令和二年度卒業論文

追跡ツール「UMATracker」への深層学習による検出器の導入

宮崎大学 工学部 情報システム工学科

比江島　惇

指導教員　椋木　雅之 教授

目次

1　はじめに

農学、医学の分野では、遺伝的変異、薬物投与等の影響を知るための実験動物として小動物(マウス)が広く利用されている。動物実験では、それらの影響を知るために、マウスの行動観察が行われている。

従来、マウスの行動観察は、研究者たちの手作業によって行われていた。この場合、観察者に多大な負担がかかるだけでなく、その観察結果も観察者の主観が入ってしまい定量的なデータが得られないという問題があった。その問題を解決するため、画像処理技術を用いた様々な追跡ツールが提供されてきた。しかし、それらのツールは、背景減算や閾値処理などといった古い手法をベースにしているため、同時に複数のマウスの追跡を行うとあまり良い精度が得られないものとなっている。そのため、それらの追跡ツールは研究者たちに受け入れられず、結局行動観察を手作業で行うという現状がある。

そこで本研究では、従来の追跡ツール「UMATracker」[1]に、深層学習による検出器を導入することで、従来の追跡ツールの複数のマウス追跡の精度改善を図った。

2 UMATracker

2.1 小動物の追跡ツール

本研究では、従来の様々な追跡ツールを調査し、それぞれの性能を評価し比較した。表 1 は、pathtracker[2]、 MOTHe[3]、ezTrack[4]、Tracktor[5]、UMATracker[1]の計 5 つの追跡ツールを用いて性能を比較した結果を表している。

pathtracker は、単体の追跡を対象とした追跡ツールである。追跡結果として、対象の位置、距離、速度、移動の軌跡などの統計情報を出力する。本研究では、複数体の追跡を対象としているため除外した。

MOTHe は、物体検出に畳み込みニューラルネットワークを用いた追跡ツールである。対象の種類、動画中の対象のピクセルサイズの変化、不均一な背景への対応などの様々な検出問題を解決できる。実際に本研究の対象であるマウスの画像を 1500 枚以上学習させたところ、作業量に見合った追跡結果が得られなかった。よって、扱いにくいツールであると判断し除外した。

ezTrack は、単体の追跡を対象としている。前処理として対象が存在しない参照フレームを生成し、それと動画中の各フレームの差で、対象の位置を把握し追跡する背景減算とよばれる手法を用いている。pathtracker と同じく、単体の追跡を対象としているため除外した。

Tracktor は、対象の検出に適応的閾値処理を用いた追跡ツールである。前処理として、適応的閾値処理をする領域サイズ、閾値、追跡対象の最小面積、追跡対象の最大面積の 4 つのパラメータを設定して追跡する。精度の高い追跡をするには、これら 4 つのパラメータの組み合わせを試行錯誤する必要がある。しかし GUI が無いため、試行錯誤したパラメータの組み合わせが良いものなのか、追跡結果が得られるまで分からないという欠点がある。よって、扱いにくいツールであると判断し除外した。

以上の実態から従来の追跡ツールの性能を評価した。GUI の有無、検出の精度に関わるパラメータ設定の難易度、オクルージョン(追跡対象が他の物体と重なること)による追跡失敗への対応の点で優れていることから、UMATracker をベースにした。

表 1 : 従来の小動物追跡ツールの性能評価表

項目	pathtracker[2]	MOTHe[3]	ezTrack[4]	Tracktor[5]	UMATracker[1]
複数体の追跡	×	○	×	○	○
作業の単純さ	○	×	○	○	○
GUI の有無	無	無	無	無	有
検出の精度	良	機械学習のデータ強化による	入力動画による	パラメータ設定による	パラメータ設定による
パラメータ設定の難易度	-	-	-	難しい	易しい
オクルージョンへの対応	×	×	×	×	○

2.2 UMATracker

UMATracker は、撮影した追跡対象の動画を入力として扱い、FilterGenerator 工程、Tracking 工程、TrackingCorrector 工程の 3 つの工程を経て、追跡を実現している。

2.2.1 FilterGenerator

FilterGenerator 工程では、追跡対象を検出するためのフィルタを作成する。あらかじめ撮影した追跡対象の動画を入力し、Google が提供するビジュアルプログラミング言語「Google Blockly」を用いて、OpenCV で提供されている閾値処理やノイズ除去、平滑化などの画像処理を組み合わせて検出フィルタを作成する。

FilterGenerator では、図 1 のように追跡対象が白画素、それ以外が黒画素となる 2 値画像を生成する画像処理手順、パラメータを与える必要がある。

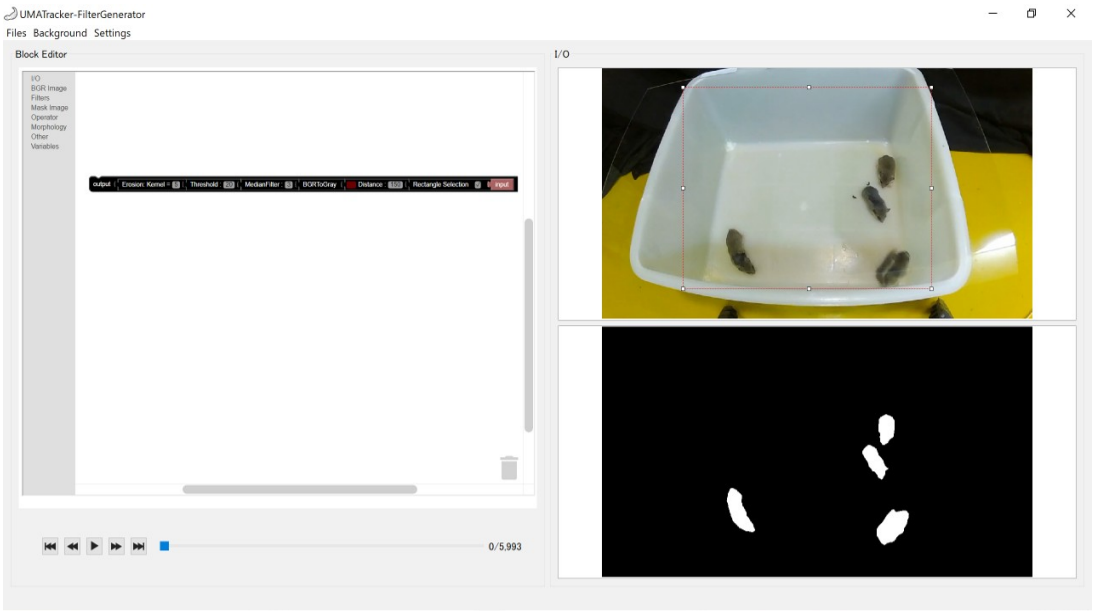


図 1 : FilterGenerator 工程

2.2.2 Tracking

Tracking 工程では、FilterGenerator 工程で作成したフィルタを用いて、追跡対象の動画から個体ごとの位置座標を抽出し、動画フレーム毎に個体識別し追跡する。Tracking 工程には、すでに 7 つの追跡アルゴリズムが実装されており、使用者の用途で使い分けることができ、さらに個人で追跡アルゴリズムを実装することもできる。

