コンピュータビジョン

第9回 ~ 第14回
プログラミング演習

担当: 井尻 敬

更新履歴

5/6 前半(1~12)の課題と雛形を作成、補足資料を作成、後半部分は作成中
5/7 提出方法を更新 (zip利用)、課題13~21を作成し公開、後半部分の雛形・実行例は準備中
5/20 問題15（領域拡張）にて、隣接画素は4画素とする旨追記。
5/20 問題13~21のテンプレートと回答例を作成&公開

締切

- 前半：課題01~課題12：締め切り 6/30 23:59 scombより提出
- 後半：課題13~課題19：締め切り 7/16 23:59 scombより提出
- 発展：課題20~課題21：締め切り 授業中，井尻 or TAに確認を受ける

提出方法:

1. 『**dm_学籍番号**』というフォルダを作成
2. その中にソースコードを書いたファイルを入れてzip圧縮
3. Scombの課題より提出．フォルダ名は全て半角．

フォルダ名の例: dm_AL190321

zipファイル名の例: dm_AL190321.zip

課題雛形: http://takashiijiri.com/classes/cv2021/

入出力　：各課題において詳細な仕様が定義されるので正しく従うこと

注意 ：

- 採点が自動化されているため，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コードは評価できないため0点扱いとなります．
- 採点は未配布のテストデータに対し正しい出力が出せるかどうかを確認します
- この課題では計算速度を重視しませんが，評価用入力データに対して20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．
- 各課題について，入出力例を用意するので，作成したプログラムのテストに利用してください．

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

- 7/01の回よりコードレビューをしてもらいます．
  - 提出したコードの処理内容を口頭で説明
  - 一人数分程度
  - 説明してくれた方は加点扱いとします（コードを人に説明する経験をしてほしいので一人一回必須としたいですが運用上なかなか難しいので．．）

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。。。

- Yahoo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※雛形ありのPython課題で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
    - 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ブレークアウトルームにて
  - 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
- ブレークアウトルームの利用を
  - 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**にブレークアウトルームで画面共有してもOK

Template matching

課題1 平均画素値の計算 *
課題2 Sum of squared difference *
課題3 Template Matching **
課題4 Template matchingによる領域探索 I ***
課題5 Template matchingによる領域探索 II ***
課題6 Template matchingによる領域探索 III ****

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グレースケール化後，その平均値と中央値を出力せよ

- 計算した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行目が中央値とすること
  - 標準出力には，計算結果以外を出力しないこと

- 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   img.png
```

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

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

- 画像は輝度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

- 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
```

課題3. Template Matching – 雛形 exer3.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

- ターゲット画像と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
- 結果は各画素にSSD値を格納した画像として出力せよ
  - テンプレートを重ね合わせられる領域を考慮し、ターゲットが *HxW*，テンプレートが *hxw*なら，出力画像サイズは (*H-h+1*) x (*W-w+1*) とせよ
  - 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]とテンプレートを重ね合わせた際のSSD値を記録すること
  - 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   exer3.py   target.png   template.png   fname_out.png
```

課題4. Template Matchingによる探索 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により最もテンプレートと適合する領域を発見せよ

- 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すること
- 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を描画し出力すること
  - 線幅2，線の色(B=255, G=0, R=0)とすること
  - 四角形描画には『関数：cv2.rectangle (img, (x1,y1), (x2,y2),(r,g,b), 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

- 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
```

※ python + OpenCVでは、色の指定はRGBではなくBGRの順なので注意

課題5. Template Matchingによる探索 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により最もテンプレートと適合する領域を発見せよ

- 入力と出力の仕様は課題4と同様
- **OpenCVの関数（cv2.matchTemplate()やcv2.minMaxLoc()など）を利用すること**

- ヒント
  - 出題意図はweb上で必要な関数の利用方法を検索できるようになること
  - Web上で発見したコードを（一部）コピペし提出してよい
  - この関数の利用方法は自身で検索すること
  - たくさん関連Web pageがあるので信用できそうなところを頼ること
  - 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(詳細は雛形を参照)

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

課題6. Template Matchingによる探索 III – 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により最もテンプレートと適合する領域を“3か所”発見せよ

