デジタルメディア処理1

担当: 井尻 敬

プログラミング演習について

- 基礎課題は各3点 (15問)
- 発展課題は各5点 (*n* 問)
- 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 1~5　: **7/11, 23:59**
  - 基礎課題 6~10 : **7/18, 23:59**
  - 基礎課題11~15: **7/26, 23:59**
  - 発展課題　　: **7/26, 23:59**

- いくつかは比較的難易度が高いため適宜取捨選択を

提出方法: 共有フォルダに 『**dm2(学籍番号)**』というフォルダを作成し，その中にソースコードの入ったファイルを置く．フォルダ名は全て半角.

フォルダ名の例: dm2AL150999

課題雛形: http://takashiijiri.com/classes/dm2018_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，教わる人はただ他人のコードをコピーするだけではなく，その部分にどのような意味があるかを確実に理解するようにしてください
- で，理解度を確認するために演習時間中に，ランダムで学生・課題をサンプリングし，井尻が口頭でコードの内容を聞きます。もし，提出したにも関わらず理解ができていないと判断される場合はその課題は不可とし，他の課題の内容を質問します.
- 少なくても一度は講義中に井尻の確認を受けるようにしてください．一度も口頭の確認が行えない場合は，課題の点数を減点します.

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，数年後には何回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

Template Matching 15

課題1.	SSDの計算	easy	3点
課題2.	Template Matching (SSD)	normal	3点
課題3.	Template Matching (NCC)	normal	3点
課題4.	Template Matching で位置検索	normal	3点
課題5.	Template Matching (cv2利用)	very easy	3点

Detection 15

課題6.	Harris 行列の計算	normal	3点
課題7.	Harrisのcorner検出	normal	3点
課題8.	Hough変換	normal	3点
課題9.	Hough変換による線描画	normal	3点
課題10.	Canny filter	very easy	3点

Segmentation 6

課題11.	ヒストグラム計算	easy	3点
課題12.	Otsu法計算	normal	3点

Detection 9

課題13.	mnistの出力	easy	3点
課題14.	knnで文字認識	normal	3点
課題15.	svmで文字認識	normal	3点

Advanced

課題16.	Seed counting	normal	5+a点
課題17.	Seam Carving	hard	5+a点

課題1. Sum of Square Differenceの計算 - 雛形 exer1.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  exer1.py  img1.png img2.png  fname_out.txt
```

課題2. Template Matching – 雛形 exer2.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  exer2.py  target.png  template.png  fname_out.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

- 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（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による位置検索 – 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，課題2や課題3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() は利用しないこと
- 四角形描画には 『cv2.rectangle ((x1,y1), (x2,y2),(r,g,b), line_width)』を利用すること

- 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による位置検索 – 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によりテンプレート画像と最も適合する位置を出力せよ

- 入力と出力の仕様は課題4と同様
- OpenCVの関数（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   exer5.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
  - ex1_1.png と ex1_2.png の SSDは，ssd12.txt (値は 4717546.0)
  - ex1_1.png と ex1_3.png の SSDは，ssd13.txt (値は 8626211.0)
- 課題2
  - target.pngとtemplate1.pngのSSD画像は，ex2_out1.png
  - target.pngとtemplate2.pngのSSD画像は，ex2_out2.png
- 課題3
  - target.pngとtemplate1.pngのCCS画像は，ex3_out1.png
  - target.pngとtemplate2.pngのCCS画像は，ex3_out2.png
- 課題4
  - target.pngとtemplate1.pngのtemplate matching結果は，ex4_1.png
  - target.pngとtemplate1.pngのtemplate matching結果は，ex4_2.png
- 課題5
  - target.pngとtemplate1.pngのtemplate matching結果は，ex5_1.png
  - target.pngとtemplate1.pngのtemplate matching結果は，ex5_2.png

Detection

課題6. Harris行列の計算 – 雛形 exer6.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7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です

※ 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  exer6.py  img_in.png  x_in  y_in  output.txt
```

課題7. Harrisのコーナー検出 – 雛形 exer7.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6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≧10,000の画素をコーナーとして検出**し，検出画素に円（半径2・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注意

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

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   exer7.py   fname_in.png  fname_out.png
```

課題8. Hough変換 – 雛形 excer8.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   excer8.py   img_in.png   img_out.png
```

