Contents

[Contents 1](#_Toc99038527)

[Medical Image Processing & Visualization 3](#_Toc99038528)

[CT Data 3](#_Toc99038529)

[Medical Image preprocessing 4](#_Toc99038530)

[ITK-SNAP 6](#_Toc99038531)

[nifti 6](#_Toc99038532)

[PyDicom 7](#_Toc99038533)

[simpleITK (simple Insight Toolkit) 7](#_Toc99038534)

[Data 7](#_Toc99038535)

[Transforms and Resamplings 9](#_Toc99038536)

[Segmentation 10](#_Toc99038537)

[Registration 13](#_Toc99038538)

[Visualization 15](#_Toc99038539)

[VTK 17](#_Toc99038540)

[Python+VTK 17](#_Toc99038541)

[VTK source code dissect 18](#_Toc99038542)

[scipy.ndimage 21](#_Toc99038543)

[scikit-image 25](#_Toc99038544)

[Pillow 32](#_Toc99038545)

[Opencv 32](#_Toc99038546)

[Linux 安装 32](#_Toc99038547)

[Ubuntu 32](#_Toc99038548)

[Via source code 33](#_Toc99038549)

[《学习OpenCV》 34](#_Toc99038550)

[《OpenCV3计算机视觉》 34](#_Toc99038551)

[基本操作 34](#_Toc99038552)

[直方图 38](#_Toc99038553)

[特征检测 39](#_Toc99038554)

[模板匹配 47](#_Toc99038555)

[图像分割 48](#_Toc99038556)

[图像修复 48](#_Toc99038557)

[轮廓 48](#_Toc99038558)

[视频分割 49](#_Toc99038559)

[目标检测与识别 50](#_Toc99038560)

[目标跟踪 53](#_Toc99038561)

[摄像机模型与标定 54](#_Toc99038562)

[机器学习 55](#_Toc99038563)

[机器学习涉及的重点 57](#_Toc99038564)

[拓展知识 58](#_Toc99038565)

[光流跟踪 58](#_Toc99038566)

[医学图像配准 60](#_Toc99038567)

[MMCV 61](#_Toc99038568)

[**《计算机视觉--模型，学习和推理》** 63](#_Toc99038569)

[应用及Tools 65](#_Toc99038570)

[OCR 65](#_Toc99038571)

[人脸检测及识别 66](#_Toc99038572)

[Interview 70](#_Toc99038573)

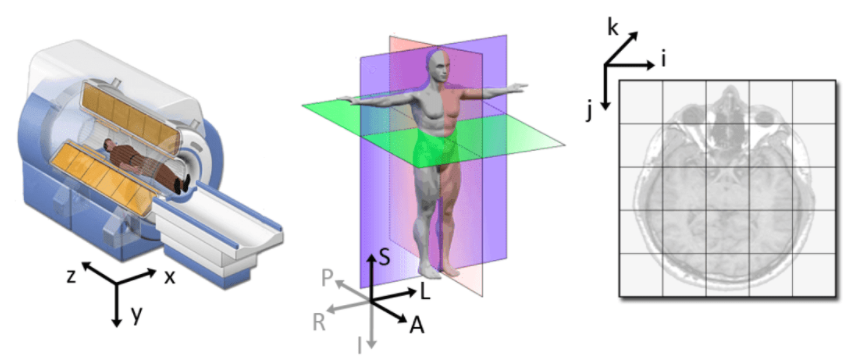
[How to detect line and circle in image? 70](#_Toc99038574)

[如何评价两张图像的相似度？ 71](#_Toc99038575)

# Medical Image Processing & Visualization

## CT Data

The coordinate systems in medical imaging



* World coordinate system
* Anatomical coordinate system

Axial plane: from foot to head (inferior – superior)

Sagittal plane: from right to left

Coronal plane: from front to back (anterior – posterior)

RAI (DICOM, ITK) coordinate system:

Voxel X Axis: Right to Left

Voxel Y Axis: Anterior to Posterior

Voxel Z Axis: Inferior to Superior

RAS (3D slicer,…) coordinate system:

From left towards right

From posterior towards anterior

From inferior towards superior

* Medical Image coordinate system (Voxel space)

World coordinate -> Voxel space :

# SITK, VTK

# formula

import dicom2nifti

dicom2nifti.convert\_directory(dcm\_dir, nii\_dir) # convert dicom to nifty

# convert dicom files (in order) to nifty

from dicom2nifti import compressed\_dicom, convert\_dicom

dicom\_input = [compressed\_dicom.read\_file(file, force=True) for file in dcm\_files]

convert\_dicom.dicom\_array\_to\_nifti(dicom\_input, nii\_file)

Affine transformation

e.g.:

# convert dicom to nifty

Affine =

<https://www.kaggle.com/kmader/show-3d-nifti-images>

## Medical Image preprocessing

<https://theaisummer.com/medical-image-processing/>

# 3D rescaling (zoom-in/out)

from scipy.ndimage.interpolation import zoom, affine\_transform

zoom(data, [s1, s2, s3], order=0)

rescaling by affine transformation

*def* random\_zoom(matrix,min\_percentage=0.7, max\_percentage=1.2):

z = np.random.sample() \*(max\_percentage-min\_percentage) + min\_percentage

zoom\_matrix = np.array([[z, 0, 0, 0],

[0, z, 0, 0],

[0, 0, z, 0],

[0, 0, 0, 1]])

*return* affine\_transform(matrix, zoom\_matrix)

import nibabel

nibabel.resample\_to\_output() 提供voxel\_size=(1, 1, 1) mm

# 3D rotation

*def* random\_rotate3D(img\_numpy, min\_angle, max\_angle):

"""

Returns a random rotated array in the same shape

:param img\_numpy: 3D numpy array

:param min\_angle: in degrees > -360

:param max\_angle: in degrees < 360

"""

all\_axes = [(1, 0), (1, 2), (0, 2)]

angle = np.random.randint(low=min\_angle, high=max\_angle+1)

axes\_random\_id = np.random.randint(low=0, high=len(all\_axes))

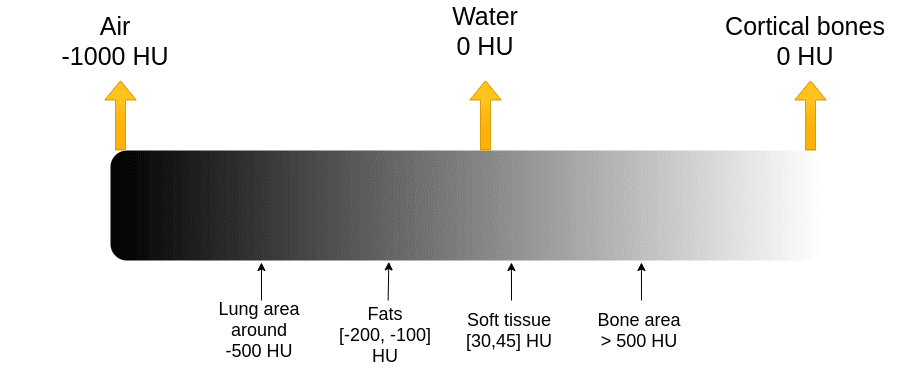
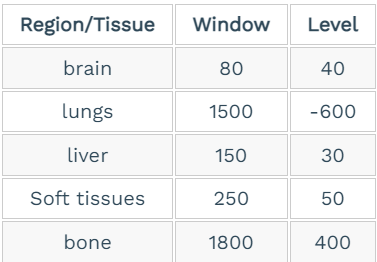
axes = all\_axes[axes\_random\_id]

*return* scipy.ndimage.rotate(img\_numpy, angle, axes=axes)

# clip intensity values (outliers)

min, max = level-window/2, level+window/2

np.clip(img, min\_value, max\_value)

# normalization

When we perform mean/std normalization we usually omit the zero intensity voxels

*def* normalize\_intensity(img\_tensor, normalization="mean"):

*if* normalization == "mean":

mask = img\_tensor.ne(0.0)

desired = img\_tensor[mask]

mean\_val, std\_val = desired.mean(), desired.std()

img\_tensor = (img\_tensor - mean\_val) / std\_val

*elif* normalization == "max":

MAX, MIN = img\_tensor.max(), img\_tensor.min()

img\_tensor = (img\_tensor - MIN) / (MAX - MIN)

*return* img\_tensor

Python文件夹文件等操作

from os import walk, path

files = []

for (dirpath, dirnames, filenames) in walk(folder):

files.extend(filenames)

os.join(folder, filename)

filename.replace(“jpg”, “npy”)

imgaug

http://imgaug.readthedocs.io/en/latest/source/installation.html

sudo pip install imgaug

## ITK-SNAP

info = pydicom.read\_file(‘\*.dcm’)

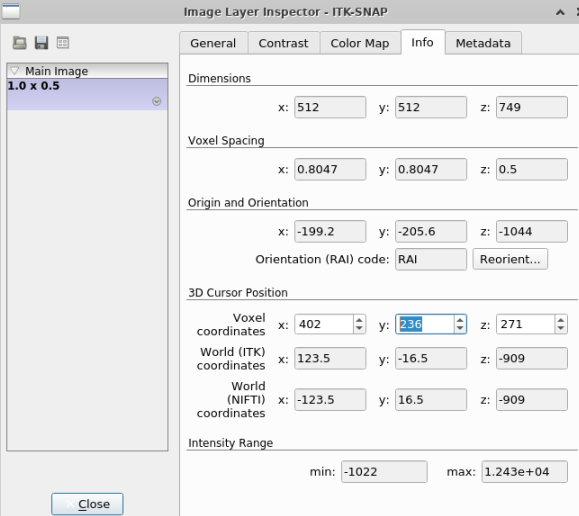
spacing = info[‘PixelSpacing’], info[‘SpacingBetweenSlices‘]

origin = info[‘ImagePositionPatient’] #inferior

orientation = RAI

voxel coordinates = (i, j, k)

world (ITK) coordinates: (x, y, z)



## nifti

$pip install dicom2nifti

## PyDicom

## simpleITK (simple Insight Toolkit)

SimpleITK provides a simplified interface to ITK in a variety of languages.

https://itk.org/Wiki/SimpleITK/GettingStarted$ sudo pip install SimpleITK

$ conda install -c simpleitk simpleitk

videos from <https://simpleitk.org/TUTORIAL/>

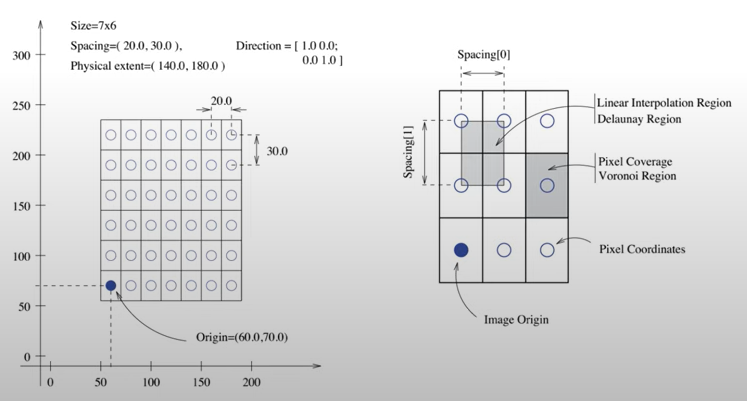
### Data

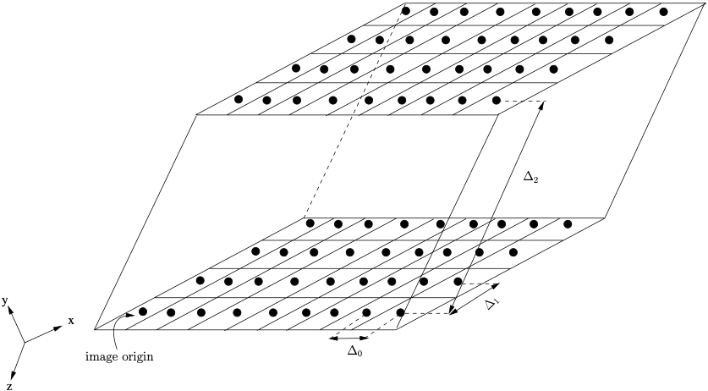
Differ from other image analysis libraries, treat image as array which has two implications:

1. voxel spacing is assumed to be isotropic
2. no notion of an image’s location in physical space.

