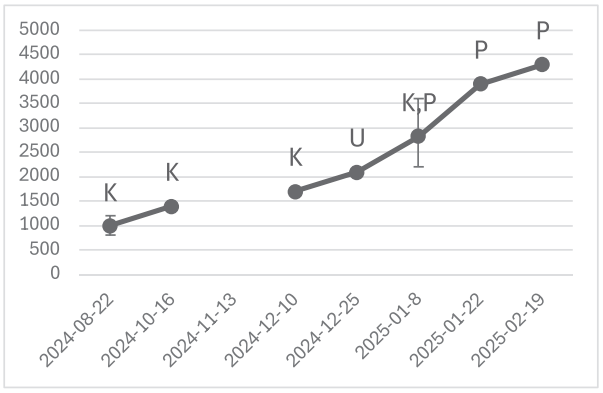
は「HORIZONTAL&VERTICAL Game」とした。なお、ゲーム中は無音だとモチベーションが下がるため音楽を流した。ゲームの制限時間は全てのゲームモードで60秒である。4つ目のゲームは操作性の悪さから使用していない。ゲームのスコアは、赤い矩形が青い矩形に当たると一律100点加点とした。この方式ではスコアを100で割れば正確な成功数が分かる。点数の変化を調べるためスコアを記録しテキストファイルに保存しておく機能をつけた。

3．スコア推移の結果と考察

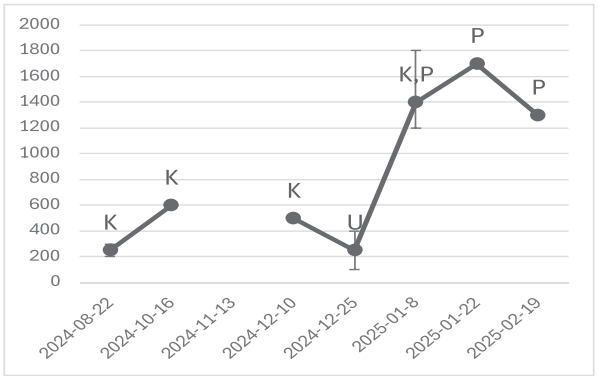
付録表５に、著者Ｂのスコアの推移と深層学習ライブラリごとの最高点と最低点を示す。ただし、Unknownのときのスコアは除いた。スコアを調べる実験は、場面に応じて、様々な要素を追加、削除、修正しながら進めたため、記録が間に合わなかった部分を Unknown（記録なし）とした。ブレンド比は、実験者のデータ（事前学習データ）と著者Ａのデータ（標準データ）の比を表している。標準データやブレンド比について詳しくは、参考文献[19]に述べた。今回これらの結果を、後方視的に考察した。



（a）HORIZONTAL Game スコア推移



（b）DOUBLE-CLICK Game スコア推移



（c）HORIZONTAL&VERTICAL Game スコア推移

図６　異なるゲームによるスコア推移。点・実線は平均点、バーは複数回実験を行った日での最高点、最低点。U は Unknown、記録なし。

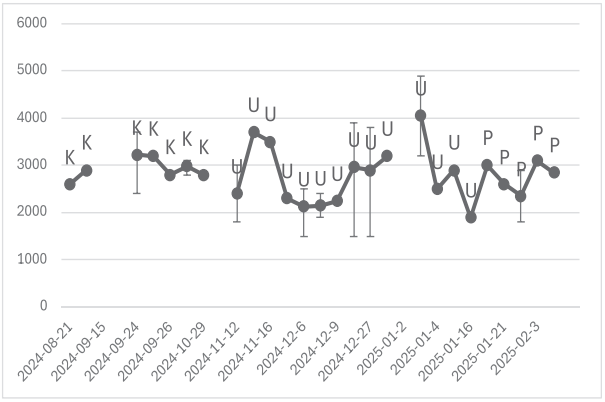
図６(a)、(b)、(c)にそれぞれ HORIZONTAL Game、DOUBLE-CLICK Game、HORIZONTAL&VERTICAL Game の著者Ｂのスコアの推移のグラフを示した。実線は、実験を行った日の平均スコアをつないでいる。1日に1回のみ実験を行ったときは、平均点とその日の値は等しい。1日2回以上実験を行った場合は、最高スコアと最低スコアを表すバーを付けている。ゲームを行っていないときは、データの欠損（データなし）として、線が途切れており結果や考察から除外した。また、グラフのデータ点の上に使用ライブラリを載せた。K は Keras で、P は PyTorch である。U は Unknown である。