- 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すること
- ターゲット画像中の最も適合する**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値が最も小さい位置，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番目に発見した領域（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（四角形）と重ならないように、3番目の領域を探索すること
  - 線幅2，線の色(255,0,0)とすること
  - 四角形描画には『関数：cv2.rectangle (img, (x1,y1), (x2,y2),(r,g,b), line_width)』 を利用せよ

- 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
```

※利用するOpenCVやPythonのバージョンの違いに寄り若干の誤差が出る可能性があります

課題1~6の入出力例

課題1~6の入出力例を「./tm」フォルダに入れて置くので自身のコードの実行結果の確認に利用してください

課題1:

> python exer1.py tm/img1.png

126.81983333333334

139.0

> python exer1.py tm/img2.png

136.5515

142.0

課題2:

> python exer1.py tm/img1.png tm/img2.png

27076984.0

課題3:

> python exer3.py tm/target.png tm/template.png tm/exer3output.png

とすると ./tm/exer3output.pngが出力される

target.png

template.png

tm/exer3output.png

課題1~6の入出力例

課題1~6の入出力例を「./tm」フォルダに入れて置くので自身のコードの実行結果の確認に利用してください

課題4:

> python exer4.py tm/target.png tm/template.png tm/exer4output.png

とすると ./tm/exer4output.png が出力される

exer4output.png

課題5:

> python exer5.py tm/target.png tm/template.png tm/exer5output.png

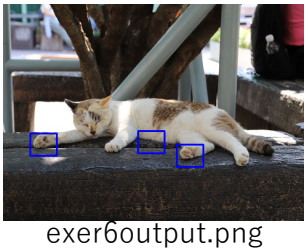
とすると ./tm/exer5output.png が出力される

exer5output.png

課題6:

> python exer6.py tm/target.png tm/template.png tm/exer6output.png

とすると ./tm/exer6output.png が出力される



exer6output.png

Detection

課題7　Harris行列の計算準備　**
課題8　Harris行列の計算　***
課題9　Harrisのコーナー検出　****
課題10　Canny Filter　*
課題11　Hough変換　***
課題12　Hough変換による線検出　****

課題7. Harris行列の準備 – 雛形 exer7.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し計算すること
- 各画素における縦横方向微分には右のSobel filterを利用すること
- 計算した勾配は，ラスタスキャン順（右図）にテキストファイルへ出力すること
- 1行にひとつのベクトルを書き，x成分とy成分の間には半角スペースを配置すること

※ 出力の詳細は雛形および解答例を確認すること

- 実行コマンドは以下の通り(詳細はひな形を参照)

```
> python  exer7.py  img_in.png  x_in  y_in  output.txt
```

$\frac{1}{4}$

-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

※「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

課題8. Harris行列の計算 – 雛形 exer8.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し計算すること
- 計算に用いるGaussianの重みは，右図のものを利用すること
- 各画素の縦横方向微分には右のsobel filterを利用すること
- 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
  - 詳細はj出力例を参考のこと

※ 行列計算部分は，OpenCVの関数を利用せず，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てしまい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

- 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
```

$\frac{1}{4}$

-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

課題9. Harrisのコーナー検出 – 雛形 exer9.py

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

- Harris行列計算の仕様は前問の通り
- 画像のふち部分（2画素分）は無視してよい
- 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≧280,000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)は利用せず，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

実行コマンドは以下の通り(詳細はひな形を参照)

```
> python   exer9.py   fname_in.png  fname_out.png
```

$\frac{1}{4}$

-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

課題10. Canny Filterによるエッジ抽出 – 雛形 exer10.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，170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

- ヒント：
  - この問題の出題意図は，ライブラリをうまく使うと非常に簡単に画像処理ができることを体験してもらうことです
  - Pythonの関数では，特定の引数の値を直接指定できます

    cv2.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   exer10.py   img_in.png   img_out.png
```

課題11. Hough変換 – 雛形 excer11.py

画像を読み込み以下の手順でHough変換画像を計算せよ

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 filteを利用する

   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

   ρ 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

$\frac{1}{4}$

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

$\frac{1}{4}$

1	0	1
2	0	2
1	0	1

Sobel filter

- 実行コマンドは以下の通り(詳細は雛形を参照)

```
> 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**75以上**の(ρ, θ) の組に対する直線を描く
   直線は，色(B=255, G=0, R=0)，線幅1とする
3. 直線を描画した画像を出力する

