キー操作を特徴とするアノテーションツールの開発とその高解像度衛星画像への適用

竿本　英貴[1]・宮本　崇[2]・齋藤　彰[3]

[1]正会員　産業技術総合研究所　活断層・火山研究部門（〒 305-8567 茨城県つくば市東 1–1–1 中央第 7）
E-mail: h-saomoto@aist.go.jp (Corresponding Author)
[2]正会員　山梨大学　大学院総合研究部工学域　土木環境工学系（〒 400-8511 山梨県甲府市武田 4–3–11）
E-mail: tmiyamoto@yamanashi.ac.jp
[3]非会員　明治大学　理工学部　機械工学科（〒 214-8571 神奈川県川崎市多摩区東三田 1–1–1）
E-mail: asaito@meiji.ac.jp

土木分野で機械学習が広く活用されている．機械学習用の教師データを作成するためのアノテーション作業には多大な労力が必要であり，労力削減のためのアノテーションツール開発が必要となる．本研究では，近年機械学習に活用されている高解像度衛星画像を対象としたアノテーションツールを開発する．まずは既往のアノテーションツールで高解像度衛生画像を取り扱い，衛星画像特有の課題 (大容量への対応，迅速な拡大・縮小機能の必要性，キーバインド機能の必要性等) を抽出した．次いで，課題に対応したアノテーションツールをプログラミング可能なオープンソース画像ビューワ napari に追加実装することで開発した．開発したツールを高解像度衛星画像に対する 3 クラス分類問題に適用し，効率的にアノテーション作業が実施できることを確認した．

Key Words: *machine learning, annotation, satellite image, GUI, classification*

1.　はじめに

土木分野において機械学習を用いた研究が広く実施されている [1)–3)]．研究内容の具体例として，コンクリートのひび割れや変状の検出 [4)–6)]，道路舗装の損傷検出 [7),8)]，橋梁部材の損傷検出 [9)] や損傷原因・補修工法の推定 [10)] などが挙げられ，一定の成果が得られている．

上記研究内容では，画像データと深層学習の組み合わせによる異常検出に関するものが多く確認できる．コンクリートのひび割れ分類・検出の機械学習タスクを例に取れば，まずは作業者が目視にて教師データとするひび割れ領域を抽出・ラベリングしておく作業が発生する (アノテーション)．当然ながら，ひび割れとそれ以外という 2 クラス分類のみならず，エフロレッセンス領域等の分類も含めた多クラス分類も実施されている [5)]．このような画像とクラスを対応させるアノテーション作業 (教師データ作成) には，一定レベル以上の時間と労力が必要となることが広く知られている．

図-1 は，データ入手から学習済み機械学習モデルの供用までの一連の機械学習タスクを構成する代表的なプロセスと，各プロセスに必要となる作業時間の目安を示している [11),12)]．機械学習アルゴリズムに関連するプロセスは全体の 20%程度にとどまっており，残りの約 80%が機械学習アルゴリズムに入力するための教師データ整備に関連していることがわかる．特に，データ整形 (Data Cleansing) とデータラベリング・アノテーション (Data Labeling, Annotation) は各 25%程度の作業時間が費やされる．

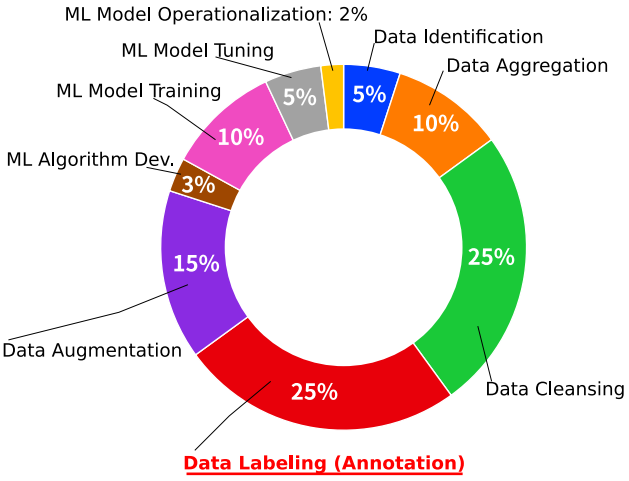

図-1: 機械学習タスクを構成する各プロセスとそれらに必要となる時間の目安 (Cognilytica 社の資料 [11),12)] を基に作成)

アノテーション作業では研究分野の特性が反映されたデータ (画像，動画，音声等) を取り扱うため，分野毎に多様なアノテーションツールが開発されているのが実態

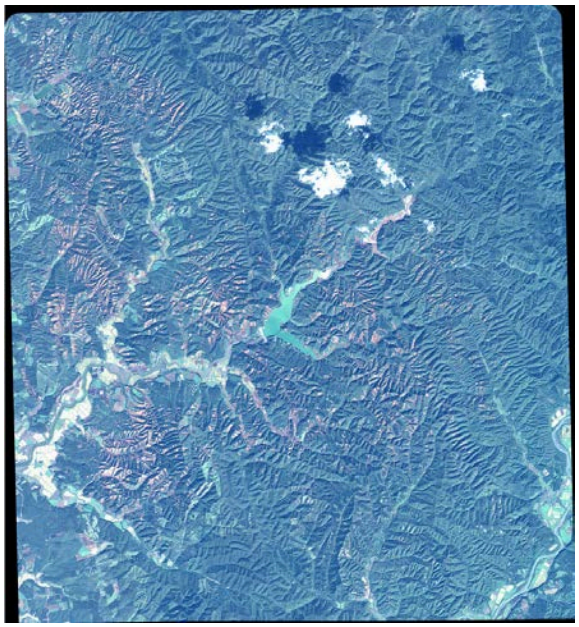

図-2: 既往アノテーションツールでの動作確認に用いた高解像度衛星画像 (13268×14396 pixels)

である (例えば，海洋生物画像に対するアノテーションツール [13]，医療用 DICOM 画像に対するアノテーションツール [14]，漫画画像に対するアノテーションツール [15] 等)．この実態は，万能なアノテーションツールの開発は困難であり，目的に応じたアノテーションツールの開発には一定の意義があることを示している．

本研究では，高解像度を有する衛星画像を対象としたアノテーションツールを開発する．この理由として，高解像度衛星画像は近年入手し易くなってきていること，機械学習を利用した災害時の状況把握に衛星画像が活用されはじめていること [16–18] が挙げられる．まずはいくつかの著名な既往アノテーションツールを使用して高解像度衛星画像を実際に取り扱い，課題を抽出した．得られた課題に対応しつつ，プログラミング可能な画像ビューワ napari[19] をベースとしてアノテーションツールを開発した (第 2 章)．第 3 章では開発したアノテーションツールを実際の衛星画像に適用し，3 クラス分類を実施した事例を紹介する．第 4 章では本研究のまとめと今後の課題を述べる．

