デジタルメディア処理2

担当: 井尻 敬

プログラミング演習について

- 基礎課題は各*a* 点 (*18*問)
- 発展課題は各*b* 点 (*3*問)　※a=2.4, b=4 程度を想定
- すべてPython OpenCV環境を利用すること
- 提出締め切りは以下の通り
  - 基礎課題　1~12　：7**/09, 23:59**
  - 基礎課題13~18　：7**/23, 23:59**
  - 発展課題19~21　：7**/23, 授業中に確認**
- いくつかは比較的難易度が高いため適宜取捨選択を
- 発展課題については，自動採点が難しいので講義中に井尻のチェックを受けてください

提出方法: scombの課題よりファイルをそのまま提出．ファイル名は，各課題の雛形に従うこと．

課題雛形: http://takashiijiri.com/classes/dm2020_2/dm2exer.zip

入出力　: 課題では入力画像を受け取り，画像またはファイルを保存するプログラムを作る．入出力ファイル名は，以下の例のようにコマンドラインより与えるものとする．（※各課題の指定に従うこと）

```
$python   exer*.py   fname_in.png   fname_out.png
```

注意　：採点は自動化されています．フォルダ名・ファイル名やプログラムの仕様は指示に厳密に従ってください．入出力の仕様を満たさないコードは評価できず0点扱いとなります．

注意　：今回は計算速度を重視しませんが，64x64程度の画像に対して30秒以上の計算時間がかかるものは，自動採点の都合上0点とします．

理解の確認について

- この課題は，知人同士で相談しながら行ってよいです．
- 教える立場の方は教えることで理解が補強でき，教えられる方は比較的難しい課題も解けるようになり，メリットは大きいです
- ただし，教わる人は他人のコードをコピーするのではなく，どのような処理を行なうかを議論し，ソースコードは自身で作成してください

- 7/9の回より，コードレビューをしてもらいます．口頭で一人数分程度．
  - 一人一回必須（としたいですが，検討中です）

注意

- 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（GitHub，SNS，個人web page）
  - これを許してしまうと，発見し次第課題を差し替える事になり，数年後には難解な課題のみが残ってしまうので。。。
- 知恵袋やteratailなどのナリッジコミュニティサイトにて，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
  - 上記のような活動を井尻が発見した場合は，しかるべき処理をとります
  - 分からない部分がある場合は，“どこがどう分からないかを自分の言葉で明確に説明し”，他者から知識を受け取ってください
- 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こちらも理解しています．カンニングの疑いをかけられないようにと不安になったり、あえて”へんな書き方”をしなくてよいです．

演習の実施方法

- 演習時間中は zoom利用とslackを併用
  - 全員に共同ホスト権限を付与
    - 分室（ブレークアウトルーム）へ自由に行き来できるように
    - その他の共同ホスト権限機能は使わない
- 質問はslackへ
  - 簡単なものはTAが解答，ややこしいものはTA分室にて
  - 質問は講義時間内でなくてもOK, TAからの解答は基本的に講義中に
- 分室の利用を
  - 知人と好きな分室へ行く
  - メインの部屋でやる
- お願い
  - 正解コードを共有しないでください（mail/zoom画面共有など）
    - できていないコードを共有するのはOK
  - できている人同士なら例外的に分室で画面共有してもOK

アイスブレーク

- Slackのアイスブレークチャンネルに…
  - 文字列を投稿してみてください
  - それを編集・削除してみてください

  - 最近はまっている・面白かったものを1つ以上書いて下さい
    （映画・ゲーム・アニメ・youtuber・vtuber スポーツ）

- Zoomのブレークアウト機能を使って
  - Room1(踏み台)にはいって
  - そこから好きなroomにはいって
  - 出て来てください

  ※要共同ホスト権限

Template Matching

課題1.	平均画素値	very easy
課題2.	SSDの計算	easy
課題3.	NCCの計算	easy
課題4.	Template Matching (SSD)	normal
課題5.	Template Matching (NCC)	normal
課題6.	Template Matching で位置検索	normal
課題7.	Template Matching (cv2利用)	very easy