※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7~12の入出力例

課題7~12の入出力例を「./det」フォルダに入れて置くので自身のコードの実行結果の確認に利用してください


img_cat.png


img_thai.png

課題7:

> **python exer7.py det/img_cat.png 216 90 det/exer7a.txt** の出力は exer7a.txt に
> **python exer7.py det/img_cat.png 85 50 det/exer7b.txt** の出力は exer7b.txt に

※ (216,90)はコーナー付近，(85, 50)はエッジ付近

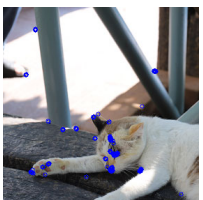
課題8:

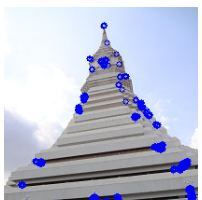
> **python exer8.py det/img_cat.png 216 90 det/exer8a.txt** の出力は exer8a.txt に
> **python exer8.py det/img_cat.png 85 50 det/exer8b.txt** の出力は exer8b.txt に

※ (216,90)はコーナー付近，(85, 50)はエッジ付近

課題 9 :

> **python exer9.py det/img_cat.png det/exer9cat.png** の出力は exer9cat.png に
> **Python exer9.py det/img_thai.png det/exer9thai.png** の出力は exer9cat.png に


exer9cat.png


exer9thai.png

課題7~12の入出力例

課題7~12の入出力例を「./det」フォルダに入れて置くので自身のコードの実行結果の確認に利用してください

課題10:

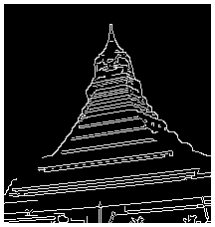
> **python exer10.py det/img_cat.png det/exer10cat.png** の出力は exer10cat.png に
> **python exer10.py det/img_thai.png det/exer10thai.png** の出力は exer10thai.png に

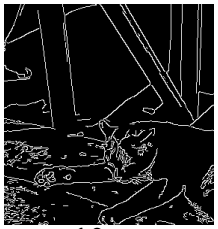
課題11:

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

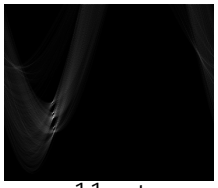
課題12:

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

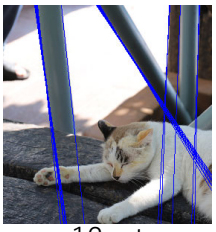

exer10thai.png


exer10cat.png


exer11thai.png


exer11cat.png


exer12thai.png


exer12cat.png

Segmentation

- 課題13 : ヒストグラムの計算 *
- 課題14 : Otsu法 ***
- 課題15 : Region Growing *****
- 課題16 : 領域の数え上げ *****

課題13. ヒストグラムの計算 – 雛形 exer14.py

画像を読み込み，グレースケール化し，そのヒストグラムを計算せよ

- 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  exer13.py  fname_in.png  output.txt
```

課題14. 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階調数は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  exer14.py  target.png  template.png  fname_out.png
```

課題15. 領域成長法 – 雛形 exer15.py

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

- 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
- 領域成長のシード画素（seed_x, seed_y）はコマンドライン引数より与えられる
- ある画素を成長させる際、その上下左右４個の画素を隣接しているとみなすこと
- 領域成長時、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
  - 閾値tはコマンドライン引数より与えられる

※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

- 実行コマンドは以下の通り(詳細はひな形を参照)

```
> python   exer15.py  img.png   seed_x   seed_y   t   fname_out.png
```

課題16. Seed Counting – 雛形 exer16.py

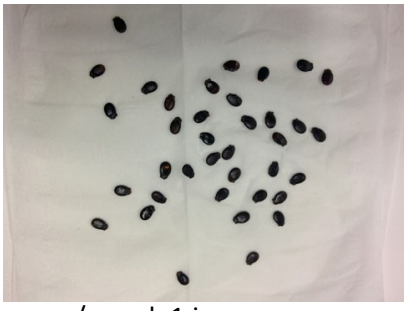
画像内の種の個数を数え標準出力に書き出すプログラムを作成せよ

- 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）
- 標準出力には種の数以外は出力しないこと

※ヒント: OpenCVやNumpyのライブラリを利用してください

- 実行コマンドは以下の通り

```
> python   exer16.py   img.jpg
```



seg/sample1.jpg

seg/sample2.jpg

課題13~15の入出力例

課題13~15の入出力例を「./seg」フォルダに入れて置くので自身のコードの実行結果の確認に利用してください


kang.png


cat.png


late.png

課題13:

