機械学習を用いた
防犯カメラ画像からの流星検出

岡山大学 大学院 自然科学研究科 地球科学専攻
41M51201 原口 美悠

2023/02/10

要旨

自然科学研究科棟の屋上に設置した防犯カメラによって記録されたグレースケール動画 (1-4fps) から，流星を自動検出する推論モデルを機械学習を用いて作成し，その検出精度の検証を行った．機械学習で使用する訓練データは，2019 年 7 月から 2022 年 3 月に取得した動画の中から，原口卒論 (2021) の手法で抽出した流星と，動画を目で見て発見した流星を用いた．学習によって推論モデルを生成したら，それを用いて 10 夜分のデータで流星検出を行い，誤検出したものを訓練データに追加し，再度学習させた．誤検出した画像を加えては学習を繰り返すことによって，誤検出を減らし，検出器の性能を向上させることができた．学習を 6 回繰り返した結果，1 夜あたりの誤検出数として 10 程度を許容するなら，回収率が 0.8 以上になる流星検出ができるようになった．

Abstract

I developed an inference model for automatic detection of meteors from gray scale videos by using a deep learning. The training data used in the machine learning were extracted whether by automatic meteor detection (Haraguchi, 2021) or by the human eye, from videos. Once the inference model was generated by training, I ran inferences on images recorded during 10 nights. Then, the detections were examined, and the false positives are classified. I added the false positives to the training data, and restart training. By repeating the training with adding false positive images, the number of false positives are reduced. After 6 times of trainings, the model detects meteors with a recovery rate of 0.8 or better if 10 false positives per night are allowed.

第1章　序論

1.1　機械学習を用いた流星検出

流れ星は人を幸せにする (図 1.1)．私はたくさんの流星を見たいと思い，FS コースの先取りプロジェクト研究で，流星を観測するカメラを設置した (原口，2020)．卒業研究では，流星を自動検出するアルゴリズムを構築し，実際にカメラで撮影したデータに適用して流星の自動検出を行った (原口，2021)．しかし，卒業研究で構築した自動検出アルゴリズムは，流星でないものを流星と誤検出する数が多く，自動検出という目標を十分に達成したと言えるものではなかった．これを改善するため，本研究では機械学習を用いて精度よく流星を検出する検出器を開発した．

現在は，第三次 AI ブームの真っ只中にある (例えば，総務省 平成 28 年版情報通信白書)．2000 年代半ばから始まった第三次 AI ブームを牽引しているのは機械学習で，現在は知識を定義する要素である特徴量を AI が自ら習得するディープラーニングが実用化されている．特徴量とは，対象を認識する際に注目するべき特徴は何か，というのを定量的に表すもので，ディープラーニングは今まで人間が機械に教えていた特徴量を機械自ら発見する．ディープラーニングを活用することで，労働力を削減したり生産性を向上させることができ，ディープラーニングは現代社会で欠かせないものになっている．このように広く活用されている機械学習を用いることで，精度の良い流星検出を行うことができるのではないかと考えた．

図 1.1: 自然科学研究科棟の屋上で撮影された流星．撮影日時は 2019 年 8 月 13 日，03 時 03 分 58 秒から 03 時 04 分 00 秒に撮影された動画の比較明合成．

第2章　データ取得

撮影システムの詳細は，原口 (2021) に記述されている．撮影に使用しているカメラは市販の防犯カメラで，防水・防塵仕様のものである (図 2.1)．このカメラを自然科学研究科棟の屋上に設置して，2019 年の 7 月から毎晩夜空を撮影している (図 2.2).

図 2.1: 撮影に使用している防犯カメラ.

図 2.2: 撮影システム．旗竿を立てる台に取り付けたカメラ．後ろに写っている銀色のドームは岡山大学天文台．

カメラのコンポジット出力は，キャプチャーボードで UVC に変換されてから，Raspberry Pi 4 に読み込まれて記録される (原口，2021)．カメラ及びキャプチャーボードの出力と，Raspberry Pi 4 で記録する動画の解像度・フレームレートを表 2.1 に示す．

表 2.1: 解像度とフレームレート．

	カメラ	キャプチャー	Raspberry Pi 4
解像度	1920 × 1080	640 × 480 **	640 × 480
フレームレート	1-4fps *	25fps	4fps

*：夜間の典型的なフレームレート．フレームレートは明るさによって変動する．
**：カメラとキャプチャーボードで解像度の縦横比が異なるため，キャプチャボードで取り込まれた画像は縦横比が本来のものとは異なる．

第3章　流星検出

ディープラーニングを用いた物体検出を行うにあたって，本研究は Darknet YOLO(https://github.com/AlexeyAB/darknet) をフレームワークとして用いた．Darknet YOLO には，動画から物体検出するプログラムも付属しているが，ここでは静止画から物体検出を行った．

3.1　静止画の生成

自然科学研究科棟の屋上に設置したカメラで撮影した動画は，まずコマに分けた後，8 コマずつ比較明合成して静止画を作成した．動画は 4fps で記録しているため，8 コマは 2 秒間になる．カメラが出力するフレームレートは空の明るさによって変化し，夜間はだいたい 1-4fps である．本来であればコマの切り替えを検出して，フレームの切り替えに合わせて比較明合成するべきであったが，コマの切り替えを検出することが難しい場合もあり，2 秒ごとに比較明合成することにした．

最初は，この 2 秒ずつ比較明合成した画像を用いて機械学習を行い流星検出を試みたのだが，飛行機を大量に誤検出する結果になった．これは，飛行機も流星と同じく空を移動する光点で，2 秒を比較明合成した画像では，飛行機と流星の区別がつかなかったためである．

本研究では，飛行機と流星を区別できるようにするため，2 秒比較明合成した画像を数 10 枚程度集めて，時刻の情報を埋め込んだカラー画像を生成することにした．カラー画像は，各画素の輝度を集めた画像の比較明合成によって決定し，各画素の色は輝度が最大となったフレームが出現した時刻に対応して付ける．色と時刻の対応は，最大輝度となったのが 1 枚目であれば赤，前から数えて 1/3 枚目であれば緑，前から数えて 2/3 枚目であれば青，とした (図 3.1)．このように色を付けることで，同じ移動する光点であっても，移動速度の違いによって異なる画像が生成される．例えば，移動速度の速い流星はほぼ単色の線になり (図 3.2)，移

動速度の遅い飛行機はグラデーションのある線になる (図 3.3).

図 3.1: 本研究で使用した時刻と色の関係．左端が最初の時刻，右端が最後の時刻に対応する．すなわち，集めたフレームのうち最初のフレームが最も明るいなら赤，前から 1/3 枚目にフレームが最も明るいなら緑，前から 2/3 枚目のフレームが最も明るいなら青.

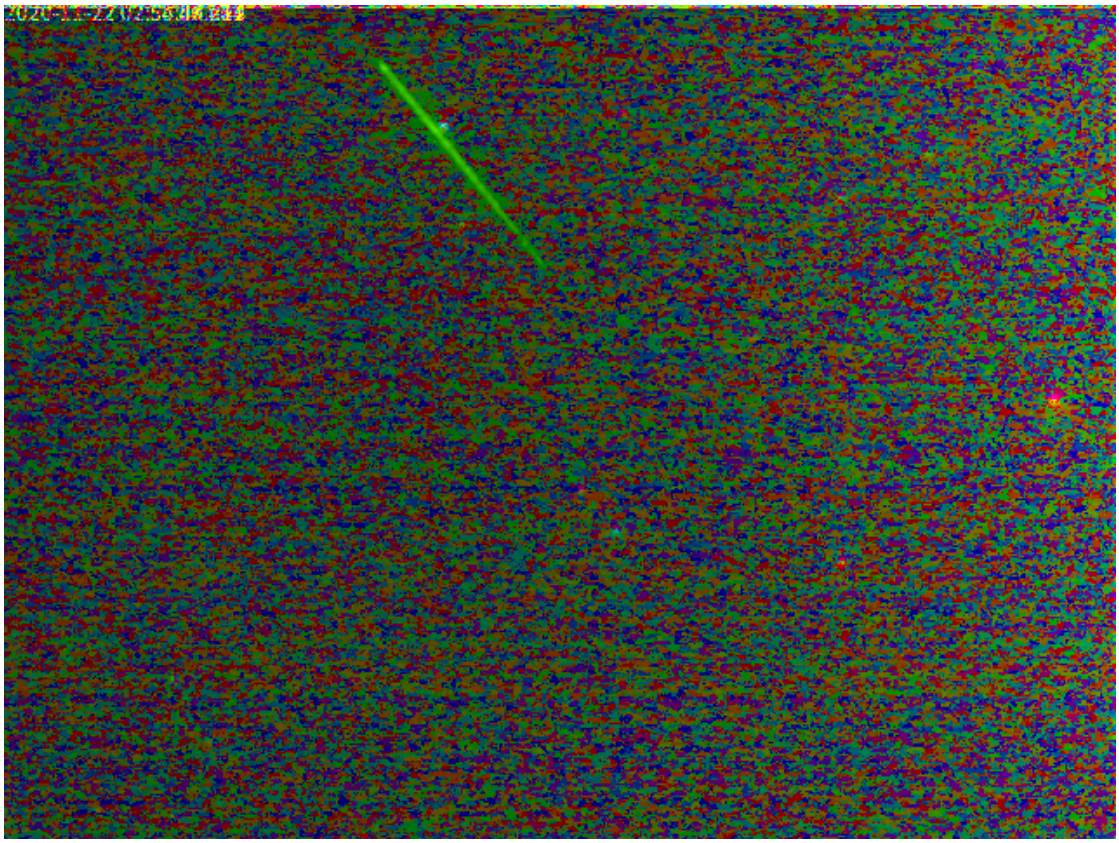

図 3.2: 比較明合成で生成した流星のカラー画像.