2. アノテーションツールの実装

(1) 既往アノテーションツールの課題

図-2 は本研究で対象とする高解像度衛星画像の例を示しており，横方向に 13268 pixel，縦方向に 14396 pixel の解像度を有している (1 pixel:1.5 m)．本画像は 16 bit の GeoTIFF の元画像 (容量約 1.5 GB) を 8 bit のカラー画像に調整したものである (容量約 760 MB)．画像の内容は 2018 年 9 月 6 日に発生した北海道胆振東部地震に起因する厚真町の地すべり発生状況を撮影したもので，画像を拡大すると様々な様式の地すべり痕を数多く確認することができる．

表-1: 検討に用いたオープンソースの既往アノテーションツール

名称	機能	対応データ	出力形式
VoTT	B[1]	画像・動画	CSV, TF[2] 他多数
Labelme	B, S[3], K[4]	画像・動画	CO[5]
LabelImg	B	画像	PV[6], Y[7], C[8]

[1] Bounding Box; [2] TensorFlow; [3] Segmentation; [4] Keypoint; [5] COCO-format; [6] Pascal VOC; [7] Yolo; [8] CreateML

表-1 に示すオープンソースのアノテーションツール (いずれもマルチプラットフォーム) に**図-2** の画像をセットし，拡大して確認できる地すべり領域のバウンディングボックスを手動で抽出することにより，高解像度衛星画像を取り扱う上での課題を抽出する．今回検討に用いたアノテーションツールは 3 種類であり，VoTT[20]，LabelMe[21]，LabelImg[22] である．なお，どのツールも同一のノート PC (富士通 LIFEBOOK WU3/E2，CPU: Core i7-10510U, RAM: 16GB メモリ) 上でそれぞれ動作させて検討を実施した．

VoTT(Visual Object Tagging Tool) は，2018 年に公開されたマイクロソフト社製のオープンソースアノテーションツールであり，GUI を通じて画像およびビデオフレームに対してアノテーション作業を実施できる．アノテーション結果を多くの形式で出力できることが特徴である．VoTT を用いた検討の結果，VoTT では**図-2** の画像を読み込むことは不可能であった (フリーズした)．また，VoTT には画像の拡大・縮小機能が無いため，頻繁に画像を拡大・縮小する必要がある高解像度衛星画像のハンドリングには対応不可能である (今回の事例では拡大して地滑り領域を確認する必要がある)．

Labelme は 2018 年に開発されたアノテーションツールあり，GUI を通じて画像およびビデオフレームに対してバウンディングボックスのみならずポリゴンや円などの図形を用いてアノテーション作業を実施することができる．Lbelme を用いた検討の結果，VoTT 同様，画像を読み込むのに多大な時間がかかった上，作業を開始しようするとフリーズしてアノテーション作業が不可能であった．

LabelImg は 2015 年に開発されたアノテーションツールあり，MIT ライセンスの元で配布されている．GUI のレイアウトは Labelme に類似しているが，拡大・縮小のボタンが配置されていることが特徴である．機能

は矩形抽出のみに限定されているものの，**図-2** の画像を読み込みアノテーション作業を実施することができた．比較的高速に動作したが，拡大・縮小では逐一マウスで GUI のボタンを押下する必要があり，拡大・縮小操作時に作業が中断されるため，この点における操作性の改善が必要と考える．また，アノテーション作業では，マウスポインタで矩形領域を設定したのち，別途用意された属性ボタン (ラベル) をマウスポインタで逐一押下して矩形領域とラベルを紐付けする必要があり，マウスポインタの多用のために迅速なアノテーション作業が阻害されている．マウスポインタで矩形領域を設定するとともに，キーボードのキーで属性を指定するなどの工夫により，より効率的なアノテーションが可能となると考える．これらの他，設定した矩形領域のハイライトが弱く，画像のどこを抽出したのかが把握し難い点も改善が必要と考える．

以上，既往のアノテーションツールを用いて高解像度衛星画像に対するアノテーション作業を実施したところ，次の 4 つの課題が抽出された．

1. 高解像度衛星画像の大容量に対応可能な画像読み込み・保存機能が必要．
2. 高解像度衛星画像に対するアノテーション作業ではスムーズな拡大・縮小機能が必要．
3. マウスポインタのみの多用を避け，キーバインドとの併用によりアノテーションを効率化することが必要．
4. 設定した矩形領域を特定の色で強調し，ラベルとの対応を明確化する等の工夫が必要．

(2) napari をベースにしたアノテーションツール開発

前節で抽出された課題に対応したアノテーションツールを開発するためには，画像や動画の取り扱いが容易となる機能，マウスポインタ座標を容易に取得できる機能，キーバインドに基づく機能追加が容易であること等の要件を満たす必要がある．これらを念頭において調査・検討を進めた結果，Python スクリプトによって機能を容易に追加できる画像ビューワ napari[19] をベースとして，これに必要な機能を追加実装することでアノテーションツールを開発することとした．napari はアノテーション用途での使用を念頭において提供されており，高解像度衛星画像用途ではないが napari をベースとするアノテーションツールが開発されている [23]．

画像を読み込む機能は napari にすでに実装されており，この点についての追加実装は必要ない．また，画像のレンダリングには GPU が使用されるため，描画が高速化されている (課題 1 への対応)．画像の拡大・縮小機能はマウスホイールを回転させることで実現されているため，マウスポインタの位置を移動させることなく画像の拡大・縮小が可能である (課題 2 への対応)．マウスポインタ座標の取得については組み込み関数が準備されており，「$y, x = viewer.cursor.position$」と記述することで画像上でのポインタの座標 (x, y) を取得できる．キーバインド機能の追加例を，プログラムリスト 1 に示す (課題 3 への対応)．このプログラムでは，F1 キーを押下することで，マウスポインタが存在している位置に予め定めたサイズ (dx, dy) の矩形 (色は黄色) を配置することができる．したがって，マウスポインタを移動させながら F1 キーを定期的に押下する等すれば，連続的なアノテーション作業に応用できる．なお，矩形領域は描画色・透過度をプログラム内で指定できるため，任意色でのハイライトが可能である (課題 4 への対応)．