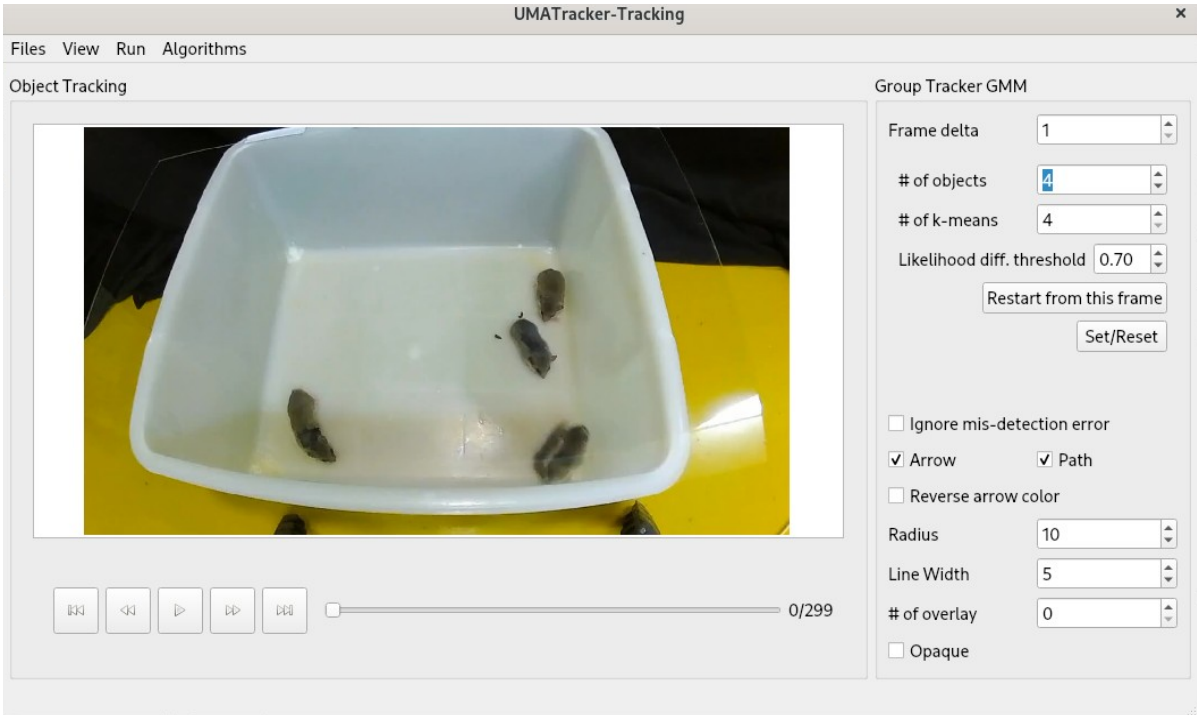


図 2 : Tracking 工程

2.2.3 TrackingCorrector

TrackingCorrector 工程では、Tracking 工程で得られた追跡結果と追跡対象の動画を見比べて、追跡の失敗があった箇所を手作業で修正する。

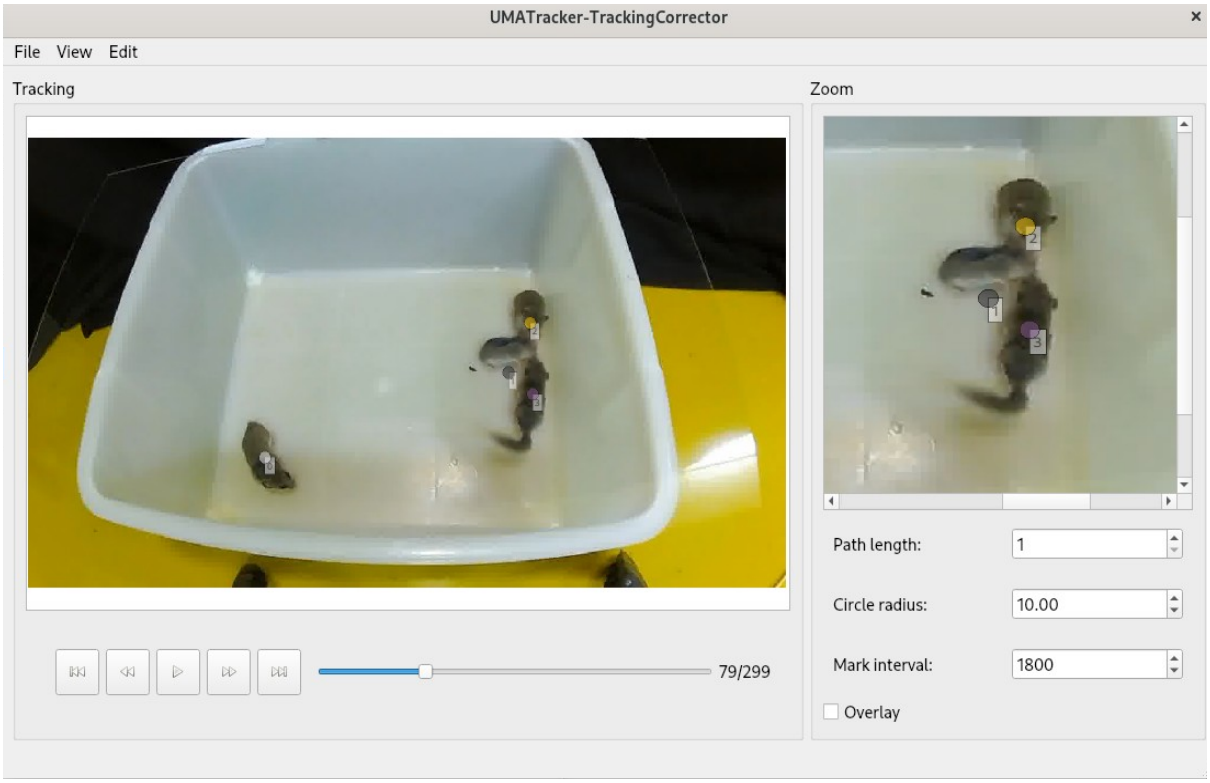


図 3 : TrackingCorrector 工程

2.2.4 追跡手法

UMATracker は、複数体の追跡アルゴリズムとして、主に「Group Tracker GMM」を用いている。フィルタを適用した追跡対象の動画から、白画素領域のデータを抽出し、それらのデータを k-means 法を用いて、ユーザが入力する追跡したい個体数だけグループ分けを行う。グループ分けされたデータを使い、個体毎に最適な割り当てを計算することで、動画内のフレーム毎の追跡を実現している。

3　従来ツールの問題点

3.1　UMATracker の追跡処理の課題

UMATracker は、2.2.1 節で述べたとおり、FilterGenerator 工程で白画素領域を抽出して追跡対象としている。

この手法の問題点は、追跡対象が複数いるとき、追跡対象同士が隣接または重なってしまうと、複数の物体を 1 つの物体として検出してしまうという点である。追跡対象であるマウスは、同じ環境に他のマウスがいると頻繁に接触をするため、従来の UMATracker の検出器では個体毎の検出が行えず、追跡は困難である。

3.2　OpenCV の追跡手法の課題

OpenCV[6]ライブラリは、TLD や CSRT など様々な追跡手法を提供している。それらの追跡手法は共通して、初期値に対象の動画から切り出した追跡対象の画像を与えて、動画のフレーム毎に既知の画像と似ている物体を探索して追跡するというものである。

このような特徴量検出をベースにした手法の問題点は、追跡対象の特徴が全個体似ているとき、対象の識別が困難になるという点である。追跡対象であるマウスの特徴は個体同士で似ているため、OpenCV の追跡手法では追跡は困難である。

3.3　深層学習による検出器の導入

