

コンピュータビジョンプログラミング課題

担当 : 井尻敬

v1公開 2024/04/28

締め切り

課題前半（問01~問11）: 07月05日

課題後半（問12~問23）: 07月26日

提出方法

『cv_学籍番号』というフォルダを作成し、その中に解答となるコードをすべて入れて、そのフォルダをzip圧縮し、出来上がったzipファイルをscombzより提出してください。

※ フォルダ名の例: cv_AL22000

※ zipファイル名の例: cv_AL22000.zip

雛形 :

講義ページ (<https://takashijiri.com/classes/cv/>) に雛形があるので、必ず利用してください。雛形ファイル名は変更しないでください（課題XXのファイル名はexerXX.pyのまま）。

準備 :

- Step1 : 講義ページより、『[python_install.pdf](#)』と『[template.zip](#)』を取得してください
- Step2 : [python_install.pdf](#) に従い Pythonをインストールしてください
- Step3 : [template.zip](#)を解凍してください（フォルダをダブルクリック > 全て展開 を選択）
- Step4 : コマンドプロンプトを開き、解凍してできたtemplate フォルダをカレントディレクトリにしてください（templateフォルダを開いて アドレスバーに「cmd」と打ち込むなど。）
- Step5 : 下記のコマンドを実行し、テンプレートを初期化してください

```
>python cv_starter.py
```

- Step6: 初期化キーの提出

実行に成功すると、以下の通り出力されます。

```
>python cv_starter.py
initialize successXXXXXXXXXXXX.XXXXXXXXXX
```

この出力された一行『initialize successXXXXXXXXXXXX.XXXXXXXXXX』をscombzの『[テスト : 課題準備](#)』より提出してください。「XXXXXXXXXXXX.XXXXXXXXXX」には、数値が入ります。

この数値を不正行為防止に利用するため、提出がない場合は採点できません。また、ここで初期化した雛形を必ず利用して解答して下さい（ファイル紛失などの際はご連絡ください）。

提出時の注意 :

- **仕様を守ってください。**採点自動化のため『フォルダ名・ファイル名が間違っているもの・入出力の仕様を満たさないもの』は評価できず0点扱いとなることがあります。各課題について、コマンドライン引数の詳細な仕様など、入出力の仕様が指定されます。正しく従ってください。
- **動作確認をしてください。**各課題に入出力例が用意されています。必ず提出前に、自身で作成したプログラムを実行し正しく動作することを確認してください。（採点時には配布したものとは別のデータを使います）
- **極端に動作が遅いコードはNG。**本課題では計算速度を重視しませんが、極端に長い時間のかかるもの（20秒以上）は、自動採点の都合上0点とします。

その他の注意 :

- **本課題および解答をweb上に公開しないでください。**この課題の解答となるコードを、この課題の解答と分かる形でWeb上に公開する事は避けてください。また、ナリッジコミュニティサイト（知恵袋やteratailなど）にて、問題文をそのまま掲載し、解答を得る事は行なわないでください。
- **解答は自分で書いてください。**学生同士が教え合うことは問題ありませんが、解答となるソースコードは必ず自分で作成して提出してください。解答となるソースコードの他人との共有や、ソースコードのコピーが発見された場合には、コピー元・コピー先の両名ともカンニングと同様の処置をとります。※Pythonはコードがどうしても似てしまうことは理解しています。していないカンニングの疑いをかけられないかと不安になったり、あえて”へんな書き方”をする必要はないです。
- **生成AIは使わないでください。**この課題は、自然言語で書かれた課題を理解しプログラムに変換する訓練を行うことで、知識と経験の蓄積を目指すものです。そのため、chatGPT等の生成系AIの利用は禁止します。（このような比較的単純な課題を解く経験を蓄積することで、研究や仕事における実問題を解くときに『こういう課題ならweb上にたくさんコードがあるので生成系AIもいい結果を出してくれそう』という予測ができる、AIの支援を受けながら作業を加速する能力が付くと思います。）

目次

Day 1

01. 平均画素値の計算	★	★ (易)
02. Sum of Squared Difference	★	★★ (普通)
03. PSNR	★	★★★ (やや難)
04. 画像復元	☆☆☆☆	★★★★ (難)

Day 2

05. Template Matching 1	★★	
06. Template Matching 2	★★	
07. Template Matching 3	★	
08. Template Matching 4	★★★	

Day 3

09. Harrisのコーナー検出 1	★★
10. Harrisのコーナー検出 2	★★★
11. Harrisのコーナー検出 3	★★★

Day 4

12. Canny Filter	★
13. Hough変換 1	★★★
14. Hough変換 2	★★★
15. seam carving (発展)	☆☆☆☆

Day 5

16. Histogramの計算	★
17. Otsu法	★★
18. Region Growing	★★★
19. 領域を数える	★★★★★

Day 6

20. パターン認識1 MNIST	★
21. パターン認識2 SVM	★★
22. パターン認識3 kNN	★★
23. Texture Synthesis (発展)	☆☆☆☆

※ 離形に多くのヒントがあります。まずは離形にあるコードを読んでください。

※ 少多少易度の高い課題もあるので、全くわからない課題があっても気を落とさないでください。（全課題を簡単に解いてしまう方が毎年数パーセントいるのも事実ではあるのですが。。。）

課題01 平均画素値の計算

画像を1枚読み込み、グレースケール化した後、全画素の平均値と中央値を出力せよ。

詳細な仕様

- 下記の実行コマンドの通り、入力ファイル名(img.png)はコマンドライン引数より取得すること ※ 下は各課題の実行コマンドを示します

```
> python exer01.py img.png
```

- 平均値と中央値は標準出力に2行で出力すること (1行目が平均値, 2行目が中央値)
- 標準出力には計算結果以外を出力しないこと

※ すべての課題にひな形 exerXX.pyが用意してあるので参考にしてください。

入出力例

```
>python exer01.py tm/img1.png  
106.76513586956521  
95.0  
>python exer01.py tm/img2.png  
106.36149456521738  
94.0  
>python exer01.py tm/img3.png  
106.1773097826087  
93.0
```