プログラムリスト 1: キーバインド機能の追加例

```
# add a rectangle by pressing "F1"
@viewer.bind_key("F1", overwrite=True)
def put_rect(viewer):
    # マウスポインタ座標の取得
    y, x = viewer.cursor.position
    xc = int(x); yc = int(y)

    # 矩形端部の座標
    xm=xc-int(dx/2); ym=yc-int(dy/2)
    xp=xc+int(dx/2); yp=yc+int(dy/2)

    p1=[ym, xm]; p2=[yp-1, xp-1]
    rect = array([p1, p2])

    # 画像上への黄色四角の描画
    viewer.add_shapes(rect,
    ↪  shape_type='rectangle',
    ↪  edge_width=0.0, face_color=Yellow)
```

プログラムリスト 1 の 2 行目と 3 行目にあるように，組み込み関数「$@viewer.bind\_key$」を用いることで任意のキーを登録でき，登録したキーを押下することでその後に続く独自に定義した関数 (ここでは「$def put\_rect$」) 内に記述した内容を処理させることができるのが napari の特長となる．この特長を活用することで複数種類の処理内容を複数キーに容易に割り当てることができる．例えば，3 クラスのラベリングを実施する場合は，F1，F2，F3 の各キーを各クラスにそれぞれ割り当てることでアノテーションの効率化が図れる．

図-3 は，napari に必要な機能を追加して作成したアノテーションツールの動作例を示している．**図-2** の画像を拡大して確認できる地すべり領域を，マウスと F1 キーの組み合わせ操作で 5 箇所 (各 200pixel 四方の黄色

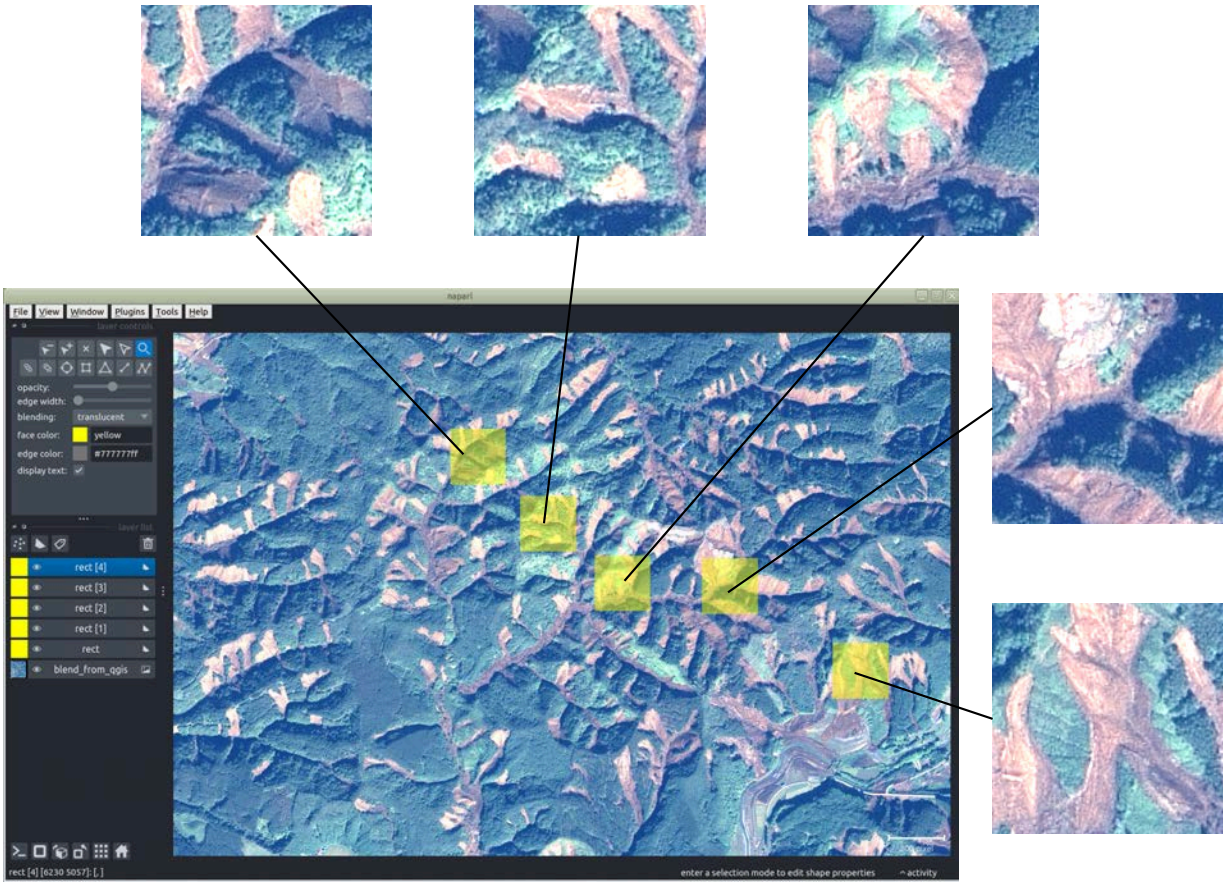

図-3: napari に機能を追加して作成したアノテーションツールの動作例 (図-2 を拡大して地すべり領域の一部をマウス・F1 キーで選択 (黄色の矩形))

の矩形領域) 抽出した．なお，各矩形領域をマウスで選択することで任意のサイズに変更可能である．本ツールでは設定した矩形領域を個別の画像として保存する機能を付与しており，矩形の画像データそのものを機械学習アルゴリズムに直接入力する際に有用である．**表-1** で示した各ツールには設定した矩形領域を個別の矩形画像として保存する機能は無いことを付記しておく．個別の矩形画像の保存とともに，各矩形画像のラベルおよび矩形端部 4 点の座標値をテキストデータとして保存する機能を付与している．また，物体認識で広く用いられる YOLOv5 形式 [24] でも各矩形領域のデータを保存するように作成している．

本アノテーションツール使用時における主要なキー操作は，以下のとおりである．

- F1, F2, F3, F4 の各キー：マウスポインタ位置に矩形領域 (サイズは予め設定) を配置，各キーは分類クラスに対応 (現時点では 4 クラス分類まで対応)
- u キー：動作の取り消し
- s キー：画像上に配置した各矩形領域を切り抜いて個別に保存，併せて各矩形領域のクラスと位置 (テキスト形式) を保存．