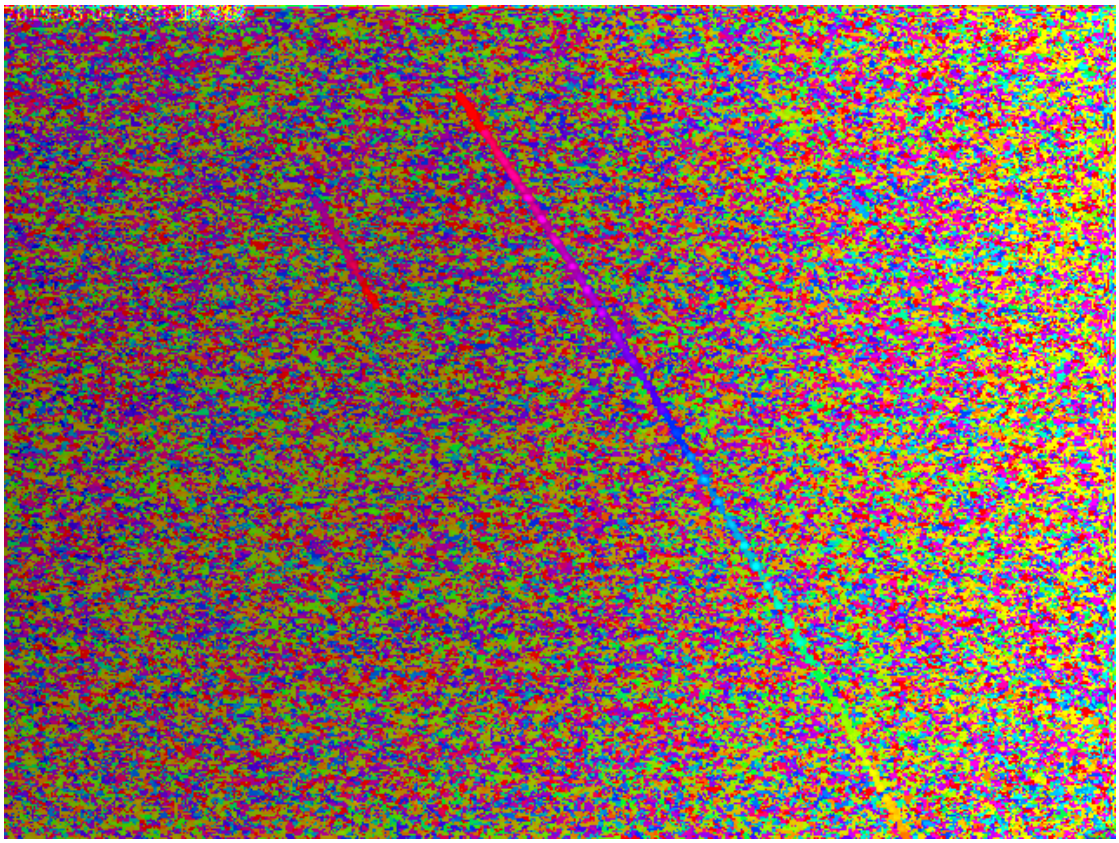

図 3.3: 比較明合成で生成した飛行機のカラー画像.

本研究はカラー画像の合成を，10 枚, 30 枚，90 枚の 3 パターンで行った．図 3.4 と図 3.5 は流星が流れた時間を切り取ったもので，図 3.4 には流星が写っているが，図 3.5 では流星が見えなくなっている．図 3.5 は，図 3.4 よりも長い時間にわたって撮影した画像を合成したため，流星が流れた場所を通過した雲によって上書きされたためである．カラー画像を合成する時に使う画像は，流星と飛行機の区別ができるくらいに長い時間で集めなければならないが，長すぎると移動する雲によって流星は消されてしまう可能性がある．

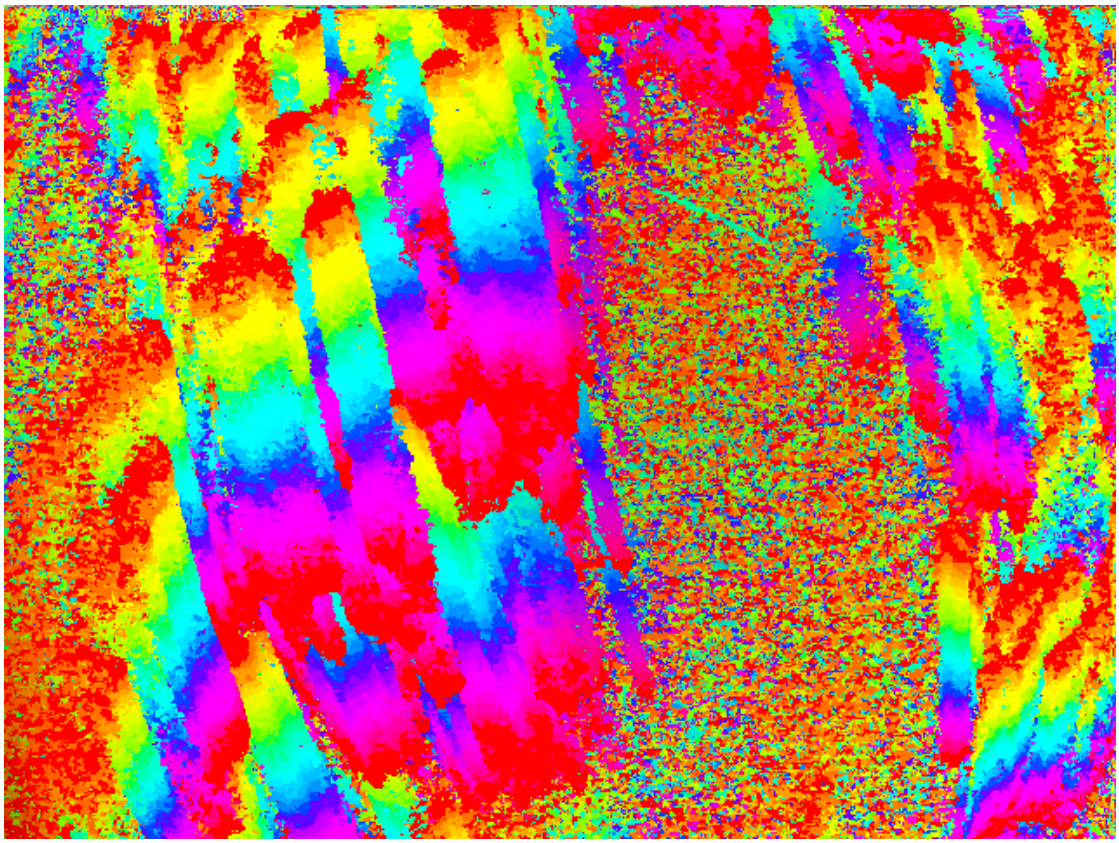

図 3.4: カラー合成 (30 枚)．画像中央上あたりに，流星 (緑色) がある．

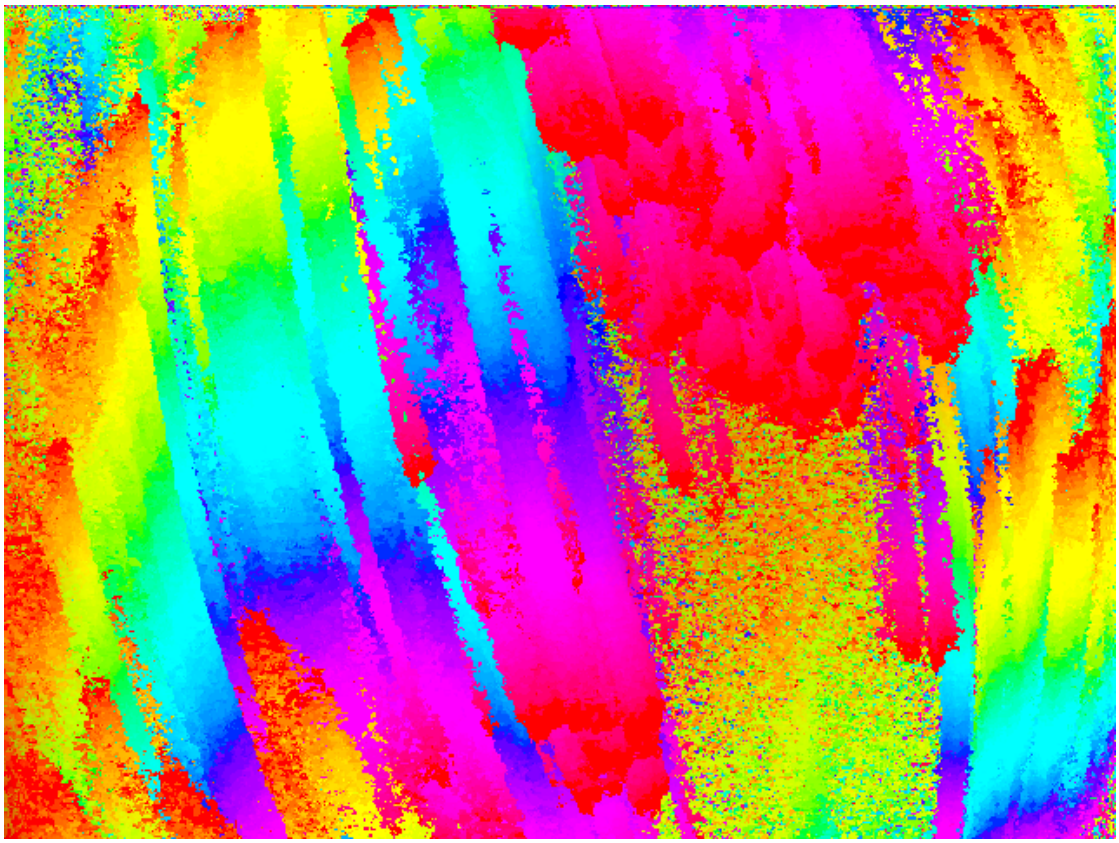

図 3.5: カラー合成 (90 枚)．図 3.4 に写っていた流星は，移動してきた雲によって隠されている．

3.2 機械学習を用いた流星検出

機械学習を用いた流星検出器の開発手順は，(1) 訓練データの作成，(2) 学習，(3) 検出器の評価，である．以下，それぞれについて説明する．

3.2.1 手順 1：訓練データ作り

訓練データとは，機械に正解を教えるものである．訓練データが大量かつ多様であるほど，検出器の検出精度は高くなる．本研究で使用した流星の訓練データは，自然科学研究科棟の屋上に設置したカメラで撮影された画像から生成した．2019

年 7 月から 2022 年 3 月に撮影された画像の中から，原口 (2021) の手法によって抽出した 271 個の流星と，動画を目で見て発見した 46 個の流星，計 317 個の流星を用いた．抽出した画像は，1 枚ずつ，画像のどこに何が写っているのかを，注釈として付与する作業を行った．この作業はアノテーションと呼ばれ，機械任せにすることはできず，全て人力で行う必要がある．本研究では，アノテーション・ツール LabelImg(https://github.com/heartexlabs/labelImg) を用いて，注釈を付与した．LabelImg はグラフィカルなインターフェースが用意されており，対象物をマウスで囲うと，座標などの情報が自動的に生成される (図 3.6).