./tmフォルダ内のimg1.pngの平均画素値は106.76513586956521、中央値は95.0です。

./tmフォルダ内のimg2.pngの平均画素値は106.36149456521738、中央値は94.0です。

./tmフォルダ内のimg3.pngの平均画素値は106.1773097826087、中央値は93.0です。

※ 各課題の入出力例を提供します。画像ファイル等は雛形に同梱してあります。

※ 自身で作成したプログラムの出力と比較することでデバッグに利用してください。

※ 数値の微妙な誤差は許容します。

02 Sum of Square Difference

サイズの同じ2枚の画像を読み込み、グレースケール画像に変換後、2枚の画像間のSSD値を出力せよ。

- 入力ファイル名(img1.png、img2.png)はコマンドライン引数より取得せよ

```
> python exer02.py img1.png img2.png
```

- 入力画像は、比較前にグレースケール画像に変換すること
- 計算したSSD値は標準出力に出力すること
- 標準出力には計算結果以外を出力しないこと

入出力例

```
> python exer02.py tm/img1.png tm/img2.png  
2249066.0
```

```
> python exer02.py tm/img1.png tm/img3.png  
12536530.0
```

./tmフォルダ内のimg1.pngとimg2のSSDは、2249066.0です。

./tmフォルダ内のimg2.pngとimg3のSSDは、12536530.0です。

03 PSNR (Peak signal-to-noise ratio)

サイズの同じ2枚の画像を読み込み、グレースケール化した後、2枚の画像間のMean Squared Error (MSE) とPeak signal-to-noise ratio (PSNR) を出力せよ。

- ふたつの入力ファイル名(img1.png、img2.png)はコマンドライン引数より取得せよ

```
> python exer03.py img1.png img2.png
```

- 入力画像は、比較前にグレースケール画像に変換すること
- MSE値を標準出力の1行目に、PSNR値を標準出力の2行目に出力すること
- 標準出力には、計算結果以外を出力しないこと
- PSNRとは、信号とノイズの比を表す数値のことであり、下記の通り計算できる。I₁とI₂は入力画像、Mは画像の取りうる最大値（ここでは255）、HとWは画像の高さと幅である。

$$\text{PSNR}(I_1, I_2) = 10 \log_{10} \frac{M^2}{\text{MSE}(I_1, I_2)}$$

$$\text{MSE}(I_1, I_2) = \frac{1}{WH} \sum_{y=0}^{H-1} \sum_{x=0}^{W-1} (I_1(y, x) - I_2(y, x))^2$$

ここでは、`sklearn.metrics.mean_squared_error`、`cv2.PSNR()`、`skimage.metrics.peak_signal_noise_ratio()`は利用せず、MSEやPSNRは自作すること。

入出力例

```
> python exer03.py tm/img1.png tm/img2.png  
61.11592391304349  
30.269259793235683  
> python exer03.py tm/img1.png tm/img3.png  
340.6665760869565  
22.807508352632627
```

./tmフォルダ内のimg1.pngとimg2のMSEとPSNRは、おおよそ61.1と30.3です。

./tmフォルダ内のimg1.pngとimg3のMSEとPSNRは、おおよそ340.7と22.8です。

※ PSNRは、例えば、『元画像』と『劣化画像』を入力として、劣化画像がどの程度元画像に近いかを評価するために利用されます。img2はimgを50%に縮小してから元のサイズに拡大したもので、img3は20%に縮小してから拡大したものです。

04 劣化画像の復元 (発展)

※余裕のある方向けの課題

※講義中にTAか教員に確認をうけること。

ランダムに選択された30%の画素が黒く塗りつぶされた劣化画像を入力とし、なるべく元画像に近い画像を復元せよ。

- 下記の通り、入力ファイル名(input.png)、出力ファイル名(output.png)はコマンドライン引数より取得すること

```
> python exer04.py input.png output.png
```

- 提出プログラムにより復元された画像と元画像とのPSNRが閾値 $T = 31.0$ 以上のものを正解とする

tmフォルダ内にて、下記の元画像と劣化画像を配布します。劣化画像のみを入力として、元画像に近い画像を復元してください。解法はいろいろあります。自分で考えて画像を復元してみてください。

※採点は違う画像で行います。

※閾値 $T=31.0$ は多少きつい可能性があるので、提出されるコードを見てから緩める方向に調整する可能性があります。

※PSNRの計算にはカラー画像を利用します。



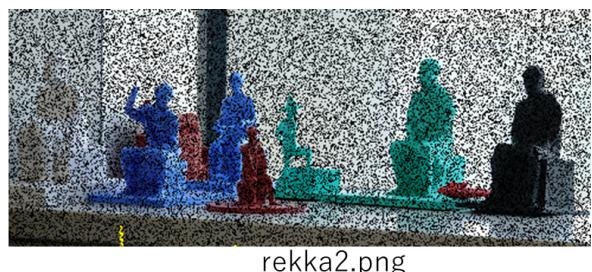
orig1.png



rekka1.png



orig2.png



rekka2.png

05 Template Matching 1

ターゲット画像とテンプレート画像を読み込みTemplate Matchingを計算せよ。

- 入力ファイル名(target.png、template.png)と、出力ファイル名(output.png)はコマンドライン引数より取得せよ

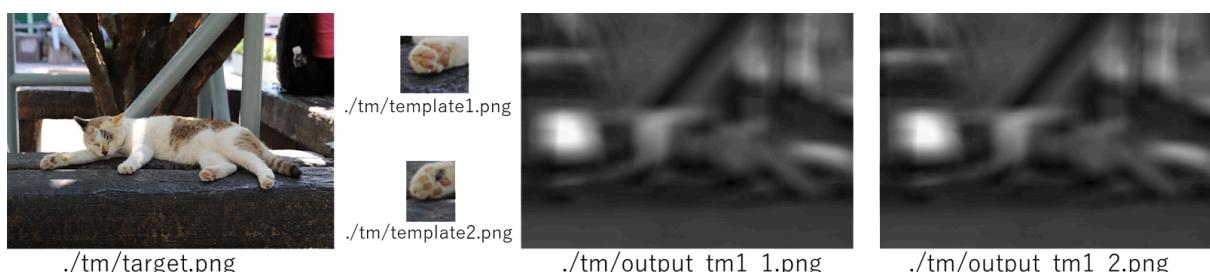
```
> python exer05.py target.png template.png output.png
```