グラフから結果を述べる。HORIZONTAL Game では、PyTorch のスコアの平均点が、Keras より明らかに上回っている。

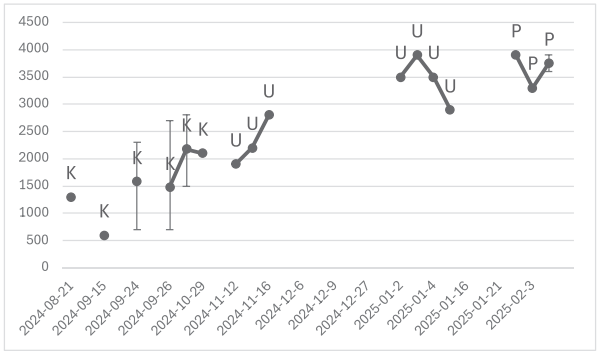
次に DOUBLE-CLICK Game では、スコアは単調に増加し、PyTorch のスコアの平均点が上回っている。HORIZONTAL&VERTICAL Game でも、HORIZONTAL Game 同様、PyTorch のスコア平均点が Keras より上回っている。

以上の結果から考察を述べる。今回、実験で測った全てのゲームモードについて、表から Keras より PyTorch で最高点が上回っていることがいえ、グラフから Keras より PyTroch のときの平均点が上回っている。2025年1月8日では、深層学習ライブラリが PyTorch のとき、データ追加後のスコアは追加前より上回っている。それ以降では、データを追加して実験を行っているため、データの追加の影響があることは無視できない。

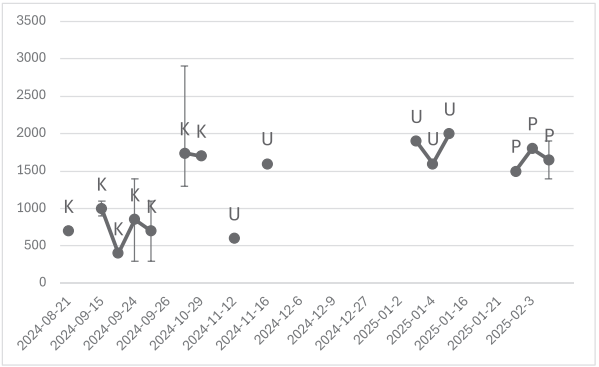
付録表６、表７に著者Ａのスコアの推移と各ライブラリでの最高点と最低点示す。スコアはゲーム動作が安定した2024年8月21日から掲載している。



（a）HORIZONTAL Game スコア推移



（b）DOUBLE-CLICK Game スコア推移



（c）HORIZONTAL&VERTICAL Game スコア推移

図７　異なるゲームによるスコア推移。点・実線は平均点、バーは複数回実験を行った日での最高点と最低点。U は Unknown、記録なし。

図 7 (a)、(b)、(c) に HORIZONTAL Game、DOUBLE-CLICK Game、HORIZONTAL&VERTICAL Game の著者Ａのスコアの推移をグラフに示した。著者Ｂのグラフ同様、実線は実験を行った日の平均点をつないでいる。1日に1回のみ実験を行ったときは、平均点とその日の値は等しい。1日2回以上実験を行った場合は、図６と同様に最高スコアと最低スコアを表すバーを付けている。また、ゲームを行っていないときは、データの欠損（データなし）として、線が途切れており結果や考察から除外した。また、先ほど同様にグラフのデータ点の上に使用ライブラリを載せた。K は Keras で、P は PyTorch である。U は Unknown である。

紙面や見やすさの問題から省略した日付がある。

グラフから読み取れることとして、HORIZONTAL Game は、小さな期間で変化はあるがスコアは横ばいである。DOUBLE-CLICK Game は小さな期間での変化はあるがスコアは全体として右肩上がりである。HORIZONTAL&VERTICAL Game も他のゲームモード同様に小さな期間での変化はあるがスコアは横ばいである。