Detection

課題8.	勾配計算	easy
課題9.	Harris 行列の計算	normal
課題10.	Harrisのcorner検出	**hard**
課題11.	Hough変換	**hard**
課題12.	Hough変換による線描画	**hard**
課題13.	Canny filter	very easy

Segmentation

課題14.	ヒストグラム計算	easy
課題15.	Otsu法計算	normal

Detection

課題16.	mnistの出力	easy
課題17.	knnで文字認識	normal
課題18.	svmで文字認識	easy

Advanced

課題19.	Seed counting	**hard+**
課題20.	Seam Carving	**hard+**
課題21.	Texture synthesis	**hard+**

Template matching

課題1. 平均画素値の計算 - 雛形 exer1.py

1枚の画像を読み込みその平均値を出力せよ

- 読み込み後画像は輝度画像に変換すること（雛形参照）
- 計算した平均画素値は，テキストファイルを作成しそこへ記入すること

- 実行コマンドは以下の通り（コマンドライン引数の詳細は雛形を参照）

```
$python   exer1.py   img1.png   fname_out.txt
```

課題2. Sum of Square Differenceの計算 - 雛形 exer2.py

2枚の画像を読み込みそのSSD値を出力せよ

- 画像は輝度画像に変換すること
- 計算したSSD値は，テキストファイルを作成しそこへ記入すること

- 実行コマンドは以下の通り（コマンドライン引数の詳細は雛形を参照）

```
$python  exer2.py  img1.png  img2.png  fname_out.txt
```

課題3. Normalized Cross Correlation の計算 - 雛形 exer2.py

2枚の画像を読み込みその正規化相互相関(NCC)を出力せよ

- 画像は輝度画像に変換すること
- 計算したNCC値は，テキストファイルを作成しそこへ記入すること

- 実行コマンドは以下の通り（コマンドライン引数の詳細は雛形を参照）

```
$python  exer3.py  img1.png  img2.png  fname_out.txt
```

課題4. Template Matching I – 雛形 exer4.py

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

- 画像は輝度画像に変換すること
- 結果は各画素にSSD値を格納した画像として出力せよ
  - 出力画像は，ターゲット画像よりテンプレート画像分だけ小さいものとなる
  - ターゲットが *W*x*H*，テンプレートが *wxh*なら，出力画像は (*W-w+1*) x (*H-h+1*) となる（ターゲットにテンプレートを重ねられる領域）
  - 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を掛けること）

※ OpenCVの関数（matchTemplate()など）は利用せず，独自に実装すること

- 実行コマンドは以下の通り（引数の詳細は雛形を参照）

```
$python  exer4.py  target.png  template.png  fname_out.png
```

課題5. Template Matching II – 雛形 exer5.py

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

- 画像は輝度画像に変換すること
- 結果は各画素に**NCC(正規化相互相関)**値を格納した画像として出力せよ
  - 出力画像は，ターゲット画像よりテンプレート画像分だけ小さいものとなる
  - ターゲットが *W*x*H*，テンプレートが *wxh*なら，出力画像は (*W-w+1*) x (*H-h+1*) となる（ターゲットにテンプレートを重ねられる領域）
  - 出力画像は値域[0,255]の範囲へ正規化せよ（NCCの最大値で割り，255を掛けること）

※ OpenCVの関数（matchTemplate()など）は利用せず，独自に実装すること

- 実行コマンドは以下の通り（引数の詳細は雛形を参照）

```
$python  exer5.py  target.png  template.png  fname_out.png
```