- ターゲット・テンプレート画像はグレースケール画像に変換してから計算すること
- 計算結果は各画素にSSD値を格納した画像として出力すること
 - 出力画像の画素 (i, j) には、ターゲット画像の窓領域 $[i:i+h, j:j+w]$ とテンプレート画像とのSSD値を記録すること
 - テンプレート画像を重ねられる領域を考慮し、ターゲット画像が $H \times W$ 、テンプレート画像が $h \times w$ なら、出力画像サイズは $(H-h+1) \times (W-w+1)$ とせよ
- 出力画像は値域[0,255]の範囲へ正規化せよ（SSDの最大値で割り、255を掛ける）

ここではOpenCVの関数（matchTemplate()など）は利用せず、独自に実装すること

入出力例

```
> python exer05.py tm/target.png tm/template1.png output_tm1_1.png  
> python exer05.py tm/target.png tm/template2.png output_tm1_2.png
```



tmフォルダに入出力例となる画像があります。

`target.png`と`template1.png`を利用してtemplate matchingをした結果→`output_tm1_1.png`

`target.png`と`template2.png`を利用してtemplate matchingをした結果→`output_tm1_2.png`

06 Template Matching 2

ターゲット画像とテンプレート画像を読み込み、Template Matchingにより最もテンプレートと適合する領域を発見し矩形を描くプログラムを作成せよ。

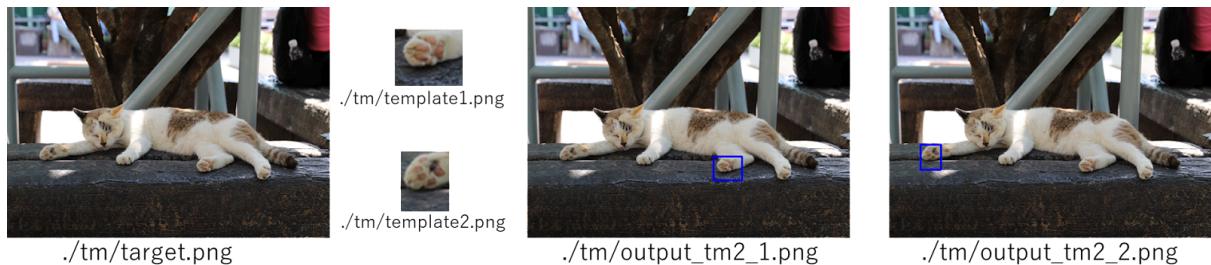
- 入力ファイル名(target.png、template.png)と、出力ファイル名(output.png)はコマンドライン引数より取得せよ

```
> python exer06.py target.png template.png output.png
```

- ターゲット・テンプレート画像はグレースケール画像に変換してから計算すること
- 出力はカラー画像とし、ターゲット画像中の最も適合する領域にテンプレートと同じサイズの四角形（線幅2、線の色(B=255, G=0, R=0)）を描画し出力すること
- 四角形描画には『cv2.rectangle (img, (x1,y1), (x2,y2),(b,g,r), line_width)』を利用せよ
※ OpenCVのmatchTemplate() と minMaxLoc() は利用せず独自に実装すること

入出力例

```
> python exer06.py tm/target.png tm/template1.png output_tm2_1.png  
> python exer06.py tm/target.png tm/template2.png output_tm2_2.png
```



tmフォルダに入出力例となる画像があります。

target.pngからtemplate1.pngを探索した例→ output_tm2_1.png

target.pngからtemplate2.pngを探索した例→ output_tm2_2.png

07 Template Matching 3

ターゲット画像とテンプレート画像を読み込み、Template Matchingにより最もテンプレートと適合する領域を発見しその部分に矩形を描画せよ。

- 入力ファイル名(target.png、template.png)と、出力ファイル名(fname_out.png)はコマンドライン引数より取得せよ

```
> python exer07.py target.png template.png fname_out.png
```

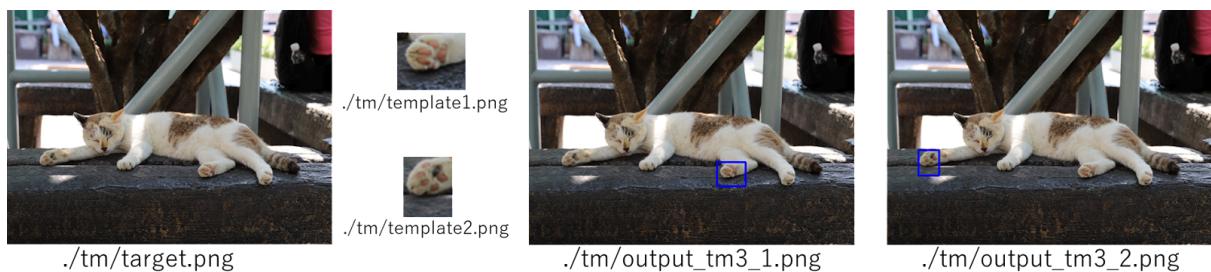
- 入出力の仕様はすべて前課題と同じ
- OpenCVのmatchTemplate() や minMaxLoc() を利用すること

