コンピュータビジョン

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

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xx：締め切り 7/16 23:59 scombより提出
- 発展：課題xx~課題xx：締め切り 授業中，井尻 or TAに確認を受ける

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
```

✓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．

✓この課題では計算速度を重視しません．ただし，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

- 7/01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※ひな形ありの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	very easy
課題2 Sum of squared difference	very easy
課題3 Template Matching	easy
課題4 Template matchingによる領域探索 1	normal
課題5 Template matchingによる領域探索 2	normal
課題6 Template matchingによる領域探索 3	normal+

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
```

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
```

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
```

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

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
```

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
```

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

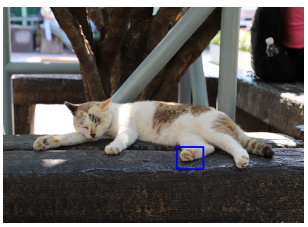
課題5:

> 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

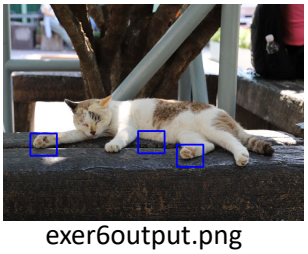
課題6:

> 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

exer4output.png



exer5output.png



exer6output.png

Detection

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

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

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

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

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
```

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
```

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:

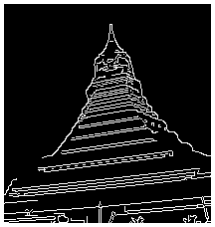
> python exer11.py det/img_cat.png det/exer11cat.txt の出力は exer11cat.png に

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

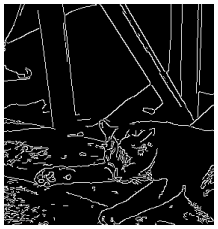
課題12:

> 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

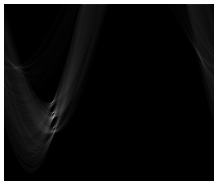


exer10thai.png



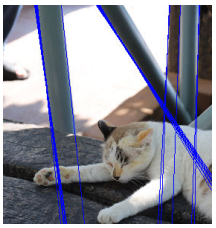
exer10cat.png

exer11thai.png



exer11cat.png

exer12thai.png



exer12cat.png