マウスポインタによって矩形領域の中心を指し示しつつ，F1～F4 キーのどれかを押すことでアノテーション作業を進めていく．

3. アノテーションツールの動作検証

本章では作成したアノテーションツールを使用して，2 種類の高解像度衛生画像それぞれについて 3 クラス分類を実施した結果を示す．

(1) 地すべり痕を含む高解像度衛生画像への適用

図-4 は，**図-2** で示した北海道厚真町の高解像度衛星画像に対し,「地すべり域」,「森林域」,「田園域」の 3 クラスに分類するためのアノテーション作業結果を示している．各クラスについて，ツールを用いてそれぞれ 50 の矩形領域 (サイズ：200pixel 四方) を指定した．各クラスについてのサンプル例 (各 3 画像) を**図-4** 右側に併せて示している．地すべり域の特徴は，森林域に比べて白色領域 (地すべり痕) が混入しているのが特徴的であり，地すべり痕の様式・サイズについては多様である．また，地すべり域・森林域に比べて田園域は直線的な道路が含まれるなど，他の 2 クラスと比較して明

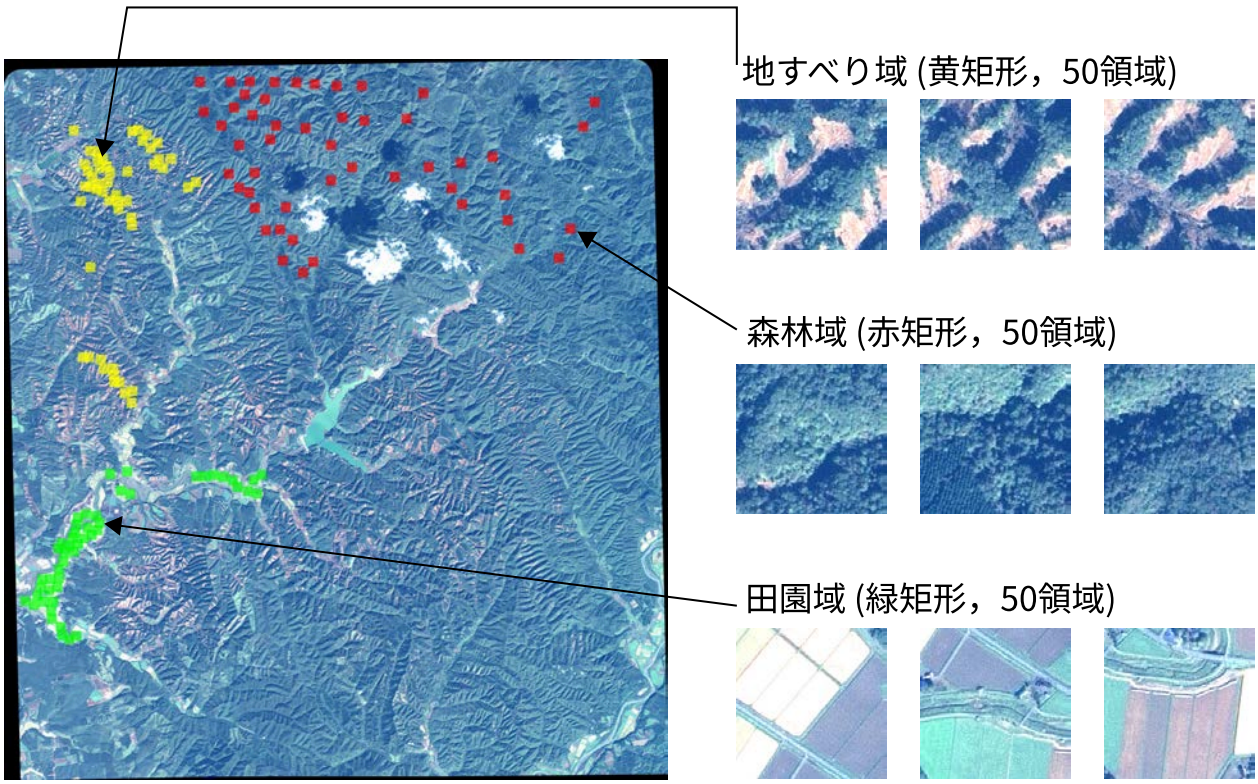

図-4: 図-2 の地すべり域，森林域，田園域 (各 50 領域) に対するアノテーション作業結果

らかに異なる特徴を有している．なお，田園の色や道路形態なども多様である．

各クラスとキーバインドの対応は，F1 キー：地すべり域，F2 キー：森林域，F3 キー：田園域と設定した．実際に作業した結果，アノテーション作業を迅速に実施するためには，可能な限り単一クラス毎に作業することが効率的な方策であるとの結論に至った．今回，この方策に基づき，まずは地すべり域について 50 領域，次で森林域について 50 領域，最後に田園域について 50 領域と順を追って作業を実施した．作業時の動作がマウスポインタを動かしながら単一キーを押下するのみに限定されるので，誤操作の発生が抑制できる．なお，ツールの仕様上，どのような順番でアノテーションを実施しても問題は生じない (例えば F1 キー，F2 キー，F1 キーの順など)．

図-4 で取得した 150 毎の各矩形画像について，どの程度特徴が異なるのかを予め把握しておくことは，その後の画像分類を実施する上で有用である．この観点から，次元削減手法 UMAP[25] によって 12 万次元 (200pixels × 200pixels × 3colors) で表される各画像の特徴を 2 次元空間に写像したものが**図-5** である．各プロットの横軸と縦軸の数値は UMAP のハイパーパラメータを変更すると変化するものであり，この図は各矩形画像が相対的に類似しているかどうかを示している．ここでは UMAP のハイパーパラメータ値はデフォルトの設定とした．なお，UMAP 以外の次元削減手法も多く提案されている

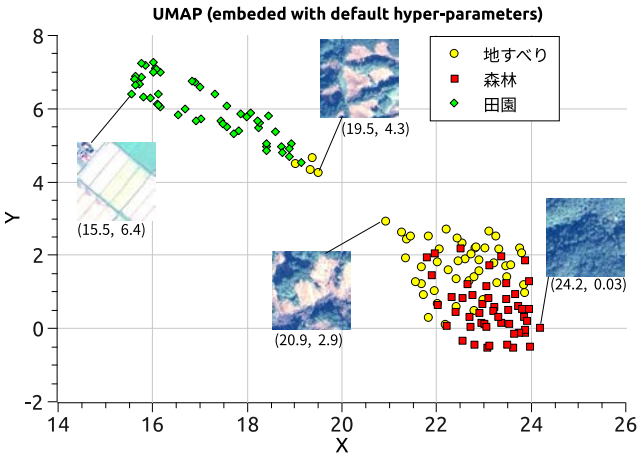

図-5: 図-4 で抽出した各矩形画像に対する UMAP による次元削減結果 (12 万次元 ↦2 次元)