※ 出題意図はweb上で必要な関数の利用方法を調べられるようになることです

※ 前課題と実行速度などを比較してみてください

入出力例

```
> python exer07.py tm/target.png tm/template1.png output_tm2_1.png  
> python exer07.py tm/target.png tm/template2.png output_tm3_2.png
```



tmフォルダに入出力例となる画像があります。

target.pngからtemplate1.pngを探索した例→ output_tm3_1.png

target.pngからtemplate2.pngを探索した例→ output_tm3_2.png

08 Template Matching 4

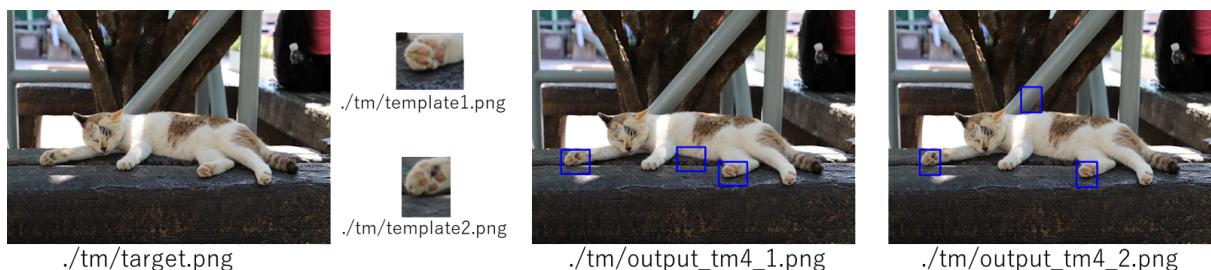
ターゲット画像とテンプレート画像を読み込み、Template Matchingによりテンプレートと最も似た領域『3か所』を発見しその部分に矩形を描画せよ。

- ふたつの入力ファイル名(target.png template.png)と、出力ファイル名(output.png)は、下記の通り、コマンドライン引数より取得すること

```
> python exer08.py target.png template.png output.png
```
- ターゲット・テンプレート画像はグレースケール画像に変換してから計算すること
- 出力はカラー画像とし、ターゲット画像中のテンプレート画像に最も似た3個の領域にテンプレートと同じサイズの四角形を描画し出力すること
- SSD値が一番小さい位置、2番目に小さい位置、3番目に小さい領域を見つけること
- 1番目の領域（四角形）と重ならないように2番目の領域を探索すること
- 1, 2番目に発見した領域と重ならないように、3番目の領域を探索すること
- 矩形描画には『cv2.rectangle (img, (x1,y1), (x2,y2),(r,g,b), line_width)』を利用し、線幅2, 線の色(255,0,0)とすること

入出力例

```
> python exer08.py tm/target.png tm/template1.png output_tm4_1.png  
> python exer08.py tm/target.png tm/template2.png output_tm4_2.png
```



tmフォルダに入出力例となる画像があります。

target.pngからtemplate1.pngを探索した例→ output_tm4_1.png

target.pngからtemplate2.pngを探索した例→ output_tm4_2.png

09 Harrisのコーナー検出 1

画像と画素位置(x,y)を読み込み、その画素を中心とするサイズ5x5の窓領域における勾配を計算し出力せよ。

- 下記の通り、入力ファイル名(img_in.png)、画素位置(x_in y_in)、出力ファイル名(output.txt)はコマンドライン引数より取得すること

```
python exer09.py img_in.png x_in y_in output.txt
```
- 入力画像をグレースケール化してから計算すること
- 各画素における縦横方向微分には、右図のSobel filter（※重みを4で割ったもの）を利用せよ
- 計算した勾配をラスタスキヤン順（右図）にテキストファイルへ出力せよ
- 1行にひとつの勾配ベクトル『x成分 y成分』を書き、x成分とy成分の間には半角スペースを配置せよ
※出力の詳細は雛形および解答例を確認すること

-1/4	-2/4	-1/4	-1/4	0	1/4
0	0	0	-2/4	0	2/4
1/4	2/4	1/4	-1/4	0	1/4

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24

※「12」を注目画素として、その周囲領域0~24における勾配ベクトルを出力する

入出力例

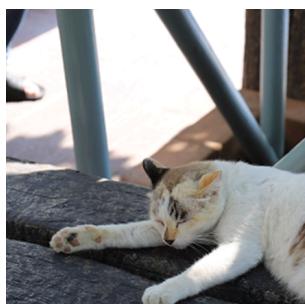
```
> python exer09.py det/img_cat.png 216 75 output_hr1_1.txt  
> python exer09.py det/img_cat.png 85 35 output_hr1_2.txt
```

./det/img_cat.pngについて、

画素(216, 75)付近の勾配がoutput_hr1_1.txtに、

画素(85, 35)付近の勾配がoutput_hr1_2.txtにあります。

(216, 75)はコーナー付近の画素、(85, 35)はエッジ付近の画素です。



./det/img_cat.png

10 Harrisのコーナー検出 2

画像と画素位置(x,y)を読み込み、その画素位置におけるHarris行列を計算し出力せよ。

- 下記の通り、入力ファイル名(img_in.png)、画素位置(x_in y_in)、出力ファイル名(output.txt)はコマンドライン引数より取得すること

```
python exer10.py img_in.png x_in y_in output.txt
```
- 入力画像はグレースケール化してから計算すること
- 計算には、右上のsobel filter (重み*1/4) と右下のGaussian重みを利用せよ
- 計算したHarris 行列 (2x2行列)は、テキストファイルへ2行で出力すること (出力例参照)

※ OpenCVの関数 (cv2.cornerHarrisなど) は利用せず自作すること

※入力画像のふち付近では5x5の窓領域がはみ出し、Harris行列が計算できない。このようなふち画素は指定されないものと仮定してよい

-1/4	-2/4	-1/4	-1/4	0	1/4
0	0	0	-2/4	0	2/4
1/4	2/4	1/4	-1/4	0	1/4

$\frac{1}{256}$	$\frac{4}{256}$	$\frac{6}{256}$	$\frac{4}{256}$	$\frac{1}{256}$
$\frac{4}{256}$	$\frac{16}{256}$	$\frac{24}{256}$	$\frac{16}{256}$	$\frac{4}{256}$
$\frac{6}{256}$	$\frac{24}{256}$	$\frac{36}{256}$	$\frac{24}{256}$	$\frac{6}{256}$
$\frac{4}{256}$	$\frac{16}{256}$	$\frac{24}{256}$	$\frac{16}{256}$	$\frac{4}{256}$
$\frac{1}{256}$	$\frac{4}{256}$	$\frac{6}{256}$	$\frac{4}{256}$	$\frac{1}{256}$

Gaussian 重み

入出力例

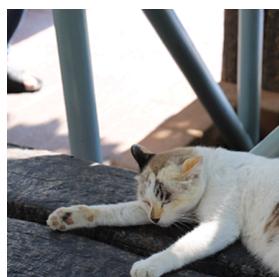
```
> python exer10.py det/img_cat.png 216 75 output_hr2_1.txt
> python exer10.py det/img_cat.png 85 35 output_hr2_2.txt
```