本研究では、UMATracker の検出器に、深層学習を用いた検出手法である YOLO[7]を導入する。深層学習を用いた検出を導入することで、追跡対象同士が重なっても個体を区別することができる。さらに、YOLO の検出結果に対応した追跡アルゴリズムを実装し、UMATracker の複数体のマウスの同時追跡を改良する。

4　YOLO を用いた小動物追跡

本研究では、UMATracker に深層学習を用いた検出手法である YOLO を導入し、それを用いた小動物追跡を実現した。

4.1　YOLO

YOLO(You Only Look Once)は、リアルタイムオブジェクト検出アルゴリズムの１つである。画像を１度だけ畳み込みニューラルネットワークに入力するだけで、物体の検出と分類を行えるため、高速で検出結果を得られる長所がある。

YOLO は今日までアップグレードされ続けており、現在 YOLOv5 がリリースされている。YOLOv5 は、YOLOv4[8]の検出精度を保ったまま、その検出速度を向上させたものとなっている。本研究では、この YOLOv5 を UMATracker に導入した。

具体的には、UMATracker の FilterGenerator 工程に、YOLOv5 の機能を持たせたブロックを実装した。

4.2　YOLO に対応した追跡プログラムの実装

本研究では、ハンガリーアルゴリズム[9]により YOLO の検出結果を個体毎に割り当てることで、追跡を実現している。

4.2.1 追跡における同一性の定義

追跡において重要なのが、多くの検出結果の中でどれがその個体のものなのかを決める個体識別である。本研究では、動画フレーム間の検出ボックスの重心同士のユークリッド距離を評価値にして個体識別を行っている。

現在の動画フレームを t 、その前のフレームを $t-1$ とする。 t の検出ボックス 1 は $t-1$ の検出ボックスのどれと同一のものなのか決めるとき、検出ボックス 1 と $t-1$ の全ての検出ボックスのユークリッド距離を計算し（図 4 (a)）、値が小さい程、同一性が高いと定義した（図 4 (b)）。

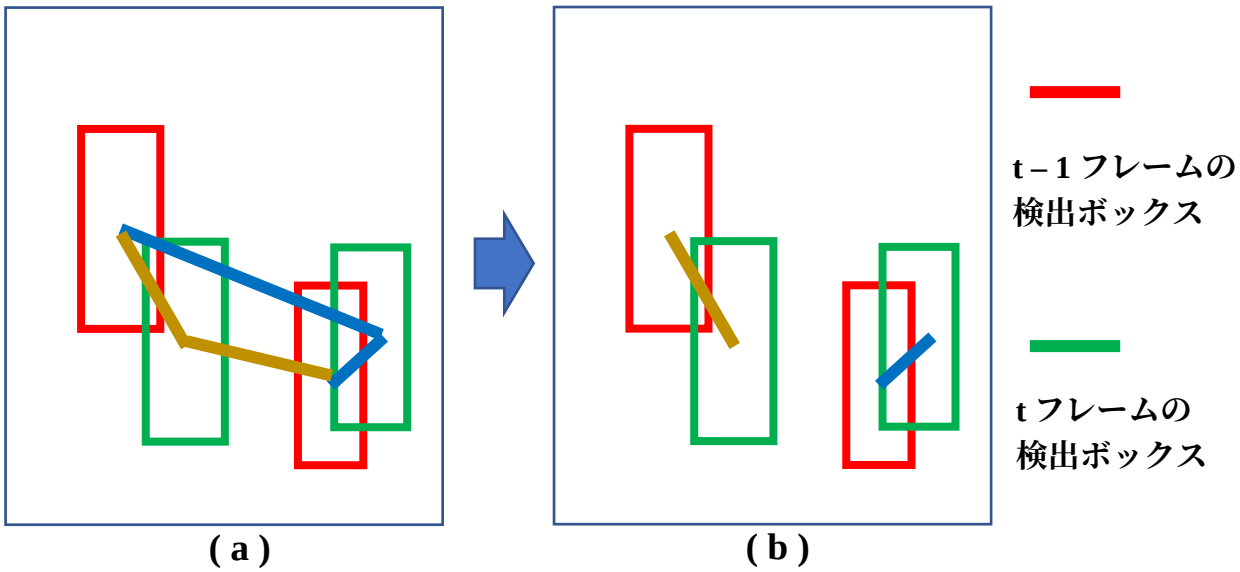


図 4 : ユークリッド距離を用いた同一性の定義

4.2.2 YOLO を用いたハンガリーアルゴリズムによる追跡

ハンガリーアルゴリズムは、 n 人に n 個の仕事を割り当てたいときに、最もコストの低い仕事の割り当て方を求める。その際、 n 人それぞれが n 個それぞれの仕事にかかるコストを並べたコスト行列が必要になる。

