NestedUNet で使用する画像をネット上から zip 形式でダウンロードする

昨日までの結果から，`UNet` に癌の細胞画像を訓練させた結果，入力に似た画像を生成できることが分かった．そこで，次に別のデータセットでも再現性があるかを確認する．そのために，データセットを `URL` を指定してダウンロードし，訓練に使用することができるように展開したい．

目標は，以下の処理が自動で行えることとする．

1. データの保存先を URL で指定
2. zip 形式で `temp` ディレクトリにダウンロード
3. 任意の場所に展開

URL Requests

python の `requests` パッケージを使用して Web 上の `HTML` に含まれる特定の情報を検索する.

ここで, `requests` パッケージとは, **人が使いやすいように作られた、エレガントでシンプルな Python の HTTP ライブラリ** である『Requests: クイックスタート』.

もう 1 つ参考『Python Requests モジュールについて』

使い方

request モジュールのインポート

```
import requests
```

web ページを取得する.

```
res = requests.get('https://github.com/timeline.json')
```

この `request` オブジェクト(`res`) から必要なすべての情報を取得することができる.

レスポンスの処理

requests を使った画像のダウンロード を参考にホームページから画像をダウンロードする関数を作成する.

そのために, `request.get` で得られたオブジェクトのヘッダの `content-type` を調べる.

```
content_type = res.headers["content-type"]
```

そして, `content_type` の中に `image` の文字列が含まれていた場合, そのコンテンツをダウンロードすることで, URL と同様の画像をダウンロードすることができる.

```
if 'image' not in content_type:
  e = Exception("Content-Type: " + content_type)
  raise e

return res.content
```

この関数を実際に動作させてみると, 次のような文字列が取得できる.

```
b' \x89PNG\r\n\x1a\n\x00\x00\x00\rIHDR\...
```

これを `type` で確認すると, `<class 'bytes'>` と表示された.

画像を保存

先ほど得られた文字列を，特定の拡張子としてファイルに書き込むことで，画像として PC に表示することができる．先ほどの HP を参考にしつつ，こちらも作成していく．

画像ファイルは，`bytes` クラスであろうと適切な拡張子が指定されていれば，`write` コマンドでファイルに書き込むことで作成できる．

すなわち，

```
def save_image(filename, image):
    with open(filename, "wb") as fout:
        fout.write(image)
```

Zip ファイルをダウンロードする

zip も `requests` を使用することで，ダウンロードすることができる．

zip ファイルをダウンロードする場合は，ファイルサイズが大きくても問題ないように `requests.get` の `steam` オプションや `iter_content` を設定する必要がある．

『Python、Requests を使ったダウンロード』によると，`res.iter_content()` を使用することで，データを数回に分けて書き込むことができるようである．

```
import requests

def download_file(url):
  """URL を指定してカレントディレクトリにファイルをダウンロードする
  """
  res = requests.get(url, stream=True)
```

作業の進捗を表示する.

大量のデータを含む zip ファイルをダウンロードする場合，処理がどの程度進んでいるかをダウンロードしたファイルの数から判断するのは，手間がかかる．そこで，作業の進捗をコマンドライン上に表示する関数を作成する．

1. 『Python でファイルをダウンロード＆プログレスバーを表示させる』
2. 『Python3 - ファイルのダウンロード』
   を参考に目標の関数を作成した．

```
def _ProgressPrint(block_count: int, block_size: int, total_size: int):
    """作業の進捗を表示するための関数
        イメージ：[=====>    ] 50.54% ( 1050KB )

    Args:
        block_count (int): 1回の処理で処理されるファイルサイズ
        block_size (int): チャンクサイズ
        total_size (int): トータルのファイルサイズ
    """
    percentage = 100 * block_count * block_size / total_size
    # 100 より大きいと見た目が悪いので...
    if percentage > 100:
        percentage = 100
    # バーは max_bar 個で 100% とする
    max_bar = 10
    bar_num = int(percentage / (100 / max_bar))
    progress_element = '=' * bar_num
    if bar_num != max_bar:
        progress_element += '>'
    bar_fill = ' ' # これで空のところを埋める
    bar = progress_element.ljust(max_bar, bar_fill)
    total_size_kb = total_size / 1024
    print(
        f'[{bar}] {percentage: .2f}% ( {total_size_kb: .0f}KB )\r', end=''
    )
```

上記の関数を zip ファイルダウンロード関数の中に配置し，実際に zip ファイルのダウンロードを行ったところ，途中でメモリリークと思われるエラーが発生した．そこで，次の 2 つのサイトを参考に関数の最後に `del` 文を追加し，それまで使用した変数をすべて削除することにした．

1. 【Python 入門】メモリの解放や効率的に使う方法をマスターしよう！
2. 【python】python でメモリ不足になったときにすること

しかし，この処理を行っても3つ目の zip ファイルをダウンロードするあたりでスタックが原因と思われる動作停止が発生した．この原因としては，コマンドラインの同じ行にすべてのファイルのダウンロード進捗状況を表示していたため，その部分でメモリリークが発生していたのではないかと考えられる．そこで，ダウンロードファイルを表示する `print` 文を追加し，ファイルごとに表示場所が1行ずつずれるようにした．

その結果，スタックすることなくスムーズに動作できるようになった．