課題6. Template Matchingによる位置検索 I – 雛形 exer6.py

ターゲット画像とテンプレート画像を読み込みTemplate Matchingを行い，ターゲット画像中のテンプレート画像と最も適合する位置に四角形を描画せよ

- 画像は輝度画像に変換すること
- Template matchingには，前課題のコードをそのまま用いてよい
- 四角形は，線幅2，線の色(255,0,0)で描画せよ
- テンプレートと同じサイズの四角形をターゲット画像中に描画すること
- OpenCVのmatchTemplate() と minMaxLoc() は利用しないこと
- 四角形描画には 『cv2.rectangle (img, (x1,y1), (x2,y2),(r,g,b), line_width)』を利用すること
- 出力はカラー画像

- 実行コマンドは以下の通り(引数の詳細は雛形を参照)

```
$python   exer6.py   target.png   template.png   fname_out.png
```

課題7. Template Matchingによる位置検索II– 雛形 exer7.py

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画像と最も適合する位置四角形を描画せよ

- 入力と出力の仕様は課題7と同様
- **OpenCVの関数（cv2.matchTemplate()など）を利用すること**
- この関数の利用方法は自身で検索すること
- この課題の意図は，web上で関数を検索できるようになることと，それを正しく利用できるようになることなので，Web上で発見したコードのコピペを提出してよい

ヒント：たくさんweb pageがあるので信用できそうなところを頼ること

ヒント：入力すべき画像の型に注意すること

- 実行コマンドは以下の通り(詳細はひな形を参照)

```
$python   exer7.py   target.png   template.png   fname_out.png
```

Template matching の実行例

入力・出力のサンプル画像を『tm』フォルダへ置きます
提出前のソースコードのテストに利用してください

- 課題1
  - img_cat.png の 平均画素値は，exer1.txt (値は 79.9159944351)
- 課題2
  - img1.png と img2.png の SSDは，exer2.txt (値は 27076704.0)
- 課題3
  - img1.png と img2.png の NCCは，exer3.txt (値は 0.900657858512)
- 課題4
  - target.pngとtemplate.pngのSSD画像は，exer4.png
- 課題5
  - target.pngとtemplate.pngのNCC画像は，exer5.png
- 課題6
  - target.pngとtemplate.pngのtemplate matching結果は，exer6.png
- 課題7
  - target.pngとtemplate.pngのtemplate matching結果は，exer7.png

上記はテスト用サンプルです．他の画像を用意するなどして色々と動かしてみてください．
利用するpython, opencvのバージョンにより結果に差が出るようです

課題8. Harris行列の準備 – 雛形 exer8.py

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

- 入力画像はグレースケール化すること
- 各画素のx方向/y方向微分にはsobel filterを利用すること
- 計算した勾配は，右図の順序（ラスタスキャン順）にテキストファイルへ出力すること

※ 出力の詳細は雛形や解答例を確認すること

※ 行列計算部分は，OpenCVの関数を利用せず，自作すること

※5x5の窓領域がはみ出すような入力画像のふち付近では，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

$\frac{1}{4}$

-1	-2	-1
0	0	0
1	2	1

$\frac{1}{4}$

1	0	1
2	0	2
1	0	1

Sobel filter

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

- 実行コマンドは以下の通り(詳細はひな形を参照)

```
$python  exer8.py  img_in.png x_in y_in  output.txt
```

課題9. Harris行列の計算 – 雛形 exer9.py

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

- 入力画像はグレースケール化すること
- 計算に用いるGaussian filterは，右図のものを利用すること
- 各画素のx方向/y方向微分にはsobel filterを利用すること
- 計算した行列は，テキストファイルへ出力すること

※ 課題9で利用するので全体のsobel filterを計算するようなコードを書いてもよい

※ 行列計算部分は，OpenCVの関数を利用せず，自作すること

※Gaussian filterがはみ出すような入力画像のふち付近では，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

$\frac{1}{4}$

-1	-2	-1
0	0	0
1	2	1

$\frac{1}{4}$

1	0	1
2	0	2
1	0	1

Sobel filter

$\frac{1}{256}$

1	4	6	4	1
4	16	24	16	4
6	24	36	24	6
4	16	24	16	4
1	4	6	4	1

Gaussian filter

- 実行コマンドは以下の通り(詳細はひな形を参照)

```
$python  exer9.py  img_in.png x_in y_in  output.txt
```

