

横路のインターンドキュメント

概要

ARとGANを組み合わせて室内の景色を変化させるシステムを開発した。

使用言語

- Python3.7
- C#

開発環境

- PyCharm 2019.1.3 x64
- Unity 2018.2.21f1

使用ライブラリ

Python

- pytorch
- pytorchvision
- PIL
- cv2
- numpy
- websocket-server

Unity

- SteamPlugin2.2
- SRWorks
- WebSocket-Sharp

深層学習モデル

- StarGAN
- DeepLabV3+

学習用データセット

StarGAN用

<https://drive.google.com/open?id=1mJrD8soJTUsm3s4yxzHiTDm-gUXe1qpe>

各ラベルを持つ画像の枚数は以下の通り

ラベル名	枚数
black_floor	634
red_brick_wall	1004
white_floor	1809
white_wall	3400
wood_floor	2914
wood_wall	486
ocean	12081
sky	18191

DeepLabV3+用

<http://sceneparsing.csail.mit.edu/>

[train/val (922MB)] からダウンロード

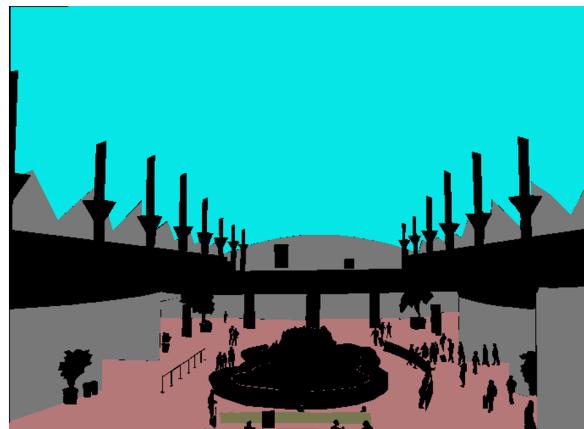
<https://github.com/yokoro13/utils/blob/master/ADE20Kto8Classes.py> を実行しラベルを変更

実行に必要なディレクトリは適宜作成

ADE20Kはインデックスカラー表現の8bit画像になっているので注意。

対応表：

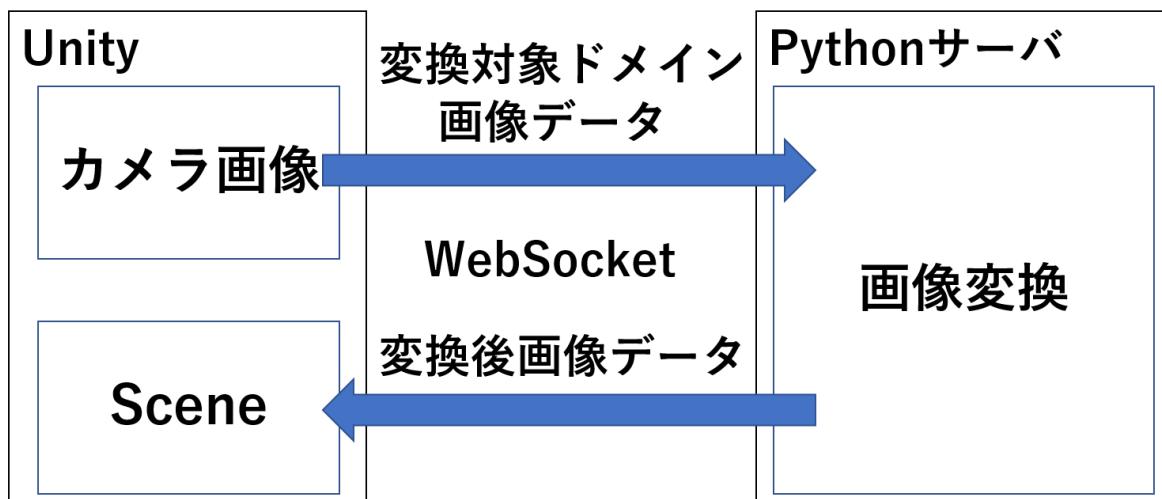
https://github.com/CSAILVision/semantic-segmentation-pytorch/blob/master/data/object150_info.csv