Images are physical objects, defined by

1. Origin - location in the world coordinate system of the voxel with all zeros indexes
2. Size - number of pixels per dimension
3. Spacing - distance between pixels along each of the dimensions
4. Direction cosine matrix - axis directions in physical space corresponding to the matrix col





**Initialize Image**

#注意与numpy区别, numpy[z,y,x], sitk[x,y,z]

# 默认Origin=(0,0,0), Spacing =(1,1,1), Direction=(1,0,0, 0,1,0, 0,0,1), intensities in all channels = 0

# the unit for Origin and Spacing is unknown (km, m, cm, mm, …)

image\_3D = sitk.Image(width, height, slice\_num, sitk.sitkInt16)

image\_rgb = sitk.Image([width, height], sitk.sitkVectorUInt8, 3)

**# SimpleITK2Numpy**

nda = sitk.GetArrayFromImage(image\_3D) # copy

sitk.GetArrayViewFromImage # view

image\_3D.GetSize(): (x, y, z) nda.shape: (z, y, x)

nda = sitk.GetArrayFromImage(image\_rgb)

image\_rgb.GetSize(): (x, y) nda.shape : (y, x, 3)

**# Numpy2SimpleITK**

new\_image = sitk.GetImageFromArray(nda)

new\_image.SetOrigin()

new\_image.SetSpacing()

new\_image.SetDirection()

nda\_rgb = np.zeros((512, 512, 3))

img = sitk.GetImageFromArray(nda\_rgb, isVector=True) # 彩色图像转化为sitk，要特别注意

**# access element or sub region**

image\_3D[0, 0, 1] # in [x, y, z] order

image\_3D[x1:x2, y1:y2, z1 :z2]

image\_2D[::2, ::2] # sub-sampling

image\_2D[512:0:-1, :] # flip horizontally

**# +-\*/, >, >=, <, <=, ==**

img1 + img2

img1 > thresh

**Read images and viewer**

file\_reader = sitk.ImageFileReader()

print(‘\n’, file\_reader) # 查看sitk支持多少数据格式

img = sitk.ReadImage(‘1.png’)

sitk.WriteImage(img, ‘1.png‘)

series\_IDs = sitk.ImageSeriesReader\_GetGDCMSeriesIDs(dcm\_dir))

img = sitk.ReadImage(sitk.ImageSeriesReader\_GetGDCMSeriesFileNames(dcm\_dir, series\_IDs[0]))

sitk.WriteImage(img, ‘data.mha‘)

# 只想读取dicom header or 部分数据

file\_reader = sitk.ImageFileReader()

file\_reader.SetFileName(‘data.mha’)

file\_reader.ReadImageInformation()

start\_index, extract\_size = zip(\*[(int(0.25\*sz), int(0.5\*sz)) for sz in file\_reader.GetSize()])

file\_reader.SetExtractIndex(start\_index)

file\_reader.SetExtractSize(extract\_size)

img\_sub = file\_reader.Execute()

# cvt to jpg list

Sitk.WriteImage(sitk.Cast(sitk.RescaleIntensity(img), sitk.sitkUInt8),

[f’{i}.jpg’ for i in range(img.GetDepth())])

z = img.GetDepth()//2

plt.imshow(sitk.GetArrayViewFromImage(img)[z,:,:], cmap=plt.cm.Greys\_r)

plt.axis(‘off’)

image\_viewer = sitk.ImageViewer()

image\_viewer.SetApplication(‘/usr/bin/itksnap’)

image\_viewer.Execute(image)

### Transforms and Resampling

**Spatial Transformations**

* Global domain transforms: translation, rotation, rigid, similarity, affine, …

1. Points are represented by vector-like data types: Tuple, Numpy array, List.
2. Matrices are represented by vector-like data types in row major order.
3. Default transformation initialization as the identity transform
4. Angles specified in radians
5. Distances specified in unknown but consistent units (nm, mm, m, km…)
6. All global transformations except translation are of the form

Matrix: the matrix A

Center : the point c

Translation: the vector t

Offset:

* Bounded domain transformations: B-spline deformable transformation, Free-Form Deformation, the displacement field transformation
* Composite transformations: global or bounded domain transforms are applied in stack order (first added, last applied)

identity\_transform = sitk.Transform() # identity trnasformation

translation = sitk.TranslationTransform(3, [-256, -256, -40])

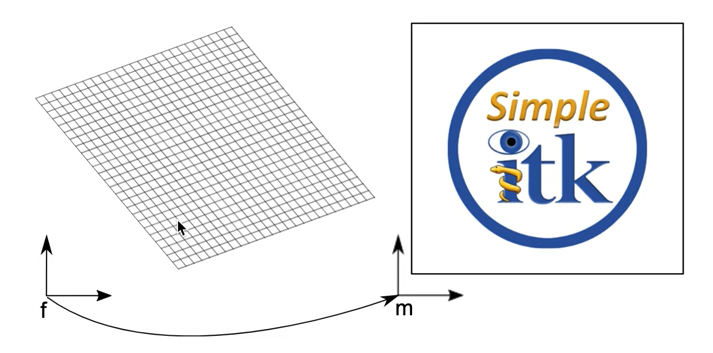
point\_transformed = translation.TransformPoint([0, 0, 0])

point = translation.GetInverse().TransformPoint(point\_transformed)

sitk.WriteTransform(transform, ‘transform.tfm‘)

transform = sitk.ReadTransform(transform.tfm‘)

**Resampling**



1. Image – the image we resample, given in coordinate system m
2. Resampling grid – a regular grid of points given in coordinates system f which will be mapped to coordinate system m

Three common resampling grid: 1) the resampled image 2) reference image 3) specify the grid using size, origin, spacing and direction cosine matrix

1. Transformation – maps points from coordinate system f to coordinate system m
2. Interpolator – method for obtaining the intensity values at arbitrary points in coordinate system

Transform = sitk.Euler2DTransform()

transform.SetCenter(image.TransformContinuousIndexToPhysicalPoint(np.array(image.GetSize())/2)

image\_resampled = sitk.Resample(image, transform)

### Segmentation

plt.hist(sitk.GetArrayViewFromImage(img).flatten(), bins=100) # histogram

#thresh\_filter = sitk.TriangleThresholdImageFilter()

#thresh\_filter = sitk.HuangThresholdImageFilter()

#thresh\_filter = sitk.MaximumEntropyThresholdImageFilter()

thresh\_filter = sitk.OtsuThresholdImageFilter()

thresh\_filter.SetInsideValue(0)

thresh\_filter.SetOutsideValue(1)

binary = thresh\_filter.Execute(img)

binary = sitk.BinaryOpeningByReconstruction(binary, [10, 10, 10]) # remove small objects

binary = sitk.BinaryClosingByReconstruction(binary, [10, 10, 10]) # fill small holes

**# watershed segmentation**

dist = sitk.SignedMaurerDistanceMap(binary!=0, insideIsPositive=False,

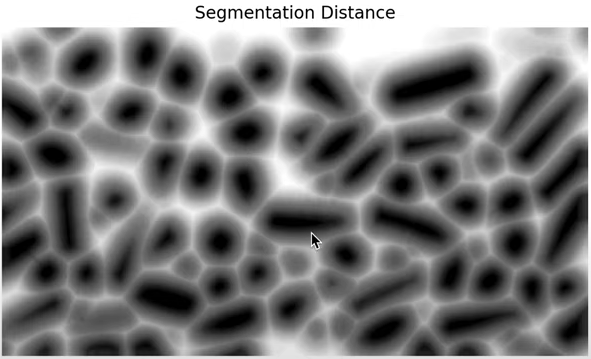
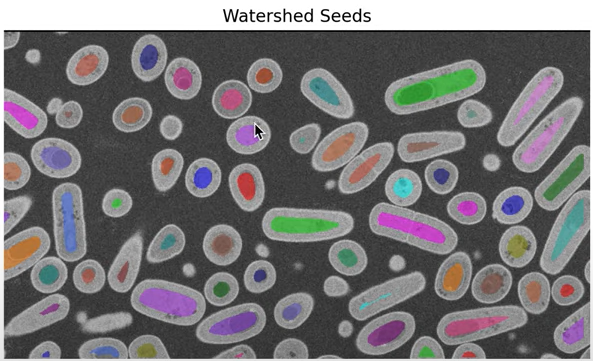
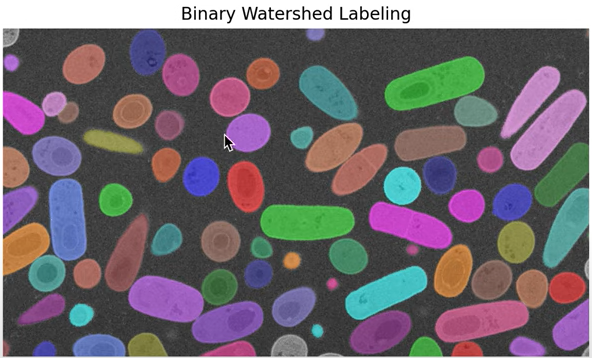
squaredDistance=False, useImageSpacing=False)

seeds = sitk.ConnectedComponent(dist < -10) # 选取远离object boundary作为seed

seeds = sitk.RelabelComponent(seeds, minimumObjectSize=15)

watershed = sitk.MorphologicalWatershedFromMarkers(dist, seeds, markWatershedLine=True)

watershed = sitk.Mask(wathershed, sitk.Cast(binary, watershed.GetPixelID()))

# 统计objects analysis

stats\_shape = sitk.LabelShapeStatisticsImageFilter()

stats\_shape.ComputeOrientedBoundingBoxOn()

stats\_shape.Execute(sitk.ConnectedComponent(watershed))

stats\_intensity = sitk.LabelIntensityStatisticsImageFilter()

stats\_intensity = sitk.Execute(watershed, img)

for label in shape\_stats.GetLabels():

stats\_shape.GetPhysicalSize(label)

stats\_intensity.GetMean(label)

**# Segmentation Evaluation (refer to videos from sitk tutorial)**

How to generate “ground truth” from multiple expert inputs

# one method is the STAPLE algorithm

ref\_prob = sitk.STAPLE(segmentations, 1)

ref = ref\_prob > threshold

# another method is majority vote

Overlap measures:

* Jaccard and Dice coefficients
* False negative and false positive errors

Surface distance measures:

* Hausdorff distance (symmetric)
* Mean, median, max and standard deviation between surfaces (detect spurious erros)

Volume measures:

* Volume similarity

Evaluation Criteria are task dependent:

* whether you are interested in detecting spurious errors (mean or max surface distance)
* whether over/under segmentation should be differentiated (volume similarity and Dice)
* what is the ratio between acceptable errors and the size of the segmented object (Dice coefficient may be too sensitive to small errors when the segmented object is small and not sensitive enough to large errors when the segmented object is large).
* In the context of segmentation challenge, algorithm rankings are often based on a weighted combination of these criteria.

overlap\_measures\_filter = sitk.LabelOverlapMeasuresIamgeFilter()

overlap\_measures\_filter.Execute(ref, seg)

overlap\_measures\_filter.GetJaccardCoefficient()

overlap\_measures\_filter.GetDiceCoefficient()

overlap\_measures\_filter.GetVolumeSimilarity()

overlap\_measures\_filter.GetFalseNegativeError()

overlap\_measures\_filter.GetFalsePositiveError()

hausdorff\_distance\_filter = sitk.HausdorffDistanceImageFilter()

hausdorff\_distance\_filter.Execute(ref, seg)

hausdorff\_distance\_filter.GetHausdorffDistance()

#use the absolute values of the distance map to compute the surface distances

# (distance map sign, outside or inside relationship, is irrelevant)

ref\_dist = sitk.SignedMaurerDistanceMap(ref, squaredDistance=False, useImageSpacing=True)

ref\_dist\_map = sitk.Abs(dist)

ref\_surface = sitk.LabelContour(ref)

seg\_dist = sitk.SignedMaurerDistanceMap(seg, squaredDistance=False, useImageSpacing=True)

seg\_dist\_map = sitk.Abs(dist)

seg\_surface = sitk.LabelContour(seg)

# multiply the binary surface segmentations with the distance maps. The resulting distance

# maps contain non-zero values only on the surface

seg2ref\_dist\_map = ref\_dist\_map \* sitk.Cast(seg\_surface, sitk.sitkFloat32)

ref2seg\_dist\_map = seg\_dist\_map \* sitk.Cast(ref\_surface, sitk.sitkFloat32)

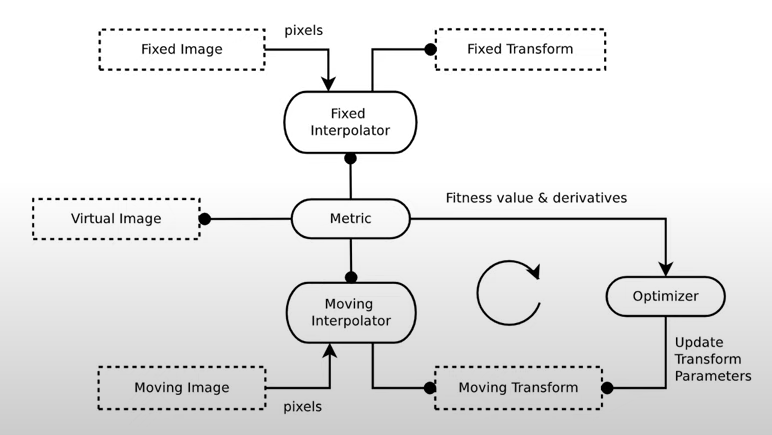
…

np.mean(all\_surface\_distance)

### Registration

The goal of registration is to estimate the transformation (from fixed image to moving image)

Successful registration is highly dependent on initialization



**Rigid Registration: estimate 3D rigid transformation between images**

* Similarity metric: mutual information
* Interpolator: sitkLinear
* Optimizer: gradient descent

method = sitk.ImageRegistrationMethod()

# similarity metric settings

method.SetMetricAsMattesMutualInformation(numberOfHistogramBins=50)

method.SetMetricSamplingStrategy(method.RANDOM)

method.SetMetricSamplingPercentage(0.01) # Sampling percentage (1%)

method.SetInterpolator(sitk.sitkLinear)

method.SetOptimizerAsGradientDescent(learningRate=1.0, numberOfIterations=100,

convergenceMinimumValue=1e-6, convergenceWindowSize=10)