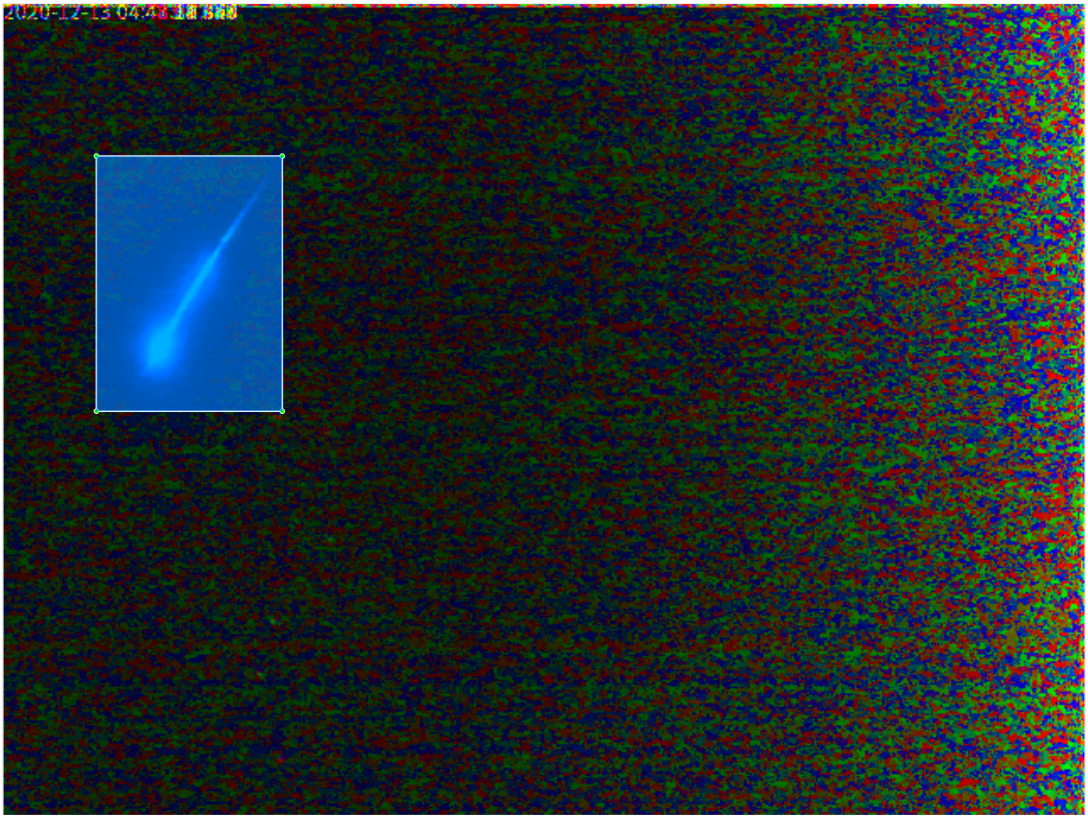

図 3.6: 流星に注釈を付与した例.

3.2.2 手順 2：学習

訓練データを与えて推論モデルを作らせることを，学習という．本研究はフレームワークとして，YOLOv3-tiny(https://github.com/ultralytics/yolov3) を用いた．

学習は訓練データを 2 つに分けて，片方 (train) を推論モデルを構築するために用い，もう片方 (valid) を推論モデルの評価に使う．train と valid に異なるデータを用いることで，未知のデータに対する推論モデルの性能を評価することができるのだが，本研究では集めることのできた流星データの数が少なかったため，train と valid で同じデータを用いて，学習させることにした．train と valid で同じデータを用いると，訓練データに特化した (訓練データでは良い推論を行うが，未知のデータには対応できない) 推論モデルが作られる可能性がある．これを避けるためには，訓練に使う流星データを増やす必要があるのだが，訓練データに使うことのできる流星の画像が少ないため，やむを得ず同じデータを用いた．

推論モデルの構築は，推論モデルを構築するパラメタの値を少し変更して推論を行い，推論結果と正解の比較に基づいてパラメタの値を修正する，といった反復 (iteration) を繰り返すことで，推論モデルの性能を上げていく．推論結果と正解の比較には valid のデータが用いられ，推論結果と正解のずれは loss と呼ばれる値によって表される．推論結果と正解のずれが小さいと，loss は小さくなる．図 3.7 は横軸に反復回数，縦軸に平均の loss をとったグラフである．反復では乱数が使われているため，loss の値はランダムに増減しているが，反復を繰り返すことによって loss は減少し，ある程度まで下がった後はほぼ一定の値をとるようになる．図 3.7 に示した学習を行った時は，反復回数が 10000 回のあたりまでは loss が減少し，これは反復によって推論の精度が向上したことを表す．10000 回を超えたあたりからは loss の値が減らなくなっており，これ以上の反復は推論の精度向上に寄与していないとわかる．

機械学習は大量の画像処理を行うため，高性能な計算機が必要となる．本研究で使用した計算機のスペックは表 3.1 の通りである．この計算機で GPU を用いて学習を行うとき，10000 回の反復にかかる時間は約 1 時間であった．

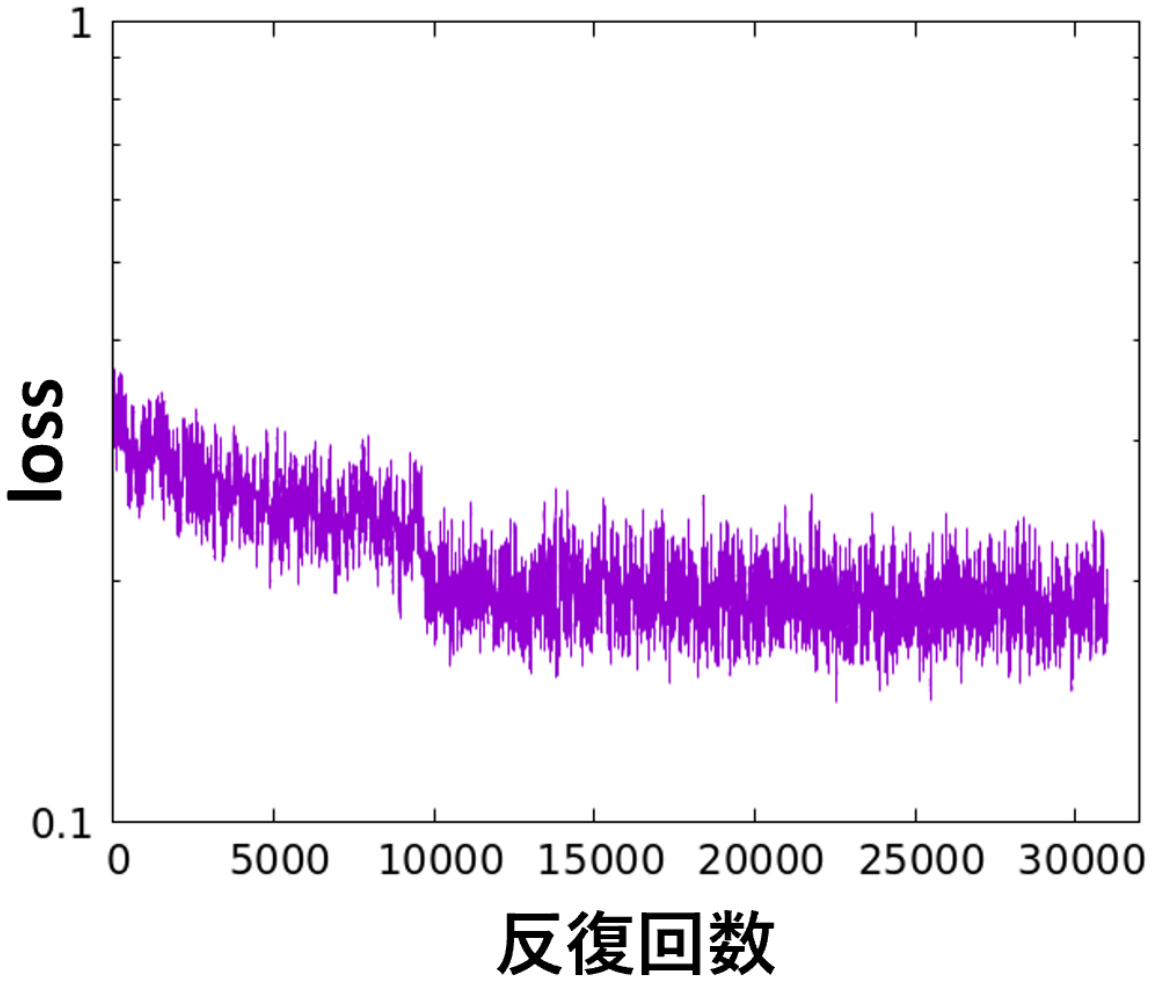

図 3.7: 反復による loss の変化.

表 3.1: 学習に使用した計算機.

CPU	Intel Xeon W-2235 (3.8-4.6GHz/6 コア/12 スレッド)
GPU	NVIDIA GeForce RTX 3090
メモリ	DDR4-2666 ECC Registered 8GBx4 (32GB)

3.2.3 手順 3：評価

検出結果の正解・不正解のパターンは 4 つに分けられる．TP(True Positive)：陽性を正しく陽性と分類したもの (真陽性)，FP(False Positive)：陰性なのに誤って陽性と分類したもの (偽陽性，誤検出)，TN(True Negative)：陰性を正しく陰性と分類したもの (真陰性)，FN(False Negative)：陽性なのに誤って陰性と分類したもの (偽陰性，見逃し)，である．このうち，TP と TN が正解，FP と FN が不正解である．流星検出で TP，FP，FN となった例を図 3.8 に示す．

TP，FP，FN を数えるためには，正解を知っている必要がある．そのため，10 夜 (2019 年 8 月 12 日，2019 年 8 月 13 日，2019 年 10 月 6 日，2019 年 10 月 22 日，2019 年 11 月 14 日，2019 年 11 月 29 日，2019 年 12 月 13 日，2020 年 1 月 29 日，2021 年 12 月 13 日，2021 年 12 月 14 日) の動画を最初から最後まで目で見て確認し，流星のあった時刻を記録した．流星検出器の評価は，この 10 夜，計 134 時間のデータを用いて行った．この 10 夜には，訓練データとして使用した 130 個の流星が含まれている．

検出結果は，TP，FP，TN，FN に分類した後，回収率と 1 夜あたりの誤検出数を求めた．回収率は流星を見逃すことなく検出した割合で，以下の式で計算される．

$$\text{回収率} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{3.1}$$

回収率は，見逃しが少ないほど 1 に近い値になる．すなわち，回収率が 1 に近く，誤検出が少ないものが，良い検出器である．

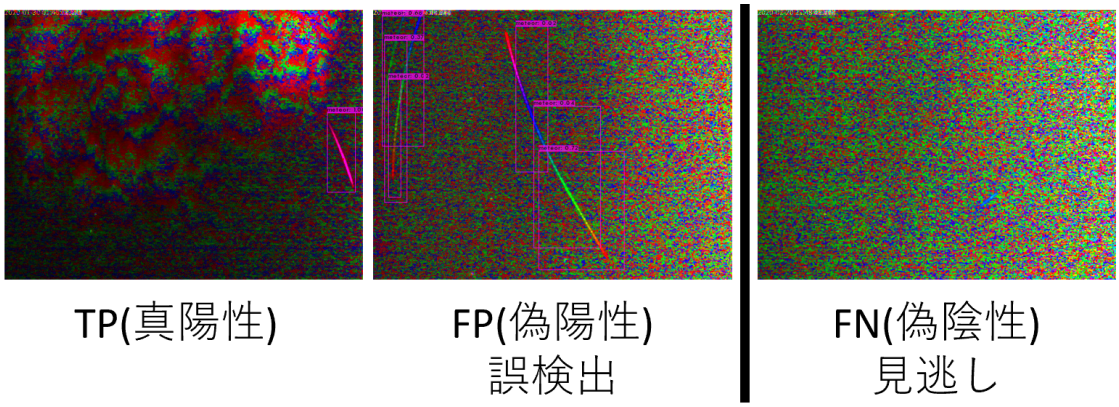

図 3.8: 検出結果の例．(左) 右端の中央に写っている流星を検出．上半分を広く覆っているのは雲．(中央) 飛行機を誤検出．(右) 中央やや右下に写っている流星 (青い線) を見逃し．