./det/img_cat.png について、

画素(216, 75)のHarris行列が output_hr2_1.txt に、

画素(85, 35)のHarris行列が output_hr2_2.txt にあります。

(216, 75)はコーナー付近の画素、(85, 35)はエッジ付近の画素です。



./det/img_cat.png

11 Harrisのコーナー検出 3

Harrisのコーナー検出法により入力画像からコーナーを検出し、コーナーに円を描画した画像を出力せよ。

- 下記の通り、入力ファイル名 (fname_in.png) と出力ファイル名 (fname_out.png) はコマンドライン引数より取得すること

```
python exer11.py fname_in.png fname_out.png
```

- Harris行列計算の仕様は前問の通り
- 評価式 R は、Harris行列の固有値 λ_1, λ_2 を用いて以下の通り計算せよ

$$R = \lambda_1 \lambda_2 - 0.15 * (\lambda_1 + \lambda_2)^2$$

- Opencvの関数(cv2.cornerHarris)は利用せず、行列計算・評価式Rの計算部分は自作すること
- $R \geq 260,000$ の画素をコーナーとして検出し、検出画素に円（半径3・色(255,0,0)・線幅1）を描画した画像を出力すること（雛形参照）

入出力例

```
> python exer11.py det/img_cat.png output_hr3_cat.png  
> python exer11.py det/img_thai.png output_hr3_thai.png
```



./detフォルダ内に入出力例となる画像があります。

12 Canny Filter

画像を読み込みCanny Filterによりエッジ画像を生成し出力せよ。

- 下記の通り、入力・出力ファイル名はコマンドライン引数により取得すること

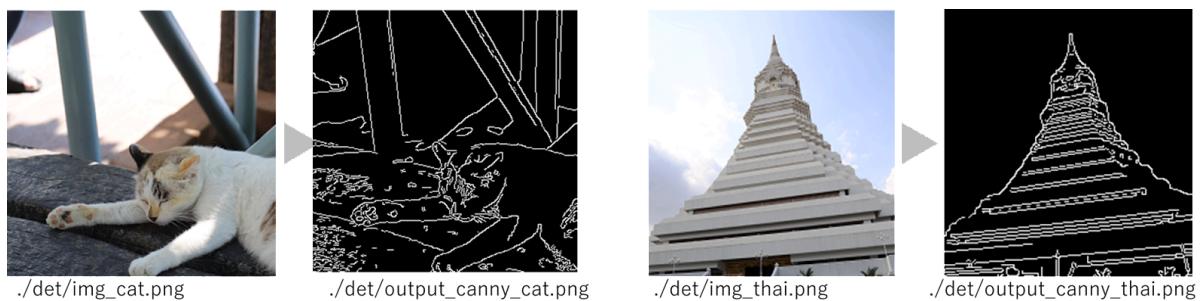
```
python exer12.py img_in.png img_out.png
```

- OpenCVの関数「cv2.Canny」を利用すること
- 二つの閾値 **TmaxとTminは、それぞれ、165と85**とすること
- 勾配の計算には 3x3 Sobel フィルタを利用すること：デフォルトのまま
- 勾配強度は L2gradient (L2ノルム)を利用すること：デフォルトではない

※ 関数の使い方を検索して使う練習をするのがこの問題の出題意図です

入出力例

```
> python exer12.py det/img_cat.png output_canny_cat.png  
> python exer12.py det/img_thai.png output_canny_thai.png
```



./detフォルダ内に入出力例となる画像があります。

13 Hough変換 1

画像を読み込み以下の手順でHough変換画像を計算せよ。

1. 下記の通り、入力ファイル名 (img_in.png) と出力ファイル名 (img_out.png) をコマンドライン引数により取得

```
python exer13.py img_in.png img_out.png
```

2. 入力画像をグレースケール画像化し、勾配強度画像を計算する（右図のsobel filterにより勾配強度を計算した後、最大値で全体を除算し[0,1]に正規化する）

-1/4	-2/4	-1/4	-1/4	0	1/4
0	0	0	-2/4	0	2/4
1/4	2/4	1/4	-1/4	0	1/4

Sobel filter

3. 勾配強度画像を閾値により二値化（**値0.35以上を前景に**）
4. 全ての前景画素(y, x)について、Hough変換画像へ投票する
 - θ の値域は[0,360]とする（1画素の幅が1度に対応）
 - ρ の値域は[0,A]とする（1画素の幅が1画素分に対応、Aは画像の対角方向の長さ）
 - 投票方法は講義中に解説したものを利用すること
5. Hough変換画像を、画像内の最大値で除算して255をかけることで[0,255]に正規化する

入出力例

```
> python exer13.py det/img_cat.png output_ht1_cat.png  
> python exer13.py det/img_thai.png output_ht1_thai.png
```



./detフォルダ内に入出力例となる画像があります。

14 Hough変換 2

入力画像を読み込み、前課題で作成したHough変換画像を利用して直線を検出し、検出した直線を入力画像に描画せよ。

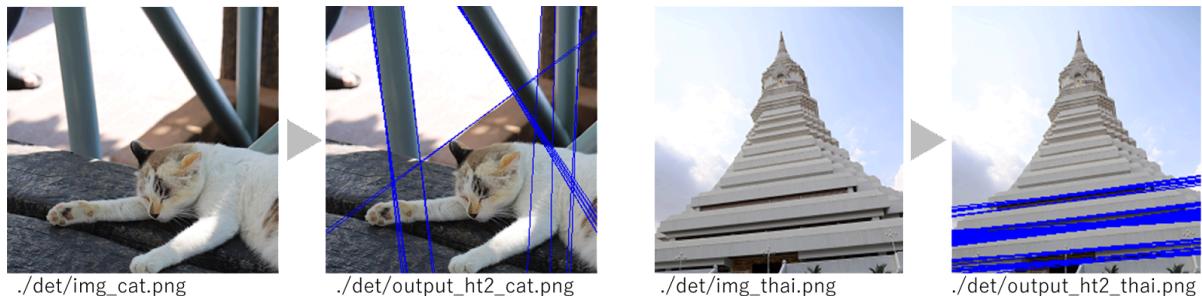
- 下記の通り、入力ファイル名（img_in.png）と出力ファイル名（img_out.png）はコマンドライン引数より取得すること

```
python exer14.py img_in.png img_out.png
```