# Initialize registration by aligning the centers of the two volumes

init\_transform = sitk.CenteredTransformInitializer(fixed\_image, moving\_image,

sitk.Euler3DTransform(), sitk.CenteredTransformInitializerFilter.GEOMETRY)

method.SetInitialTransform(init\_transform, inPlace=False)

‘’’

Initialize registration by manually specified points

fixed\_image\_points = [(x1, y1, z1), …]

moving\_image\_points = [(x1, y1, z1), …]

init\_transform = sitk.LandmarkBasedTransformInitializer(sitk.VersorRigid3DTransform(),

fixed\_image\_points, moving\_image\_points)

‘’’

# setup for the multi-resolution framework

method.SetShrinkFactorsPerLevel(shrinkFactors=[4,2,1])

method.SetSmoothingSigmasPerLevel(smoothingSigmas=[2,1,0])

method.SmoothingSigmasAreSpecifiedInPhysicalUnitsOn()

final\_transform = method.Execute(fixed\_image, moving\_image)

moving\_resampled = sitk.Resample(moving\_image, fixed\_image, final\_transform)

**Non-rigid Registration:** Free Form Deformation, and Demons

* Free Form Deformation: Define a BSplineTransform using a sparse set of grid points overlaid onto the fixed image to deform it

Input: fixed\_image, fixed\_image\_mask, moving\_image, moving\_image\_mask

Output: transformed\_seg 比如只对CT肺部做配准，即将CT lung seg 进行变形

method = sitk.ImageRegistrationMethod()

# similarity metric settings

method.SetMetricAsMeanSquares()

method.SetMetricSamplingStrategy(method.RANDOM)

method.SetMetricSamplingPercentage(0.01)

method.SetMetricFixedMask(fixed\_image\_mask)

method.SetInterpolator(sitk.sitkLinear)

method.SetOptimizerAsLBFGS2(solutionAccuracy=1e-2, numberOfIterations=100,

deltaConvergenceTolerance=0.01)

init\_transform = sitkBSplineTrnasformInitializer(…)

method.SetInitialTransformAsBSpline(init\_transform, …)

# setup for the multi-resolution framework

method.SetShrinkFactorsPerLevel(shrinkFactors=[4,2,1])

method.SetSmoothingSigmasPerLevel(smoothingSigmas=[2,1,0])

method.SmoothingSigmasAreSpecifiedInPhysicalUnitsOn()

final\_transform = method.Execute(fixed\_image, moving\_image)

transformed\_seg = sitk.Resample(moving\_image\_mask, fixed\_image, final\_transform,

sitk.sitkNearestNeighbor, …)

* Demons Based Registration (please refer to sitk tutorial): displacement

**Registration evaluation:**

* TRE (Target Registration Error)

Given the transformation and corresponding points in the two coordinate system (which not used in the registration process)

initial\_TRE = TRE(sitk.Transform(), fixed\_points, moving\_points)

final\_TRE = TRE(final\_transform, fixed\_points, moving\_points)

assert final\_TRE < initial\_TRE

* use segmentation for registration evalution, we independently segment the structures of interest in the two images

ref\_seg = fixed\_image\_mask

seg\_before = moving\_image\_mask

seg\_after = transformed\_seg

metric\_before = fun(ref\_seg, seg\_before)

metric\_after = fun(ref\_seg, seg\_after)

assert dice\_after > dice\_before

### Visualization

# map [level-win/2, level+win/2] to [0, 255]

img = sitk.Cast(sitk.IntensityWindow(img, windowMinimum=2, windowMaximum=657,

outputMinimum=0, outputMaximum=255), sitk.sitkUInt8)

def make\_isotropic(image, interpolator=sitk.sitkLinear):

original\_spacing = image.GetSpacing()

if all(spc == original\_spacing[0] for spc in original\_spacing): return sitk.Image(image)

# make image isotropic via resampling

original\_size = image.GetSize()

min\_spacing = min(original\_spacing)

new\_spacing = [min\_spacing] \* image.GetDimension()

new\_size = [int(round(osz\*ospc/min\_spacing)) for osz, ospc in zip(original\_size, original\_spacing)]

return sitk.Resample(image, new\_size, sitk.Transform(), interpolator,

image.GetOrigin(), new\_spacing, image.GetDirection(), 0,

image.GetPixelID())

# combine two images

Alpha blending:

sitk.Cast(sitk.Compose(img1, img1, img3), sitk.sitkVectorUInt8)) three grays -> rgb

# 伪彩色编码

dist\_map = sitk.SignedMaurerDistanceMap(mask)

color\_dist = sitk.ScalarToRGBColormap(dist\_map, sitk.ScalarToRGBColormapImageFilter.Jet)

# overlay segmentation onto image

* Map the segmentation labels to a color image and alpha blend onto the original image
* Overlay the segmentation boundaries onto the original image

sitk.LabelToRGB(seg)

pink = [255, 105, 180]

green = [0, 255, 0]

gold = [255, 215, 0]

img\_mask = sitk.LabelOverlay(img, seg, opacity=0.5,

backgroundValue=0, # background label not overlaid

colormap=pink+green+gold)



contour = sitk.LabelContour(seg, fullyConnected=True, backgroundValue=255)

contour\_rgb = sitk.LabelToRGB(contour, colormap=red+green+blue, backgroundValue=255)

# show contour

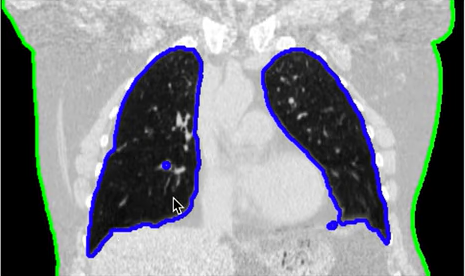
msk = sitk.Cast(seg, sitk.sitkLabelUInt8)

img = sitk.IntensityWindow(img, window\_min, window\_max)

img = sitk.Cast(img, sitk.sitkUInt8)

contour\_overlaid\_image = sitk.LabelMapContourOverlay(msk, img, opacity=1, contourThickness=[4,4],

dilationRadius=[3,3], colormap=red+green+blue)



# show image

plt.imshow(sitk.GetArrayViewFromImage(img)[z,:,:], cmap=plt.cm.Greys\_r)

plt.axis(‘off’)

image\_viewer = sitk.ImageViewer()

image\_viewer.SetApplication(‘/usr/bin/itksnap’)

image\_viewer.Execute(image)

# show seq

tile\_w = int(np.sqrt(num\_slices))

tile\_h = int(np.ceil(num\_slices/tile\_w))

tile\_image = sitk.Tile(seq, (tile\_w, tile\_h))

## VTK

### Python+VTK

最简单方式：

http://stackoverflow.com/questions/15383666/installing-vtk-for-python

$sudo apt-get install python-vtk

Example: <http://www.vtk.org/Wiki/VTK/Examples/Python>

source → filter → mapper → actor → render → window → interactor

Keypress j/t toggle between joystick (position sensitive) and trackball (motion sensitive)

shift + mouse pan

middle mouse zoom

Keypress e/q exit or quit the application

Keypress r reset the camera view

官方：

<http://www.vtk.org/download/>

download vtkpython-7.1.0-Linux-64bit.tar.gz

### VTK source code dissect

Observer/command 模式

Command 模式

对象（vtkRender）发送一个命令,然后执行观察者相应的动作

Void StartEvent() -> ExecuteStart()

Void EndEvent() -> ExecuteEnd()

若有很多命令，同时有相应的动作，如何统一一个接口？

Void InvokeEvent(Event) -> Execute()

命令映射，从而使每个具体命令有唯一标识符（命令的名字作为主键）,用带命令标识符参数的函数统一接口InvokeEvent(Event),对于被调用的对象依次发起的每个命令(如StartEvent, EndEvent, ProcessEvent)，在客户端注册的观察者集合中查找匹配的命令，并执行客户端的动作

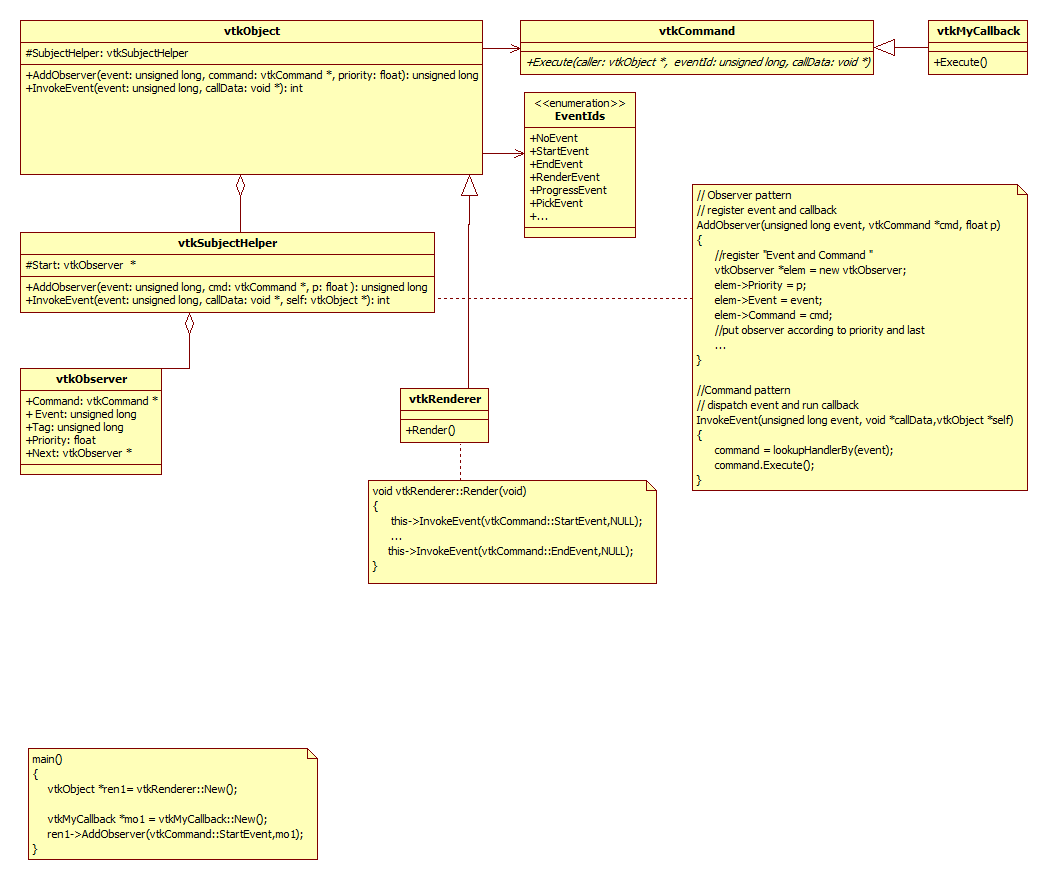
动作通过接口继承统一接口Execute().

Observer模式

客户端注册命令和相应的动作（事件和相应的回调函数作为一个观察者），从而可以在对象运行时，了解对象的状态。（多个观察者接注册顺序添加到对象的变量列表中如vtkRender的vtkObservers，从而有机会被通知到干活）

应用程序运行时，会依次发起客户端注册的事件，从而触发客户端的回调函数.(一对多)

（另一个例子是界面语言切换，发起事件，从而每个注册的界面切换语言）



VTK 使用心得

VTK source directory Structure

InfoVis

-----classes for information visualization

Views

-----classes for viewing data including filters, visualization, interaction and selection

Common

-----core classes

Filtering

-----classes related to data processing in the visualization pipeline

GenericFiltering

-----an adaptor framework to interface VTK to external simulation packages

GeoVis

-----view, sources and other objects useful in terrain visualization

Graphics

-----filters that process 3D data

GUISupport

-----classes for using VTK with the MFC and Qt user interface packages

Hybrid

-----complex classes that depend on classes in multiple other directories

Imaging

-----image processing filters

IO

-----classes for reading and writing data

Parallel

-----classes used to render

Utilities

-----supporting software like expat, png, jpeg, tiff and zlib

VolumeRendering

-----classes used for volume rendering

Widgets

-----3D widget classes

Wrapping

-----support for Tcl, Python, and Java wrapping.

Examples

-----examples, grouped by topic

CMake

-----configuration files for cross-platform building

Application software: ParaView

Except creating VTK application using the Tcl ( in this case, pre-compiled binaries may be available for the windows platform), you will have to compile and link the source code to produce libraries and executables.