3.2.4 訓練データの追加

「手順3：評価」を行った時に検出された画像を用いて，訓練データの追加を行った．まず，検出されたものの中に動画を目視で確認した際に見逃していた流星があったため，それらに「流星」のラベルを貼って，訓練データに追加した．次に，誤検出したものを確認し，特に多かった，雲 (図 3.9)，飛行機 (図 3.10)，ノイズ (図 3.11) について，誤検出した画像を集め，それぞれにラベルを貼って訓練データに追加した．

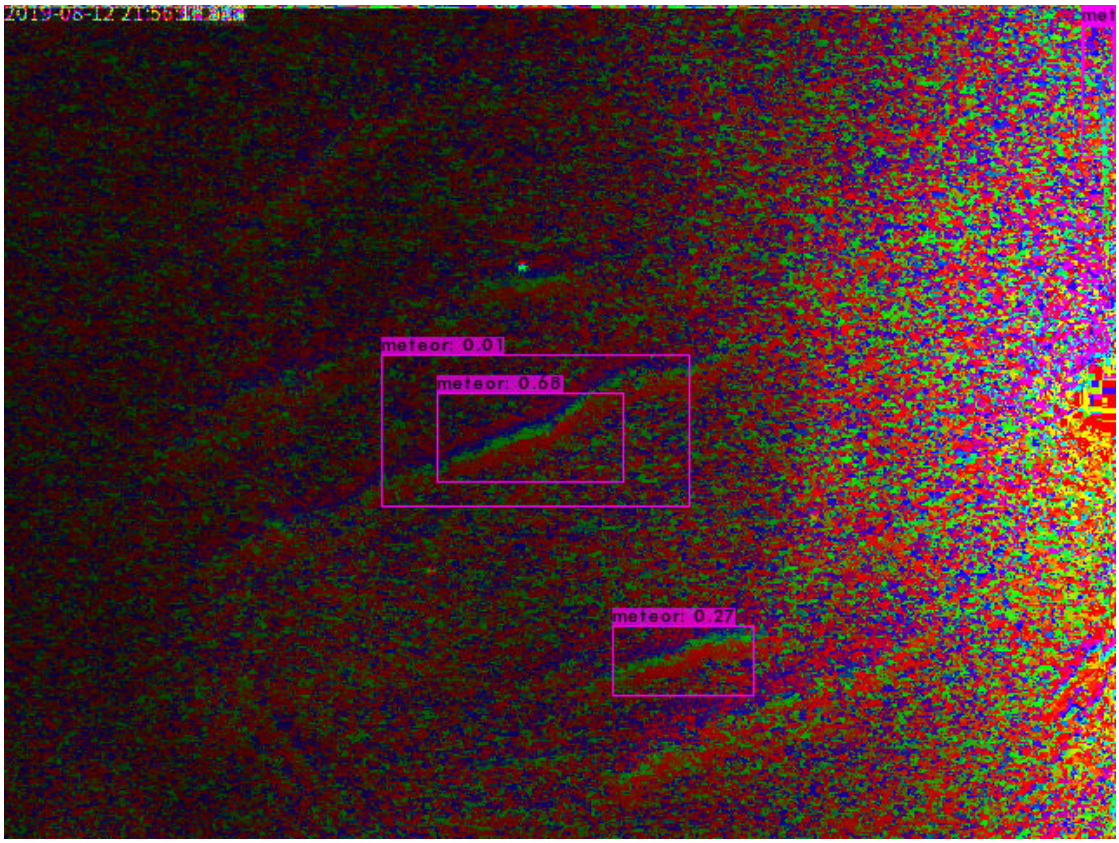

図 3.9: 雲を流星と誤検出した例.

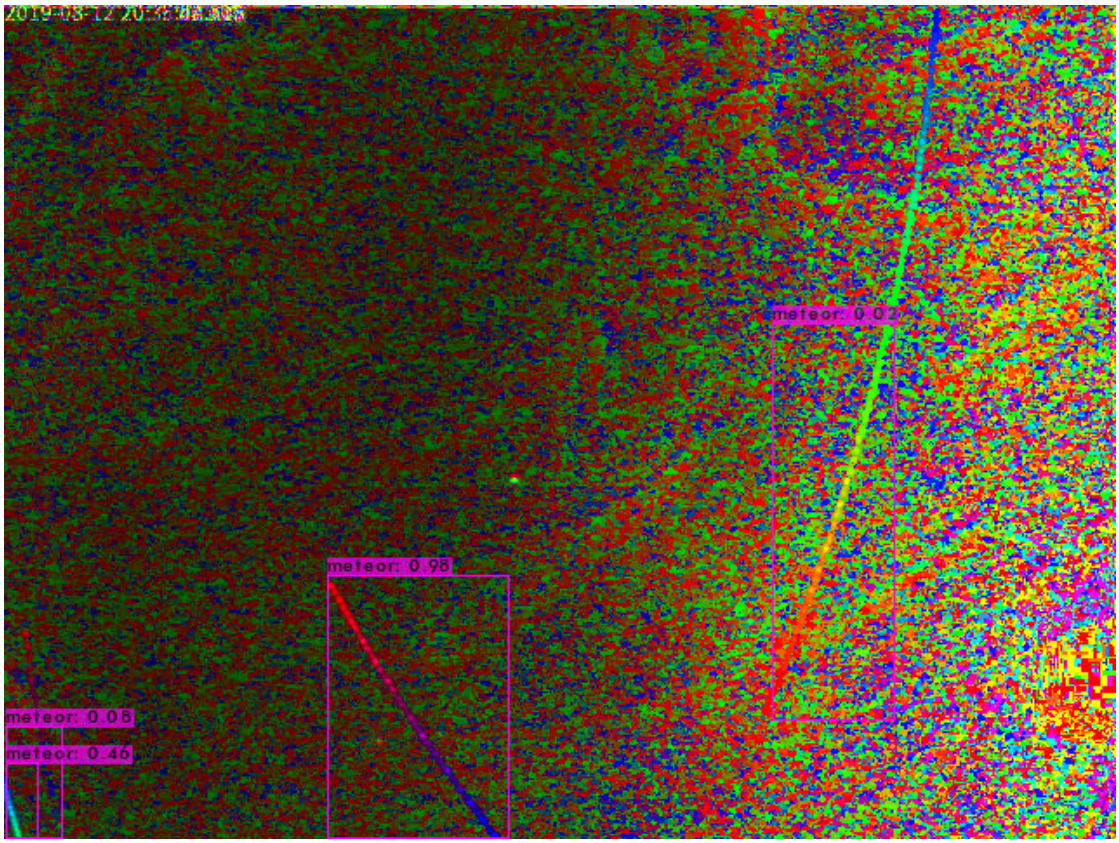

図 3.10: 飛行機を流星と誤検出した例.

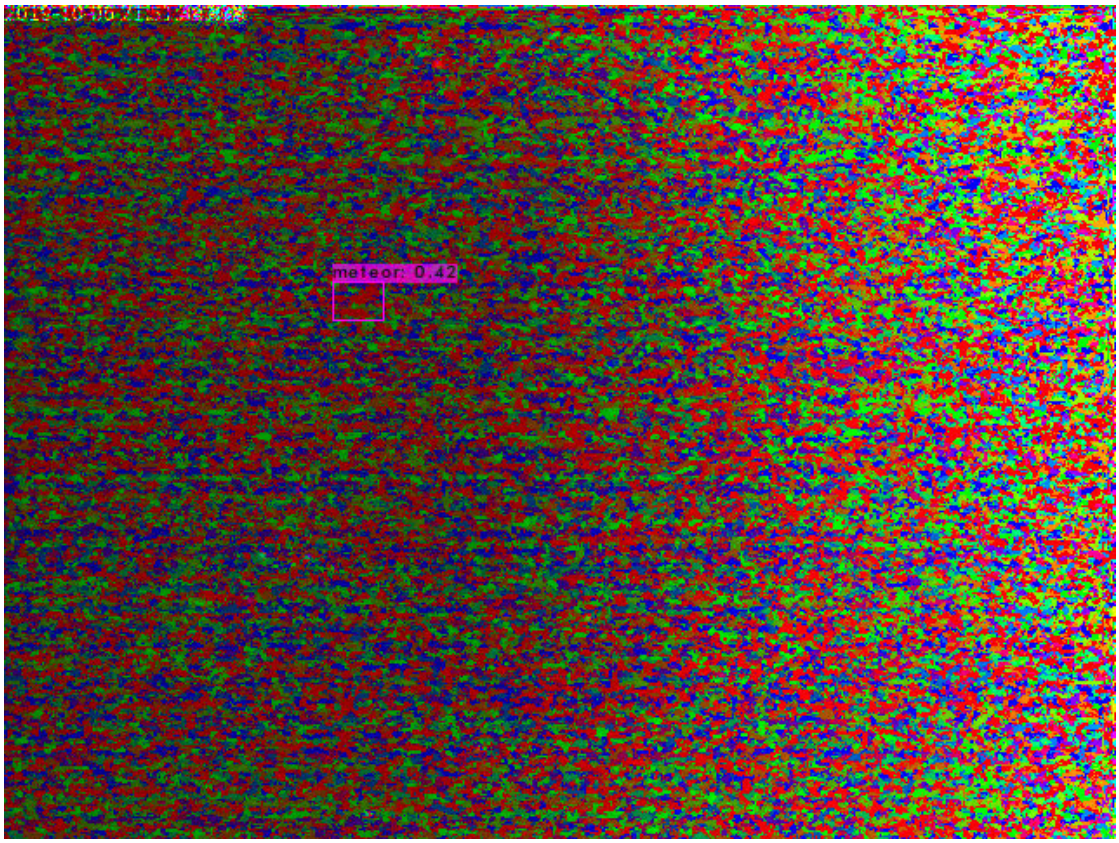

図 3.11: ノイズを流星と誤検出した例.

3.2.5 学習の繰り返し

学習が終わるたびに流星検出を行って，誤検出したものを訓練データに追加し，再度学習させることを繰り返した．各回の学習で使用した訓練データの枚数を，表 3.2 と表 3.3 に示す．2〜4 回目の学習では，訓練データを train と valid に 8：2 の割合で振り分けて学習を行ったが，これを行うと回収率が低下したため，5 回目以降は再び train と valid で同じデータを用いるようにした.

表 3.2: 学習に使用した訓練データの内訳 (30 枚カラー合成の場合).

学習回数	mteor	cloud	plane	noise	合計
1	317	0	0	0	317
2	323	152	214	112	795
3	327	156	230	137	844
4	328	173	241	144	880
5	329	184	250	151	908
6	332	198	256	159	939

表 3.3: 学習に使用した訓練データの内訳 (10 枚カラー合成の場合).