本研究ではコスト行列において、人をマウスに、仕事のコストを 4.2.1 節で述べた各 YOLO 検出ボックスのユークリッド距離に見立ててハンガリーアルゴリズムを適用する。

4.2.3 YOLO 検出結果の補填

YOLO の検出は、動画の全フレームで必ず検出に成功するわけではなく、複数の対象がいる中で 1 体だけ検出できていないといった検出漏れが発生する。ハンガリーアルゴリズムでは割り当てる仕事の数は、必ず人数以上でなければならないため、検出漏れはハンガリーアルゴリズムには致命的である。そこで、発生した検出漏れを前の動画フレームの検出ボックスで補填することにより解決する。

5　追跡実験

YOLO を用いたハンガリーアルゴリズムによる追跡の精度を調べるため、対象同士が重なることのない「オクルージョンなし動画」を追跡させ UMATracker の従来の追跡手法である Group Tracker GMM(以下 GMM)と追跡精度を比較した。また、実装した追跡手法を「オクルージョンあり動画」にも適用して、その結果を考察した。

5.1　実験準備

実験するために、YOLO にマウスを学習させるための訓練用画像、テスト用動画が必要になる。そこで、データ拡張、アノテーション、テスト用動画を準備した。

5.1.1 データ拡張

データ拡張とは、元画像を回転させたり、明るさを変えたりした画像を教師データに追加し、教師データを多様化させる方法である。

本研究では、テスト用動画とは別に訓練用動画を用意し、その訓練用動画からオクルージョンのシーンを含む 300 フレームを抽出して、それを訓練用の元画像として扱っている。それら元画像に回転、マスク付与、拡大縮小などの処理を織り交ぜてデータ拡張を行った。元画像 300 枚とデータ拡張済み画像 940 枚の計 1240 枚の訓練用画像を YOLO に学習させた。

5.1.2 アノテーション

アノテーションとは、機械学習の教師データを作成することである。訓練用画像のどこに対象がいるかを表す座標、その対象の名称であるラベルを全ての訓練用画像で作成する。本研究では、オープンソースのアノテーションツール LabelImg を用いてアノテーションを行った。

5.1.3 テスト用動画の準備

テスト用動画の元動画として、4 匹のマウスが映っている動画を準備した。その元動画から、対象同士が重なることのないシーン（オクルージョンなし動画）300 フレームと、対象同士が重なることがあるシーン（オクルージョンあり動画）114 フレームを抽出して、それらをテスト用動画として用いた。各フレームは 1280×760 の大きさで、マウスの大きさは 150×75 程度である。

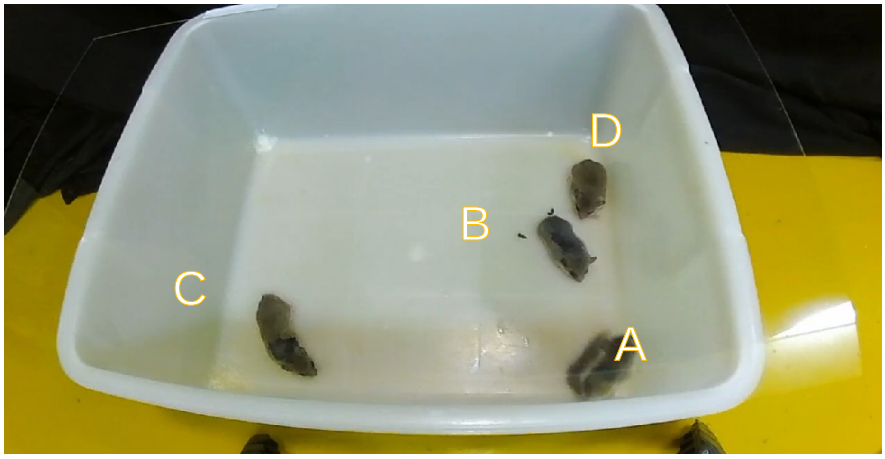


図 5 : テスト用動画の例

5.2 追跡結果

オクルージョンなし動画を GMM、YOLO を用いた追跡で追跡させた結果を表 2 に示す。また、オクルージョンあり動画を、YOLO を用いた追跡で追跡させた結果を表 3 に示す。

図 6 は、オクルージョンなし動画の GMM での各マウスの誤差の推移、図 7 は YOLO での各マウスの誤差の推移である。図 8 は、オクルージョンあり動画の YOLO での各マウスの誤差の推移である。

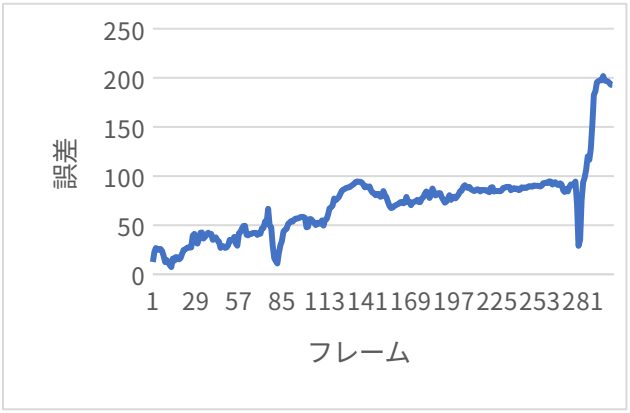
追跡誤差は、手作業で与えた追跡対象の正解座標と追跡結果の座標との差である。

表 2：オクルージョンなし動画での追跡誤差の比較

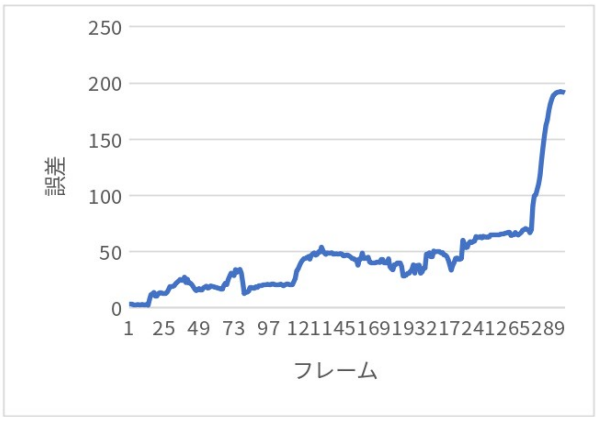
	GMM	YOLO
マウス A	71.26	7.20
マウス B	45.47	7.89
マウス C	16.13	17.81
マウス D	6.99	9.79
全体の平均誤差	34.96	10.67