VTK = visualization pipeline + rendering engine

Visualization pipeline is used to acquire or create data, process that data, and either write the results to a file or pass the results to the rendering engine for display

Rendering engine is responsible for creating a visual representation of the data

Actors: serves to group rendering attributes such as surface properties(e.g., ambient, diffuse, and specular color), representation(e.g., surface or wireframe), texture maps, and a geometric definition(a mapper)

Mappers: geometric definition using analytic primitives such as points, lines, polygons and triangle strips, the mapper terminates the visualization pipeline and serves as the bridge between the visualization subsystem and the graphics subsystem

Coordinate systems:

Display: x-y pixel values in the rendering window, the original is the lower-left corner

View: x-y-z(-1,1) values in camera coordinates(z is depth)

Operation performed on image data in VTK: image processing + geometry extraction + direct rendering

# scipy.ndimage

建立在numpy基础上

**2D可以交给opencv, muti-dimensional交给ndimage操作**

* **input/output**

from scipy.miscimport imread,

imsave,

imresize,

imfilter,

imrotate,

imshow,

bytescale,

central\_diff\_weights,

comb,

derivative,

factorial,

factorial2,

fromimage,

info,

logsumexp,

pade,

toimage,

source,

who

# Read an JPEG image into a numpy array

img = imread('assets/cat.jpg')

from scipy.io import loadmat,

savemat,

whosmat,

readsav,

mminfo,

mmread,

mmwrite,

FortranFile,

netcdf\_file,

netcdf\_variable,

hb\_read,

hb\_write,

wavefile,

read,

write,

arff,

loadarff,

MetaData

data =scipy.io.loadmat(‘test.mat’) load matlab data

# 自带图像

from scipy.misc import lena,

ascent,

face

# Resize the image to be 300 by 300 pixels.

img\_tinted = imresize(img, (300, 300))

# Write the tinted image back to disk

imsave('assets/cat\_tinted.jpg', img\_tinted)

# Show the original image

plt.subplot(1, 2, 1)

plt.imshow(img)

* **Filters**

output = ndimage.convolve(input, weights) input可以3D

correlate

gaussian\_filter

gaussian\_gradient\_magnitude

gaussian\_laplace

generic\_filter

generic\_gradient\_magnitude

generic\_laplace

laplace

prewitt

sobel

# 邻域内最大值，size定义邻域shape, footprint可以正定义邻域，类似mask

output = ndimage.maximum\_filter(input, size=None, footprint=None)

minimum\_filter

median\_filter

uniform\_filter

percentile\_filter

rank\_filter

Morphology

generate\_binary\_structure/iterate\_structure, binary\_closing/binary\_dilation/binary\_erosion/binary\_opening

binary\_fill\_holes,

binary\_hit\_or\_miss,

black\_tophat/white\_tophat,

binary\_propagation,

distance\_transform\_bf/distance\_transform\_cdt/ distance\_transform\_edt, grey\_closing/grey\_dilation/grey\_erosion/grey\_opening, morphological\_gradient/morphological\_laplace

Fourier filters

fourier\_ellipsoid

fourier\_gaussian

fourier\_shift

fourier\_uniform

基于各种坐标变换进行插值

from scipy.ndimage.interpolation import affine\_transform,

geometric\_transform,

map\_coordinates,

rotate,

shift,

spline\_filter,

zoom

zoom(data, [s1, s2, s3], order=0)

* **Measurements**

基于label（也就是基于各区域）计算统计量

center\_of\_mass

extrema

find\_objects

histogram

label

maximum/minimum

maximum\_position/minimum\_position

sum/mean/median/standard\_deviation/variance

watershed\_ift

# scikit-imagehttp://scikit-image.org/docs/stable/

scikit-image.org/docs/dev/

The scikit-image SciKit (toolkit for SciPy) **extends scipy.ndimage** to provide a versatile set of image processing routines

# $sudo apt-get install python-skimage

$sudo pip install scikit-image

* **图像输入输出**

from skimage.io import imread,

imread\_collection,

imread\_collection\_wrapper,

imsave,

show,

imshow,

imshow\_collection,

load\_sift,

load\_surf,

concatenate\_images,

ImageCollection,

MultiImage

moon = imread(filename) 返回ndarray

out = img\_as\_uint(sobel(image))

plt.imshow(out)

from skimage.viewer import CollectionViewer,

ImageViewer

CollectionViewer([data.coins(), data.astronaut()]).show() 显示图片序列

# 自带图像

from skimage.data import astronaut,

binary\_blobs,

camera,

checkerboard,

chelsea,

clock,

coffee,

coins,

horse,

hubble\_deep\_field,

moon,

page,

rocket,

stereo\_motorcycle,

text

* **与OpenCV区别**

OpenCV image data can be accessed (without copying) in NumPy (and, thus, in scikit-image). **OpenCV uses BGR (instead of scikit-image’s RGB) for color images**, and its dtype is uint8 by default

image = image[:, :, ::-1] #Converting BGR to RGB or vice versa

from skimage import img\_as\_float

**image = img\_as\_float(any\_opencv\_image) # skiimage <- opencv image**

from skimage import img\_as\_ubyte

**cv\_image = img\_as\_ubyte(any\_skimage\_image) # opencv image <- skiimage**

# 索引，ndarray该有的都有，因为返回的就是ndarray表示图像

camera[10, 20]

mask = camera < 87

lower\_half = row > cnt\_row

camera[np.logical\_and(lower\_half, mask)] = 255

图像数据类型及转换

from skimage import dtype\_limits,

img\_as\_bool,

img\_as\_float,

img\_as\_float32,

img\_as\_float64,

img\_as\_int,

img\_as\_ubyte,

img\_as\_uint

Data type Range Function name

uint8 0 to 255 img\_as\_ubyte()

uint16 0 to 65535 img\_as\_uint()

uint32 0 to 232

float -1 to 1 or 0 to 1 img\_as\_float ()

int8 -128 to 127

int16 -32768 to 32767 img\_as\_int()

int32 -231 to 231 - 1

min, max = dtype\_limits(image)

* **维数次序，2D/3D处理**

Matplotlib: standard Cartesian coordinates, where x is the horizontal coordinate, y the vertical, and the origin is on the bottom left.

Two-dimensional (2D) grayscale images: are indexed by (row, col), with the lowest element (0, 0) at the top-left corner

Dimension name and order conventions in scikit-image

Image type coordinates

2D grayscale (row, col)

2D multichannel (eg. RGB) (row, col, ch)

3D grayscale (pln, row, col)

3D multichannel (pln, row, col, ch)

2D color video (t, row, col, ch)

3D multichannel video (t, pln, row, col, ch)

# 2D 处理方式

from skimage import filters

edges = np.zeros\_like(im3d)

for pln, image in enumerate(im3d):

# iterate over the leading dimension (planes)

edges[pln] = filters.sobel(image)

# 3D 处理方式， skimage很多图像操作支持3D处理

from scipy.ndimage import label

from skimage import morphology

seeds = label(im3d < .1)[0]

ws = morphology.watershed(im3d, seeds)

Image processing pipeline

def custom\_func(image):

image = img\_as\_float(image)

from skimage import img\_as\_float

processed\_image = custom\_func(func1(func2(image)))

Image adjustment: transforming image content

图像Augment

from skimage.util import random\_noise 添加噪声

from skimage.util import crop, pad,

* **图像顡色空间转换**

from skimage.color import gray2rbg,

rbg2gray,

rgb2hsv,

rgba2rgb,

…

* **图像灰度变换**

from skimage.exposure import adjust\_gamma,

adjust\_log,

adjust\_sigmoid,

cumulative\_distribution,

equalize\_adapthist,

equalize\_hist,

histogram,

is\_low\_contrast,

rescale\_intensity

Gamma Correction: 0 = I \*\* gamma

Logarithmic Correction: 0 = gain \* log(1 + I)

Sigmoid Correction: 0 = 1/(1 + exp(gain\*(cutoff-I)))

equalize\_adapthist: Contrast Limited Adaptive Histogram Equalization (局部对比度增强）

equalize\_hist: 直方图均衡化

is\_low\_contrast: 亮度范围与数据类型范围的比率低于阀值

[0, 127, 255] = rescale\_intensity(np.array([51, 102, 153]) [min, max] → [0, 255]

[0., .5, 1.] = rescale\_intensity(np.array([51., 102., 153.]) [min, max] → [0, 1]

[.2, .4, .6] = rescale\_intensity(np.array([51., 102., 153.], in\_range=(0, 255)) in\_range → [0, 1]

[.5, 1., 1.] = rescale\_intensity(np.array([51., 102., 153.], in\_range=(0, 102)) 截断

* **特征提取**

from skimage.feature import blob\_dog,

blob\_doh,

blob\_log, corner\_fast,

corner\_foerstner,

corner\_harris,

corner\_kitchen\_rosenfeld,

corner\_moravec,

corner\_orientations,

corner\_shi\_tomasi,

CENSURE,

corner\_peaks,

peak\_local\_max,

corner\_subpix,

daisy,

draw\_multiblock\_lbp,

greycomatrix,

greycoprops, hessian\_matrix,

hessian\_matrix\_det,

hessian\_matrix\_eigvals,

structure\_tensor,

structure\_tensor\_eigvals,

local\_binary\_pattern,

multiblock\_lbp,

match\_descriptors,

match\_template,

plot\_matches,

register\_translation,

BRIEF,

ORB,

shape\_index,

canny,

hog

* **滤波**

from skimage.filters import pply\_hysteresis\_threshold,

threshold\_adaptive,

threshold\_isodata,

threshold\_li,

threshold\_local,

threshold\_mean,

threshold\_minimum,

threshold\_niblack,

threshold\_otsu,

threshold\_sauvola,

threshold\_triangle,

threshold\_yen,

frangi,

gabor,

gabor\_kernel,

wiener,

LPIFilter2D,

gaussian,

hessian,

laplace,

median,

prewitt,

prewitt\_h,

previtt\_v,

Roberts,

roberts\_net\_diag,

roberts\_pos\_diag,

scharr,

scharr\_h,

scharr\_v,

sobel,

sobel\_h,

sobel\_v,

inverse

* **形态学**

from skimage.morphology import ball,

cube,

diamond,

disk,

octagon,

octahedron,

rectangle,

square,

star,

binary\_closing,

binary\_dilation,

binary\_erosion,

binary\_opening,

binary\_closing,

thin,

black\_tophat,

white\_tophat,

dilation,

erosion,

opening,

h\_maxima,

h\_minima,

label,

recontruction,

watershed,

remove\_small\_holes,

remove\_small\_objects,

local\_maxima,

local\_minma,

medial\_axis,

convex\_hull\_image,

convex\_hull\_object,

skeletonize,

skeletonize\_3d,

* **测量**

from skimage.measure import approximate\_polygon,

block\_reduce,

compare\_mse,

compare\_nrmse,

compare\_psnr,

compare\_ssim,

correct\_mesh\_orientation,

find\_contours,

grid\_points\_in\_poly,

label,

marching\_cubes\_classic,

marching\_cubes\_lewiner,

mesh\_surface\_are,

moments,

moments\_central,

moments\_hu,

moments\_normalized,

perimeter,

points\_in\_poly,

regionprops,

shannon\_entropy,

subdivide\_polygon,

CircleModel,

EllipseModel,

LineModelND

* **画图形**

from skimage.draw import bezier\_curve,

circle,

circle\_perimeter,

circle\_perimeter\_aa,

ellipse,

ellipse\_perimeter,

ellipsoid,

line,

line\_aa,

polygon,

polygon\_perimeter,

rectangle,

rr, cc = bezier\_curve(1, 5, 5, -2, 8, 8, 2) (1,5)起点，(5, -2)控制点，(8, 8)终点

img = np.zeros((10, 10), dtype=np.uint8)

img[rr, cc] = 1

rr, cc = circle(4, 4, 5) (4, 4)中心点， 5半径

img[rr, cc] = 1 圆盘

rr, cc = circle\_perimeter(4, 4, 5)

img[rr, cc] = 1 圆形

rr, cc = ellipse(5, 6, 3, 5, rotation=np.deg2rad(0)) (5, 6)中心点，(3, 5)长短轴半径

ellip = ellipsoid(a, b, c) 隋球体

vol, surf = ellipsoid\_stats(a, b, c) 隋球体体积和表面积

rr, cc = line(1, 1, 8, 8)

r = np.array([x0, x1, x2, …])

c = np.array([y0, y1, y2, ...])

rr, cc = polygon(r, c) 多边形区域

* **图论**

from skimage.graph import route\_through\_array,

shortest\_path,

MCP,

MCP\_Connect,

MCP\_Flexible,

MCP\_Geometric

# Pillow

Pillow-style img is column-major while numpy-style img is row-major

from PIL import Image

img = Image.open(‘1.jpg‘)

img\_gray = img.convert(‚L’)

img.show()

img.save(‘img.jpg’)

data = np.array(img) # img -> numpy.adarray

img = Image.fromarray(np.uint8(data)) # adarray -> Image

# Opencv

Linux 安装

<http://www.pyimagesearch.com/2016/10/24/ubuntu-16-04-how-to-install-opencv/>

Ubuntu

$ sudo pip install opencv-contrib-python

Via source code

Step #1: Install OpenCV dependencies on Ubuntu 16.04

refresh and upgrade and pre-installed packages/libraries

$ sudo apt-get update

$ sudo apt-get upgrade

developer tools

$ sudo apt-get install build-essential cmake pkg-config

图像导入依赖库

$ sudo apt-get install libjpeg8-dev libtiff5-dev libjasper-dev libpng12-dev

视频导入依赖库

$ sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev

GUI依赖库

$ sudo apt-get install libxvidcore-dev libx264-dev libgtk-3-dev

数值计算依赖库

$ sudo apt-get install libatlas-base-dev gfortran

Python development headers and libraries

$ sudo apt-get install python2.7-dev python3.5-dev

Step #2: Download the OpenCV source

$ cd ~

$ wget -O opencv.zip https://github.com/Itseez/opencv/archive/3.1.0.zip

$ unzip opencv.zip

$ wget -O opencv\_contrib.zip https://github.com/Itseez/opencv\_contrib/archive/3.1.0.zip

$ unzip opencv\_contrib.zip

Step #3: Setup your Python environment — Python 2.7 or Python 3

Step #4: Configuring and compiling OpenCV on Ubuntu 16.04

$ cd ~/opencv-3.1.0/

$ mkdir build

$ cd build

$ cmake -D CMAKE\_BUILD\_TYPE=RELEASE \

-D CMAKE\_INSTALL\_PREFIX=/usr/local \

-D INSTALL\_PYTHON\_EXAMPLES=ON \

-D INSTALL\_C\_EXAMPLES=OFF \

-D OPENCV\_EXTRA\_MODULES\_PATH=~/opencv\_contrib-3.1.0/modules \

-D PYTHON\_EXECUTABLE=~/.virtualenvs/cv/bin/python \

-D BUILD\_EXAMPLES=ON ..