- 前の課題の手順でHough変換画像を作成し、Hough変換画像は正規化せず、投票数が78以上の(ρ, θ)の組に対する直線を描くこと
- 出力はカラー画像とし、直線は『色(B=255, G=0, R=0), 線幅1』とする
※ cv2.HoughLines(), cv2.HoughLinesP()を利用せず、Hough変換画像の生成や直線の検出部分は自作すること

入出力例

```
> python exer14.py det/img_cat.png output_ht2_cat.png  
> python exer14.py det/img_thai.png output_ht2_thai.png
```



./detフォルダ内に入出力例となる画像があります。

15 Seam Carving (発展)

※ 余裕のある方向けの発展課題

※ 解けたら、講義中にTAか教員に確認をうけること

Seam Carvingを行うプログラムを実装せよ

- 下記の通り、入力画像名をコマンドライン引数として取得すること

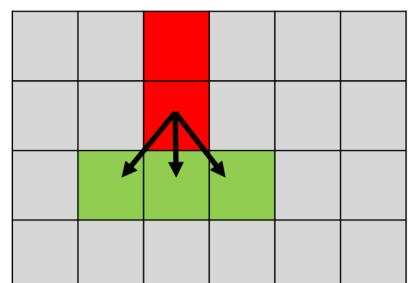
```
python exer15.py img.png
```

- Seam Carvingアルゴリズムについては『Shamir, Ariel (July 2007). "Seam carving for content-aware image resizing | ACM SIGGRAPH 2007". Siggraph 2007』を参照
- Seam Carving関数を除いたコードを雛形として配布するので利用すること
- aキーを押すと、Seam Carvingを行い画像を横方向に1画素分縮小する
- bキーを押すと、現在の画像が[out.png]という名前で保存される
- 削除する画素の重要度 $e(x,y)$ は、論文中の式(1) $\left| \frac{\partial I(x,y)}{\partial x} \right| + \left| \frac{\partial I(x,y)}{\partial y} \right|$ を利用すること (Iは輝度値画像)
- skimage.transform.seam_curveなどの外部関数は利用せず、seam探索については自作すること

Seam carvingアルゴリズムの解説

- 例としてサイズ 6×4 の小さな画像を、seam carvingにより横方向に1画素分小さくする問題を考える
- 画素 (x,y) には、重要度 $e(x,y)$ が定義されている
- 検索したいシームが満たす条件は。。。
 - 上端の画素と下端の画素をつなぐ
 - ある画素からひと画素分下に移動するとき、左隣・真下・右隣の3通りに移動できる
 - 上記2条件を満たすもののうち、重要度の総和が最小となる
- このようなシームは動的計画法により高速に計算可能
 - Step1：ラスタ順に各画素をたどり、『上端から着目画素までたどる際の重要度の総和』と『着目画素の上段の画素の位置』を記録する
 - Step2：下端の画素のうち最も重要度の総和が小さいものを発見し、『着目画素の上段の画素の位置』の情報をを利用して上方向にたどりシームとする

		($x^0, 0$)			
		($x^1, 1$)			
			($x^2, 2$)		
				($x^3, 3$)	



※ 詳細は第1回の講義資料を参考にしてください。

16 Histogram

画像を読み込み、グレースケール化した後、そのヒストグラムを出力せよ。

- 下記の通り、入力ファイル名と出力ファイル名はコマンドライン引数より取得せよ

```
python exer16.py input.png output.txt
```

- グレースケール画像の階調数は256 [0,255]とすること
- 計算結果は、テキストデータとして出力すること
- 出力データの各行に、グレースケール値と画素数を記入すること

※グレースケール値と画素数の間に半角スペースを配置すること

※出力したヒストグラムをエクセルなどで可視化してみてください(提出不要)

—入出力例————

```
> python exer16.py seg/kang.png output_histo_kang.txt  
> python exer16.py seg/cat.png output_histo_cat.txt  
> python exer16.py seg/late.png output_histo_late.txt
```



./seg/kang.png



./seg/cat.png



./seg/late.png

3件の画像ファイルとそのHistogramがsegフォルダ内にあります。

17 Otsu Method

画像を読み込み、グレースケール画像に変換後、Otsu法により画像を二値化せよ。

- 下記の通り、入力ファイル名と出力ファイル名はコマンドライン引数より取得せよ

```
python exer14.py input.png output.png
```

- グレースケール画像の階調数は256 [0,255]とし、Otsu法適用のためのヒストグラムは、前課題のものを利用せよ
- 出力は二値化画像とし、前景画素は255、背景画素は0とすること

入出力例

```
> python exer17.py seg/kang.png output_otsu_kang.png  
> python exer17.py seg/cat.png output_otsu_cat.png  
> python exer17.py seg/late.png output_otsu_late.png
```

3件の画像ファイルとそれらの二値化した結果がsegフォルダ内にあります。



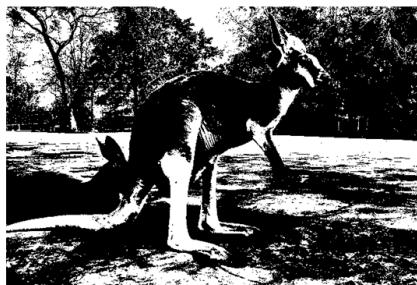
./seg/kang.png



./seg/cat.png



./seg/late.png



./seg/output_otsu_kang.png



./seg/output_otsu_cat.png



./seg/output_otsu_late.png

18 Region Growing

画像・シード画素位置・閾値を読み込み、画像をグレースケール化後、シードより領域成長を行い画像を二値化せよ

- 下記の通り、入力画像 (img.png) 、シード位置 (seed_x seed_y) 、閾値 (t) 、出力ファイル名 (output.png) はコマンドライン引数より取得すること

```
python exer18.py img.png seed_x seed_y t output.png
```