表 3：オクルージョンあり動画での追跡誤差

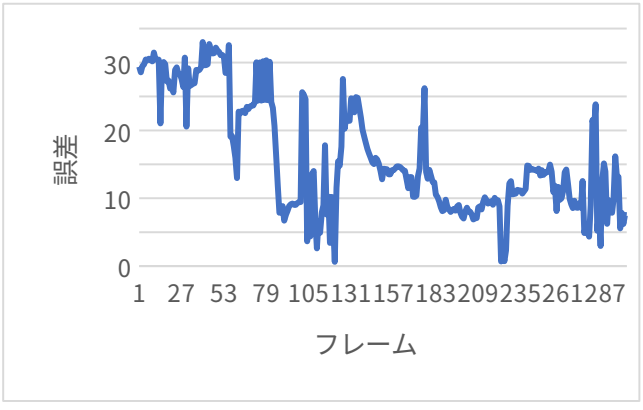
	YOLO
マウス A	8.76
マウス B	72.77
マウス C	8.12
マウス D	4.34
全体の平均誤差	23.50



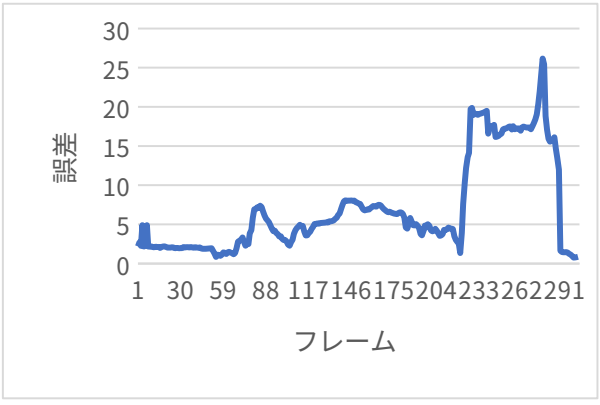
(a) マウス A



(b) マウス B

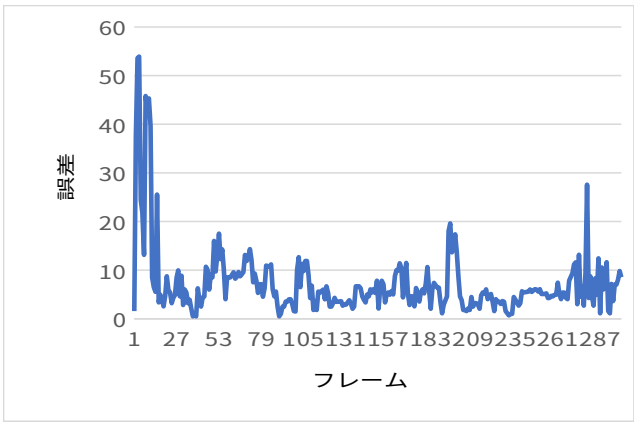


(c) マウス C

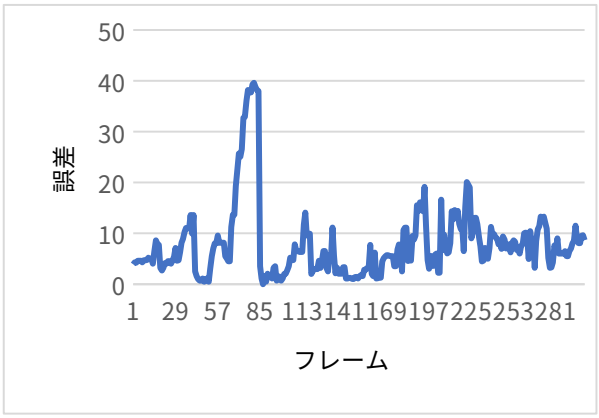


(d) マウス D

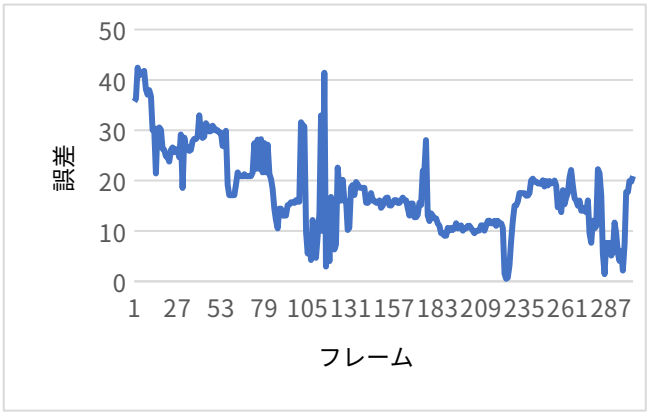
図 6 : GMM での各マウスの誤差の推移（オクルージョンなし動画）



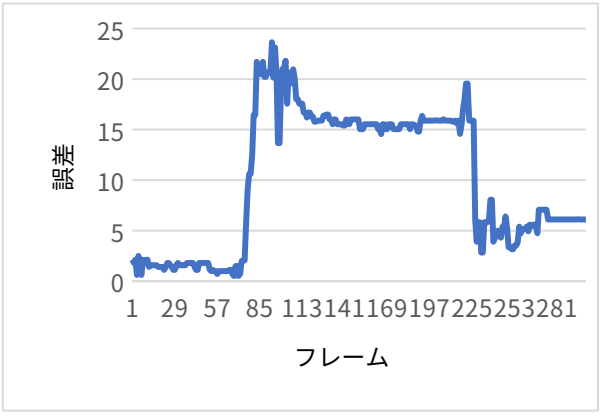
(a) マウス A



(b) マウス B

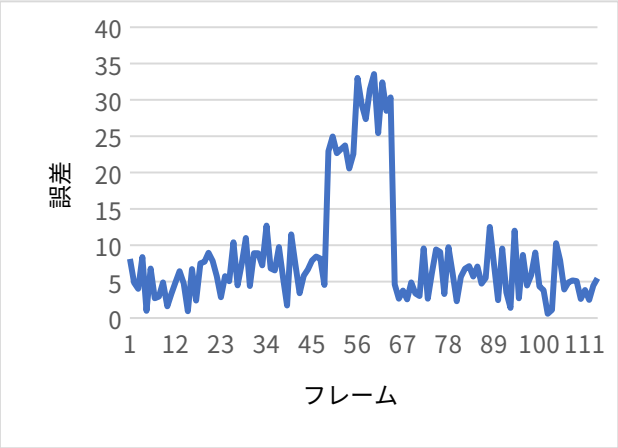


(c) マウス C



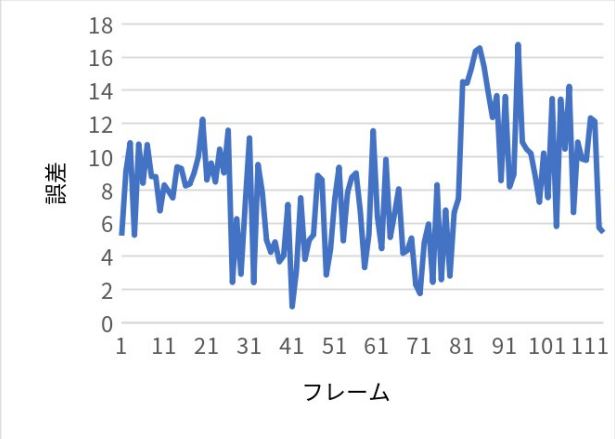
(d) マウス D

図 7 : YOLO での各マウスの誤差の推移 (オクルージョンなし動画)

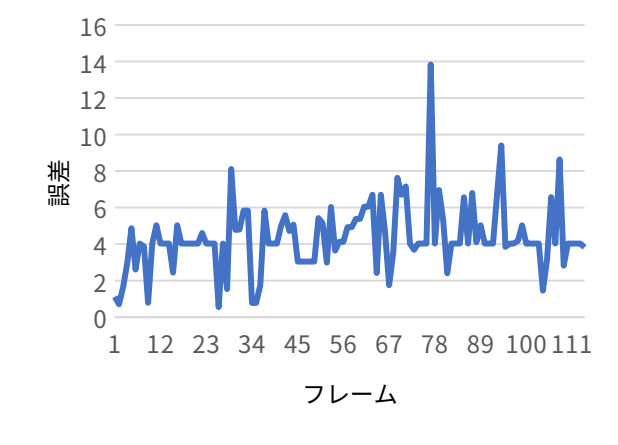


(a) マウス A

(b) マウス B



(c) マウス C



(d) マウス D

図 8 : YOLO での各マウスの誤差の推移 (オクルージョンあり動画)

5.3 考察