課題10. Harrisのコーナー検出 – 雛形 exer10.py

画像を読み込み，Harrisの手法により検出したコーナーに円を描画した画像を出力せよ

- Harris行列計算の仕様は，課題9と同様
- 画像のふち部分（2画素分）は計算しなくてよい
- 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定義する

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

- **R≧300,000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

※Opencvの関数(cv2.cornerHarris)は利用しないで，行列計算・評価式Rの計算部分は自作すること

※グレースケール化した画像には色付きの円を描けないので元画像に書いて出力すること

$$\frac{1}{4}\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad \frac{1}{4}\begin{bmatrix} 1 & 0 & 1 \\ 2 & 0 & 2 \\ 1 & 0 & 1 \end{bmatrix}$$

Sobel filter

$$\frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

Gaussian filter

```
$python  exer10.py  fname_in.png  fname_out.png
```

課題11. Hough変換 – 雛形 excer11.py

画像を読み込み，Hough変換画像を計算せよ（手順は以下の通り）

1. 入力画像をグレースケール画像化
2. グレースケール画像の勾配強度画像を計算
   勾配計算には右図のsobel filterを利用し
   最大値で全体を除算し[0,1]に正規化する
3. 勾配強度画像を閾値により二値化（**値0.4以上**を前景に）
4. 前景画素の位置を利用し，以下の通りHough変換画像へ投票
   Θ の値域は[0,360]で，1画素の幅が1度分に対応
   P の値域は[0,A]で，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）
   *Hough*変換画像には投票数（直線の本数）を登録し，最後に最大値を利用して[0,255]に正規化すること

$$\frac{1}{4}\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad \frac{1}{4}\begin{bmatrix} 1 & 0 & 1 \\ 2 & 0 & 2 \\ 1 & 0 & 1 \end{bmatrix}$$

Sobel filter

- 実行コマンドは以下の通り(詳細はひな形を参照)

```
$python  excer11.py  img_in.png  img_out.png
```

課題12. Hough変換 – 雛形 excer12.py

画像を読み込み，課題8のHough変換画像を利用して直線を検出し，
入力画像に直線を描画して出力せよ

1. 課題7の手順でHough変換画像を作成
2. Hough変換画像は正規化せず，投票数が**80以上**の(ρ, θ) の組に対する直線を描く
   直線は，色(255,0,0)，線幅1とする

直線を描画した画像を出力すること

※cv2.HoughLines(), cv2.HoughLinesP()を利用しないこと

※入力画像を一度グレースケール化してしまうと色付きの直線が描けないので出力時には元のカラー画像を利用すること

- 実行コマンドは以下の通り(詳細はひな形を参照)

```
$python   excer12.py   img_in.png   img_out.png
```

課題13. Canny Filterによるエッジ抽出 – 雛形 exer13.py

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

- OpenCVの関数(cv2.Canny)を利用すること
- 二つの閾値 **T_{max}とT_{min}は，それぞれ，180と90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

  ※ Pythonの関数では，特定の引数の値を直接指定できます

  Canny (arg1, arg2, arg3, L2gradient = True)

  引数L2gradient のデフォルト値はFalseなので，何もしないと L2gradientでなく簡易的なノルムが適用されます。

- 実行コマンドは以下の通り(詳細はひな形を参照)

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

Detectionの出力結果
提出前のソースコードのテストに利用してください

ファイルはすべて『det』フォルダにあります

課題8

python exer8.py det/img_cat.png 216 90 exer8a.txt の出力は exer8a.txt に
python exer8.py det/img_cat.png 85 50 exer8b.txt の出力は exer8b.txt に
※ (216,90)はコーナー付近，(85, 50)はエッジ付近


img_cat.png

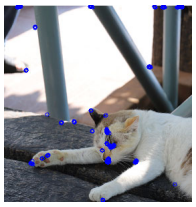

Img_tni.png

課題9

python exer9.py det/img_cat.png 216 90 exer9a.txt の出力は exer9a.txt に
python exer9.py det/img_cat.png 85 50 exer8b.txt の出力は exer9b.txt に
※ (216,90)はコーナー付近，(85, 50)はエッジ付近

