DeepLabCutの使い方を解説する

執筆者：松井 大 (Matsui Hiroshi)
慶應義塾大学社会学研究科 (2019年3月現在博士3年)
連絡先：matsui.h1004@gmail.com

はじめに
本稿は DeepLabCut に関する非公式の日本語マニュアルである。

基本的には bioRxiv 論文として公開されているユーザーガイドの情報の一部を日本語にしてあるだけだが、所々自分がつまずいたところも入れておいた。なお、ここに書かれた情報は 2019 年 3 月 4 日時点のものである。

ユーザーガイドが非常に優秀なので、正直そっちを見た方が良いと思うが、Quick Tutorial として日本語で残しておく (ほぼ自分用であることに留意してほしい)。
ただ、何か誤り、追加すべき情報、あるいは本稿では解決不可能な問題があれば、上記の連絡先にコンタクトをとってもらえれば幸いである。

基本情報
Webpage：https://alexemg.github.io/DeepLabCut/
ユーザーガイド：https://www.biorxiv.org/content/early/2018/11/24/476531

バージョン管理
以下の組み合わせであれば動くことは確認した。Ubuntu ではまだ試していない。

OS	Python	tensorflow	GPU	CUDA	cuDNN
Windows 10	3.6.7	1.10	GTX 1060	9.0	7.3.0
Windows 10	3.6.1	1.10	RTX 2080	10.0	7.30

このツールボックスはバージョンにかなりうるさいので注意してほしい。例えば、私はPython 3.6.5, Python 3.7.1 では DeepLabCut のインストールの時点でつまずいた (setup.py がないとか、意味不明なエラーを吐かれる)。

tensorflow については、製作者らは 1.0, 1.4, 1.8, 1.10, 1.11 で動作確認済みらしい。

DeepLabCut は Python のモジュールとして提供されているが、Python の知識は極々基本的なことがわかっていればいい。例えば「リスト形式というデータ型がある」など。

DeepLabCut のインストール
適切なバージョンの Python + その他が入っていれば、以下のコマンドで DeepLabCut をインストールできる。

>> pip install deeplabcut

エラーが出たとしたら、可能性としては以下の 2 つが考えられる
・ソフトウェアのバージョンの相性が悪い
・必要なパッケージが入っていない
意味不明なメッセージが出たら前者で、「◯◯が足りない」という旨の説明が出たら後者だと思われる。
おそらく、以下のものも手動でインストールする必要がある。

```
>> pip install tensorflow==1.10
>> pip install tensorflow-gpu==1.10
>> pip install -U wxPython
```

"==1.10" の部分でバージョンを指定してインストールできる。

DeepLabCut Quick Start

① まず、deeplabcut を使用可能状態にする。
必要なパッケージが足りていなけらば、ここでエラーが起きる

```
>> import deeplabcut as dlc
```

なお、解析したい動画はテキトーな場所に保存しておく。
動画の拡張子は、avi、mpg、mov では動くことを確認した。細かい話だが、Mac で作った mp4 では動かなかった。

② create_new_project() する (bioRxiv 論文の 7p 付近)
create_new_project() は分析プロジェクトを作成する関数だ。

引数としては、

```
dlc.create_new_project(‘Name of the project’,‘Name of the experimenter’,
[‘Full path of video 1’,‘Full path of video2’,‘Full path of video3’,...],
working_directory=‘Full path of the working directory’,copy_videos=True/False)
```

となっている。

'Full path of video' のところには動画のパスを入れる。複数の動画がある場合は、Python のリスト形式で複数個入れる。

実行すると、working_directory に指定した場所にプロジェクトフォルダが作成される。

例えば、このように実行する。copy_videos はデフォルトが False なので、打たなくても構わない (ただ、True が推奨)。

```
>> config_path = dlc.create_new_project('PigeonCPP', 'HM', ['desktop/pigeonCPP'],
working_directory='desktop/pigeonCPP', copy_videos=False)
```

ここで作った情報は後でも使うので、config_path という名前の変数に格納している。

※ 注意点
config_path にはプロジェクトフォルダの中にある config.yaml ファイルへのパスが格納されている。一旦 python を落として再開してワークスペースの変数が消えていると、dlc.create_new_project() をやっても「既にそのプロジェクトはあるよ」とエラーが返ってくる。従って、一度プロジェクトを作成して、PC を閉じたり IDE を閉じたりした後作業を再開するときは、以下のように直接 yaml ファイルへのパスを通す必要がある。

>> config_path = 'User/kokoni/file/no/basyo/wo/ireru/config.yaml

初歩的な話だが、パスの場所は、config.yaml を右クリックして「プロパティ」を見たら載っている。

③ プロジェクトの設定を行う (bioRxiv 論文の 9p 付近)
まず、config.yaml ファイルを開く。このファイルにトラッキングのための様々な設定情報が書き込まれている。ソフトウェアはなんでもいいが、私は Visual Studio Code で開いている。
細かい情報はユーザーガイドの Box1 に載っているが、絶対チェックすべきなのは以下の箇所だ。