オクルージョンなし動画について、動画内の 120 フレーム付近でマウス A とマウス B が隣接するシーンがあった。GMM での追跡で、図 6 (b)の 120 フレーム付近の誤差を見ると急激に上昇している。この時点で 3.1 節で述べたように、隣接したマウス A とマウス B を 1 つの物体として検出してしまい、追跡が破綻していた。一方 YOLO での追跡で、図 7 (b)の同じ 120 フレーム付近の誤差を見ると、誤差を 15 以内に収めることができている。よって、実装した追跡手法により、追跡対象同士が隣接した時の追跡精度を改善できたと言える。

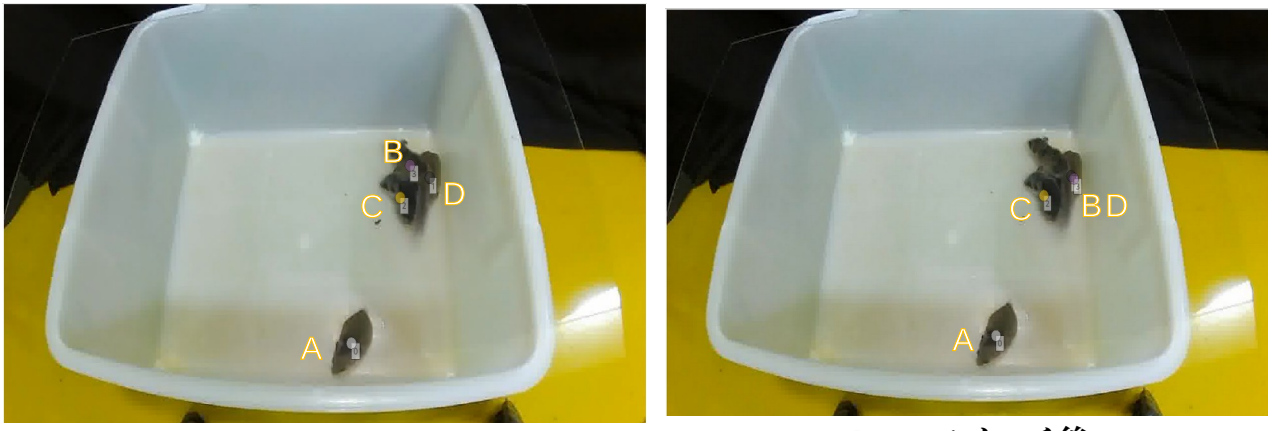
しかし、YOLO での追跡でも、図 7 (a)の 1 フレーム付近、図 7 (b)の 60 フレーム付近ではそれぞれ、誤差の急激な上昇が見られる。図 7 (a)の場合は、マウス A の尻尾の検出ボックスを追跡してしまったために起きた。これは、YOLO の誤検出により、マウス A の胴体と尻尾のそれぞれを一個体として検出してしまい、尻尾の検出ボックスの重心と前フレームのマウス A の検出ボックスの重心とのユークリッド距離が小さかった為である。この現象は、一個体に発生する複数の検出ボックスを一つにまとめる手法を加えることで解決できる。図 7 (b)の場合は、4.2.3 節で述べた YOLO の検出漏れが原因である。YOLO の検出漏れは、前フレームの検出ボックスで補っているため、場合によっては誤差が大きくなることは必然である。この YOLO 検出の補填がなければ、マウス B の 60 フレーム付近での追跡はできていないため、図 7 (b)は YOLO 検出の補填が有効に働いた結果と言える。