課題10

python exer10.py det/img_cat.png det/exer10cat.png の出力は exer10cat.png に
python exer10.py det/img_thai.png det/exer10thai.png の出力は exer10thai.png に


exer10cat.png


exer10thai.png

Detectionの出力結果
提出前のソースコードのテストに利用してください

ファイルはすべて『det』フォルダにあります

課題11

python exer11.py det/img_cat.png exer11cat.png の出力は exer11cat.png に
python exer11.py det/img_thai.png exer11thai.txt の出力は exer11thai.png に


Img_cat.png

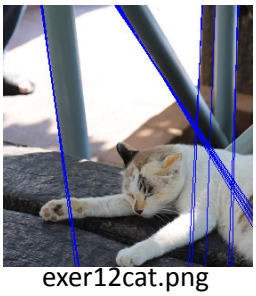

exer11cat.png

課題12

python exer12.py det/img_cat.png exer12cat.png の出力は exer12cat.png に
python exer12.py det/img_thai.png exer12thai.txt の出力は exer12thai.png に

課題13

python exer13.py det/img_cat.png exer13cat.png の出力は exer13cat.png に
python exer13.py det/img_thai.png exer13thai.txt の出力は exer13thai.png に


exer12cat.png


exer13cat.png

Segmentation

課題14. ヒストグラムの計算 – 雛形 exer14.py

画像を読み込み，グレースケール画像に変換後，そのヒストグラムを計算せよ

- グレースケール画像の色深度は256 [0,255]とすること
- 計算結果は，text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書く事
- 出力したヒストグラムをエクセルなどで可視化してみること(提出不要)

- 実行コマンドは以下の通り(詳細はひな形を参照)

```
$python  exer14.py   fname_in.png  output.txt
```

課題15. Otsu法の実装 – 雛形 exer15.py

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

- グレースケール画像の色深度は256 [0,255]とすること
- 出力は二値化画像とし，前景画素は(255,255,255)，背景画素は(0,0,0)とすること
- Otsu法適用のためのヒストグラムは，前課題のものを利用するとよい

- 実行コマンドは以下の通り(詳細はひな形を参照)

```
$python  exer15.py  target.png  template.png  fname_out.png
```

Segmentationの出力結果

提出前のソースコードのテストに利用してください

img.png

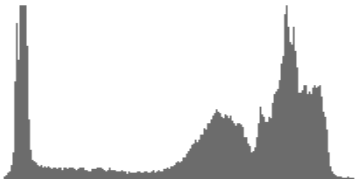
ファイルはすべて『segm』フォルダにあります

課題14

python exer14.py segm/img.png exer14img.txt の出力は exer14img.txtに

python exer14.py segm/img_cafe.png exer14cafe.txt の出力は exer14cafe.txtに

ヒストグラムの可視化結果は右図の通り



ヒストグラムの可視化結果

課題15

python exer15.py segm/img.png segm/exer15img.png の出力は **exer15img.png**に

python exer15.py segm/img_cafe.png segm/exer15cafe.png の出力は **exer15cafe.png**に

exer14img.png

Pattern recognition

準備：MNIST database とは

- 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
- URL: http://yann.lecun.com/exdb/mnist/

  - 数字は画像の中心に配置され，数字のサイズは正規化されている
  - 各画像のサイズは 28x28
  - データ数：トレーニング用：60000文字 / テスト用：10000文字
  - データは独自のバイナリ形式(pythonによる読み込みは簡単)

準備：PythonでMNISTを読む

1. http://yann.lecun.com/exdb/mnist/ からデータをダウンロード
   - 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 – バイナリ形式ですべて読んで，行列の形に整形

```
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