以上の結果から考察を述べる。HORIZOTAL Game、HORIZONTAL&VERTICAL Game では、著者Ｂと著者Ａでは結果が異なる。この要因として、2022年から2025年までシステム作成のため著者Ａは何度も実験を繰り返している。そのため2年以上実験を頻繁に行った著者Ａでは2024年では握りと反りのジェスチャーによるカーソルの動かし方が上達している。したがって HORIZONTAL Game と HORIZONTAL&VERTICAL Game のスコアはとりうる上限付近にとどまっていると考えられる。DOUBLE-CLICK Game に関しては、カーソルの移動より方向の切り替えは試行回数が少ない。そのため、著者Ａはそれほど上達していなかったと考えられる。

以上のスコアの推移から著者Ａでは2年以上のカーソル操作の経験からゲームスコアが上限に達しており PyTorch への変更の効果がスコアの推移として現れにくいと考えられる。

４．制限事項

プログラムの検査を行った結果、PyTorch の学習において softmax ＋ CrossEntropyLoss（softmax が含まれる）の冗長な実装が見つかった。ただし、学習自体が不安定となる処理を行っても、データ追加による動作識別精度が向上したため、結果と考察ではそのままのデータを掲載する。また、冗長な処理を削除し、ゲームのスコアを再測定した。ゲームスコアは同様であった。

５．結論

本研究の実験システムで深層学習用ライブラリを変更することの動作識別精度への影響は分かっていなかった。そのため今回、ゲームを用いてスコアの推移を通してライブラリ間の性能比較を行った。著者Ａ、著者Ｂの主観的なゲームの操作性は PyTorch の方が上回っていた。また、2.5節で述べた計算グラフの構築方式による違いで、結果に差がでた可能性も考えられる。

著者Ｂでの結果と考察からも、今回の実験で用いた深

層学習用ライブラリのバージョンにおいては、Keras より PyTorch の性能が上回っている。また、ゲームへの上達が性能評価の課題となることが分かった。

深層学習用ライブラリは、改良や進歩が著しく速いため、本論文の知見もすぐに古いものとなる可能性が高い。これからも、リアルタイム性の向上のために、深層学習用ライブラリをフォローしていきたい。

最後に、本研究では高齢者や障がい者に有用な道具を作ることを目的としている。引き続き研究開発を続けたい。

謝辞

研究の場や機会を下さった先生や大学の皆様、深く感謝いたします。また、支えてくれた家族に感謝したいと思います。

参考文献

［1］伊藤正剛，北本卓也（2023）「筋電位を利用した深層学習による手の動作識別について」，山口大学教育学部研究論叢 第72巻 227-235

［2］伊藤正剛，北本卓也（2024）「様々な機械学習を用いた手の動作識別について」，山口大学教育学部研究論叢 第73巻 91-100

［3］伊藤正剛，北本卓也（2025）「筋電位での機械学習によるリアルタイム動作識別について」，山口大学教育学部研究論叢 第74巻 163-171

［4］C.Mビショップ著，元田浩，栗田多喜夫，樋口知之，松本裕治，村田昇監訳（2012）『パターン認識と機械学習 上』丸善出版

［5］Large Scale Visual Recognition Challenge 2012 https://www.image-net.org/challenges/LSVRC/2012/results.html（2025年7月23日アクセス）

［6］Jeffery Dean（2022）“A Golden Decade of Deep Learning: Computing Systems & Applications”, Daedalus 151(2): 58-74.

［7］鈴木亮太（2019）「第2回 機械学習と開発環境：深層学習フレームワークの動向」，計測と制御 第58巻 第5号 387-391

［8］Mai H. Abdelaziz, Wael A. Mohamed, Ayman S. Selmy（2024）“Hand Gesture Recognition Based on Signals and Deep Learning Techniques”, Journal of advances in Information Technology. Vol. 15. No. 2

［9］Y V Ponomarchuk, I V Kuznetsov（2021）“Comparative of Efficiency of the Machine Learning Methods for Gesture Recognition Using Double-Channel Electromyography”, J. Phys: Conf. Ser.2134 012010

［10］Tools and Frameworks for Deep Learning CPU Benchmarks https://www.analyticsvidhya.com/blog/2025/01/deep-learning-cpu-benchmarks/（2025年6月10日アクセス）

［11］Seongsoo Kim, Hayden Wimmer, Jongyeop Kim,（2022）“Analysis of Deep Learning Libraries: Keras, PyTorch, and MXnet”,IEEE/ACIS 20th International Conference on Software Engineering Research, Management and Applications（SERA）, 54-62