> **python exer13.py seg/kang.png seg/exer13kang.txt** の出力は exer13kang.txt に
> **python exer13.py seg/late.png seg/exer13late.txt** の出力は exer13late.txt に
> **python exer13.py seg/cat.png seg/exer13cat.txt** の出力は exer13cat.txt に


exer14kang.png

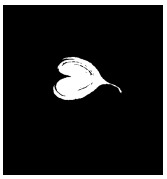

exer14cat.png


exer14late.png

課題14:

> **python exer14.py seg/kang.png seg/exer14kang.png** の出力は exer14kang.png に
> **python exer14.py seg/late.png seg/exer14late.png** の出力は exer14late.png に
> **python exer14.py seg/cat.png seg/exer14cat.png** の出力は exer14cat.png に


exer15cat.png


exer15late.png

課題15:

> **python exer15.py seg/late.png 180 180 140 seg/exer15lage.png** の出力は exer15late.png に
> **python exer15.py seg/cat.png 200 180 120 seg/exer15cat.png** の出力は exer15cat.png に

※cat.pngは領域拡張がはみ出さないように少しだけふちを書き込んでいます

課題16の入出力例

課題16の入力例を「./seg」フォルダに入れて置くので自身のコードの実行結果の確認に利用してください

課題16:

> **python exer16.py seg/seeds1.jpg**
> **python exer16.py seg/seeds2.jpg**

標準出力に種の数を出力して下さい

Pattern Recognition

- 課題17 : MNISTデータの読み込み　　**
- 課題18 : SVMによる文字認識　　***
- 課題19 : kNNによる文字認識　　***

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17. MNISTデータの読み込み – 雛形 exer17.py

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

- 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/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   exer17.py   n
```

課題18. kNNによる文字認識 – 雛形 exer18.py

『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

- 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読み込み部分は同じなので、exer17.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，雛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   exer18.py   k    img1.png  img2.png  img3.png  output.txt
```

課題19. SVMによる文字認識 – 雛形 exer19.py

『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

- プログラムファイル（exer19.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、exer17.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使ってください
  - 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   exer19.py   img1.png  img2.png  img3.png  output.txt
```

課題18, 19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

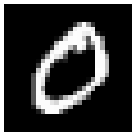
knnは5000個のデータを利用しましょう
knn.fit(x_train[0:5000], t_train[0:5000])

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])

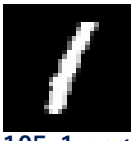
上記の　『knn.』『svm.』のところには皆さんが定義した変数が入ります.

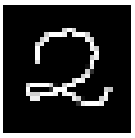
課題17~19の入出力例

課題17~19の入出力例を「./pr」フォルダに入れて置くので
自身のコードの実行結果の確認に利用してください

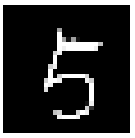

1_0.png


25_2.png


105_1.png


img1.png


img2.png


img3.png

課題17:

> **python exer17.py 1** の出力は pr/1_0.png に
> **python exer17.py 25** の出力は pr/25_2.png に
> **python exer17.py 105** の出力は pr/105_1.png に

課題18:

> **python exer18.py 3 seg/img1.png seg/img2.png seg/img3.png seg/exer18.txt**
の出力は exer18.txt に

課題19:

> **python exer19.py seg/img1.png seg/img2.png seg/img3.png seg/exer19.txt**
の出力は exer19.txt に

最新のsklearnを利用した場合、このデータでも推定に成功すると思います。
もしうまく行かない場合TAまたは私まで連絡を

Advanced

- 課題20 Texture Synthesis　　******
- 課題21 Seam Carving　　******

この課題を解いた方は、講義時間中に教員またはTAの確認を受けてください。

ここまでの課題が簡単すぎた方向けです
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20. Seam Carving – 雛形 exer20.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とし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