つづいて、マウス D は動画内で動いていないにもかかわらず、GMM よりも追跡誤差が大きいという結果になった。全てのマウスに言えることだが、YOLO の検出ボックスは動画全体で同じ形をとることはない。そのため、対象が動いていないにも関わらず、検出

ボックスの重心に多少のずれが発生する。これにより、マウス D の追跡誤差が GMM より劣るという結果になった。

オクルージョンあり動画について、マウス B の追跡誤差のみ大きいことが表 3 から分かる。対象同士が重なっても YOLO により対象を区別して検出し追跡できているのだが、重なっていたマウス B が離れた直後、離れたマウスの追跡が他のマウスに移ってしまう現象（ID スイッチ）が起きた（図 9 、図 8 (b) 80 フレーム付近）。この現象は、4.2.3 節で述べた YOLO の検出結果の補填が有効に働かなかったことで起きた。

この失敗は 2.2.3 節で述べた TrackingCorrector 工程で修正することができる。



(a) ID スイッチ前　　**(b) ID スイッチ後**

図 9: オクルージョンあり動画での ID スイッチ

6　教師データ数の違いによる追跡精度の変化

深層学習を用いた検出をするにあたり、欠かせないのが 5.1.2 節で述べたアノテーションである。本研究では、300 枚の元画像にアノテーションを施し、データ拡張で 1240 枚の訓練用画像を作成したが、その作業時間は 3 時間以上を超える膨大なものである。実用性を求めると、なるべく少ない訓練用画像で追跡を行えるようにしたい。そこで、教師データ数を変化させることで追跡精度がどのように変化するか実験した。

6.1　評価方法

5.1.3 節で扱っているオクルージョンなし動画 300 フレームを用いて、教師データ数を変化させることで追跡精度がどのように変化するか実験した。評価値として、5 章の追跡実験と同様に追跡結果の座標と正解座標との誤差を用いている。表 4 は、それぞれの教師データに含まれる元画像の枚数とデータ拡張済み画像の枚数を表している。

表 4 : 教師データに含まれる元画像とデータ拡張済み画像

教師データ数	元画像	データ拡張済み画像
10	**5**	**5**
50	**25**	**25**
100	**50**	**50**
150	**75**	**75**
300	**150**	**150**
600	**150**	**450**
1240	**300**	**940**

6.2　実験結果

表 5 はそれぞれの教師データ数で追跡させた時の、それぞれのマウスの追跡誤差の平均を表している。表 5 に記載している「-」は、追跡不可を表している。追跡不可は、図 10 のように動画の先頭フレームで、一部のマウスを追跡対象として検出できないために発生する。UMATracker では、追跡対象の数は指定できるが、それらのマウスが先頭フレームでどこにいるかは指定できない。そのため、YOLO での検出が正確に行えないと、図 10 のマウス A のように、動画中の同一のマウスを追跡対象として複数の追跡が行われてしまう。その場合のマウス A の追跡誤差は、マウス A の正解座標とマウス A に付いている追跡点それぞれの追跡結果との差を計算し、より小さい値のものを採用している。

表 5 : 教師データ数の変化による追跡誤差

教師データ数	マウス A	マウス B	マウス C	マウス D
10	検出なし	検出なし	検出なし	検出なし
50	31.42	-	218.31	-
100	27.33	-	21.98	-
150	39.50	-	17.43	62.39
300	39.22	-	15.76	16.63
600	100.95	31.43	15.35	10.83
1240	7.20	7.89	17.81	9.79

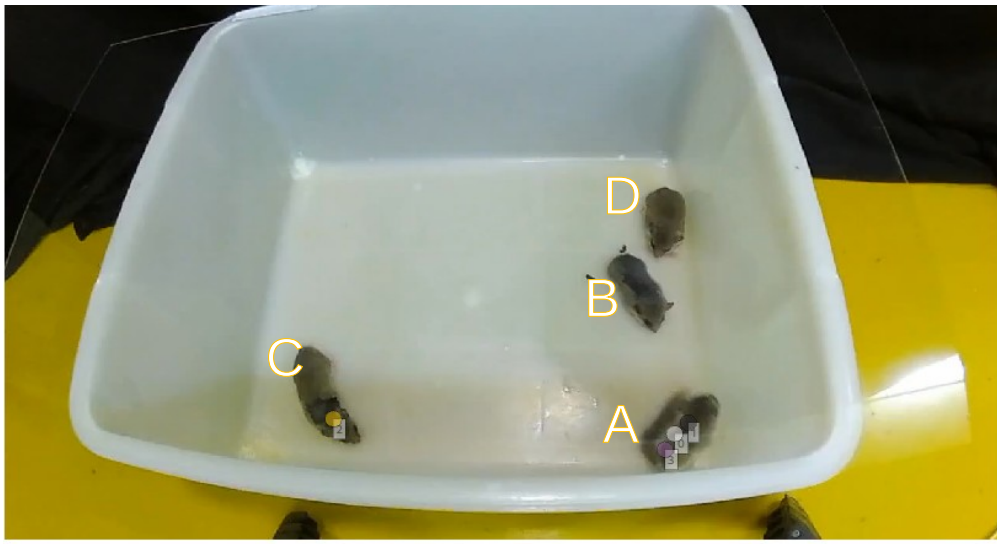


図 10: 先頭フレームの追跡の様子（教師データ数 50）

6.3 考察

表 5 から、教師データ数 1240 の時以外は、安定した追跡結果が得られていないことがわかる。教師データ数 10 の場合は、YOLO の学習不足により動画の全フレームでマウスを検出できなかった。教師データ数 50 以降は、図 11 のように動画の先頭フレームで全てのマウスを検出することができ、一部のマウスには複数の検出結果が出ている。しかし、図 10 のように先頭フレームで全てのマウスに追跡点を付けられず、追跡不可のマウスが生まれてしまった。