```
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
```

課題16. MNISTデータの読み込み – 雛形 exer16.py

MNISTのトレーニングデータを読み *n*番目のデータの画像とラベルを出力せよ

- プログラムファイル（exer16.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）
  - ../mnist/train-images-idx3-ubyte.gz
  - ../mnist/train-labels-idx1-ubyte.gz

  採点時に必要な仕様なので守ってください
- データの番号 *n* は，コマンドライン引数で与えること
- 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

- 実行コマンドは以下の通り(詳細はひな形を参照)

```
$python   exer16.py   n
```

課題17. kNNによる文字認識 – 雛形 exer17.py

MNISTのトレーニングデータを読み，さらにラベル値の不明な画像を3枚読み込み，kNNによりラベルの値を推定せよ

- プログラムファイル（exer17.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）
  - MNISTロード部分は同じなので、exer16.py を転用すること
- 入力画像は，28x28のグレースケール画像（背景黒、文字が白）とすること
- 推定対象の画像ファイル名 と kNNの近傍数 *k* はコマンドライン引数より入力すること
- 推定結果は，テキストファイルに書き出すこと（画像三枚分を3行に分けて書き出す）
- kNNは sklearnの KNeighborsClassifierを利用すること（使い方は各自webを検索，又は，ひな形に例を載せておくので参照のこと）
- 次次ページの指示に従って訓練データを5000個に縮小すること

- 実行コマンドは以下の通り(詳細はひな形を参照)

```
$python  exer17.py  k  img1.png  img2.png  img3.png  output.txt
```

課題18. SVMによる文字認識 – 雛形 exer18.py

MNISTのトレーニングデータを読み，さらにラベル値の不明な画像を3枚読み込み，SVMによりラベルの値を推定せよ

- プログラムファイル（exer18.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）
  - MNISTロード部分は同じなので、exer16.py を転用すること
- 入力画像は，28x28のグレースケール画像（背景黒、文字が白）とすること
- 推定対象の画像ファイル名 はコマンドライン引数より入力すること
- 推定結果は，テキストファイルに書き出すこと（画像三枚分を3行に分けて書き出す）
- 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用いてよい
  - 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
- 次ページの指示に従って訓練データを1500個に縮小すること

- 実行コマンドは以下の通り(詳細はひな形を参照)