学習回数	mteor	cloud	plane	noise	合計
1	327	0	0	0	327
2	328	106	128	0	562
3	328	145	215	76	763
4	329	162	251	183	924
5	331	183	262	220	995
6	332	190	267	233	1021

3.2.6 検出器の評価

図 3.12 と図 3.13 は，各回の学習で生成した検出器を使って，流星検出を行った結果である．検出器は，回収率が高いものほど良く，誤検出が少ないものほど良いため，図の右下の角に近いものが良い検出器ということになる．色は学習回数を表し，図の各点は，流星とみなす確信度 (confidence) の閾値を 0.0 から 1.0 まで 0.1 刻みで変えた場合の結果を表している．確信度は，推論の確からしさを表す数値で，1 に近いほどより確からしい推論であることを示す．確信度の閾値を大きくすると，より確からしいものだけを流星として検出するようになり，誤検出を減らすことはできるが，回収率は下がってしまう．逆に閾値を小さくすると，やや怪しいものまで流星として検出するようになり，回収率は上がるが，誤検出が増えてしまう．閾値を変更することで，検出器の性能を調整することはできるが，閾値の調整は回収率と誤検出数のトレードオフになっている．

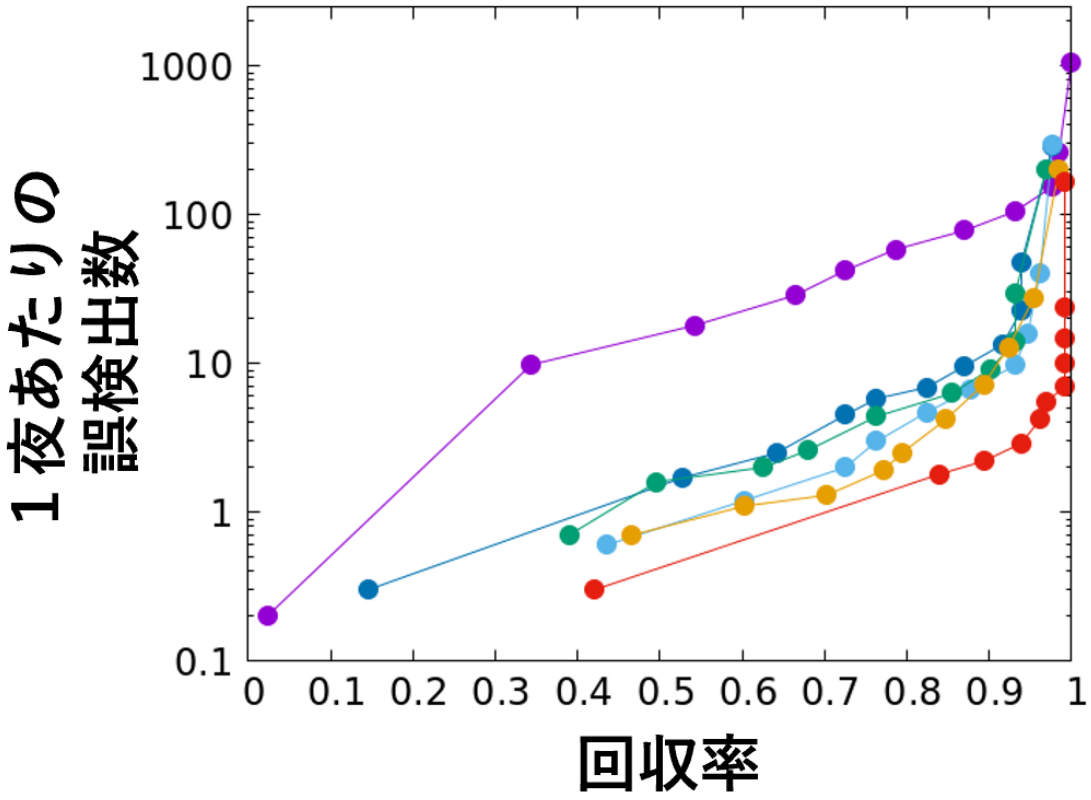

図 3.12: 流星検出器の評価．学習に使用した流星 130 個を含む 10 夜のデータで流星検出を行った結果．カラー合成は，2 秒ずつ比較明合成した画像を 30 枚使用した場合．色は学習回数を表しており，1 回目：紫，2 回目：青，3 回目：水色，4 回目：緑，5 回目：橙，6 回目：赤．

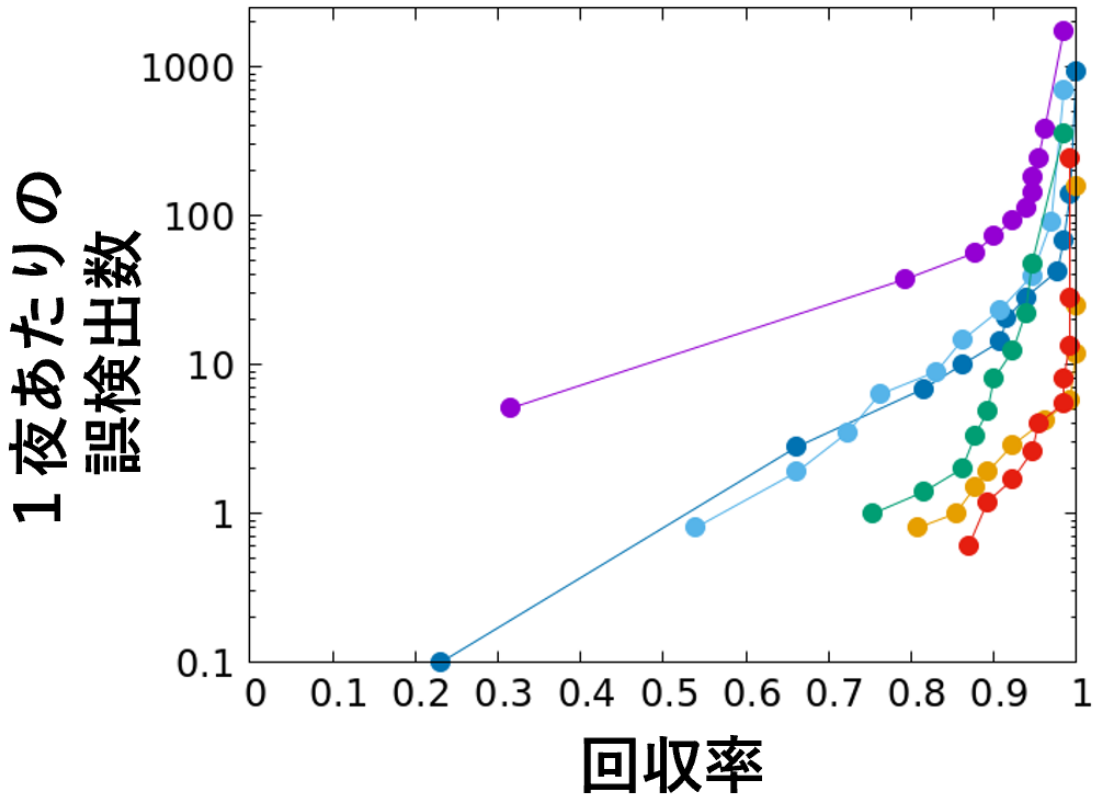

図 3.13: 流星検出器の評価．学習に使用した流星 130 個を含む 10 夜のデータで流星検出を行った結果．カラー合成は，2 秒ずつ比較明合成した画像を 10 枚使用した場合．色は学習回数を表しており，1 回目：紫，2 回目：青，3 回目：水色，4 回目：緑，5 回目：橙，6 回目：赤．

学習を繰り返した効果について見ると，学習回数が増すごとに，検出結果は右下の角に近づいており，学習によって検出器の性能は向上したことがわかる．誤検出の減少は，雲や飛行機などを訓練データに追加したことによる．一方の回収率の上昇は，新たに発見した流星を訓練データに追加したことによる．

カラー合成に使用した枚数は，30 枚よりも 10 枚の時の方が，検出性能は良いものになっている．とはいえ，差は大きくないので，検出性能の差が有意であるのかどうかはわからない．

3.2.7 検出器の試験

機械学習で生成した推論モデルの実力を評価するため，学習に使っていない 10 夜 (2022 年 4 月 4 日，2022 年 6 月 19 日，2022 年 7 月 20 日，2022 年 8 月 7 日，2022 年 8 月 12 日，2022 年 10 月 30 日，2022 年 11 月 3 日，2022 年 11 月 23 日，2022 年 12 月 9 日，2022 年 12 月 14 日) のデータを用いて，流星検出を行った (図 3.14，図 3.15)．データは 130 時間で，この中には学習に使っていない 17 個の流星が含まれている．

まず，学習に用いた流星を含むデータを用いた場合 (図 3.12，図 3.13) に比べて，回収率は下がった．回収率が下がった原因は，訓練データとして用意した流星の多様性が足りなかった場合と，過学習になっている場合の 2 つが考えられる．訓練データの不足が原因であるなら，新しい流星を訓練データに追加して補わなければならない．過学習は，訓練データに特化した学習をしてしまって，訓練データ以外のデータに対する適応が悪くなることである．過学習が原因であるなら，学習における反復回数を減らすなど，学習のやり方を修正することで，回収率を上げることができるかもしれない．

学習回数を増やすことは，検出器の性能を向上させることに若干の寄与があるように見えるが，その効果は学習に使った流星を検出した場合 (図 3.12，図 3.13) ほどではない．学習回数の効果が見えにくくなっている原因は不明だが，過学習が原因であるかもしれない．

カラー合成した枚数と検出器の性能を見ると，合成枚数を 10 枚とした方 (図 3.15) が，合成枚数を 30 枚とした場合 (図 3.14) よりも，良い結果を出している．合成枚数の違いがどのように影響したのか不明だが，訓練データとして用いた画像の枚数が少し違うこと (表 3.2，表 3.3) が，影響したのかもしれない．そうであるとしたら，誤検出したものをさらに訓練データに追加することで，検出器の性能を向上させることができるかもしれない．

最後に，本研究で作成した流星検出と，原口 (2021) の流星検出の性能を比較する．カラー合成を 10 枚で行った場合，本研究の流星検出は原口 (2021) よりも良い結果となった．回収率を原口 (2021) と同じ 0.76 にするなら，1 夜あたりの誤検出を 5.0 個まで減らすことができる．また，1 夜あたりの誤検出を 10 個程度まで許容するなら，回収率を 0.9 くらいまで上げることができる．