システム構成

Unity で表示を行い、Pythonサーバで変換処理を行う。

システムの構成を以下の図に示す。

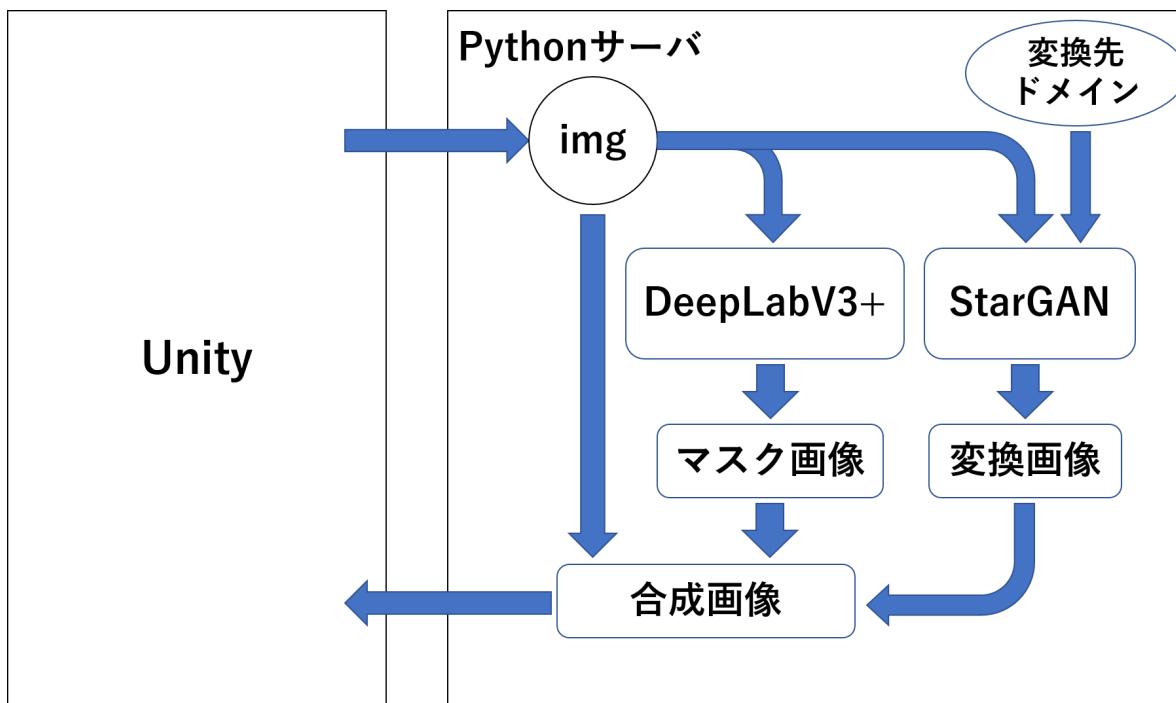


Python処理

Pythonサーバでの処理を以下の図に示す。

処理の流れとしては以下の通り

1. Unityから受信した画像データをStarGANで画像変換を行う
2. 対象物体のみに変換を適用させるためにDeepLabV3+で対象の物体に対してのマスク画像を作成する
3. マスク画像を使用して元画像と変換画像の合成を行い、合成画像を作成する
4. 合成画像をJPEGでエンコードしてUnity側にバイト配列で送信する