が，UMAP は高速に動作すること，クラス毎の分類精度が高い等の特徴を持つ．

図-5 より，田園域の矩形画像は $y \geq 4$ の領域に集中しており，このクラスの画像は分類が容易であることが期待される．一方，地すべり域と森林域については，$y = 1$ 付近でプロットが混在しており，場合によっては地すべり域と森林域の分類が困難であることを示唆している．

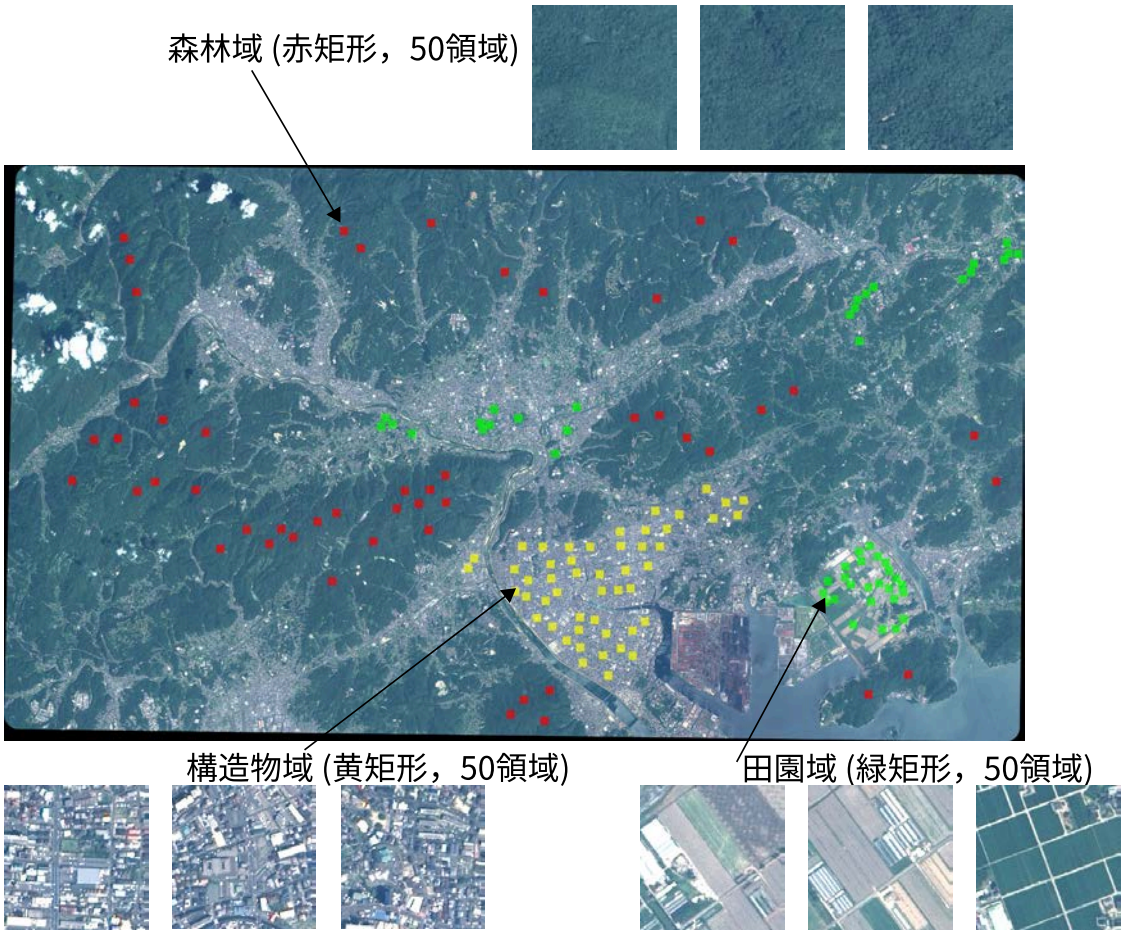

図-6: 構造物域，森林域，田園域 (各 50 領域) に対するアノテーション作業結果

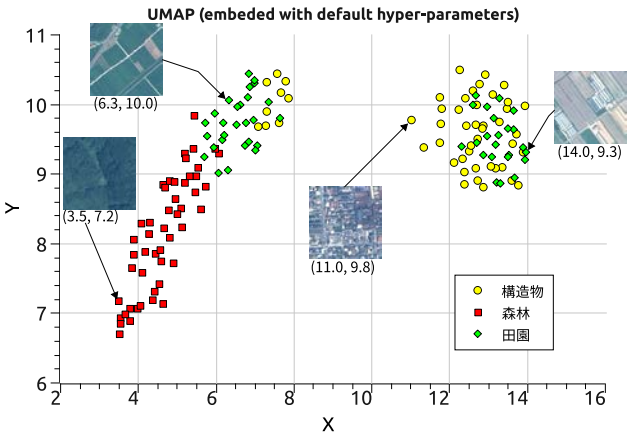

図-7: 図-6 で抽出した各矩形画像に対する UMAP による次元削減結果 (12 万次元 ↦2 次元)

(2) 市街地を含む高解像度衛生画像への適用

図-6 は広島県福山市の高解像度衛星画像を示している．画像の容量は 1.4 GB であり，サイズは横 24660 pixels，縦 13896 pixels(1pixel = 1.5 m) である．この画像に対し，アノテーションツールを用いて前節と同様の作業を実施し，市街地における構造物域，森林域，田園域について各 50 領域の矩形画像 (200pixels 四方) を抽出した．大規模な田園域が**図-6** 右下に確認できるが (高密度の緑矩形)，住宅街にも田園域が点在している．画像のサイズが極めて大きいため，このような点在している田園域を抽出するには画像を拡大しながら目視で確認するほかないが，本ツールによってスムーズに抽出することができた．

図-7 は**図-5** と同様，保存した矩形画像を UMAP に入力し，12 万次元で表される各画像の特徴を 2 次元空間に写像したものである．UMAP のハイパーパラメータはデフォルト値を使用した．今回の事例では $x \leq 6$ の領域に森林域の画像の多くが存在していることが確認できる．一方，構造物域と田園域は混在しているため，場合によってはこれらの区別は困難であることが推察できる．