```
$python  exer18.py  img1.png  img2.png  img3.png  output.txt
```

課題17, 18について

Mnistの訓練データ全てを利用するとPC室の計算機では非常に時間がかかります
そこで，学習部分を以下の通り書き直してください

knnは5000個のデータを利用しましょう

knn.fit(x_train[0:5000], t_train[0:5000])

Svmは1500個のデータを利用しましょう

svm.fit(x_train[0:1500], t_train[0:1500])

上記の　『knn.』『svm.』のところには皆さんが定義した変数が入ります.
また，svmでは学習データが足らずに識別には失敗しますが，ここでは気にしないでください.

Pattern recognitionの実行例


10_3.png


20_4.png


1000_0.png


img1.png


img2.png


img3.png

実行例に関するファイルはすべて『pr』フォルダにあります
提出前のソースコードのテストに利用してください

課題16

- n=10 を指定した場合 → 10_3.png が出力される
- n=20 を指定した場合 → 20_4.pngが出力される

課題17

- k=3, pr/img1.png pr/img2.png pr/img3.pngを指定
  → exer17.txtが出力される (全て正解する)

課題18

- pr/img1.png pr/img2.png pr/img3.png exer18.txt を指定

→ exer18.txtが出力され中身は「2,8,5」（scikit-learn が新しい場合）
→ exer18.txtが出力され中身は「7,7,7」（scikit-learn が古い場合）
※pythonのバージョンによっては不正解となる事があります（scikit-learn 0.23.1では正解します）
※訓練データを増やしたり，カーネルを変えたりしてみてください

発展課題

発展課題は基本課題が簡単すぎた人用です
多少実装に時間がかかると思うのですがこれでも簡単だったら申し訳ない。。

課題19. Seam Carving – 雛形 exer19.py

Seam Carvingを行うプログラムを実装せよ

- 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

※ Seam Carving関数を除いたコードを雛形にて配布するので参照のこと

- Seam Carvingアルゴリズムについては原著論文 [Avidan and Shamir, 2007] を参照

※ 削除する画素のエネルギーは，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，自作すること（numpyなどのライブラリは利用してOk）

- 実行コマンドは以下の通り(詳細はひな形を参照)

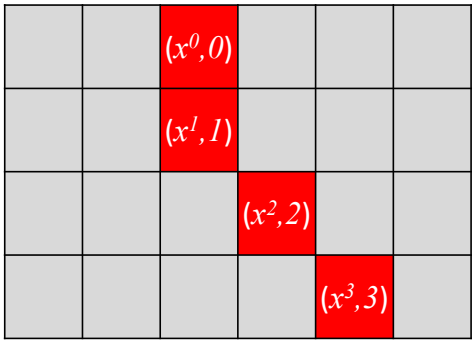
```
$python   exer19.py  img.png
```

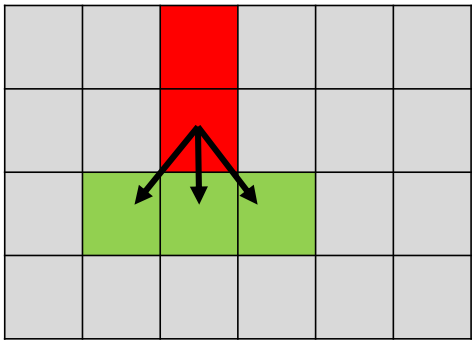
参考: *Avidan, Shai; Shamir, Ariel (July 2007).*
"Seam carving for content-aware image resizing | ACM SIGGRAPH 2007 papers". Siggraph 2007

Seam Carving algorithmの解説

例としてサイズ 6 x 4 の小さな画像をseam carving algorithm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 $e(x,y) = \left|\frac{\partial I(x,y)}{\partial x}\right| + \left|\frac{\partial I(x,y)}{\partial y}\right|$ が定義されている
- 画像の上端 のある画素$(x^0,0)$から開始し，下端のある画素までをつなぐシームを検索する
  - シーム上の画素位置は，$(x^0,0)$ - $(x^1,1)$ - $(x^2,2)$ - $(x^3,3)$ と表せる
  - シームにおいて，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
  - 画像の上端から下端をつなぐシームは多く存在するが，その中でもシームが通る画素上の重要度の総和が一番小さなものを出力する
    $$\underset{x^0,x^1,x^2,x^2}{\operatorname{argmin}}\ e(x^0,0) + e(x^1,1) + e(x^2,2) + e(x^3,3)$$
  - このようなシームは動的計画法により高速に計算可能
    - → Pythonで書くとあまり早くないかも
    - → 詳細は論文へ, or TAや井尻へ質問してください
- 発見したシームを削除し，画像の縮小が完了する





課題20. Seed Counting – 雛形 exer20.py

写真内の種の個数を数え，テキストファイルに書き出すプログラムを作成せよ

- 入力される画像はスイカの種であり，ほかの種類の種に対応する必要はない
- 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
- 入力画像は，常にサンプル画像のような角度で撮影される
- 照明条件や対象の大きさ（カメラからの距離）は若干変化する可能性がある
- サンプル画像(sample1.jpg, sample2.jpg)を配布する
- 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2%以内であれば合格とする（100個の種を含む画像に対して 98 ~ 102を出力）

※どのような外部ライブラリを利用してもよい

- 実行コマンドは以下の通り