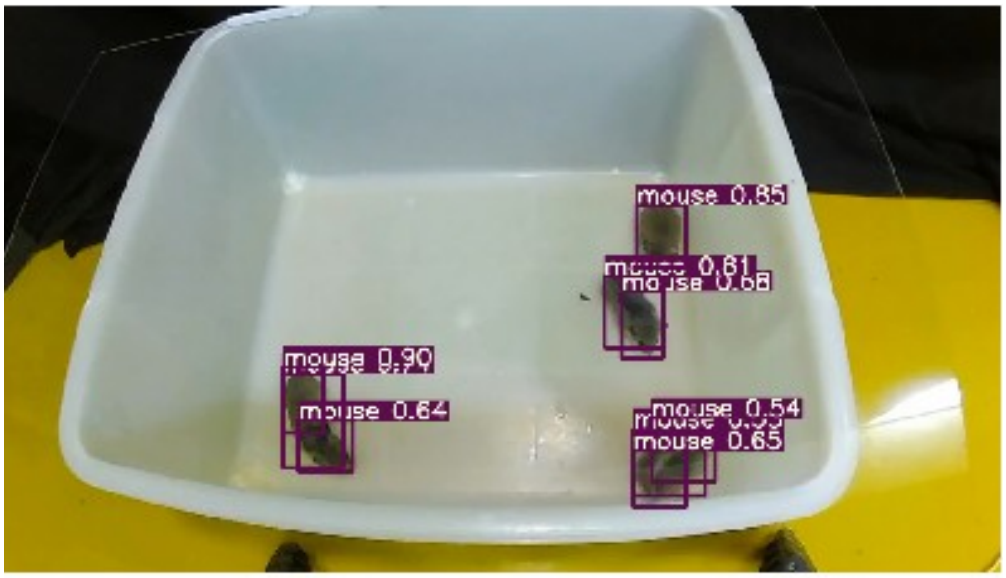


図 11: 先頭フレームの検出の様子（教師データ数 50）

今回のような追跡不可の原因は、UMATracker の仕様にある。UMATracker の Tracking 工程では、動画にフィルタを適用する前に、ユーザは追跡したい個体数を入力する必要がある。従来の UMATracker の追跡では、2.2.4 節で述べているようにフィルタから抽出した白画素領域を、グループ分けするために個体数を入力していた。しかし、本研究の YOLO を用いた追跡では、全てのマウスそれぞれに 1 個だけ検出ボックスがあるという前提で、個体数を入力している。そのため、図 11 のように 1 匹のマウスに複数の検出結果が出てしまうと、入力した個体数以上ある検出ボックスに追跡点がランダムに割り振られてしまい、追跡不可のマウスが生まれてしまう。この問題を解決するために今後の課題として、Tracking 工程で個体数を入力するだけでなく、動画の先頭フレームで追跡点をどの検出結果に割り当てるかを決めるインターフェースを追加することが挙がる。これを実現できれば、教師データ数 50 以降でも先頭フレームで全てのマウスに追跡点を割り当てることができる。しかし、教師データ数 50 では、YOLO の学習不足による検出漏れが多発してしまうため、たとえ先頭フレームで追跡点を割り当てたとしても追跡が困難である。表 5 より、教師データ数 150 あたりから、マウス B を除く全てのマウスに、先頭フレームで追跡点を割り当てることができている。つまり、教師データ数 150 以降からは検出が安定してきている可能性がある。よって、上記のインターフェースの改良を加れば、教師データ数 150 でも追跡できる可能性が高い。

7　おわりに

追跡ツール「UMATracker」の FilterGenerator　工程に深層学習を用いた検出手法である YOLO を導入した。また、Tracker 工程に YOLO を用いたハンガリーアルゴリズムによる追跡を実装した。YOLO を導入したことで、従来の UMATracker の課題であった追跡対象同士が隣接したときの検出を改善し、追跡の精度を大幅に改善することができた。

今後の課題としては、表 3 の結果からオクルージョンあり動画では ID スイッチが起きてしまい、追跡に失敗することが分かったので、YOLO 検出結果のオクルージョンを考慮した割り当ての改良による追跡誤差の改善が挙がる。今回の実験では、背景が無色の場合で追跡を行ったが、YOLO には背景と対象の物体を区別できる特徴がある。 その特徴を活かし、背景が無色でない場合での追跡も課題として挙がる。そして、6.3 節で述べた UMATracker の Tracing 工程のインターフェースの改良も行う必要がある。

謝辞

本研究を進めるにあたり、多くの方々のご協力を得ました。まずは、指導教員である椋木雅之 教授に感謝致します。お忙しい中、本研究の方針へのアドバイス、実験結果の考察のご協力、機材トラブルへの対応などをして頂きありがとうございました。次に工学部教育研究支援センター 技術専門職員の高塚佳代子 さん 、毎週のミーティングにご協力していただき、ありがとうございました。つづいて椋木研究室の方々、ミーティングでのアドバイス、研究室内でのご協力ありがとうございました。そして、研究背景の設定、マウスの動画の提供にご協力頂いた、農学部の坂本信介 先生、右京里那 様に感謝致します。最後に、本研究で用いた「UMATracker」の開発者である Osamu Yamanaka 氏、Rito Takeuchi 氏に感謝致します。

参考文献

[1]Osamu Yamanaka , Rito Takeuchi 『UMATracker : an intuitive image-based tracking platform』Journal of Experimental Biology (2018).

[2]Aaron M.T.Harmer , Daniel B.Thomas『PATHTRACKR : An R package for video tracking and analysing animal movement』Methods in Ecology and Evolution (2019).

[3]Akanksha Rathore , Ananth Sharma , Nitika Sharma , Vishwesha Guttal『Multi-Object Tracking in Heterogeneous environments(MOTHe) for animal space-use studies』BioRxiv (2020)

[4]Zachary T. Pennington , Zhe Dong , Yu Feng , Lauren M. Vetere , Lucia Page-Harley , Tristan Shuman , Denise J. Cai『ezTrack : An open-source video analysis pipeline for the investigation of animal behavior』Scientific Reports (2019)

[5]Vivek Hari Sridhar , Dominique G. Roche , Simon Gingins『Tracktor : Image-based automated tracing of animal movement and behaviour』Mehods in Ecology and Evolution (2019)

[6]https://learnopencv.com/object-tracking-using-opencv-cpp-python/

[7] https://deepsquare.jp/2020/09/yolo

[8]Alexey Bochkovskiy , Chien-Yao Wang , Hong-Yuan Mark Liao『YOLOv4: Optimal Speed and Accuracy of Object Detection』arXiv 2004.10934v1[cs.CV] (2020)

[9]https://qiita.com/m__k/items/ 8e2cb9067ec5d720c30d