If download ippicv\_linux\_20151201.tgz failure

Please download and *cp ippicv\_linux\_20151201.tgz ~/opencv-3.1.0/3rdparty/ippicv/downloads/linux-808b791a6eac9ed78d32a7666804320e/ippicv\_linux\_20151201.tgz*

$ make -j8

若出现编译出法成功，可能是cuda8与opencv3.1中的算法Graphcut冲突，解决方法：

<https://devtalk.nvidia.com/default/topic/986950/opencv-installation-problem-nppigraphcutinitalloc-not-declared/>

将文件opencv-3.1.0/modules/cudalegacy/src/graphcuts.cpp

#if !defined (HAVE\_CUDA) || defined (CUDA\_DISABLER) || (CUDART\_VERSION >= 8000)

$ sudo make install

$ sudo ldconfig

>>> import cv2

>>> cv2.\_\_version\_\_

若你的服务器装过anaconda，可能存在一些 冲突，如果使用上面这条命令出现：Makefile:160: recipe for target 'all' failed的报错。Shell环境变换export PATH=… 屏蔽anaconda

《学习OpenCV》

《OpenCV3计算机视觉》

### 基本操作

图像读写与显示

import cv2 as cv

# imread()函数会删除所有alpha通道的信息（透明度）

grayImage = cv.imread('1.png', cv.IMREAD\_GRAYSCALE)

cv.imshow(‘image’, grayImage)

cv.waitKey()

cv.destroyAllWindows()

# bmp格式要求每个通道有8 bits, 而png允许每个通道有8 bits adn 16 bits

cv.imwrite('tmp.png', grayImage)

注意：导进来的图像是numpy数组，可以基于numpy操作, opencv是基于BGR顺序，网页颜色采用RGB

img[0, 0] = [255, 0, 0]

视频文件读写，可以获取帧率和帧大小

摄像头读，参数是摄像头的设备索引device index, 摄像头的数量和顺序由系统决定,同时OpenCV没有提供任何查询摄像头数量和属性的方法

在帧率未知的情况下，通常在开始获取帧的时候会跳过几个帧，以得到估计帧速率的时间

通常对图像应用低通滤波器之后，与原始图像计算差值，从而获得高通滤波器

blurred = cv.GaussianBlur(img, (11, 11), 0)

g\_hgf = img – blurred

定制核做卷积

kernel = numpy.array([[…], …])

cv.filter2D(src, -1, kernel, dst)

对彩色图像来说，filter2D会对每个通道都用同样的核

注意：核权重加起来为1,如果不想改变图像的亮度就应该这样。如果使权重加起来为0，就会得到一个边缘检测核

阈值分割

diff = cv.threshold(diff, 25, 255, cv.THRESH\_BINARY)

Canny边缘检测：

1. 使用高斯滤波器对图像进行去噪
2. 计算梯度
3. 在边缘上使用非最大抑制NMS
4. 在检测到的边缘上使用双阈值去除假旧性
5. 最后还会分析所有的边缘及其之间的连接，以保留真正的边缘并消除不明显的边缘

cv.Canny(img, low\_threshold, high\_threshold)

直线检测和圓检测

lines = cv.HoughLinesP(edge, 1, np.pi/180, 100, minLineLength, maxLineGap)

for x1, y1, x2, y2 in lines[0]:

cv.line(img, (x1, y1), (x2, y2), ...)

circles = cv.HoughCircles(img, cv.HOUGH\_GRADIENT, 1, 120, ...)

Hough变换与广义hough变换

hough变换一般适用于有解析表达式的几何形状目标检测：如直线，圆，椭圆等

流程：

1. 霍夫变换的核心就是把图像空间的直线变换到参数空间（也叫霍夫空间）

f(x,alpha)=0,其中x 是图形上点，alpha则是解析表达式参数

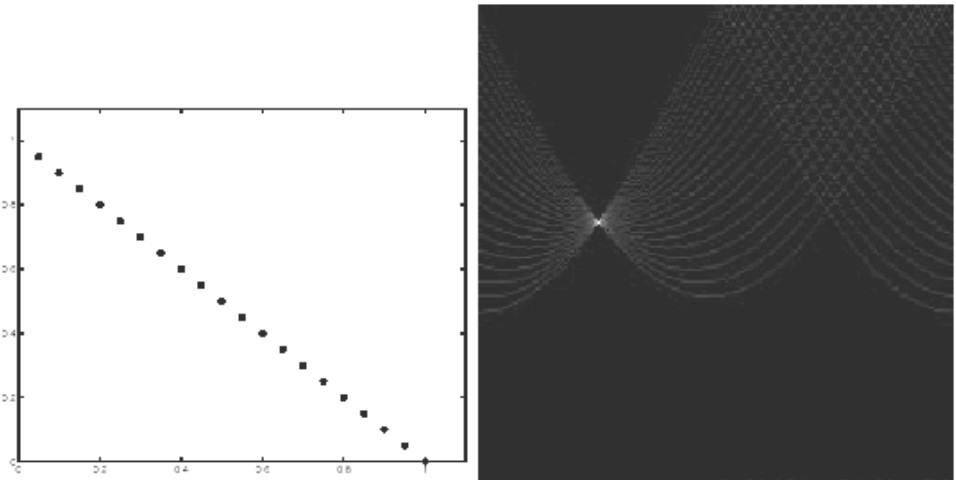
图像空间几何形状 -> hough空间

直线：边缘点 -> (r, theta): 正弦线交点

圆：边缘点 -> (x0, y0, r): 圆锥体交点

椭圆：边缘点 -> (x0, y0, a, b, theta)

1. 求hough空间极大值



代码：

For each edge pixel (x,y) in image

For θ= -90 to 90

ρ= x cosθ+ y sin θ

++ H(iθ, jρ);

end

end

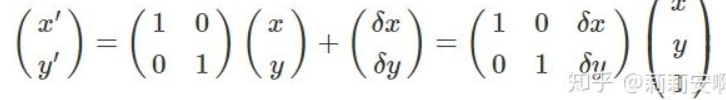
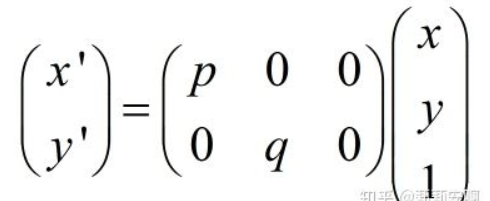
广义hough变换

核心思想:把图像中属于某种图形的点集（二维）映射到一个点（可以是高维）上，这个点记录了点集中点的数目，使得程序通过搜索峰值找到该点，这个点就是图形参数

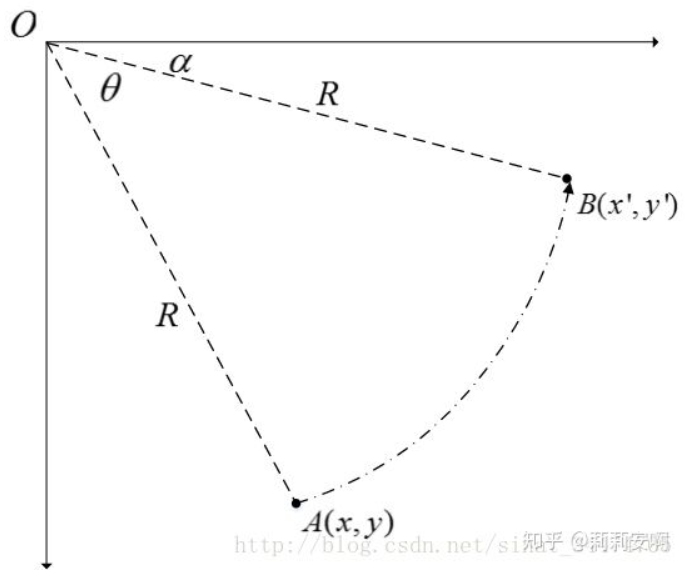
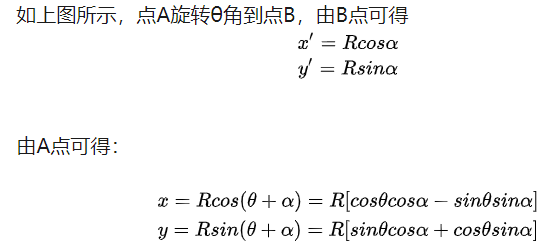
几何变换

仿射变换（Affine transformation）

平移（translation）缩放（Scaling）

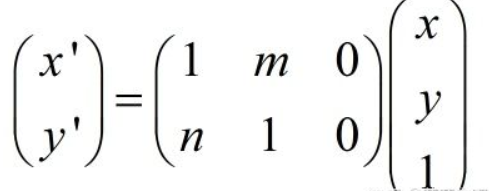
 

旋转（Rotation）

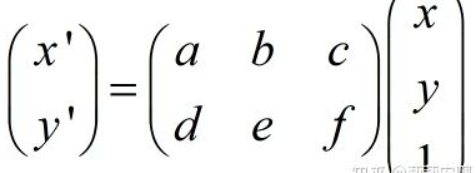
剪切(Shear)

剪切变换指的是类似于四边形不稳定性那种性质，方形变平行四边形。任意一边都可以被拉长，以一定比例的x补偿y，也以一定比例的y补偿x。



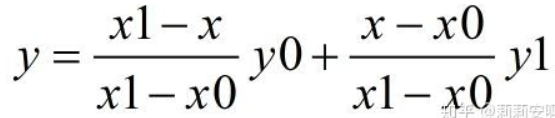
仿射变换（Affine transformation）

当6个参数取其上述变换以外的值时，为一般的仿射变换，效果相当于从不同的位置看同一个目标。



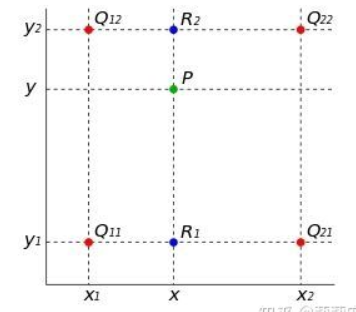
线性插值

已知点 (x0, y0) 与 (x1, y1)，要计算 [x0, x1] 区间内某一位置 x 在直线上的y值

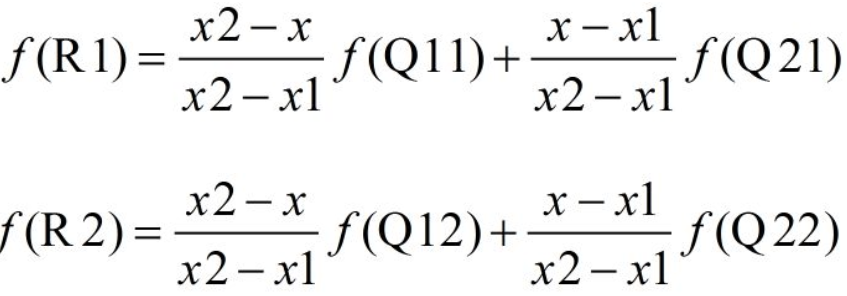
 （可以由一阶Taylor推导）

f(x) = f(x0) + f’(x0)(x-x0) while f’(x0) = f(x1)–f(x0) / (x1–x0)

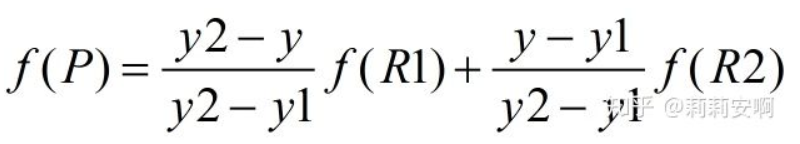
双线性插值



已知Q11、Q12、Q21、Q22四点的坐标，要求点P的坐标。分成两步，首先在 x 方向进行线性插值



然后在 y 方向进行线性插值



最终得：

### 直方图

在分析图像、物体和视频信息的过程中，我们常常把眼中看到的物体用直方图表示。**直方图可以用来描述物体的色彩分布、物体边缘梯度模板、以及表示目标位置的当前假设的概率分布等**

在每帧中，从输入的视频中检测感兴趣的色彩区域，然后计算这些感兴趣区域周围的边缘梯度方向，将得到的边缘梯度方向放到一个方向直方图相应的bin中，然后将该直方图与手势模板进行匹配，从而识别出各种手势