```
$python   exer20.py    sample.jpg     out.txt
```

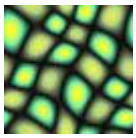
adv/sample1.jpg

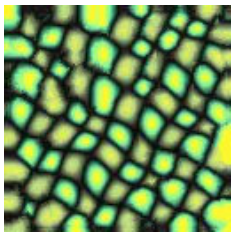
adv/sample2.jpg

課題21. Texture Synthesis – 雛形なし ファイル名はexer21.py

小さなテクスチャを読み込み，以下のアルゴリズムにより２倍の大きさのテクスチャを合成せよ

```
$python   exer21.py    sample.png   R   output.png
```





0. 入力と出力

入力: サンプル画像 (sample.png)，パッチ半径R, 出力画像ファイル名

出力: サンプルより高・幅が２倍大きな合成画像

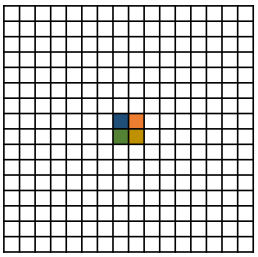
1. 初期化

- サンプル画像 S(i,j) に比べ高さと幅がそれぞれ２倍の出力画像 I(i.j) を生成する
- 出力画像 I(i, j)中央の2x2画素を乱数により初期化する
- 初期化した2x2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

2. 合成処理

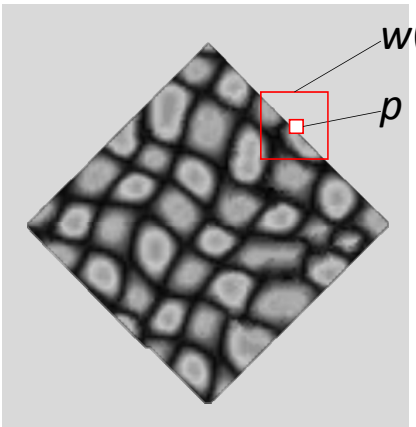
- Qが空になるまで以下を繰り返す
- Qから一つ画素をpopしこれをpとする
- pを中心とする RxRの矩形パッチに対して最も似たパッチをサンプルより検索（次ページ参照）
- 発見したパッチの中心の画素をpにコピー
- pの近傍のうち，計算前かつQに入っていないものをQにプッシュ

課題21. Texture Synthesis – 雛形なし ファイル名はexer21.py

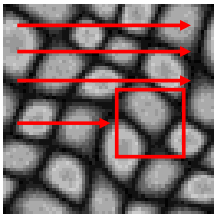


1. 画像の初期化

中央の2x2画素をランダムな色で初期化する



合成中の画像



合成中の画像

2. パッチの探索

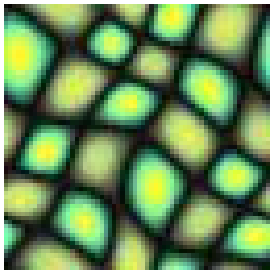
合成中画像の注目画素 p の周囲に(2R+1) x (2R+1)のパッチ$w(p)$を作成

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）

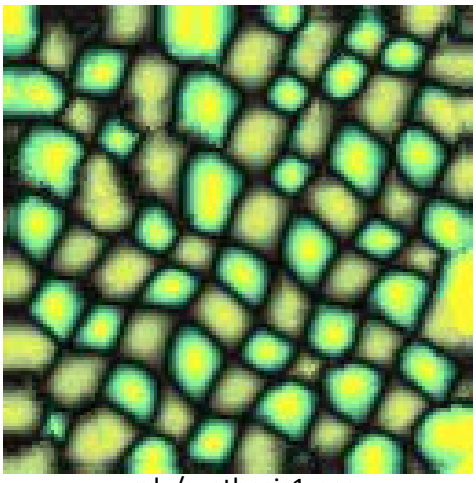
ただし，合成画像中の未合成画素は類似度計算には利用しない

課題21. Texture Synthesis – 雛形なし ファイル名はexer21.py

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以上かかっても良い
※小さなsample2.pngでテストするのをお薦めします



adv/sample1.png



adv/synthesis1.png

参考: *Efros and Thomas K. Leung, Texture Synthesis by Non-parametric Sampling, ICCV 99.*