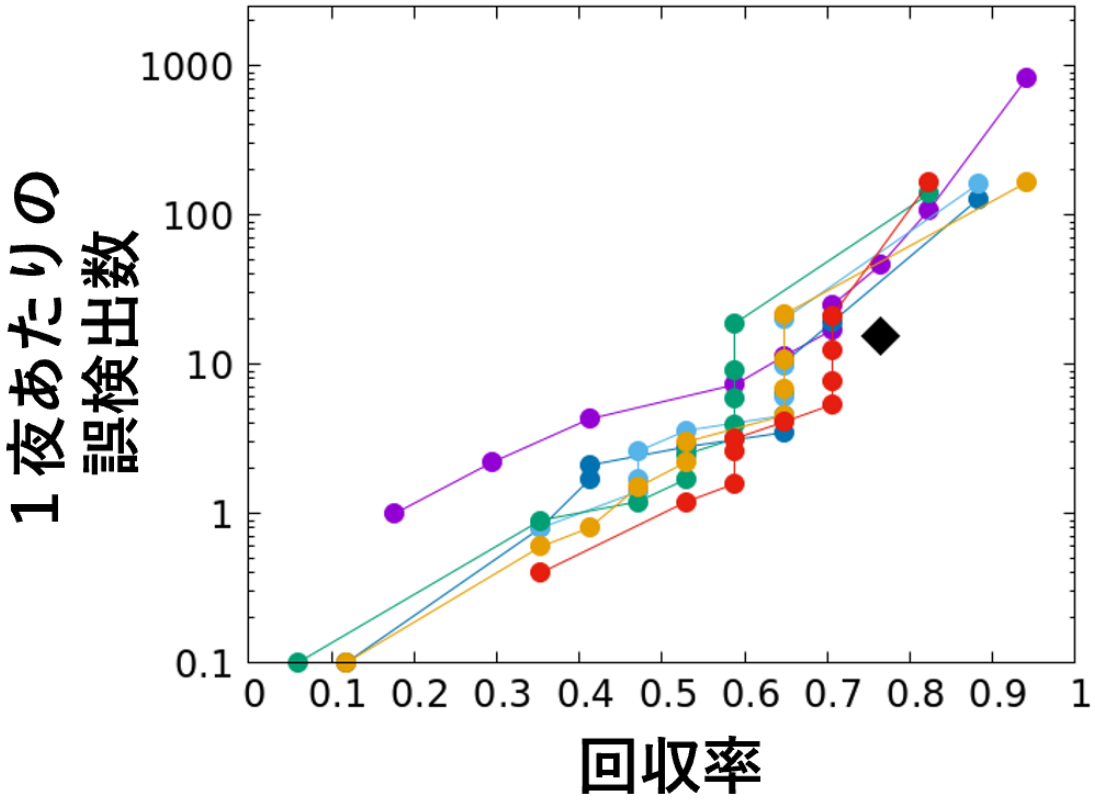

図 3.14: 流星検出器の評価．学習に使用していない流星 17 個を含む 10 夜のデータで流星検出を行った結果．カラー合成は，2 秒ずつ比較明合成した画像を 30 枚使用した場合．色は学習回数を表しており，1 回目：紫，2 回目：青，3 回目：水色，4 回目：緑，5 回目：橙，6 回目：赤．原口 (2021) の結果を黒い菱形で描いている．

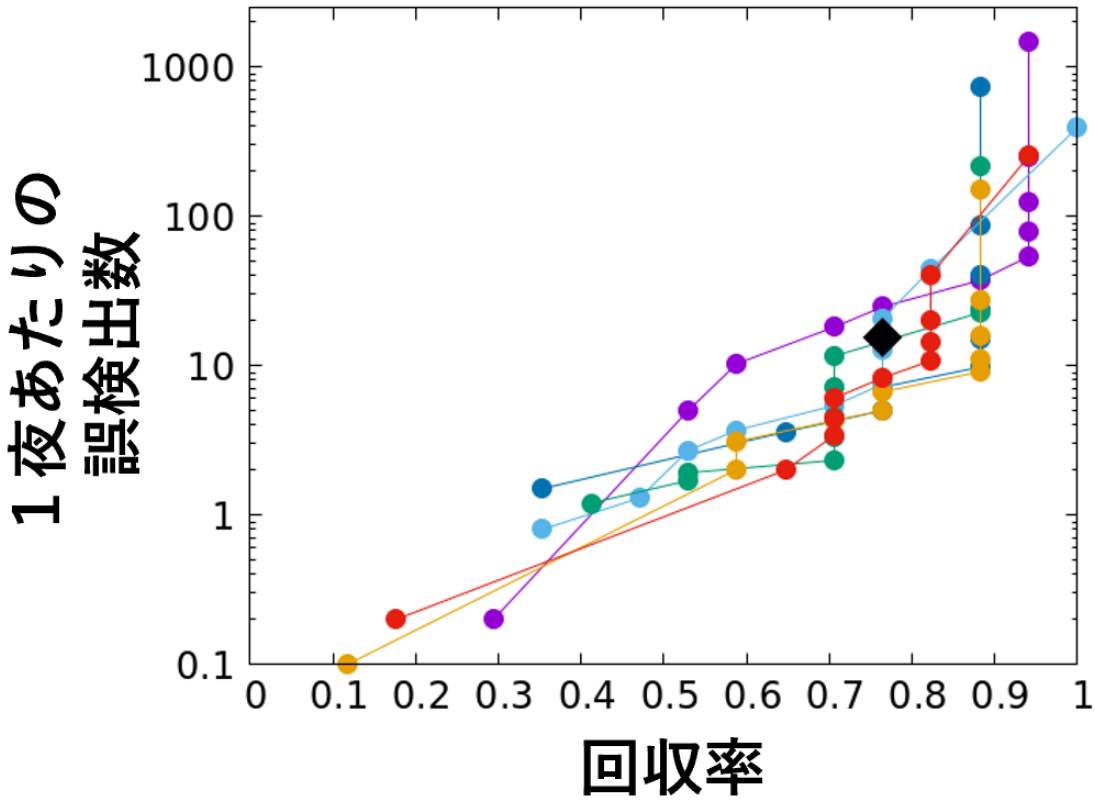

図 3.15: 流星検出器の評価．学習に使用していない流星 17 個を含む 10 夜のデータで流星検出を行った結果．カラー合成は，2 秒ずつ比較明合成した画像を 10 枚使用した場合．色は学習回数を表しており，1 回目：紫，2 回目：青，3 回目：水色，4 回目：緑，5 回目：橙，6 回目：赤．原口 (2021) の結果を黒い菱形で描いている．

第4章　まとめ

防犯カメラで撮影した動画から，流星を検出する推論モデルを機械学習を用いて作成した．初めは，流星画像のみを訓練データとして機械に与えることによって推論モデルを生成した．生成した推論モデルを使って流星検出を行うと，多数の誤検出が出た．そのため，誤検出した画像を分類，アノテーションし，訓練データに追加し，再度学習を行った．この作業を繰り返すことによって，誤検出の数は減少し，検出器の性能が向上した．学習させていない流星を検出させてみると，原口卒論 (2021) では，回収率 0.76，1 夜あたりの誤検出数が 15.5 個であった．それに比べて本研究では，回収率 0.76 の時，1 夜あたりの誤検出数は 5.0 個にまで抑えられ，卒論より性能の良い検出器の作成に成功した．また，誤検出数を 10 個程度許容すると，回収率は 0.88 になり，9 割近い流星を回収できる．

謝辞

本研究を行うにあたり，ご指導いただきました指導教官である　はしもとじょーじ　教授に心より感謝いたします.

参考文献

原口美悠 (2020)：流星観測システムの構築
岡山大学理学部地球科学科　先取りプロジェクト研究報告書.

原口美悠 (2021)：流星自動検出パイプラインの構築
岡山大学理学部地球科学科　卒業論文.

総務省 平成 28 年版情報通信白書
https://www.soumu.go.jp/johotsusintokei/whitepaper/ja/h28/html/nc142120.html

heartexlabs/labelImg
https://github.com/heartexlabs/labelImg

AlexeyAB/darknet
https://github.com/AlexeyAB/darknet

ultralytics/yolov3
https://github.com/ultralytics/yolov3

付録1

3.1で述べた「静止画の生成」の手順は，(1) 動画をコマ画像に分割，(2) 比較明合成，(3) カラー画像の合成，である．以下，それぞれについて説明する．

1. 動画をコマに分割

1-4fpsで撮影した動画をコマ画像に分割する．以下のソースコードを使用した．

ソースコード 1.1: mkpgm.sh

keepspaces

```
#!/bin/bash


OUTDIR = コマ画像を生成するディレクトリの絶対パス
MOVFILE = 処理する動画の絶対パス

FFMPEG = /usr/bin/ffmpeg
LOGLEVEL = "-loglevel quiet"


#コマ画像を生成するディレクトリを作成.
mkdir $OUTDIR

#コマ画像を生成するディレクトリに移動.
cd $OUTDIR


 #動画をコマ画像に分割.
 $FFMPEG $LOGLEVEL -i $MOVFILE -f image2 part%08d.pgm > /dev/null
      2>&1 < /dev/null
```

2. 比較明合成

次に，ノイズを低減するため，分割したコマ画像を比較明合成する．まずは，以下のソースコードで合成に使うコマ画像の総数を数え，生成する比較明合成画像の枚数を計算する．

ソースコード 1.2: addframe.sh

keepspaces

```
1  #!/bin/bash
2
3
4  SRCDIR = 合成するコマ画像のあるディレクトリの絶対パス
5  FNAME = 合成するコマ画像に共通する名前
6  OUTDIR = 比較明合成した画像を生成するディレクトリの絶対パス
7  NO = 最初のコマ画像はスキップするために，0を入れる
8  NF = 比較明合成に使用するコマ画像の枚数
9  METHOD = 比較明合成する際の合成方法
10 NP = 使用する計算機のコア数
11
12
13 CONVERT = /usr/bin/convert
14 SEQ = /usr/bin/seq
15
16
17 #比較明合成した画像を生成するディレクトリを作成.
18 mkdir $OUTDIR
19
20
21 #合成するコマ画像の総数を数え,その数を変数IXMAX に入れる.
22 IXMAX=$( find ${SRCDIR} -name "${FNAME}*.pgm" | wc -l )
23
24 #変数IXMAX を NF で割り，1 追加した数を IXMAX に上書きして入れる.
25 IXMAX=$(( (IXMAX/NF) + 1 ))
26
27 #NP コア使って，IXMAX 枚について add.sh を実行.
28 $SEQ $IXMAX | xargs -P $NP -I{} ./add.sh {} $SRCDIR $FNAME
       $OUTDIR $NO $NF $METHOD
29
30
31 #add.sh の実行終了後に，合成に使用したコマ画像のあるディレクトリを削除.
32 rmdir $SRCDIR
```

指定された枚数の比較明合成画像を，以下のソースコードで生成した．

ソースコード 1.3: add.sh

keepspaces

```
#!/bin/bash


ix = 生成する比較明合成画像の枚数
SRCDIR = 合成するコマ画像のあるディレクトリの絶対パス
FNAME = 合成するコマ画像に共通する名前
OUTDIR = 比較明合成した画像を生成するディレクトリの絶対パス
NO = 最初のコマ画像はスキップするために，0を入れる
NF = 比較明合成に使用するコマ画像の枚数
METHOD = 比較明合成する際の合成方法


CONVERT = /usr/bin/convert
SEQ = /usr/bin/seq


#変数ix を 8 桁で表し，それを idx に入れる．
idx=$( printf %08d $ix )

#合成するコマ画像を抽出し，FLIST とする．
ISTR=$( printf %08d $(( ($ix - 1)*$NF + 1 + $NO )) )
IEND=$( printf %08d $(( ($ix )*$NF + $NO )) )
FLIST=$( $SEQ -w $ISTR $IEND | xargs -I{} echo ${SRCDIR}/${FNAME
    }{}.pgm )

#FLIST の画像を比較明合成．
$CONVERT $FLIST -evaluate-sequence ${METHOD} ${OUTDIR}/${METHOD}$
    {idx}.pgm


#FLIST を削除．
rm $FLIST
```