```
> python   exer20.py  img.png
```

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

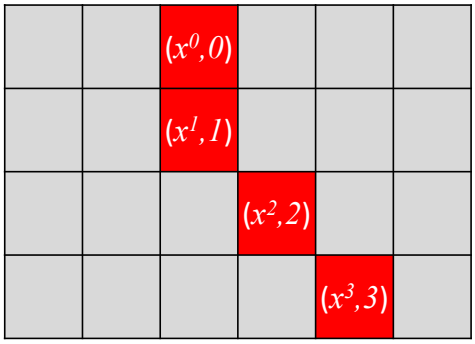
Seam Carving algorithmの解説

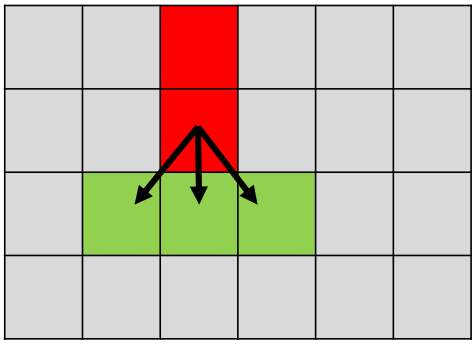
例としてサイズ 6 x 4 の小さな画像をseam carving algorithmにより，横方向に1画素分小さくする問題を考える

- 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

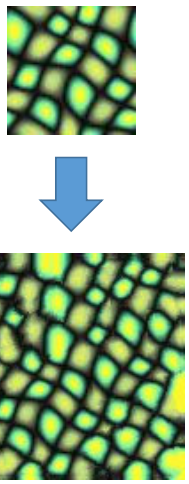




課題21. Texture Synthesis – 雛形なし ファイル名はexer21.py

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

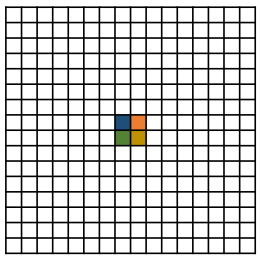
1. 初期化

- 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

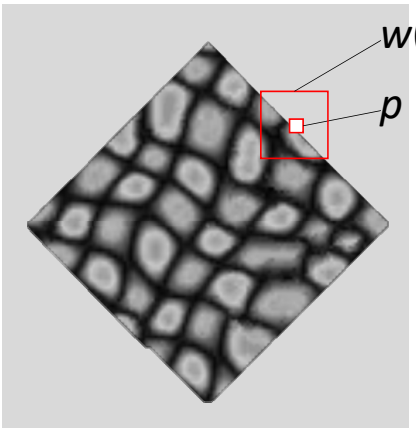
- Qが空になるまで以下を繰り返す
- Qから一つ画素をpopしこれをpとする
- pを中心とする RxRの矩形パッチに対して最も似たパッチをサンプルより検索（次ページ参照）
- 発見したパッチの中心の画素をpにコピー
- pの近傍のうち，計算前かつQに入っていないものをQにプッシュ

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

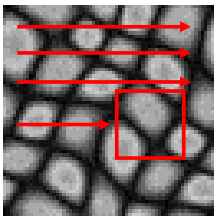


1. 画像の初期化

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

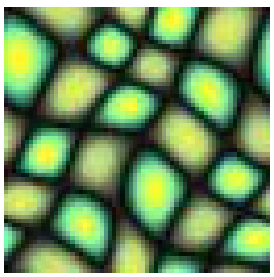
課題21. Texture Synthesis – 雛形なし ファイル名はexer21.py

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

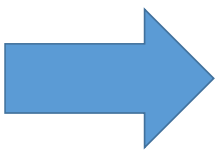
adv/sample2.png について R=3として縦横2倍にした画像がadv/synthesis2.png

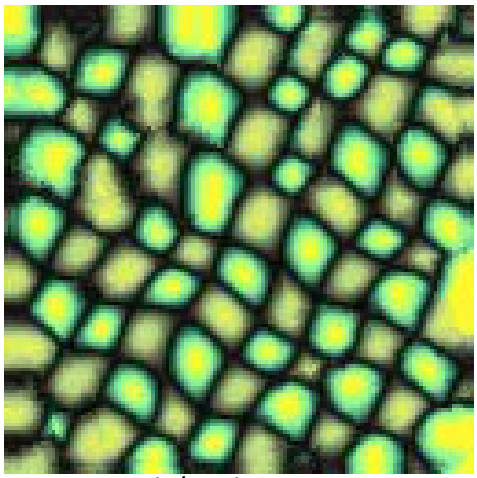
※この課題は恐らく長い処理時間を要するので実行に30sec以上かかっても良いです

※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.*