- グレースケール画像の階調数は[0,255]とする (雛形参照)
- 出力は二値化画像とし、前景は白(255), 背景は黒(0)とすること
- 領域成長時について
 - ある画素について、その上下左右4個の画素を隣接しているとみなすこと
 - 成長中の領域に隣接する画素のうち、**閾値 t以上** のものを加えること

※ここで行う領域成長法は、講義中に説明した領域成長法（二値化）なので注意してください

入出力例

```
> python exer18.py seg/late.png 180 180 140 output_rg_late.png  
> python exer18.py seg/cat.png 200 180 120 output_rg_cat.png
```

上記2件のコマンドにより、2件の画像ファイルに対してRegion Growingを適用した結果がsegフォルダ内にあります。



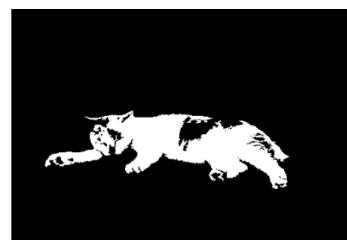
./seg/late.png



./seg/output_rg_late.png



./seg/cat.png



./seg/output_rg_cat.png

19 領域を数える

画像内の種の個数を数え標準出力に書き出すプログラムを作成せよ。

- 下記の通り、入力画像はコマンドライン引数より取得すること

```
python exer19.py img.png
```

- 入力画像について
 - サンプル画像(seeds1.jpg,seeds2.jpg)を配布する
 - 入力画像は、サンプル画像とほぼ同じ角度・証明条件で撮影される
 - 種どうしがなるべくバラバラになるよう撮影するが、種同士の多少の接触は残る可能性がある
- 画像内の種の数を標準出力に出力すること（標準出力には種の数以外は出力しない）
- 提出されたコードをテスト画像（非公開）に対して適用し、そのエラーが3%以内であれば正解とする（100個の種を含む画像に対して 98 ~ 102を出力すれば正解）

入出力例

```
python exer19.py seg/seeds1.jpg
```

37

```
python exer19.py seg/seeds2.jpg
```

122

segフォルダ内にテスト用の種画像（seeds1.pngとseeds2.png）があります。



※小松菜の種です。

準備：MNISTデータの読み込み方法

MNISTデータ

- パターン認識の勉強によく利用される手書き数字画像のデータセット
- 画像サイズ 28x28 (数字は画像の中心に配置される)
- データ数：トレーニング用60000文字、データとテスト用10000文字
- 形式：データは独自のバイナリ形式
- 本家のURL：<http://yann.lecun.com/exdb/mnist/>

MNISTデータの読み込み方法

1. <http://yann.lecun.com/exdb/mnist/> からデータをダウンロード (アクセスできない場合、別の方法を授業中に指示します。)

```
train-images-idx3-ubyte.gz : 60000個のTraining data (画像)
train-labels-idx1-ubyte.gz : 60000個のTraining data (ラベル)
t10k-images-idx3-ubyte.gz : 10000個のTest data (画像)
t10k-labels-idx1-ubyte.gz : 10000個のTest data (ラベル )
```

2. 画像データの読み込み - バイナリ形式で読みこみ、行列の形に整形する

```
def open_mnist_image(fname) :
    f = gzip.open(fname, 'rb')
    data = np.frombuffer( f.read(), np.uint8, offset=16)
    f.close()
    return data.reshape((-1, 784)) # (n, 784)の行列に整形, nは自動で決定
```

3. ラベルデータの読み込み - バイナリ形式ですべて読んで、1次元配列の形に整形する

```
def open_mnist_label(fname):
    f = gzip.open(fname, 'rb')
    data = np.frombuffer( f.read(), np.uint8, offset=8 )
    f.close()
    return data.flatten() # (n, 1)の行列に整形, nは自動で決定
```

sklearnの準備

下記のコマンドを打ち込みskleanrをインストールしてください。

```
> python -m pip install scikit-learn
```

20 パターン認識1 MNIST

MNISTのトレーニングデータを読み n番目の画像とラベルを出力せよ。

- 下記の通り、整数値nはコマンドライン引数より取得すること

```
python exer20.py n
```

- ソースファイル（exer20.py）があるフォルダのひとつ上のフォルダに『mnist』という名前のフォルダを作成し、MNISTデータはそこから読むこと（パスはプログラム内にハードコードすること、雛形参照）

path: ../mnist/train-images-idx3-ubyte.gz

path: ../mnist/train-labels-idx1-ubyte.gz

- n番目の画像をpng画像として出力し、ファイル名は『n_[label値].png』とすること
- 例、n=20の画像のラベル値が4なら、ファイル名は『20_4.png』となる
- MNISTデータにアクセスできない場合ダウンロード方法を別途指示します。

入出力例

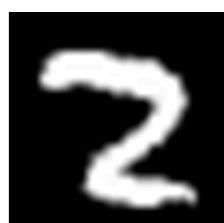
```
> python exer20.py 25  
> python exer20.py 28  
> python exer20.py 32
```

上記のように、25番目、28番目、32番目の訓練データを指定すると、下記のような画像が保存されます。ファイル名にも注意してください。

上記コマンドによる出力ファイル例が ./pr フォルダにあります。



25_2.png



28_2.png



32_6.png

21 パターン認識2 kNN

『MNISTのトレーニングデータ』と『ラベル値の不明な画像3枚』を読み込み、kNNによりラベルの値を推定せよ。

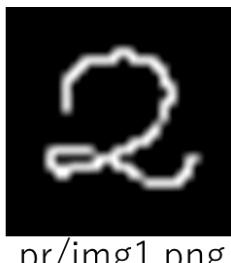
- 下記の通り、kNNのkの値（k）、3枚の入力画像名（img[1/2/3].png）はコマンドライン引数により取得すること

```
python exer21.py k img1.png img2.png img3.png
```