・bodyparts：身体の部位で、ここをいじることで何点トラッキングするかが決まる
・numframes2pick：学習に使うフレーム数で、デフォルトは 20 になっている。何枚必要かは、ビデオの複雑さに依存するが、100-200くらいはほしい。
・cropping：ビデオのいらない箇所を切り取る。なければ False のままでよい

オプショナルだが、以下も変えてよい。
・resnet：ニューラルネットワークの深さ (50 か 101 にできる)
・batch_size：一度の学習で読み込ませるフレーム数

④ データ選択 (bioRxiv 論文の 9p 付近)
学習用の画像データを作成する。dlc.extract_frames() という関数が用意されている。
extract_frames(config_path, ‘automatic/manual’,‘uniform/kmeans’)

・automatic にすれば自動、manual は自分で選ぶ
・uniform だとビデオからランダムに抜き出す
・kmeans だと画像の特徴量からクラスタリングをかけた上で抜き出してくれる
kmeans の方が色々な行動をしている場面を抜き出してくれるので、「たまにしか出ないがちゃんとトラッキングしてほしい行動」などがある場合は、そっちを使った方がうまくいきやすい。ただし、k-means クラスタリングをかけるので、時間がかかる。動画が長いとかなり長い時間がかかるので要注意。

とにかく素早く試したいならば、以下のように実行する。

>> dlc.extract_frames(config_path, ‘automatic’,‘uniform’)

うまく走れば、プロジェクトフォルダの labeled-data に画像が抽出される。
枚数は numframes2pick -1 枚出てくる。

うまくトラッキングをするにはコツがある。
・異なるセッションのデータを含める
・異なる個体のデータを含める
・カメラが複数ある場合、異なるカメラのデータを含める
・いろんな姿勢の状態のフレームを入れる
・できるだけたくさん取る！ (numframes2pick を増やす)
・トラッキング失敗した動画のフレームを学習データに追加する (⑩ の工程)

⑤ データに手動でラベルを貼る (ぽちぽち作業)
学習データの点を自分で打つ。次のコードを実行すればよい。

>> dlc.label_frames(config_path)

うまくいけば、GUI でぽちぽち点を打つための画面が出てくる。
load で画像が入ってあるフォルダ (labeled-data) を選べば画像が取り込まれる。
あとは、ひたすら右クリックで点を打ち、左クリック+ドラッグで点の位置を移動させる。点の数と名前は ③ のyaml ファイルで設定した "bodyparts" と同じになっているはずだ。

※【重要】 身体が隠れている、あるはフレームアウトなどで点が打てない場合
何も打たずに、右上のラジオボタンで次のトラッキングポイントに移ろう。
内部でそこは欠損した点として扱ってくれる。

Zoom をするとドラッグした場所を拡大して見ることができるので便利だ。

⑥ 手動ラベルのチェック
自分がぽちぽちした点がうまく入っているかをチェックする。
この工程は飛ばしても解析に影響はないが、確認のため行った方が良いだろう。

>> dlc.check_labels(config_path)

これを実行すると、プロジェクトフォルダの「labeled-data」フォルダに「test_labeled」というフォルダが作成される。

フォルダの中には、⑤ で自分が打った点が画像に重ねられた画像が保存されている。
誤った点がないかを確認すると良いだろう。

⑦ ネットワークの訓練

これで下準備は完了だ。
ようやく Deep Neural Network の訓練を行う。

まず、GPU を python が認識しているかを確認しよう。
次のコードを実行し、自分が刺している GPU が出力されれば大丈夫だ。

```
>> from tensorflow.python.client import device_lib
>> device_lib.list_local_devices()
```

GPU が出てこない (CPU しか認識されていない) 場合、tensorflow (と tensorflow-gpu)、あるいは CUDA + cuDNN のバージョンが合わない可能性が考えられる。また、一度再起動したらなぜか認識したこともあった。なので、ちゃんと確認しておくのが安パイだろう。
学習には次のコードを走らせる。

```
>> dlc.create_training_dataset(config_path, num_shuffles=1)
>> dlc.train_network(config_path, gputouse=1)
```

gputouse は、使う GPU の数らしい。CPU で回す場合には None と入れる。2 台以上使えるのかは未検証。

学習は iteration が 1030 万回まで行くと自動的にストップするらしいが、そこまでやる必要はない。ただ、二日くらい放置していたらそこまでいって自動的にストップしていることがある。
学習をやめるタイミングとしては、loss があまり変わらなくなる (プラトーになる) まで回ったら止めればよい。製作者は、おおよそ 20 万以上は欲しいと勧めている。止めるには、ctrl + C を押せばよい。
止まらない場合は、一旦 IDE なり command line なりを落としても、訓練の結果は保存されているので大丈夫だ。