［12］https://keras.io/keras_3/（2025年7月23日アクセス）

［13］落合翼，末廣達也，松田繁樹，片桐滋（2017）「音声言語処理における深層学習ツールキット解説」，日本音響学会誌73巻1号，pp.63-72

［14］伊理正夫，久保田光一（1991）「高速自動微分法（Ⅰ）」，応用数理1(1),17-35

［15］Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, Soumith Chintala（2019）“PyTorch: An Imperative Style, High-Performance Deep Learning Library”, arXiv:1912.01703v1

［16］得居誠也（2016）「ニューラルネットワークの実装―深層学習フレームワークとChainerの設計思想―」，人工知能31巻2号

［17］Aeiya Tokui, Ryosuke Okuta, Takuya Akiba, Yusuke Niitani, Toru Ogawa, Shunta Saito, Shunji Suzuki, Kota Uenishi, Brain Vogel, Hiroyuki Yamazaki Vincent（2019）“Chainer A Deep Learning Framework for Accelerating the Research Cycle”, In The 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining（KDD'19）,August4-8, Anchorage, AK, USA. ACM, New York,NY,USA

［18］Zachary DeVito, Janson Ansel, Will Constable, Michael Suo, Ailing Zhang, Kim Hazelwood（2021）“Using Python for Model Inference in Deep Learning”, arXiv:2104.002254v1［cs.LG］1 Apr

［19］伊藤正剛，北本卓也（2026）「筋電位でのジェスチャー操作におけるデータ量と計算コストの定量評価」，山口大学教育学部研究論叢 第75巻 投稿予定

表５　著者Ｂのゲームスコア推移、Unknown は記録なし
‡ Keras 最高点　† Keras 最低点　** PyTorch 最高点　* PyTorch 最低点
（最高点、最低点 Unknown は除く）　スコアは未加工

実験日	実験参加者	HORIZONTAL	DOUBLE-CLICK	HORIZONTAL&VERTICAL	制限時間(秒)	深層学習用ライブラリ	ブレンド比
2024-08-22	著者B	1400†	800†	200†	60	Keras	1:0
2024-08-22	著者B	2200	1200	300	60	Keras	1:0
2024-10-16	著者B	2000			60	Keras	1:0
2024-10-16	著者B	2200	1400	600	60	Keras	1:0
2024-11-13	著者B	1600			60	Keras	1:0
2024-11-13	著者B	1900			60	Keras	1:0
2024-12-10	著者B	1300*			60	PyTorch	1:0
2024-12-10	著者B	1700	1700*	500*	60	PyTorch	1:0
2024-12-25	著者B	1200	2100	100	60	Unknown	Unknown
2024-12-25	著者B	1500		400	60	Unknown	Unkonwn
2025-01-8	著者B	2500	2700	1200	60	PyTorch	1:0
2025-01-8	著者B	2400‡	2200‡	1200‡	60	Keras	1:0
2025-01-8	著者B	2800	3600	1800**	60	PyTorch	1:2
2025-01-22	著者B	2400	3900	1700	60	PyTorch	1:5
2025-02-19	著者B	3100**	4300**	1300	60	PyTorch	1:4

表６　著者Ａのスコア推移、Unknown は記録なし
‡ Keras 最高点　† Keras 最低点　** PyTorch 最高点　* PyTorch 最低点
（最高点、最低点 Unknown は除く）　スコアは未加工