3. カラー画像の合成

次に，比較明合成した画像を数10枚ずつ抽出し，時刻の情報を埋め込んだカラー画像を生成する，まずは，以下のソースコードで，比較明合成した画像の総数を数え，生成するカラー画像の枚数を計算する．

ソースコード 1.4: mkpict4ml.sh

keepspaces

```
1  #!/bin/bash
2
3
4  SRCDIR = 比較明合成した画像のあるディレクトリの絶対パス
5  FNAME = 比較明合成した画像に共通する名前
6  OUTDIR = カラー合成した画像を生成するディレクトリの絶対パス
7  NF = カラー合成に使用する比較明合成した画像の枚数
8  NP = 使用する計算機のコア数
9
10
11 MKPICT = mkpict.f の実行型ファイルの絶対パス
12
13 SEQ = /usr/bin/seq
14
15
16 #カラー合成した画像を生成するディレクトリを作成.
17 mkdir $OUTDIR
18
19
20 #カラー合成に使用する比較明合成した画像の総数を数え,その数を変数
       IXMAX に入れる.
21 IXMAX=$( find ${SRCDIR} -name "${FNAME}*.pgm" | wc -l )
22
23 #変数IXMAX を定義.
24 if [ $(( IXMAX % (NF/2) )) = 0 ]; then
25     IXMAX=$(( (IXMAX/(NF/2)) - 1 ))
26 else
27     IXMAX=$(( (IXMAX/(NF/2)) ))
28 fi
29
30
31 #NP コア使って，IXMAX 枚について mkpict.sh を実行.
32 $SEQ $IXMAX | xargs -P $NP -I{} ./mkpict.sh {} $SRCDIR $FNAME
       $OUTDIR $NF $MKPICT
```

指定された枚数のカラー画像を，以下のソースコードで生成した．

ソースコード 1.5: mkpict.sh

keepspaces

```
#!/bin/bash


ix = 生成するカラー画像の枚数
SRCDIR = 比較明合成した画像のあるディレクトリの絶対パス
FNAME = 比較明合成した画像に共通する名前
OUTDIR = カラー合成した画像を生成するディレクトリの絶対パス
NF = カラー合成に使用する比較明合成した画像の枚数
MKPICT = mkpict.f の実行型ファイルの絶対パス


CONVERT = /usr/bin/convert
SEQ = /usr/bin/seq


#変数ix を 8 桁で表し，それを idx に入れる．
idx=$( printf %08d $ix )

#合成に使う比較明合成した画像を抽出し，FLIST とする．
ISTR=$( printf %08d $(( ( $ix - 1 )*( $NF / 2 ) + 1 )) )
IEND=$( printf %08d $(( ( $ix - 1 )*( $NF / 2 ) + 1 + $NF - 1
    )) )
FLIST=$( $SEQ -w $ISTR $IEND | xargs -I{} echo ${SRCDIR}/${FNAME
    }}.pgm )


#カラー画像を生成するディレクトリの下に変数idx のディレクトリを作成．
mkdir ${OUTDIR}/${idx}

#ディレクトリidx に移動．
cd ${OUTDIR}/${idx}


#抽出した画像を画素値をdata.txt に書き込む．
  for file in $FLIST ; do
      if [ -e $file ]; then
          $CONVERT $file -compress none tmp.pgm
          cat tmp.pgm >> data.txt
          rm tmp.pgm
      fi
  done
```

```
40
41   #data.txt に対して，mkpict.out 実行.
42   $MKPICT < data.txt
43
44   #mkpict.out で生成した R，G，B 画像を合成してカラー画像を生成.
45   $CONVERT Rimage.pgm Gimage.pgm Bimage.pgm -set colorspace gray
         -combine -set colorspace RGB ${OUTDIR}/ml${idx}.png
46
47
48 #1つ上のディレクトリに移動.
49 cd ..
50
51 #ディレクトリidx を削除.
52 rm -r ${OUTDIR}/${idx}
```

mkpict.out を生成したソースコードは以下の通りである.

ソースコード 1.6: mkpict3.f

keepspaces

```
1       program mdetect
2       implicit none
3
4  c 解像度の大きさの最大値を宣言
5       integer NXMAX
6       parameter (NXMAX=1920)
7       integer NYMAX
8       parameter (NYMAX=1080)
9  c 合成する画像の枚数の最大値を宣言
10      integer NFMAX
11      parameter (NFMAX=256)
12 c 何位まで画素値の順位付けをするかを宣言
13      integer NM
14      parameter (NM=3)
15      integer NMMAX
16      parameter (NMMAX=NM+1)
17
18 c 書き込む新しいファイルを宣言
19      integer IFILE
20      parameter (IFILE=21)
21
22 c 変数を宣言
23      integer NF
24      integer NX, NY
25      integer NP
26      integer npmax(NXMAX,NYMAX,NMMAX)
27      integer nmaxframe(NXMAX,NYMAX)
28
29      integer nnx, nny, nnp
30      integer npix(NXMAX)
31
32      real*8 xpfac(NXMAX,NYMAX)
33
34      real t
35      real Rval, Gval, Bval
36      integer Rpix(NXMAX,NYMAX), Gpix(NXMAX,NYMAX), Bpix(NXMAX,
            NYMAX)
37
38      integer i, j, k, m
39      integer ii, jj
40
```

```
41
42 c 1枚目の画像
43       read(*,*,end=910)
44       NF = 1
45       read(*,*) NX, NY
46       if (NX.gt.NXMAX) then
47          write(*,*) 'NX is larger than NXMAX'
48          stop
49       end if
50       if (NY.gt.NYMAX) then
51          write(*,*) 'NY is larger than NYMAX'
52          stop
53       end if
54       read(*,*) NP
55       do 120 j = 1, NY
56         read(*,*) (npix(i), i=1,NX)
57         do 110 i = 1, NX
58           npmax(i,j,1) = npix(i)
59           do 109 m = 2, NM
60             npmax(i,j,m) = 0
61  109 continue
62           nmaxframe(i,j) = NF
63  110 continue
64  120 continue
65
66 c 2枚目以降の画像
67  130 continue
68       read(*,*,end=200)
69       NF = NF + 1
70       if (NF.gt.NFMAX) then
71          write(*,*) 'NF is larger than NFMAX'
72          stop
73       end if
74       read(*,*) nnx, nny
75       if (nnx.ne.NX) then
76          write(*,*) 'image size is inconsistent (NX)'
77          stop
78       end if
79       if (nny.ne.NY) then
80          write(*,*) 'image size is inconsistent (NY)'
81          stop
82       end if
83       read(*,*) nnp
84       if (nnp.ne.NP) then
85          write(*,*) 'image max value is inconsistent (NP)'
86          stop
```

```
87        end if
88        do 150 j = 1, NY
89          read(*,*) (npix(i), i=1,NX)
90          do 140 i = 1, NX
91            do 139 m = NM, 1, -1
92              if ( npix(i) .gt. npmax(i,j,m) ) then
93                  npmax(i,j,m+1) = npmax(i,j,m)
94                  npmax(i,j,m ) = npix(i)
95                  if (m.eq.1) then
96                      nmaxframe(i,j) = NF
97                  end if
98              end if
 99  139 continue
100  140 continue
101  150 continue
102       go to 130
103
104 c 最後の画像の読み込み終了時
105  200 continue
106
107
108 c 規格化
109       do 320 j = 1, NY
110         do 310 i = 1, NX
111            xpfac(i,j) = ( npmax(i,j,1) * 1.0d0 ) / ( NP * 1.0d0
                  )
112  310 continue
113  320 continue
114
115
116 c R, G, B 画像の画素値を計算
117       do 420 j = 1, NY
118         do 410 i = 1, NX
119           t = 3.0 * ( nmaxframe(i,j) - 1.0 ) / ( NF - 1.0 )
120           if ( t .le. (1.0) ) then
121               Rval = - t + 1.0
122               Gval = t - 0.0
123               Bval = 0.0
124             else if ( t .le. (2.0) ) then
125               Rval = 0.0
126               Gval = - t + 2.0
127               Bval = t - 1.0
128             else
129               Rval = t - 2.0
130               Gval = 0.0
131               Bval = - t + 3.0
```

```
132           end if
133           Rpix(i,j) = Rval * npmax(i,j,1) * xpfac(i,j)
134           Gpix(i,j) = Gval * npmax(i,j,1) * xpfac(i,j)
135           Bpix(i,j) = Bval * npmax(i,j,1) * xpfac(i,j)
136  410 continue
137  420 continue
138
139 c R, G, B 画像の画素値を新しいファイルに書き込み，画像を生成
140  810 format(2A)
141       NP = 255
142 c R 画像生成
143       open(IFILE,file='Rimage.pgm')
144       write(IFILE,810) 'P2'
145       write(IFILE,*) NX, NY
146       write(IFILE,*) NP
147       do 820 j = 1, NY
148         write(IFILE,*) (Rpix(i,j), i=1,NX)
149  820 continue
150       close(IFILE)
151 c G 画像生成
152       open(IFILE,file='Gimage.pgm')
153       write(IFILE,810) 'P2'
154       write(IFILE,*) NX, NY
155       write(IFILE,*) NP
156       do 830 j = 1, NY
157         write(IFILE,*) (Gpix(i,j), i=1,NX)
158  830 continue
159       close(IFILE)
160 c B 画像生成
161       open(IFILE,file='Bimage.pgm')
162       write(IFILE,810) 'P2'
163       write(IFILE,*) NX, NY
164       write(IFILE,*) NP
165       do 840 j = 1, NY
166         write(IFILE,*) (Bpix(i,j), i=1,NX)
167  840 continue
168       close(IFILE)
169
170       stop
171
172 c エラーの場合
173  910 continue
174       write(*,*) 'No data'
175       stop
176
177       end
```

カラー合成した画像は，コントラストが甘く，アノテーションする際に流星が見えにくくなっている．そのため，コントラストを強調する処理を以下のソースコードで行った．まずは，コントラストを強調する画像の枚数を数える．

ソースコード 1.7: mklevel.sh

keepspaces

```
#!/bin/bash


SRCDIR = カラー合成した画像のあるディレクトリの絶対パス
FNAME = カラー合成した画像に共通する名前
OUTDIR = コントラストを強調した画像を生成するディレクトリの絶対パス
NP = 使用する計算機のコア数


SEQ = /usr/bin/seq


#コントラストを強調した画像を生成するディレクトリを作成.
mkdir $OUTDIR

#コントラストを強調する画像の総数を数え,その数を変数IXMAX に入れる.
IXMAX=$( find ${SRCDIR} -name "${FNAME}*.png" | wc -l )

#NP コア使って，IXMAX 枚について level.sh を実行.
$SEQ $IXMAX | xargs -P $NP -I{} ./level.sh {} $SRCDIR $FNAME $OUTDIR
```

指定された枚数で，カラー画像のコントラストを強調した画像を，以下のソースコードで生成した．

ソースコード 1.8: level.sh

keepspaces

```
#!/bin/bash


ix = 生成する画像の枚数
SRCDIR = カラー合成した画像のあるディレクトリの絶対パス
FNAME = カラー合成した画像に共通する名前
OUTDIR = コントラストを強調した画像を生成するディレクトリの絶対パス


CONVERT = /usr/bin/convert


#変数ix を 8 桁で表し，それを idx に入れる．
idx=$( printf %08d $ix )

#カラー画像のコントラストを強調．
$CONVERT -level 0%,20% $SRCDIR/${FNAME}${idx}.png $OUTDIR/${FNAME}${idx}.png
```

最後に，カラー合成がうまくいっているかどうか確認する場合のために，カラー画像を繋げて動画にする．以下のソースコードで行った．

ソースコード 1.9: mkmovie.sh

keepspaces

```
#!/bin/bash


MOVIEDIR = 動画を生成するディレクトリの絶対パス
SRCDIR = 動画を生成する画像のあるディレクトリの絶対パス


AWK = /usr/bin/awk
FFMPEG = /usr/bin/ffmpeg


#動画を生成するディレクトリに移動.
cd $MOVIEDIR

  #カラー画像から動画を生成.
  ls ${SRCDIR}/*.png | $AWK '{printf "ln -s %s %08d.png/n", $0, NR}' | sh
  $FFMPEG -r 10 -i ./%08d.png -vcodec libx264 -pix_fmt yuv420p movie.mp4
  rm *.png
```

付録2

GPU を用いて，YOLO で物体検出する手順を以下に示す.