⑧ ネットワークの評価

```
>> dlc.evaluate_network(config_path,shuffle=[1], plotting=True)
```

訓練データに対してトラッキングの結果を返す。自分が打った点と、ネットワークが予測した点が重ねられたファイルが labeled-data フォルダに作成される。ここでちゃんと点が打たれていなかったら、まずうまくいかない。

⑨ ビデオ解析と結果の図示
結果の出力を行う

>> dlc.analyze_videos(config_path,[‘/kokoni/videono/path/woireru.avi’], shuffle=1, save_as_csv=True, videotype=.avi)

ビデオのトラッキング結果が、python の pandas 形式と csv で出力される。R や julia など他の言語で解析をする場合、save_as_csv はデフォルトでは False なので True にしておいた方が良い。
結果の図示のための関数もある。

>> dlc.plot_trajectories(config_path,[‘/kokoni/videono/path/woireru.avi’])

matplotlib (python の描画パッケージ) で描かれた結果が、出力される。
とはいえ、座標データを自分で解析をする人は、やらなくても構わない。

逆に、こっちはやっておいた方が良い。
>> dlc.create_labeled_video(config_path,[‘/kokoni/videono/path/woireru.avi’])

create_labeled_video() 関数は、トラッキングした座標を元の動画に重ねて、トラッキング動画を作成する関数だ。動画は video フォルダに保存される。うまくいけばかなりクールな動画ができあがるので、学会発表で使えばモテる。モテる必要のない人も、トラッキングがうまくいっているのかが一目瞭然でわかるので、やっておいた方が良い。

⑩ 外れ値フレームの検出と修正
残念ながら、一発で全てのフレームが完璧にトラックできることは少ない。
そこで、修正が必要になる。具体的には、トラッキングがうまくいかなかったフレームを取り出してきて、そのフレームを手動で修正する。修正したフレームを学習データに追加して、再学習を行う。
外れ値フレームの検出は、次の関数で実行できる。

>> dlc.extract_outlier_frames(config_path,[‘videofile_path’])

ヘルプを見るとこの関数は引数が多いが、outlieralgorithm という引数が特に重要だ。ここを変えると、外れ値の検出方法が変わる。

'fitting'：ARIMA モデルという時系列解析をかけて、点の残差が大きいところを拾う
'jump'：とにかく移動量が大きいところを拾う
'uncertain'：トラッキングした場所の尤度が低いところを拾う
外れ値を拾ったのち、そのフレームの点を自分で修正する。

>> dlc.refine_labels(config_path)

GUI が開くので、Load Labels し、おかしい点をドラッグして直す。
点を右クリックすると削除することができる。点が隠れて見えない場合は欠損させてしまおう。

修正した点を保存したら次の関数で元の学習データと修正したデータを統合する。

>> dlc.merge_datasets(config_path)

dlc.merge_datasets() がうまくいったら、⑦ に戻って、もう一度ネットワークを訓練する(同じコードを実行し直せばよい)。この再学習を踏めば、トラッキングの精度は大幅に上がるはずだ。
あとは、自分の望む精度になるまで 7-10 の工程を繰り返す。

ハマりそうなところとその解決策 (松井もよくわかっていないところもあり)

・まず、CUDA とか cuDNN とかよくわからない

CUDA は GPU を使った計算を行うための開発環境で、ようは GPU とプログラムを繋いでくれるソフトウェアだ。例えば、tensorflow を用いた計算を GPU で行うには、CUDA が必要になる。cuDNN は CUDA を用いたディープラーニングのライブラリらしい。

この辺あたり参照

https://www.souya.biz/blog/2016/04/26/cuda%E3%81%A8%E3%81%AF/
https://www.quora.com/What-is-CUDA-and-cuDNN

とはいえ、ユーザーレベルではあまりに気にしなくていい。とにかく適切なバージョン (これが難しいわけだが) のものをインストールしておけばいい。

・import deeplabcut がうまく回らない

何か依存パッケージが足りない場合、そのパッケージ名がエラーメッセージで表示されるはず。そのパッケージ名をコピーして、コマンドプロンプト上で 'pip install パッケージ名 ' をするとうまくいくかもしれない。

・GPU を認識しない。device_lib.list_local_devices() で GPU が表示されない。

PC を再起動したり、IDE を一度落とすと認識したことがあった。
GPU と CUDA のバージョン次第で認識しないこともあるようだ。例えば、RTX 2080 では CUDA 9.0 はエラーが返ってくる。

・refine labels で間違って点を削除してしまった！

残念ながら、現バージョンでは refine の段階で消してしまった点は元に戻せない。気になる場合は一度 Quit してもう一度最初からやり直すしかない。