実行結果（床を海へ変換）：



入力画像



出力画像

Unity処理

Unity側での処理は以下の通り

- カメラ画像をサーバに送信する
- 受信したデータをテクスチャとして反映する
- 変換先ドメインをコントローラで設定し、サーバに送信する

SRWorksでのカメラの画像データの取得方法

以下の変数から取得可能

```
namespace Vive.Plugin.SR
{
    public class ViveSR_DualCameraImageRenderer : MonoBehaviour
    {
        // 略

        private void Update()
        {
            // 略
            # region Undistored Image
            // 略
            // ここで操作でカメラ画像を配列に格納している
            ViveSR_DualCameraImageCapture.GetUndistortedTexture(out
TextureUndistorted[(int)DualCameraIndex.LEFT], out
TextureUndistorted[(int)DualCameraIndex.RIGHT], out FrameIndexUndistorted, out
TimeIndexUndistorted, out PoseUndistorted[(int)DualCameraIndex.LEFT], out
PoseUndistorted[(int)DualCameraIndex.RIGHT]);

            // 略
            // この配列にTexture2Dで格納されている
            TextureUndistorted[(int)DualCameraIndex.LEFT]

            // 略
        }
    }
}
```

StarGAN

複数ドメインでの画像変換を行うために、StarGANを使用した。

ソースコード: <https://github.com/yokoro13/stargan>

main.pyと同じ階層にdataという名前のディレクトリを作成し、そこにダウンロードしたデータセットを置く。

学習の際、出力画像のサイズは256x256にした。

実行結果



DeepLabV3+

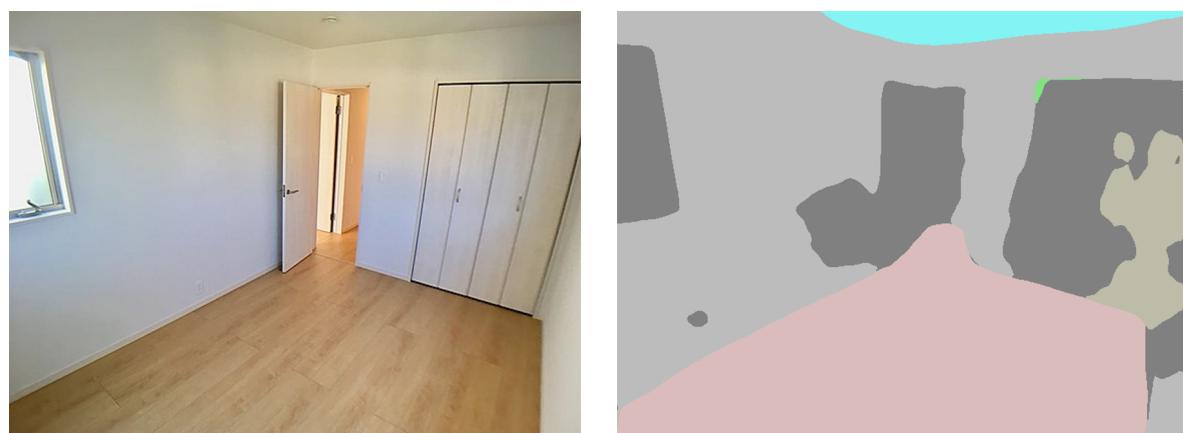
マスク画像を作成するためにDeepLabV3+を使用した。

ソースコード: <https://github.com/yokoro13/pytorch-deeplab-xception>

mypath.pyに記述してある場所にデータセットを置く。

学習の際、出力画像のサイズは512x512にした。

実行結果



データ通信

PythonサーバとUnityの通信を行うために, WebSocketを使用した.

WebSocket用ライブラリ

Python用

websocket-server はbyte配列での通信が行えないので改良した.

以下のコマンドを実行してインストール

```
pip install git+https://github.com/yokoro13/python-websocket-server
```

Unity用

既存ライブラリのwebsocket-sharpではデータの送信量に制限がかけられていたので改良した.

<https://github.com/yokoro13/websocket-sharp> をcloneしビルドする.