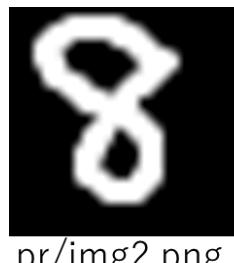
- MNISTの読み込みには前課題のものを利用すること（ソースファイルがあるフォルダのひとつ上のフォルダに『mnist』フォルダを作成し、データはそこから読む）
- 入力画像は、28x28のグレースケール画像（背景黒、文字が白）とする
- 推定結果となる3つの数値は、**標準出力に半角スペースで区切って出力すること**
- kNNは sklearnの KNeighborsClassifierを利用すること（使い方は各自webを検索、又は，雛形に例を載せておくので参照のこと）
- 全データを使うと処理が重いので、下記の通り訓練データを5000個に縮小すること

```
knn.fit( x_train[0:5000], t_train[0:5000])  
#knnのところには皆さんが定義した変数名が入ります。
```

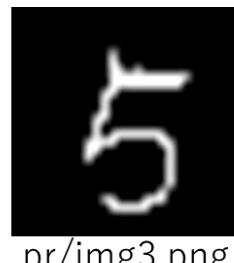
入出力例



pr/img1.png



pr/img2.png



pr/img3.png

./prフォルダに、3枚のテスト画像（下記）が配置しております。これらに対してmnistデータセットを利用してkNN（k=3）で推論をすると、『2 8 5』と正しく推論されます。

```
>python exer21.py 3 pr/img1.png pr/img2.png pr/img3.png  
2 8 5
```

22 パターン認識3 SVM

『MNISTのトレーニングデータ』と『ラベル値の不明な画像3枚』を読み込み、SVMによりラベルの値を推定せよ。

- 3枚の入力画像名 (img[1/2/3].png) はコマンドライン引数により取得すること

```
python exer22.py img1.png img2.png img3.png
```

- MNISTの読み込みには前課題のものを利用すること (ソースファイルがあるフォルダのひとつ上のフォルダに『mnist』フォルダを作成し、データはそこから読む)
- 入力画像は、28x28のグレースケール画像（背景黒、文字が白）とする
- 推定結果となる3つの数値は、**標準出力に半角スペースで区切って出力すること**
- SVMは sklearnのsvm.SVC()を利用すること（使い方はWebで検索を）
 - カーネルなどのパラメータはデフォルトのものを使うこと
 - パラメータの意味については、余裕があれば独自に調べてください
- 全データを使うと処理が重いので、下記の通り訓練データを5000個に縮小すること

```
svm.fit( x_train[0:5000], t_train[0:5000])  
#svmのところには皆さんが定義した変数名が入ります。
```

入出力例



pr/img1.png



pr/img2.png



pr/img3.png

./prフォルダに、3枚のテスト画像（下記）が配置してあります。これらに対してmnistデータセットを利用してkNN（k=3）で推論をすると、『2 8 5』と正しく推論されます。

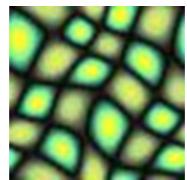
```
>python exer22.py 3 pr/img1.png pr/img2.png pr/img3.png  
2 8 5
```

23 Texture Synthesis (発展)

サンプルテクスチャを読み込み、その2倍の大きさのテクスチャを合成せよ

- 入力テクスチャ(sample.png)、パッチ半径(R)、出力ファイル名(output.png)はコマンドライン引数により取得せよ

```
python exer23.py sample.png R output.png
```

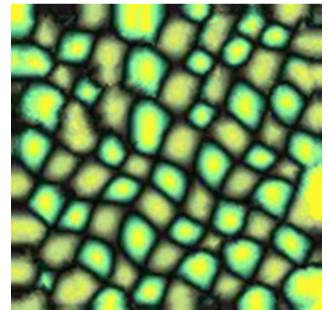


- 入力画像サイズ ($W \times H$) とすると、出力画像のサイズは $(2W \times 2H)$ とせよ

- 下記のテクスチャ合成アルゴリズムを利用せよ

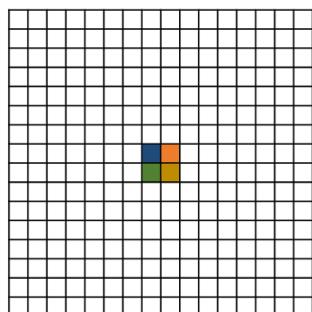
初期化

- サンプル画像 $S(i,j)$ に対し、サイズが 2 倍の出力画像 $I(i,j)$ を用意する(ゼロ初期化)
- 出力画像 $I(i, j)$ の中央 2×2 画素を乱数により初期化する
- 初期化した 2×2 画素に隣接する画素(8 画素)を FIFOキュー Q にプッシュする



合成処理

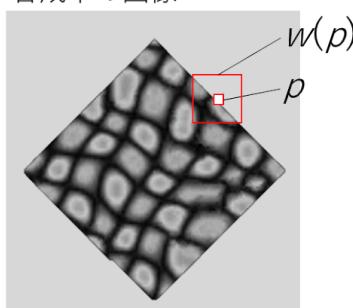
- Q が 空になるまで以下を繰り返す
 - Q から画素をpopしこれをpとする
 - pを中心とする $R \times R$ の矩形パッチに最も似たパッチをサンプルより検索
※下図参照 (検索、未合成部分は類似度計算に利用しない)
 - 発見したパッチの中心の画素をpにコピー
 - pの4近傍のうち、計算前かつQに入っていないものをQにプッシュ



1. 画像の初期化

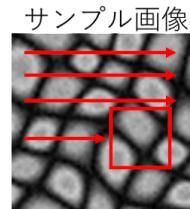
中央の 2×2 画素をランダムな色で初期化する

合成中の画像



2. パッチの探索

注目画素 p の周囲に $(2R+1) \times (2R+1)$ のパッチ $w(p)$ を作成
サンプル画像に対してラスタスキャン順にパッチを重ね合わせ、
もっとも似たパッチを検索する (SSDなどを利用する)
ただし、合成画像中の未合成画素は類似度計算には利用しない



入出力例

adv/sample1.png について $R=3$ として縦横2倍にした画像がadv/synthesis1.png

adv/sample2.png について $R=3$ として縦横2倍にした画像がadv/synthesis2.png

※この課題は恐らく長い処理時間を要するので実行に30sec以上かかるても良いです

※小さなsample2.pngでテストするのをお薦めします