以上，開発したアノテーションツールを高解像度衛星画像に適用し，キーバインド機能によって効率的にアノテーション作業が実施できることを確認した．また，設定した各矩形領域を個別画像として保存する機能によって，アノテーション後の機械学習タスク (ここでは個別画像の UMAP への入力) が効率的に実行できることを確認した．

4. まとめと今後の課題

高解像度衛星画像を対象としたアノテーションツールの開発を目的として，アノテーションツールの実態調査からプロトタイプ開発までを実施した．本研究を通じて得られた知見は以下のとおりである．

1. アノテーションに必要となる労力や分野別の事例について情報収集を実施し，目的別に特化されたアノテーションツールが多く存在することを確認した．
2. 高解像度衛星画像に既往のアノテーションツールを適用し，4 つの課題を抽出した．
3. 得られた課題の解決を念頭において，画像ビューワ napari をベースとして，マウス操作とキーバインド機能によってアノテーションが実施できるツールを開発した．
4. 開発したツールを実際の高解像度衛星画像に適用し，意図したとおりに動作することを確認した．

今回作成したアノテーションツールは 4 クラス分類までしか対応させていないため，より多くの分類に対応させる予定である．また，現時点では矩形領域のみによるアノテーションが可能であるが，今後はセグメンテーションやキーポイント抽出に対応するため，ポリゴンやポイントを取り扱えるように拡張したい．

謝辞: 本研究では,「土木学会 地震工学委員会 防災・減災への AI・IoT 技術の利活用に関する研究小委員会」の活動の一環として購入された高解像度衛星画像を使用させていただきました．ここに記して謝意を表します．

付録 I　アノテーションツール開発のための napari への追加実装 (python スクリプト)