通过标记帧与帧之间显著的边缘和颜色的统计变化，直方图被用来检测视频中场景的变换。边缘、色彩、角等直方图构成了可以被传递给目标识别分类器的一个通用特征类型。

直方图的正确性依赖于网格大小：如果网格太宽，则直方图统计中有太多的空间平均，如果网格太窄，则因太小的平均产生尖锐和单个效果

**求直方图相似度？（直方图匹配）**

在对比直方图之前，都应该自行进行归一化操作。

相关，卡方，直方图相交以及Bhattacharyya距离

当光线变化能引起图像颜色值的漂移，尽管这些漂移没有改变颜色直方图的形状，但是这些漂移引起了颜色值位置的变化，从而导致前述匹配策略失效。可以**用直方图的距离测量来代替直方图的匹配策略**：EMD（Earth Mover’s Distance）度量的是怎样将一个直方图的形状转变为另一个直方图的形状

基于直方图的反射投影？

对于归一化直方图模型来说，结果图像可以被解释为一个概率图，它表示目标是否可能出现

cvCalcBackProject：为一个特定像素是否可能是一个特定目标类型的成员（以一个直方图作为该目标类型的模型建模） --计算一个像素是否是一个已知目标的一部分

cvCalcBackProjectPatch：考虑图像子区域以及子区域的特征（比如颜色）直方图，并且想知道子区域特征的直方图是否与模型的直方图匹配 --计算一块区域是否包含已知的目标

一量我们得到了目标图像的概率值，又应怎样利用该图像来找到目标呢？

cvMinMaxLoc()寻找最大值的位置（第一步先平滑）最有可能是目标在图像中的位置

### 特征检测

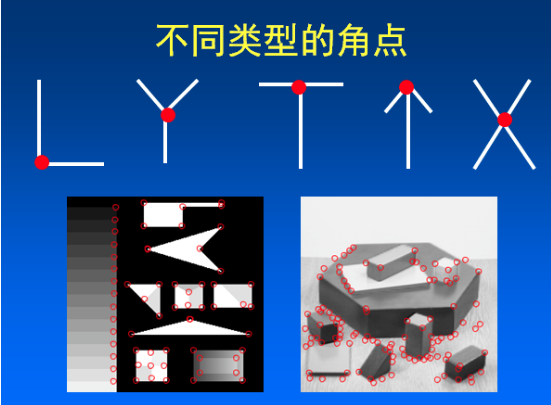
提取图像特征，作为图像描述符，代表图像的内容，常用于：图像搜索，全景图像拼接，图像配准，运动目标跟踪，物体识别，三维重建等

特征就是有意义的图像区域，该区域具有独特性或易于识别性。因此，角点及高密度区域是很好的特征，而大量重复的模式或低密度区域（例如图像中的蓝色天空）则不是很好的特征，常见的特征有：

* Orientations (SIFT, HOG, ...)
* Frequency
* Shapes
* Texture
* Illumination

角点

**如果墙上的所有点都是一样的或者是相似的，我们就不会有太好的运气能在随后的视频帧中跟踪到这个点了。相反，如果选择一个独一无二的点，那么再找到这点的几率就非常大**



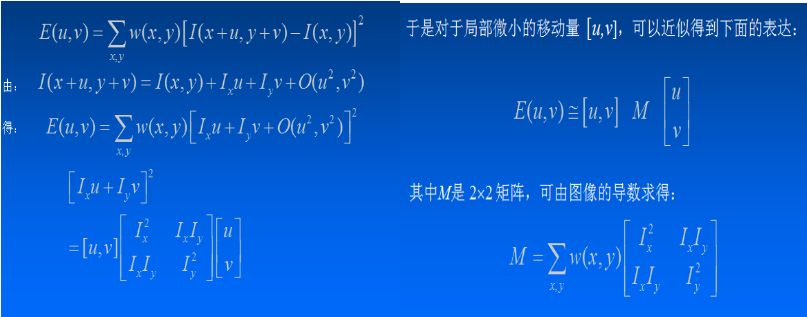
Harris角点

[1988]A Combined Corner and Edge Detector

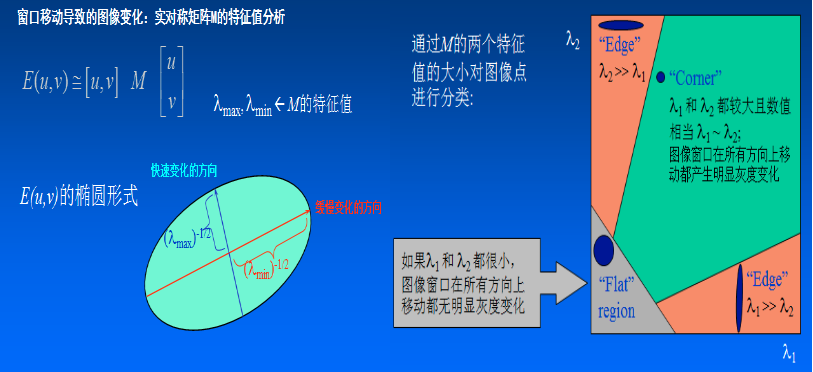
感性判断角点是图像在各个方向灰度有明显变化。

基本原理：利用局部窗口在图像上进行移动判断灰度发生较大的变化

数学模型：计算移动窗口的灰度差值，如下：



注意：M其实就是Hessian矩阵，**通过判断Hessian矩阵特征值的大小，从而判定像素的属性**



cv.cornerHarris(gray, 2, 23, 0.04)

23定义了Sobel算子的中孔。简单地说，该参数定义了角点检测的敏感度，其取值必须是介于3和31之间的奇数

**SIFT**: scale-invariant feature transform (检测斑点blob) 1999年提出2004年完善总结

[2004]Distinctive image features from scale-invariant keypoints

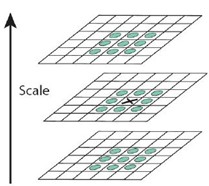
图像的局部特征，对旋转、尺度缩放、亮度变化保持不变

1. **构建DoG尺度空间，寻找关键点(x, y, size)**

大尺度对应图像的概貌特征，小尺度对应图像的细节特征。所以对不同尺度的图像检测关键点，最终得到的sift特征点具有**尺度不变性**

* 使用高斯卷积的形式来表现尺度空间: G(x,y,size) \* I(x,y); G为高斯函数，size决定图像的平滑程度
* 寻找尺度空间的极值点，每一个采样点要和它所有的相邻点比较，看其是否比它的图像域和尺度域的相邻点大或者小

尺度空间的极值点代表这样一类点，如：角点、暗区域的亮点以及亮区域的黑点,边缘点...



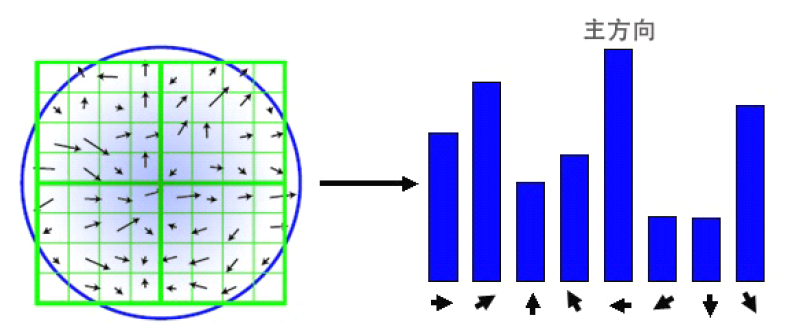
* 剔除不稳定的特征点，比如边缘响应点

因为一个物体的边缘在不同的图像中或者在同一副图像中都可能会有变化。E.g. 一个正方形，在一幅图像中可以是两条水平线以及两条垂直的线组成，而在另一幅图像中，可以是有角度的旋转，类似于普通菱形。而其实它们都是同一个图像，如果利用边缘去做识别的话，因为4条边完全不一样，那就有可能识别错误。所以我们需要将这些边缘特征尽可能的删除，留下最具代表性的角上的点

在边缘梯度的方向上主曲率值比较大，而沿着边缘方向则主曲率值较小，候选特征点的DoG函数D(x)的主曲率与2×2Hessian矩阵H的特征值成正比。可以通过hessian矩阵的大小特征值比例过滤

1. **求取keypoint的主方向 {(x,y,size,t1), (x,y,size,t2)...}**

* 关键点子领域，高斯加权（使关键点附近的梯度幅值有较大权重），计算梯度幅度和方向
* 统计梯度方向直方图：横轴是梯度方向角，纵轴是剃度方向角对应的梯度幅值累加值。梯度方向直方图将0°~360°的范围分为36个柱，每10°为一个柱。



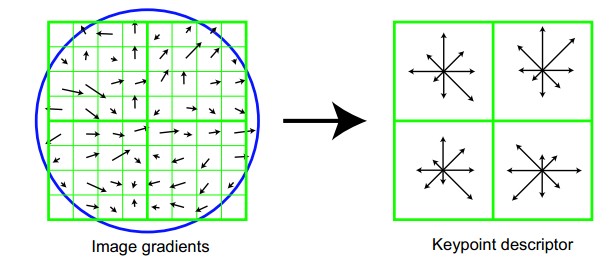
直方图峰值代表该关键点邻域内图像梯度的主方向，当存在另一个相当于主峰值 80%能量的峰值时，则认为这个方向是该关键点的辅方向。所以一个关键点可能检测得到多个方向，这可以增强匹配的鲁棒性。Lowe的论文指出大概有15%关键点具有多方向，但这些点对匹配的稳定性至为关键。具有多个方向的关键点可以复制成多份，然后将方向值分别赋给复制后的关键点。

* 为了使**sift特征点具有旋转不变性**，要以特征点为中心，在附近邻域内旋转θ角，即旋转为特征点的方向。

IMG_256

1. **生成归一化的特征描述子**

旋转后以主方向为中心取 8×8的窗口。下图所示，左图的中央为当前关键点的位置，每个小格代表为关键点邻域所在尺度空间的一个像素，求取每个像素的梯度幅值与梯度方向，箭头方向代表该像素的梯度方向，长度代表梯度幅值，然后利用高斯窗口对其进行加权运算。最后在每个4×4的小块上绘制8个方向的梯度直方图，计算每个梯度方向的累加值，即可形成一个种子点，如右图所示。每个特征点由4个种子点组成，每个种子点有8个方向的向量信息。这种**邻域方向性信息联合增强了算法的抗噪声能力**，同时对于含有定位误差的特征匹配也提供了比较理性的容错性。



对每个关键点使用4×4个种子点，每个种子点有8个方向，则一个关键点产生4\*4\*8=128维的SIFT特征向量。

1. **对特征向量进行归一化处理，去除光照变化的影响**

sift = cv.xfeatures2d.SIFT\_create()

keypoints, descriptor = sift.detectAndCompute(gray, None)

img = cv.drawKeypoints(image=img, outImage=img, keypoints, …)

SIFT对象会使用DoG检测关键点，并且对每个关键点周围的区域计算特征向量。

关键点的属性如下：

* pt表示关键点的x坐标和y坐标
* size属性表示特征的直径
* angle属性表示特征的方向
* response属性表示关键点的强度

SURF （检测斑点blob） 2006年发表

fd = cv.xfeature2d.SURF\_create(…)

keypoints, descriptor = fd.detectAndCompute(gray, …)