実験日	実験参加者	HORIZONTAL	DOUBLE-CLICK	HORIZONTAL&VERTICAL	制限時間(秒)	深層学習用ライブラリ	ブレンド比
2024-08-21	著者A	2600	1300	700	60	Keras	1:0
2024-09-2	著者A	2900			60	Keras	1:0
2024-09-15	著者A			900	60	Keras	1:0
2024-09-15	著者A		600†	1100	60	Keras	1:0
2024-09-22	著者A			400	60	Keras	1:0
2024-09-24	著者A		1100	1400	60	Keras	1:0
2024-09-24	著者A	2400†	700	600	60	Keras	1:0
2024-09-24	著者A	3200	2300	800	60	Keras	1:0
2024-09-24	著者A		2100	1400	60	Keras	1:0
2024-09-24	著者A	3600	1200	800	60	Keras	1:0
2024-09-24	著者A	3700‡	2100	300†	60	Keras	1:0
2024-09-24	著者A			1000	60	Keras	1:0
2024-09-24	著者A			300†	60	Keras	1:0
2024-09-24	著者A			1100	60	Keras	1:0
2024-09-25	著者A	3200		300†	60	Keras	1:0
2024-09-25	著者A			1100	60	Keras	1:0
2024-09-26	著者A	2800	700		60	Keras	1:0
2024-09-26	著者A		1100		60	Keras	1:0
2024-09-26	著者A		2700		60	Keras	1:0
2024-09-26	著者A		1800		60	Keras	1:0
2024-09-26	著者A		1100		60	Keras	1:0
2024-10-20	著者A	3000	1500	2900‡	60	Keras	1:0
2024-10-20	著者A	3100	2300		60	Keras	1:0
2024-10-20	著者A	3000	2200	1600	60	Keras	1:0
2024-10-20	著者A	2800	2400	1300	60	Keras	1:0
2024-10-20	著者A	3000	1500		60	Keras	1:0
2024-10-20	著者A	2900	2300		60	Keras	1:0
2024-10-20	著者A	3100	2200		60	Keras	1:0
2024-10-20	著者A	3000	2400	1600	60	Keras	1:0
2024-10-20	著者A		2800‡	1300	60	Keras	1:0
2024-10-29	著者A	2800	2100	1700	60	Keras	1:0
2024-11-5	著者A				60	Unknown	1:0
2024-11-12	著者A	3000	1900	600	60	Unknown	1:0
2024-11-12	著者A	2400			60	Unknown	1:0

表７　著者Ａのスコア推移（続き）、Unknown は記録なし
‡ Keras 最高点　† Keras 最低点　** PyTorch 最高点　* PyTorch 最低点
（最高点、最低点 Unknown は除く）　スコアは未加工

実験日	実験参加者	HORIZONTAL	DOUBLE-CLICK	HORIZONTAL&VERTICAL	制限時間(秒)	深層学習用ライブラリ	ブレンド比
2024-11-12	著者A	1800		600	60	Unknown	1:0
2024-11-15	著者A	3700	2200		60	Unknown	1:0
2024-11-16	著者A	3500	2800	1600	60	Unknown	1:0
2024-11-24	著者A	2300			60	Unknown	1:0
2024-12-6	著者A	2500			60	Unknown	1:0
2024-12-6	著者A	1500			60	Unknown	1:0
2024-12-6	著者A	2400			60	Unknown	1:0
2024-12-8	著者A	1900			60	Unknown	1:0
2024-12-8	著者A	2400			60	Unknown	1:0
2024-12-9	著者A	2300			60	Unknown	1:0
2024-12-9	著者A	2200			60	Unknown	1:0
2024-12-11	著者A	3500			60	Unknown	1:0
2024-12-11	著者A	3900			60	Unknown	1:0
2024-12-11	著者A	1500			60	Unknown	1:0
2024-12-27	著者A	3800			60	Unknown	Unknown
2024-12-27	著者A	3400			60	Unknown	Unknown
2024-12-27	著者A	1500			60	Unknown	Unknown
2025-01-1	著者A	3200			60	Unknown	Unknown
2025-01-2	著者A		3500		60	Unknown	Unknown
2025-01-3	著者A	4900	3900	1900	60	Unknown	Unknown
2025-01-3	著者A	3200			60	Unknown	Unknown
2025-01-4	著者A	2500	3500	1600	60	Unknown	Unknown
2025-01-11	著者A	2900	2900	2000	60	Unknown	Unknown
2025-01-16	著者A	1900			60	PyTorch	Unknown
2025-01-16	著者A	1900			60	PyTorch	Unknown
2025-01-19	著者A	3000			60	Keras	Unknown
2025-01-21	著者A	2600			60	Keras	1:0
2025-01-31	著者A	2900	3900**	1500	60	PyTorch	1:0
2025-01-31	著者A	1800*			60	PyTorch	1:0
2025-02-3	著者A	3100**	3300*	1800	60	PyTorch	1:0
2025-02-17	著者A	2900	3600	1900**	60	PyTorch	1:0
2025-02-17	著者A	2800	3900**	1400*	60	PyTorch	1:0