本研究で開発したアノテーションツールは，以下の python スクリプト (約 200 行) によって動作する．実行方法に関する説明を含むスクリプト一式は次の URL からダウンロードできるため，必要に応じて確認されたい．https://staff.aist.go.jp/h-saomoto/rect_N_class.zip

```
# napari に基づくアノテーションツールの作成 (産総研, 竿本)
import napari
import skimage
from numpy import *
from decimal import Decimal, ROUND_HALF_UP

viewer = napari.Viewer()
viewer.scale_bar.visible = True
viewer.scale_bar.unit = "pixel"

# size of crop tile
dx=200 # 200 pix => 300 m  (1.5m/pix)
dy=200

# colors (R, G, B, Opacity)
Red =     [1.0, 0.0, 0.0, 0.8]
Green =   [0.0, 1.0, 0.0, 0.8]
Blue =    [0.0, 0.0, 1.0, 0.8]
White =   [1.0, 1.0, 1.0, 0.8]
Yellow =  [1.0, 1.0, 0.0, 0.8]
Orange =  [1.0, 0.4, 0.0, 0.8]
Purple =  [0.5, 0.0, 1.0, 0.8]
Magenta = [1.0, 0.0, 1.0, 0.8]

rect_stack = []

# add a rectangle by pressing "F1"
@viewer.bind_key("F1", overwrite=True)
def cut_tile(viewer):
    y, x = viewer.cursor.position
    yc = int(y)
    xc = int(x)

    xm=xc-int(dx/2); ym=yc-int(dy/2)
    xp=xc+int(dx/2); yp=yc+int(dy/2)

    p1=[ym, xm]; p2=[yp-1, xp-1]
    rect = array([p1, p2])
    viewer.add_shapes(rect,
    ↪  shape_type='rectangle', edge_width=0.0,
    ↪  face_color=Yellow)
    # 黄色四角描画 (R, G, B, A) 色の最大値: 1.0

    rect_stack.append('0')

    r_id = len(viewer.layers)-1 # 黄色四角の ID
    r_name=viewer.layers[r_id].name # 黄色四角のレ
    ↪  イヤー名

    viewer.status = "Number of rect: %d" %
    ↪  (r_id) # 左下への黄色四角 ID の表示

# add a rectangle by pressing "F2"
@viewer.bind_key("F2", overwrite=True)
def cut_tile(viewer):
    y, x = viewer.cursor.position
    yc = int(y)
    xc = int(x)

    xm=xc-int(dx/2); ym=yc-int(dy/2)
    xp=xc+int(dx/2); yp=yc+int(dy/2)

    p1=[ym, xm]; p2=[yp-1, xp-1]
    rect = array([p1, p2])
    viewer.add_shapes(rect,
    ↪  shape_type='rectangle', edge_width=0.0,
    ↪  face_color=Red)
    # 赤四角描画 (R, G, B, A) 色の最大値: 1.0

    rect_stack.append('1')

    r_id = len(viewer.layers)-1 # 赤四角の ID
    r_name=viewer.layers[r_id].name # 赤四角のレイ
    ↪  ヤー名

    viewer.status = "Number of rect: %d" %
    ↪  (r_id) # 左下への赤四角 ID の表示

# add a rectangle by pressing "F3"
@viewer.bind_key("F3", overwrite=True)
def cut_tile(viewer):
    y, x = viewer.cursor.position
    yc = int(y)
    xc = int(x)

    xm=xc-int(dx/2); ym=yc-int(dy/2)
```

```
    xp=xc+int(dx/2); yp=yc+int(dy/2)

    p1=[ym, xm]; p2=[yp-1, xp-1]
    rect = array([p1, p2])
    viewer.add_shapes(rect,
    ↪  shape_type='rectangle', edge_width=0.0,
    ↪  face_color=Green)

    rect_stack.append('2')

    r_id = len(viewer.layers)-1 # 四角の ID
    r_name=viewer.layers[r_id].name # 四角のレイ
    ↪  ヤー名

    viewer.status = "Number of rect: %d" %
    ↪  (r_id) # 左下への四角 ID の表示

# add a rectangle by pressing "F4"
@viewer.bind_key("F4", overwrite=True)
def cut_tile(viewer):
    y, x = viewer.cursor.position
    yc = int(y)
    xc = int(x)

    xm=xc-int(dx/2); ym=yc-int(dy/2)
    xp=xc+int(dx/2); yp=yc+int(dy/2)

    p1=[ym, xm]; p2=[yp-1, xp-1]
    rect = array([p1, p2])
    viewer.add_shapes(rect,
    ↪  shape_type='rectangle', edge_width=0.0,
    ↪  face_color=Blue)

    rect_stack.append('3')

    r_id = len(viewer.layers)-1 # 四角の ID
    r_name=viewer.layers[r_id].name # 四角のレイ
    ↪  ヤー名

    viewer.status = "Number of rect: %d" %
    ↪  (r_id) # 左下への四角 ID の表示


# remove a rectangle at the end of list by
↪  pressing "u" (undo)
@viewer.bind_key("u", overwrite=True)
def remove_tile(viewer):
    r_id = len(viewer.layers)-1 # 最後に追加した黄色
    ↪  四角の ID
    r_name=viewer.layers[r_id].name # 最後に追加し
    ↪  た黄色四角のレイヤー名

    viewer.layers.remove(r_name) # 最後に追加した黄
    ↪  色四角レイヤーの削除
    rect_stack.pop() # ボタン種別 (F1, F2,,) リストの
    ↪  末尾削除
    r_id = len(viewer.layers)-1 # 最後に追加した黄色
    ↪  四角の ID
    viewer.status = "Number of rect: %d" %
    ↪  (r_id) # 左下への黄色四角 ID の表示

# save the stacked tiles as croped figures with
↪  "s" key (save)
@viewer.bind_key("s", overwrite=True)
def save_tile(viewer):
    img = viewer.layers[0].data #ベース画像の画素値
    ↪  読み込み
    path_name =
    ↪  viewer.layers[0].source.path.split("/")
    ↪  #ベース画像の画素値読み込み
    base_name= path_name[-1].split(".")[0]

    tate = img.shape[0] # pixel size for
    ↪  y-direction
    yoko = img.shape[1] # pixel size for
    ↪  x-direction

    f_coord_name =base_name+"_coord.txt"
    fout=open(f_coord_name, "w") # save the rect
    ↪  coordinates

    f_yolo_name =base_name+".txt"
    fyolo=open(f_yolo_name, "w") # save the rect
    ↪  coordinates

    ifile=0
    print("------------------------")
    print(path_name[-1]) # ファイル名
    for irect in range(1,len(viewer.layers)): #
    ↪  0 ではなく 1 からスタート
        ifile=ifile+1

        coord = viewer.layers[irect].data[0] #
        ↪  get 4 points coordinates of a
        ↪  rectangle
        icolor=rect_stack[irect-1]

        y0 = coord[0,0]; x0 = coord[0,1] # 左上の
        ↪  y, x (実数なので以下で四捨五入)
        y1 = coord[1,0]; x1 = coord[1,1] # 左下の
        ↪  y, x (実数なので以下で四捨五入)
        y2 = coord[2,0]; x2 = coord[2,1] # 右下の
        ↪  y, x (実数なので以下で四捨五入)
        y3 = coord[3,0]; x3 = coord[3,1] # 右上の
        ↪  y, x (実数なので以下で四捨五入)

        y0_int =
        ↪  Decimal(str(y0)).quantize(Decimal('0'),
        ↪  rounding=ROUND_HALF_UP)
        x0_int =
        ↪  Decimal(str(x0)).quantize(Decimal('0'),
        ↪  rounding=ROUND_HALF_UP)

        y1_int =
        ↪  Decimal(str(y1)).quantize(Decimal('0'),
        ↪  rounding=ROUND_HALF_UP)
        x1_int =
        ↪  Decimal(str(x1)).quantize(Decimal('0'),
        ↪  rounding=ROUND_HALF_UP)

        y2_int =
        ↪  Decimal(str(y2)).quantize(Decimal('0'),
        ↪  rounding=ROUND_HALF_UP)
        x2_int =
        ↪  Decimal(str(x2)).quantize(Decimal('0'),
        ↪  rounding=ROUND_HALF_UP)

        y3_int =
        ↪  Decimal(str(y3)).quantize(Decimal('0'),
        ↪  rounding=ROUND_HALF_UP)
        x3_int =
        ↪  Decimal(str(x3)).quantize(Decimal('0'),
        ↪  rounding=ROUND_HALF_UP)

        ym = int(y0_int); yp = int(y1_int)+1
        xm = int(x0_int); xp = int(x3_int)+1
        fout.write("%s %d %d %d %d\n"%(icolor,
        ↪  xm, xp, ym, yp))

        xc = (0.5*(xm+xp))/float(yoko); yc =
        ↪  (0.5*(ym+yp))/float(tate)
        ww = float(xp-xm)/float(yoko); hh =
        ↪  float(yp-ym)/float(tate)
        fyolo.write("%s %f %f %f %f\n"%(icolor,
        ↪  xc, yc, ww, hh))

        #i_tile = img[int(c_ind), ym:yp, xm:xp]
        ↪  # 複数チャンネルの場合
        i_tile = img[ym:yp, xm:xp, 0:3] # 黄色四角
        ↪  領域の画素値 タテ・ヨコ・RGB(0,1,2)
        tile_name="./crop_img/tile%04d.png" %
        ↪  (ifile)
```

```
            print(f_coord_name, i_tile.shape, [ym,
            ↪  yp, xm, xp], icolor)

            skimage.io.imsave(tile_name, i_tile) # ク
            ↪  ロップ画像のセーブ

    fout.close()
    fyolo.close()

    viewer.status = "coordinates of tiles are
    ↪  saved!"

if __name__ == '__main__':
    napari.run()
```