ビルド方法はここを参照

<https://qiita.com/oishihiroaki/items/bb2977c72052f5dd5bd9>

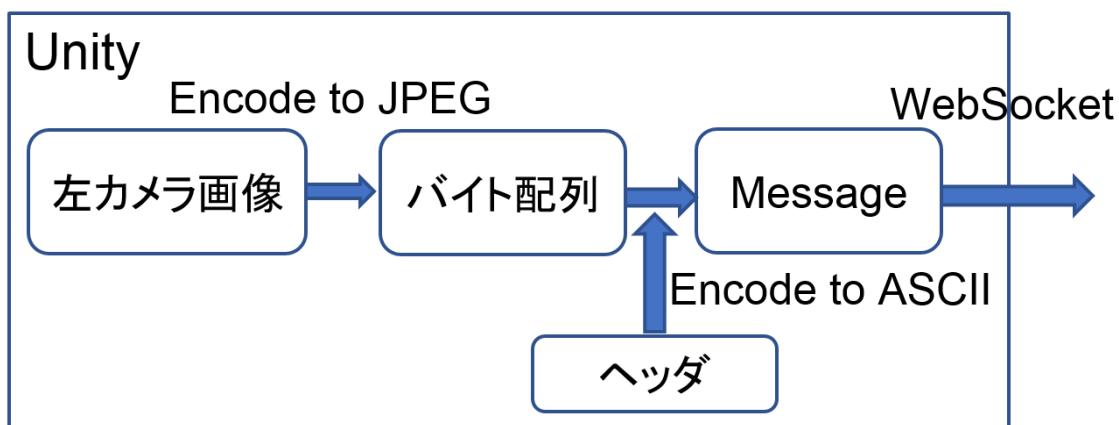
通信方法

ヘッダの設定ができなかつたため, データの先頭6バイトをヘッダとして扱うようにした.

ヘッダの種類は以下の通り

ヘッダ	内容
/cmr00	左カメラの画像データ
/cmr01	右カメラの画像データ
/label	ターゲットドメイン

データ送信時の流れ



送信データの内容



ヘッダの判定方法

ヘッダの判定は以下のように実装

```
header = message[:6]      # 始め 6byte は data の種類を記述しておく
d_header = header.decode("ascii")
contents = message[6:]    # 6byte より後ろは data の内容

# labels = [黒床, 赤レンガ壁, 白床, 白壁, 木床, 木壁, 海面, 空]
# /label のときはlabelsのインデックスが空白で区切って送られてくる
# (ex. 黒床、白壁の場合 "0 3" が送られてくる)
if d_header == "/label":
    target_labels = contents.decode("ascii")
    print(target_labels)
    target_labels = target_labels.split(" ")
    target_labels = target_labels[:len(target_labels) - 1]
    translation.set_target_labels(target_labels)
# cmr00 は左カメラ cmr01は右カメラ
elif d_header == "/cmr00" or d_header == "/cmr01":
    image = Image.open(io.BytesIO(contents))

    # 処理
    image = ImageOps.flip(image)
    # image = translation.test_translation(image)
    image = translation.trans_interior(image)
    image = ImageOps.flip(image)

    buf = io.BytesIO()
    image.save(buf, "JPEG")
    message[6:] = buf.getvalue()

server.send_message(client, message)
```

```
private void Start()
{
    isReceivedMessage = new bool[2];
    header = new byte[6];

    isReceivedMessage[0] = false;
    isReceivedMessage[1] = false;

    ws = new WebSocket("ws://127.0.0.1:5000/");
    ws.OnOpen += (sender, e) => {
        Debug.Log("WebSocket Open");
    };
    ws.OnMessage += (sender, e) => {
        Debug.Log("On Message");
        if
(Encoding.GetEncoding("ascii").GetString(e.RawData.Take(6).ToArray()) ==
"/cmr00") // 先頭6バイトはヘッダ
    {
        isReceivedMessage[0] = true;
        receive_L = e.RawData.Skip(6).ToArray(); // 6バイトより後ろはデータ
    }
    else
```

```
        {
            isReceivedMessage[1] = false;
            receive_R = e.RawData.Skip(6).ToArray(); // 6バイトより後ろはデータ
        }
    };
    ws.OnError += (sender, e) => {
        Debug.Log("WebSocket Error Message: " + e.Message);
    };
    ws.OnClose += (sender, e) => {
        Debug.Log("WebSocket Close" + e);
        print(ws);
    };
    ws.Connect();
}
```