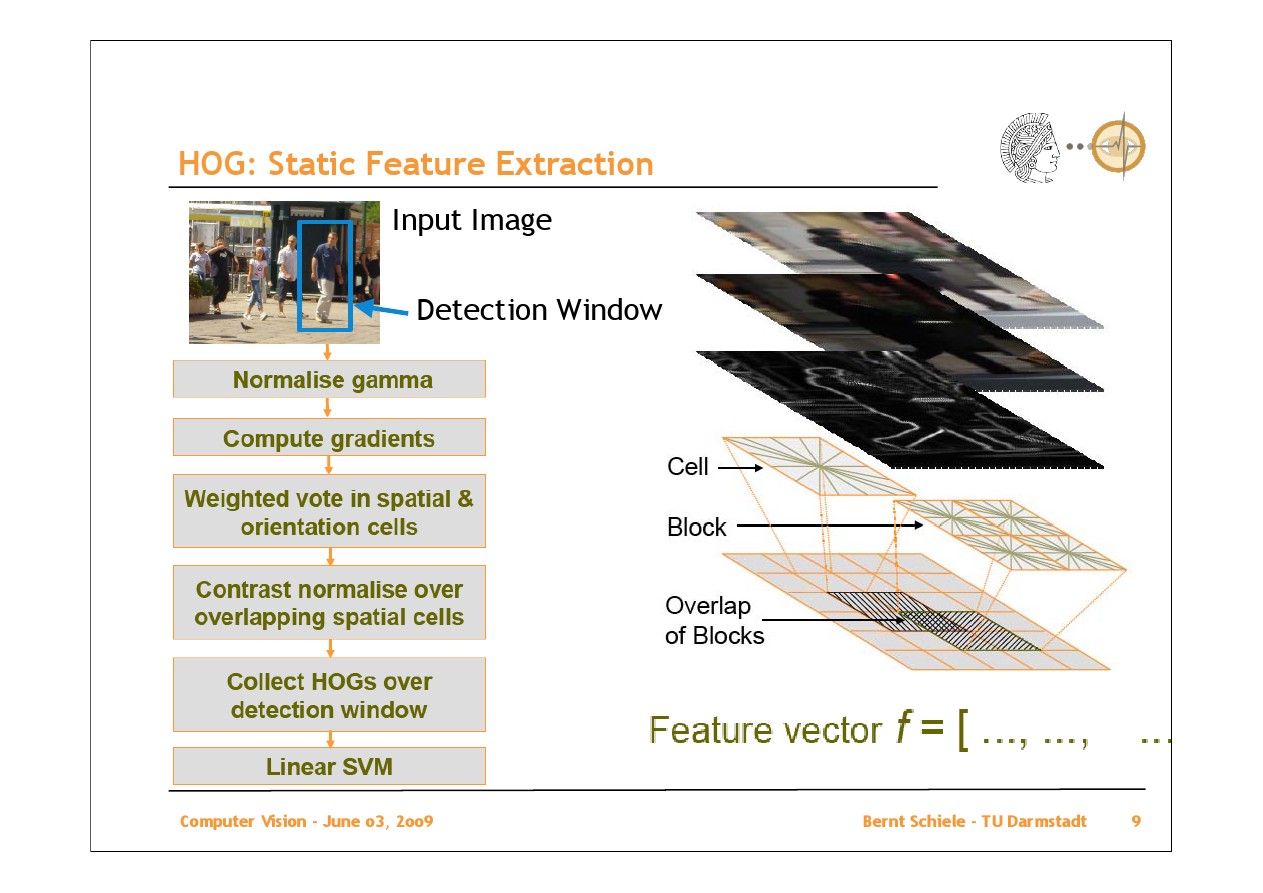
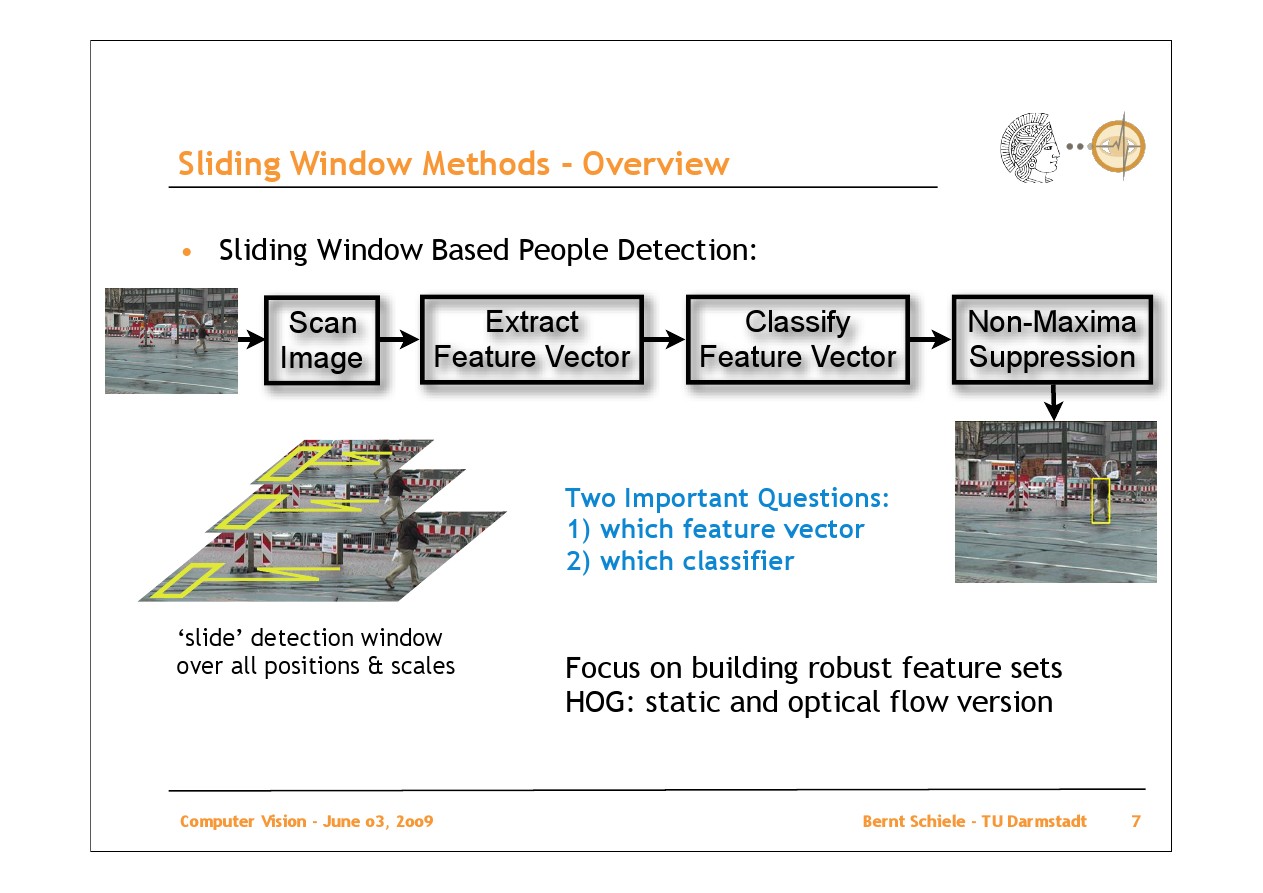
ORG （检测角点和斑点） 2011年发表

ORG将基于FAST（Features from Accelerated Segment Test）关键点检测的技术和基于BRIEF(Binary Robust Independent Elementary Features)描述符的技术相结合

HOG (histogram of oriented gradient)图像梯度方向直方图 2005 CVPR

[2005]Histograms of Oriented Gradients for Human Detection

应用于人体目标检测，使用HOG特征来表达人体，提取人体的外形信息和运动信息，形成丰富的特征集



基本观点是：**局部目标的外表和形状可以被局部梯度或边缘方向的分布很好的描述**，即使我们不知道对应的梯度和边缘的位置

检测窗口大小：64\*128, 1 block = 2\*2 cell, 1 cell = 8\*8 pixels, 块步长8 pixels,

HOG生成步骤：

1. 图像归一化，计算图像梯度
2. 统计cell的方向梯度直方图

检测窗口划分为若干block(有重叠), block划分若干cell，cell由若干像素点组成

梯度强度：M(x, y) = |Gx| + |Gy|

梯度方向：theta(x, y) = arctan(Gx / Gy)

统计cell的方向梯度直方图 （方向bin在0-180均分，幅度作为权重）

为什么不在0-360(有符号梯度）均分？因为对于人体检测来说，衣服和背景颜色的多变可能使得梯度符号信息无意义，但对于其他目标检测，如汽车，摩托车是有用的

1. HOG归一化的特征向量

对直方图进行对比度归一化（注意是针对block归一化块内的所有cell）

由于局部光照的变化，以及前景背景对比度的变化，使得梯度强度的变化非常大，需要对梯度做局部对比度归一化。

**检测窗口内所有块内的cell的直方图构成的向量作为最终描述子**

数据集和研究方法

1239个行人图片以及他们的左右翻转图作为训练的正样本，所以总共2478个正样本。

从1218个没有行人的图片中随机截取12180个检测窗口大小的子图作为初始的负样本。

用正负样本训练一个初始的分类器，然后用初始分类器在负样本原图上进行行人检测，检测出来的矩形区域自然都是分类错误的负样本，这就是所谓的难例(hard examples）。然后，**把误报的负样本(难例)集加入到初始的负样本集中，重新训练**，生成最终的分类器。这种二次训练的处理过程显著提高了每个检测器的表现(在我们的默认检测器中使每个窗口的误报率(FPPW False Positives Per Window)下降了5%)。

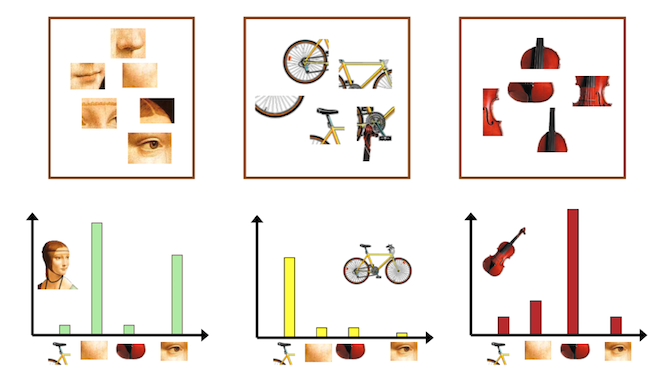
讨论：

最重要的细胞单元是包含主要的**人体轮廓(特别是头部、肩部和足部**)的那些，用这些细胞单元相对于轮廓外的块进行归一化。也就是说，不管训练图片中的背景如何复杂，**检测器检测的主要是人体轮廓相对于背景的差异，而不是内部的边缘或轮廓相对于前景的差异**。衣服上的图案和身体姿势的变化使得人体轮廓内部的区域不适合作为可靠的检测特征，而且前景到轮廓的过度可能由于平滑阴影而混淆

**HOG和SIFT特征有个优点，它们提取的边缘和梯度特征能很好的抓住局部形状的特点，并且由于是在局部进行提取，所以对几何和光学变化都有很好的不变性**：变换或旋转对于足够小的区域影响很小。对于人体检测，在粗糙的空域采样(coarse spatial sampling)、精细的方向采样(fine orientationsampling)和较强的局部光学归一化(stronglocal photometric normalization)这些条件下，只要行人大体上能够保持直立的姿势，就容许有一些细微的肢体动作，这些细微的动作可以被忽略而不影响检测效果。

词袋描述子Bag of Feature 由浅层特征 -> 高层特征

借鉴了文本分类的思路，**从图像抽象出很多具有代表性的「关键词」**，形成一个字典，再统计每张图片中出现的「关键词」数量，得到图片的特征向量



文本分类:Bag of Words 模型

一般来讲，如果我们要了解一段文本的主要内容，最行之有效的策略是**抓取文本中的关键词，根据关键词出现的频率确定这段文本的中心思想**。比如：如果一则新闻中经常出现「iraq」、「terrorists」，那么，我们可以认为这则新闻应该跟伊拉克的恐怖主义有关。而如果一则新闻中出现较多的关键词是「soviet」、「cuba」，我们又可以猜测这则新闻是关于冷战的

**关键词，就是「Bag of words」中的 words ，它们是区分度较高的单词**。根据这些 words ，我们可以很快地识别出文章的内容，并快速地对文章进行分类

训练集 -> 字典

首先我们要找到图像中的关键词，而且这些关键词必须具备较高的区分度。实际过程中，通常会采用「SIFT」特征。

从大量的训练集中提取关键词特征，然后通过聚类算法获取聚类中心，从而形成字典codebook

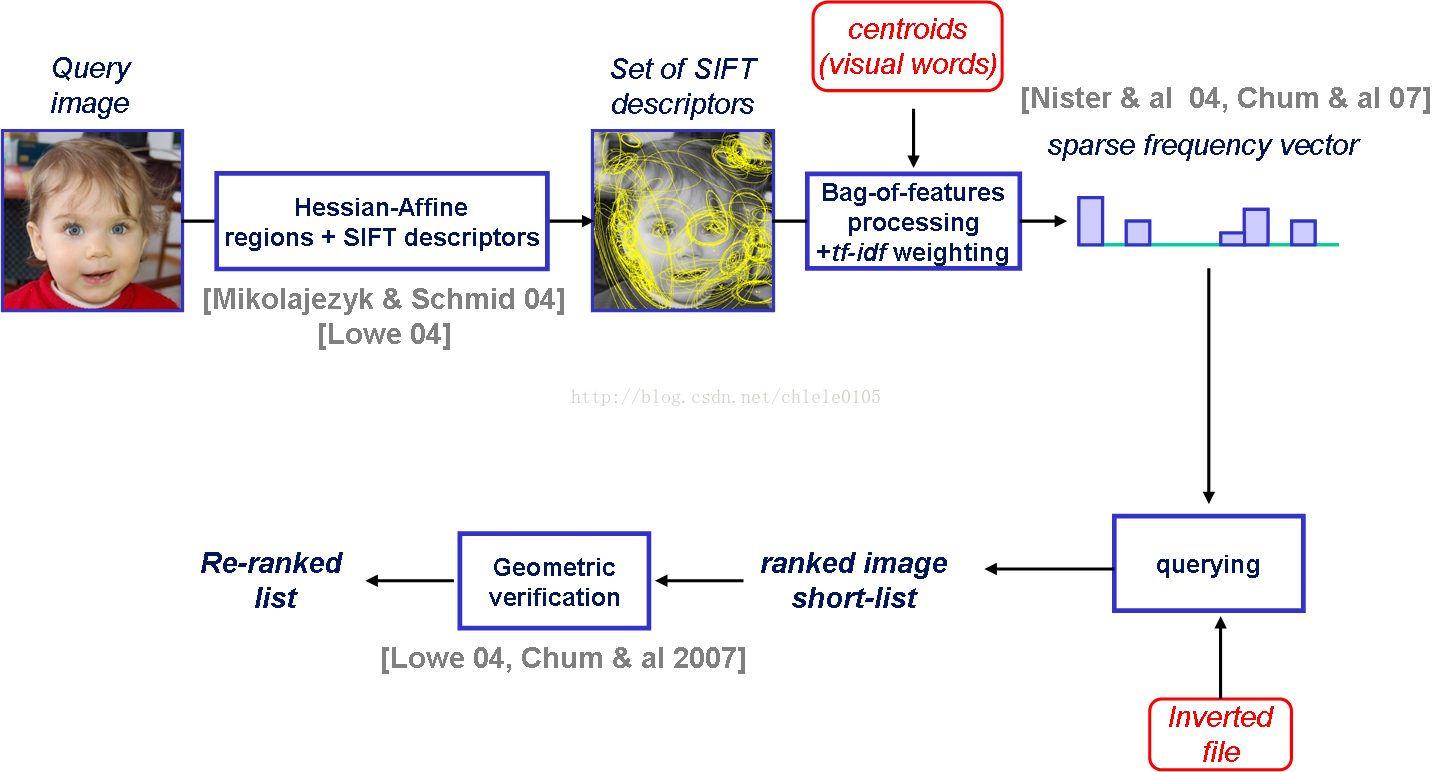
图像 -> bag of feature:

**提取关键词特征，找最相似的聚类中心**，统计聚类中心出现的次数作为直方图即bag of feature

bag of feature常作为图像描述子，进行图片分类和检索

(图像,label) -> (bag of feature, label) -> 分类器

图像检索



1. 图片的直方图: h={hj, j=0,1,…,k}
2. 加权 BOF

借鉴TF-IDF思路：检索某篇文章，先将文章按单词统计出现次数的比例（即词频term-frequency)，TF越高，文章的相关性就越强；同时要考虑词的信息量，比如停止词「的」对主题的检索几乎没有作用。专业词「原子能」比通用词「应用」含有更多的信息. 信息量log(1/p)，p = 单词出现的文章数/文章总数， 称为IDF: Inverse Document Frequency

从而相关性公式： TF1\*IDF1 + TF2\*IDF2 + ...