課題9. Hough変換 – 雛形 excer9.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**40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   excer9.py   img_in.png   img_out.png
```

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，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   exer10.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

- exer6

python exer6.py det/box.png 10　20 ex6a.txt の出力は ex6a.txt に

python exer6.py det/box.png 102 29 ex6b.txt の出力は ex6b.txt に (コーナー付近)

- exer7

python exer7.py det/box.png det/boxcorner.png の出力は boxcorner.pngになる

- exer8

python exer8.py det/box.png det/boxhough.png の出力は boxhough.png に

python exer8.py det/cat.png　det/cathough.png の出力は cathough.png に

- exer9

python exer9.py det/box.png det/boxline.png の出力は boxline.png に (線なし)

python exer9.py det/cat.png　det/catline.png の出力は catline.png に

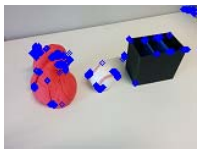
- exer10

python exer10.py det/box.png　det/boxedge.png の出力は boxedge.png に

python exer10.py det/cat.png　det/catedge.png の出力は catedge.png に


cat.png


box.png


boxcorner.png


boxline.png


catline.png


boxledge.png


catledge.png

課題11. ヒストグラムの計算 – 雛形 exer11.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フ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  exer11.py  fname_in.png  output.txt
```

課題12. Otsu法の実装 – 雛形 exer12.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，課題6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   exer12.py   target.png   template.png   fname_out.png
```

Segmentationの出力結果

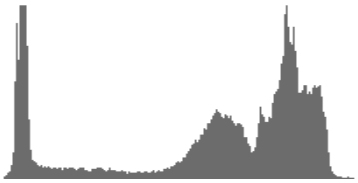
提出前のソースコードのテストに利用してください

img.png

ファイルはすべて『segm』フォルダにあります

- exer11

python exer11.py segm/img.bmp segm/histo.txt の出力は histo.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

- exer12

python exer12.py segm/img.bmp segm/ex7.png の出力は ex7.pyの通り

ex7.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13. MNISTデータの読み込み – 雛形 exer13.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（exer14.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
- K番目の画像は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3なら，ファイル名は『20_3.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  exer13.py  n
```

課題14. kNNによる文字認識 – 雛形 exer14.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一枚読み込み，kNNによりラベルの値を推定せよ

- プログラムファイル（exer14.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、exer13.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（数字ひと文字を書き出す）
- 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）

-
```
$python   exer14.py   k   img.png   output.txt
```

課題15. SVMによる文字認識 – 雛形 exer15.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一枚読み込み，SVMによりラベルの値を推定せよ

- プログラムファイル（exer15.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、exer13.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（数字ひと文字を書き出す）
- 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

- 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   output.txt
```

課題15. 追記

mnist全てを利用すると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

s.fit(x_train[0:1000], t_train[0:1000])

これは前半1000個のデータのみを利用するという事です
この書き直しをすると

- pr/img5.pngを指定 → out.txt に 7が書き込まれる
- pr/img7.pngを指定 → out.txt に 7が書き込まれる
- pr/img9.pngを指定 → out.txt に 7が書き込まれる

となります．
つまり，学習データが足らず判定を失敗します．これは，全てを学習に利用する事で回避可能ですが，自動採点プログラムの都合上[0:1000]で学習をしてください

Pattern recognitionの実行例

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

- exer13.py
  - n=10 を指定した場合 → 10_3.png が出力される
  - n=10001を指定した場合 → 10001_8.pngが出力される
- exer14
  - k=3, pr/img5.pngを指定 → knn5.txtが出力される
  - k=3, pr/img7.pngを指定 → knn7.txtが出力される
  - k=3, pr/img9.pngを指定 → knn9.txtが出力される
- exer15
  - pr/img5.pngを指定 → svm5.txtが出力される
  - pr/img7.pngを指定 → svm7.txtが出力される
  - pr/img9.pngを指定 → svm9.txtが出力される

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16. Seed Counting – 雛形 exer16.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4%以内であれば合格とする（100個の種を含む画像に対して 96 ~ 104を出力）

※発展問題の得点は5点以上（未定）
※ 外部ライブラリの利用制限はなし

- 実行コマンドは以下の通り

```
$python   exer16.py    sample.jpg     out.txt
```

課題17. Seam Carving – 雛形 exer17.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   exer17.py    img.png
```

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