参考文献

1) 全邦釘: 土木工学分野における人工知能技術活用のために解決すべき課題と進めるべき研究開発, AI・データサイエンス論文集, Vol.1, No.J1, pp.9–15, 2020.
2) Avci, O., Abdeljaber, O., Kiranyaz, S., Hussein, M., Gabbouj, M., and Inman, D. J.: A review of vibration-based damage detection in civil structures: From traditional methods to machine learning and deep learning applications, *Mech. Syst. Signal Process.*, Vol.147, pp.107077, January 2021.
3) Akinosho, T. D., Oyedele, L. O., Bilal, M., Ajayi, A. O., Delgado, M. D., Akinade, O. O., and Ahmed, A. A.: Deep learning in the construction industry: A review of present status and future innovations, *Journal of Building Engineering*, Vol.32, pp.101827, November 2020.
4) 全邦釘, 井後敦史: Random forest によるコンクリート表面ひび割れの検出, 土木学会論文集 F3 (土木情報学), 2015.
5) 横山傑, 松本高志: Deep learning によるコンクリートの変状自動検出器の開発と web システムの実装, 土木学会応用力学論文集 A2, Vol.73, No.2, pp.I_781–I–789, 2017.
6) 青島亘佐, 河村伸哉, 中野聡, 中村秀明: 深層学習による画像認識を用いたコンクリート構造物の変状検出に関する研究, 土木学会論文集 E2（材料・コンクリート構造）, Vol.74, No.4, pp.293–305, 2018.
7) 前田紘弥, 関本義秀, 瀬戸寿一, 樫山武浩, 小俣博司: 機械学習とスマートフォンを用いた道路の損傷画像のリアルタイム検出と修繕対応基準における各特徴量の重要度比較, 交通工学論文集, Vol.4, No.3, pp.A_1–A_8, 2018.
8) 全邦釘, 井後敦史, 南免羅裕治, 黒木航汰, 大窪和明: 車載カメラにより撮影された舗装画像からのディープラーニングによるひび割れ率評価, 土木学会論文集 E1（舗装工学）, Vol.73, No.3, pp.I_97–I_105, 2017.
9) 鈴木達也, 西尾真由子: 橋梁定期点検における部材損傷度判定への深層学習の適用に関する検討, 土木学会論文集 F3（土木情報学）, Vol.75, No.1, pp.48–59, 2019.
10) 龍田斉, 横山広, 永見武司, 桝谷浩, 近田康夫, 山田宗明: 勾配ブースティング決定木を用いた橋梁損傷原因および補修工法の推定と分析, AI・データサイエンス論文集, Vol.1, No.1, pp.63–70, 2020.
11) Cognilytica Inc.: Data engineering, preparation, and labeling for AI 2019, 2019.
12) Cognilytica Inc.: White paper: AI data engineering lifecycle checklist, 2020.
13) Gomes-Pereira, J. N., Auger, V., Beisiegel, K., Benjamin, R., Bergmann, M., Bowden, D., Buhl-Mortensen, P., De Leo, F. C., Dionísio, G., Durden, J. M., Edwards, L., Friedman, A., Greinert, J., Jacobsen-Stout, N., Lerner, S., Leslie, M., Nattkemper, T. W., Sameoto, J. A., Schoening, T., Schouten, R., Seager, J., Singh, H., Soubigou, O., Tojeira, I., van den Beld, I., Dias, F., Tempera, F., and Santos, R. S.: Current and future trends in marine image annotation software, *Prog. Oceanogr.*, Vol.149, pp.106–120, December 2016.
14) Dong, Q., Luo, G., Haynor, D., O'Reilly, M., Linnau, K., Yaniv, Z., Jarvik, J. G., and Cross, N.: DicomAnnotator: a configurable Open-Source software program for efficient DICOM image annotation, *J. Digit. Imaging*, Vol.33, No.6, pp.1514–1526, December 2020.
15) 藤本東, 小川徹, 山本和慶, 松井勇佑, 山崎俊彦, 相澤清晴: 学術用アノテーション付き漫画画像データセットの構築と解析, 映像情報メディア学会技術報告, Vol.41.5, pp.35–40, 2017.
16) 郷右近英臣, Joachim, P., Enrico, S., Sandro, M., Twele, A., Mück, M., 越村俊一: TerraSAR-X 画像の機械学習による津波被災地の自動検出, 土木学会論文集 B2(海岸工学), Vol.69, No.2, pp.I_1441–I_1445, 2013.
17) Miyamoto, T. and Yamamoto, Y.: Using multimodal learning model for earthquake damage detection based on optical satellite imagery and structural attributes, *IGARSS 2020 - 2020 IEEE International Geoscience and Remote Sensing Symposium*, pp. 6623–6626, September 2020.
18) Miyamoto, T. and Yamamoto, Y.: Using 3-D convolution and multimodal architecture for earthquake damage detection based on satellite imagery and digital urban data, *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, Vol.14, pp.8606–8613, 2021.
19) Sofroniew, N., Lambert, T., Evans, K., Nunez-Iglesias, J., Bokota, G., Winston, P., Peña-Castellanos, G., Yamauchi, K., Bussonnier, M., Doncila Pop, D., Can Solak, A., Liu, Z., Wadhwa, P., Burt, A., Buckley, G., Sweet, A., Migas, L., Hilsenstein, V., Gaifas, L., Bragantini, J., Rodríguez-Guerra, J., Muñoz, H., Freeman, J., Boone, P., Lowe, A., Gohlke, C., Royer, L., Pierré, A., Har-Gil, H., and McGovern, A.: napari: a multi-dimensional image viewer for Python, May 2022.
20) Microsoft Corp.: VoTT: Visual object tagging tool, 2018.
21) Wada, K.: labelme: Image polygonal annotation with python, 2018.
22) Tzutalin: LabelImg, Free Software: MIT License, 2015.
23) Hollandi, R., Diósdi, Á., Hollandi, G., Moshkov, N., and Horváth, P.: AnnotatorJ: an ImageJ plugin to ease hand annotation of cellular compartments, *Mol. Biol. Cell*, Vol.31, No.20, pp.2179–2186, September 2020.
24) Jocher, G., Chaurasia, A., Stoken, A., Borovec, J., NanoCode, Kwon, Y., TaoXie, Michael, K., Fang, J., imyhxy, Lorna, Wong, C., 曾逸夫 (zengYifu), Abhiram, V., Montes, D., Wang, Z., Fati, C., Nadar, J., Laughing, UnglvKitDe, tkianai, yxNONG, Skalski, P., Hogan, A., Strobel, M., Jain, M., Mammana, L., and xylieong: ultralytics/yolov5: v6.2 - YOLOv5 Classification Models, Apple M1, Reproducibility, ClearML and Deci.ai integrations, August 2022.
25) McInnes, Healy, Saul, and Grossberger: UMAP: Uniform manifold approximation and projection, *J. Open Source Softw.*, Vol.3, No.29, pp.861, 2018.

DEVELOPMENT OF ANNOTATION TOOL INCLUDING KEYBINDINGS FOR FAST LABELING ON HIGH-RESOLUTION SATELLITE IMAGES

Hidetaka SAOMOTO, Takashi MIYAMOTO and Akira SAITO

Machine learning based on deep learning with image data is widely used in the civil engineering field. Annotation is one of the time-consuming processes in such machine learning research, and it is demanded to reduce the annotating cost using GUI annotation tools. This study aims to develop an annotation tool for high-resolution satellite images, which have recently been used in machine learning for hazard risk analysis. First, we identify requirements specific to high-resolution satellite images using existing annotation tools: quick response irrespective of huge file size; quick zoom-in/out; keybindings to avoid many mouse operations. Next, an annotation tool that meets these requirements is realized by adding functionalities to an open-source image viewer (napari). Finally, performance verification is conducted through two classification problems with the developed annotation tool, demonstrating rapid workability by virtue of the keybindings.