パスを通す.
$ cd
$ echo 'export CUDA_PATH=/usr/local/cuda-11.8' >> ./.bashrc
$ echo 'export LD_LIBRARY_PATH=/usr/local/cuda-11.8/lib64:${LD_LIBRARY_PATH}'
>> ./.bashrc
$ echo 'export PATH=/usr/local/cuda-11.8/bin:${PATH}' >> ./.bashrc

インストールする.
$ mkdir yolo
$ cd yolo
$ mkdir AlexeyAB
$ cd AlexeyAB
$ git clone https://github.com/AlexeyAB/darknet.git
$ cd darknet

Makefile を書き換える.
GPU, CUDNN, CUDNN_HALF, AVX, OPENMP の数字を 0 から 1 にする.
「# GeForce RTX 3070, 3080, 3090」の次の行の,
「# ARCH= -gencode arch=compute_86,code=[sm_86,compute_86]」
の先頭にある # を削除する.

修正したら以下のようになる.
GPU=1
CUDNN=1
CUDNN_HALF=1

AVX=1
OPENMP=1
ARCH= -gencode arch=compute_86,code=[sm_86,compute_86]

最後に，以下のコマンドを実行する．
$ make

動作を確認する．
重みファイルを以下のサイトからダウンロードする．
$ wget https://pjreddie.com/media/files/yolov3.weights

検出してみる．
thresh の後ろの数字は閾値で，小さくすると不確かなものまで，大きくすると確からしいものだけを表示する．
$ ./darknet detect ./cfg/yolov3.cfg ./yolov3.weights -thresh 0.25 ./data/dog.jpg

図 2.1: dog.jpg.

推論結果がテキストと画像ファイル (predictions.jpg) で出力される.

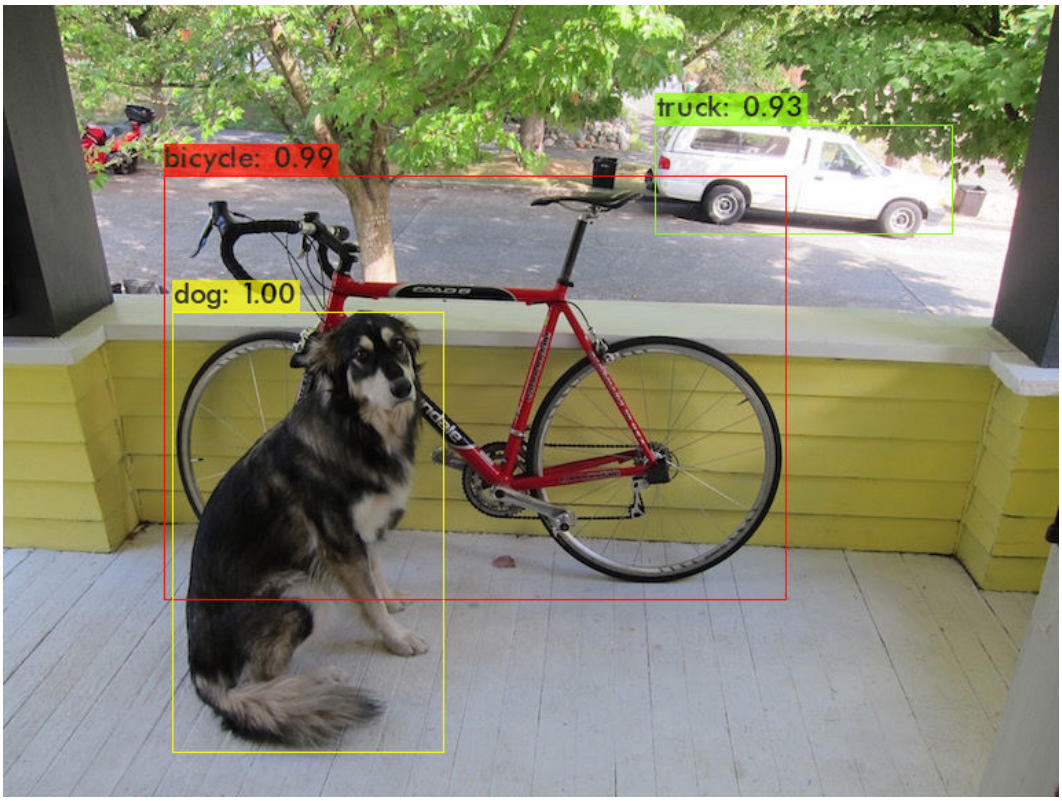

図 2.2: predictions.jpg.

動作確認ができたら，次は学習させる．

まずは，ファイルを用意する．

学習に必要なファイルの一覧．
~/AlexeyAB/darknet/task
~/AlexeyAB/darknet/task/backup
~/AlexeyAB/darknet/task/datasets
~/AlexeyAB/darknet/task/datasets/ < 対象画像ファイル名 > .txt
~/AlexeyAB/darknet/task/datasets/ < 対象画像ファイル名 > .png
~/AlexeyAB/darknet/task/class.txt
~/AlexeyAB/darknet/task/datasets.data
~/AlexeyAB/darknet/task/train.txt
~/AlexeyAB/darknet/task/test.txt
~/AlexeyAB/darknet/task/yolov3-custom.cfg

task ディレクトリを生成．
$ cd
$ cd AlexeyAB
$ cd darknet
$ mkdir task
$ cd task

backup ディレクトリを作成．
$ mkdir backup
$ touch ./backup/yolov3-custom.backup

datasets ディレクトリを作成．
$ mkdir datasets

トレーニングと検証に使用する画像をアノテーションする.
$ cd datasets
$ ln -s < imagefile.png > .
$ ln -s < imagefile.txt > .

imagefile.png
トレーニングと検証に使用する画像.

imagefile.txt
YOLO 形式のアノテーションデータは，imagefile.png のアノテーションを
imagefile.txt とする.
imagefile.txt の中身は以下の通りである.

< object-class > < x_center > < y_center > < width > < height >

---

object-class は 0 から始まる (最大は class - 1) である.
x, y, width, height は画像サイズを 1 として 0.0-1.0 の数値で与える.

train.txt を生成.
トレーニングに使う画像のファイル名の一覧.
ファイル名 (フルパス) を 1 行に 1 ファイルずつ記載する.

test.txt を生成.
検証に使う画像のファイル名の一覧.
ファイル名 (フルパス) を 1 行に 1 ファイルずつ記載する.

トレーニングと検証の枚数比は，8:2 や 7:3 にする.
画像が足りない時はトレーニングと検証が同じでもよいと考え，本研究ではトレーニングと検証を同じにした.

datasets.data を生成.
データセットの設定ファイルの一覧.

---

classes= < 分類の数 >
train = < train.txt のフルパス >
valid = < test.txt のフルパス >
names = < class.txt のフルパス >
backup = < backup ディレクトリのフルパス >

---

cfg の作成.
$ cp yolov3.cfg yolov3-custom.cfg

以下を変更する.

width, height
使用する画像のサイズで，32 の倍数でなければならない.

max_batch
クラス数と画像数によって数を調整する.
以下の 3 つのうち最大のものを使う.
class 数 x 2000
学習画像の枚数
6000

steps
max_batch の 80%と 90%の数値にする.

filters
”activation=linear”の 1 つ前の行にある”filter”3 箇所全て
(classes + 5) x 3
を入れる.

classes
”filter”の数値を変更した”activation=linear”の後ろにある”classes”3 箇所全て
classes の数
を入れる.

必要に応じて，以下の項目について変更する.

batch
メモリエラーが出るとき数を減らす.
subdivisions
メモリエラーが出るとき数を増やす.

ファイルの用意ができたら，学習させる.
$ cd
$ cd AlexeyAB
$ cd darknet
$ ./darknet detector train ./cfg/task/datasets.data
./cfg/task/yolov3-custom.cfg ./cfg/task/backup/yolov3-custom.backup
-dont_show > log.txt

学習済みのモデル (yolov3.weights) から始めるときは,
$ ./darknet detector train ./cfg/task/datasets.data
./cfg/task/yolov3-custom.cfg ./yolov3.weights -dont_show > log.txt

学習の途中で止めたとき，続きを学習させるなら,
$ ./darknet detector train ./cfg/task/datasets.data
./cfg/task/yolov3-custom.cfg ./cfg/task/backup/yolov3-custom.backup
-dont_show >> log.txt

オプションで -dont_show をつけると途中経過をグラフにして表示しない (リモートサーバで学習する場合に付ける).

出力されるログの例.

293: 3.451290, 3.651849 avg, 0.000007 rate, 342.257036 seconds, 18752 images

---

左から，バッチ回数: このバッチでの loss, 平均の loss avg, 学習率 rate, このバッチにかかった計算時間 seconds, これまでの学習画像枚数 images.
「平均の loss」に nan が出たら，学習に失敗している.
#それ以外に出た nan は気にしないでよい.

学習がうまくいかないときは,
  yolov3.cfg
にある,
  learning_rate
の値を調整する.

nan が出たときは,
  learning_rate
を小さくする.

学習が進まないときは,
  learning_rate
を大きくする.

100 ステップ毎に結果が,
  ./cfg/task/backup/yolov3-custom.backup
に保存される.
100 ステップ毎に結果は上書きされる.
#途中経過を保存するなら，手動で保存する.

backup の下に保存された weights を使うことで，計算のリスタートが可能である.

Loss
正解とどれくらい離れているかを表す数値.
この値の減り具合によって，学習の続行と中断を決定する.

学習できたら，検出する
$ cd
$ cd AlexeyAB
$ cd darknet
$ ./darknet detect ./cfg/task/yolov3-custom.cfg
./cfg/task/yolov3-custom.weights -thresh 0.25 ./data/test.jpg

重みは,
./cfg/task/yolov3-custom.weights
を使う.

閾値を 0.25(確信度 25%以上と推論したもののみを検出) に設定して,
./data/test.jpg
について物体検出をする.