给字典里的每个向量（visual word）设置权重。

权重的计算方法如出一辙：IDF = log(N/fj)

其中N 是图片总数，fj 表示字典向量 j 在 多少张图片上出现过

可以这样理解：假设我们要检索汽车图片，而汽车一般是放在地面上的，也就是说，在众多类似图片中，地面对应的 visual word 应该会经常出现，而这种特征对于我们检索汽车而言是没有帮助的

加权 BOF: hj=(hj/∑hi)log(N/fj) 公式左边是TF， 公式右边是IDF

1. 相似性度量： 向量之间的夹角(余弦定理) cos(x1, x2) = <x1, x2> / ||x1||\*||x2||

特点：

对于不同类别的图片，bag of feature具有较大的区分度

字典大小的选择是个问题：**字典过大，单词缺乏一般性，对噪声敏感且计算量大；字典太小，单词区分性能差且对相似的目标特征无法表示**

将图像表示成一个无序局部特征集的特征包方法，丢掉了所有的关于空间特征布局的信息

光流：

* 稠密光流dense optical flow：可以将图像中的每个像素与速度关联，或者等价地，与表示**像素在连续两帧之间的位移关系**。
* 稀疏光流sparse optical flow: 稀疏光流的计算需要在被跟踪之前指定一组点。如果这些点具有明显的特征，如角点，斑点，那么跟踪就会相对稳定和可靠

Lucas-Kanade算法： 只需要每个感兴趣点周围小窗口的局部信息。三个基本假设：

* **亮度恒定**（假设像素被逐帧跟踪时其亮度不发生变化）；
* **时间连续**或者运动是“小运动”（运动相对于帧率是缓慢的）；
* **相邻的点保持相邻**（一个场景中同一表面上邻近的点具有相似的运动

对于大多数30Hz的摄像机，大而不连贯的运动是普遍存在的，所以Lucas-Kanade光流正因为这个原因在实际中的跟踪效果交不是很好，我们需要一个**大的窗口来捕获大的运动，而大窗口往往会违背运动连贯的假设**！图像金字塔可以解决这个问题，即最初在较大的空间尺度上进行跟踪，再通过对图像金字塔向下直至图像像素的处理来修正初始运动速度的假定

建议的跟踪方法：在图像金字塔的最高层计算光流，用得到的运动估计结果作为下一层金字塔的起始点，重复这个过程直至到达金字塔的最底层

### 模板匹配

通过在输入图像上滑动图像块对实际的图像块和输入图像进行匹配

方法：平方差匹配法、相关匹配法

**一个好的匹配位置附近应该有许多好的匹配位置，因为模板的轻度变化不应该有不同的匹配位置。在寻找最小（对于平方差度量来说）或最大值（对于互相关或互相关系数来说）之前要对结果图像进行平滑操作**

暴力匹配： 比较两个描述符，并产生匹配结果的列表。遍历**两个描述符的距离**值

暴力匹配非常简单：遍历描述符，确定描述符是否已经匹配，然后计算匹配质量（距离）并排序，这样就可以在一定置信度下显示前n个匹配，以此得到哪两幅图像是匹配的

org = cv.ORB\_create()

kp1, des1 = orb.detectAndCompute(img1, …)

kp2, des2 = orb.detectAndCompute(img2, …)

bf = cv.BFMatcher(cv.NORM\_HAMMING, …)

matcher = bf.match(des1, des2)

matches = sorted(matches, key = lambda x: x.distance)

img = cv.drawMatches(img, kp1, img2, kp2, matches[:40]…)

K-最近邻匹配

matches = bf.knnMatch(des1, des2, k=2) 返回k个匹配

img3 = cv.drawMatchesKnn(img1, kp1, img2, kp2, matches, …)

FLANN匹配：Fast Library for Approximate Nearest Neighbors近似最近邻的快速库

flann = cv.FlannBaseMatcher(…)

matches = flann.knnMatch(des1, des2, k=2)

单应性匹配

### 图像分割

金字塔分割：cvPyrSegmentation

分水岭算法:

把图像低密度区域想象成山谷，图像高密度区域想象成山峰。开始向山谷中注入直至不同的山谷中的水开始汇聚。为了阻止不同山谷的水汇聚，可以设置一些栅栏，最后得到的栅栏就是图像分割

cv.watershed(img, markers)

GraphCut:

cv.grabCut(…)

均值漂移分割: cvPyrMeanShiftFiltering

### 图像修复

**利用已被破坏区域的边缘的颜色和结构**，繁殖和混合到损坏的图像里面。如果被破坏区域并不太大，并且在被破坏区域边缘包含足够多的纹理和颜色，那么Inpainting可以很好地恢复图像。

cvInpaint

### 轮廓

OpenCV允许得到的轮廓被聚合成一个轮廓树，从而把包含关系编码到树结构中

轮廓参数：

* 轮廓类型（检测最外的轮廓，检测所有的轮廓并保存成list, hierarchy, or tree）
* 轮廓如何被近似

轮廓表示有两种：

* 由一系列顶点的序列表示
* Freeman链码表示（多边形被表示为一系列的位移，每一个位移用8个方向，Freeman链码对于识别一些形状的物体很有帮助）

轮廓的多边形逼近cvApproxPoly，长度cvContourPerimeter，面积cv.contourArea，矩形框cvBoundingRect，最小外接矩形cvMinAreaRect2, 拟合椭圆cvFitEllipse2, 凸包cvConvexHull2, 凸缺陷cvConvexityDefects

求**轮廓的相似度？（轮廓匹配）**

**比较轮廓矩** （通过对轮廓上所有点进行积分运算而得到的特征）

简单的矩依赖于所选坐标系，这意味着物体旋转后就无法正确匹配

Hu不变矩: 归一化中心距的线性组合。之所以这样做是为了能够获取代表图像某个特征的矩函数，**这些矩函数对某些变化如缩放、旋转和镜像映射具有不变性**

使用Hu矩进行轮廓匹配: cvMatchShapes

**Freeman链码比对物体？**

Freeman链码编码：是对一个多边形序列如何“移动”的描述，每个这样的移动有固定的长度和特定的方向。

链码编码直方图chain code histogram：用来统计一个轮廓的Freeman链码编码每一种走法的数字，具有良好的性质，如将物体旋转45度，那么新的直方图是老直方图的循环平移。这就提供了一个不被此类旋转影响的形状识别

img, contours, hierarchy = cv.findContours(edge, cv.RETR\_TREE, cv.CHAIN\_APPROX\_SIMPLE)

cv.drawContours(color\_image, contours, -1, (0, 255, 0), 2)

for c in contours:

(x, y, w, h) = cv.boundingRect(c)

cv.rectangle(img, (x, y), (x+w, y+h), ...)

rect = cv.minAreaRect(c)

box = cv.boxPoints(rect)

box = np.int0(box)

cv.drawContours(img, [box], ...)

(x, y), radius = cv.minEnclosingCircle(c)

cv.circle(img, (int(x), int(y)), radius, ...)

eps = 0.01\*cv.arcLength(cnt, True)

approx = cv.approxPoly(ent, eps, True) 近似多边形

hull = cv.convexHull(cnt)

### 视频分割

从图像中分割出前景目标

“背景”在不同的应用场合下是一个很难定义的问题。例如，若正在观测一条高速公路，那么或许平均流动的车流应该被认为是背景。通常情况下，背景被认为是在任何所感兴趣的时期内，场景中保持静止或周期运行的目标。

高级的场景建模：要对前景状态和背景状态定义多重指标，以时间为基础将不变的前景模型缓慢转换为背景模块。当场景完全发生变化时我们还必须检测并建立一个新的模型。

通常，一个场景模型可能包含许多层次，从“新的前景”到旧的背景再到背景。一个新的前景目标就会放进“新前景”目标级别，标识一个真目标或一个空洞。在没有任何前景物体的地方，我们将继续更新我们的背景模型。如果一个前景物体在给定的时间内没有发生移动，就将它降级为“旧的前景”

**方法1：帧差**

用前一帧减去当前帧，然后 将足够大的差别标为前景（作阈值化处理），忽略像素值受到噪声和波动的影响（作形态学处理），连通域法清理离散噪声（因为噪声不会有很大的空间相关性，这些信号有大量的非常小的区域来描述）

**方法 2: 计算每个像素的平均值和标准差作为它的背景模型**

以上简单的方法只能用于背景场景中不包含运行的部分（比如摆动的窗帘和在风中摇曳的树）。而且这种方法还要求光线保持不变（如在室内静止的场景）。

更高级的背景建模：

很多背景场景都包含复杂的运动目标，诸如摇曳在风中的树，转动的风扇，摆的窗帘等，通常这样的场景中还包含光线的变化。比如云彩掠过，门窗中照进来不同的光线。

解决这种问题的转好方法是得到每个像素或一组像素的时间序列模型。

**方法3: 均值漂移**

acc(x, y) = (1-alpha)\*acc(x, y) + alpha\*image(x, y) if mask(x, y) != 0

可以作为跟踪器，给予最近值较大的权值，解释：

(2, 3, 4)，均值为3，而均值漂移值为3.25

**方法4: codebook方法 （具体见OpenCV书）**

Codebook不能很好处理不同模式的光（如早晨、中午和傍晚的阳光，或在室内有人打开和熄灭灯）

方法5: 背景分割器

KNN:

MOG2: Mixture of Gaussians

GMG: Geometric Multigid

bs = cv.createBackgroundSubtractorKNN(detectShadows=True)

# bs = cv.createBackgroundSubtractorMOG2()

while(1):

ret, frame = cap.read()

fgmask = bs.apply(frame)

BackgroundSubtractor类是专门用于视频分析的，即BackgroundSubtractor类会对每帧的环境进行学习，例如可用GMG来指定用于初始化视频分析的帧数，默认为120帧。BackgroundSutractor类常用来对不同帧进行比较，并存储以前的帧，可按时间推移方法来提高运动分析的结果

### 目标检测与识别

在图像和视频处理中常常会进行目标检测。流程如下：

图像滑动窗口，计算窗口内的特征描述符（比如梯度直方图Histogram of Oriented Gradient），确定窗口的评分（训练分类系统如svm，对这种分类会有一个置信度评分）

解决尺度问题：图像金字塔(image pyramid)

解决位置问题：滑动窗口(sliding window)

解决检测出的区域重叠问题(overlapping region): NMS（非最大抑制）

对这些矩形按评分进行排序，从评分最高的矩形开始，消除所有重叠超过一定阈值的矩形，消除的规则是计算相交的区域，并看这些相交区域是否大于某一阈值

BOW(bag of words词袋)：统计在一系列文档中计算每个词出现的次数，然后用这些次数构成向量来重新表示文档，这些向量可以看成是文档的直方图表示或被当作特征，这些特征可用来训练分类器

BOVW(bag of visual words计算机视觉中的BOW):

取一个样本数据集

对数据集中的每幅图像提取描述符（如SIFT, SURF）

将每个描述符都添加到BOW训练器中

将描述符聚类到k簇中（聚类的中心就是视觉单词）

以汽车检测为例：具体见《opencv3计算机视觉》第7章目标检测与识别

1. 获取一个训练数据集
2. 创建BOW训练器并获得视觉词汇
3. 采用词汇训练SVM
4. 尝试对测试图像的图像金字塔采用滑动窗口进行检测
5. 对重叠的矩阵使用非最大抑制

def detector():

flann = cv.FlannBasedMatcher(...)

detect = cv.xfeatures2d.SIFT\_create()

extract = cv.xfeatures2d.SIFT\_create()

bow\_kmeans\_trainer = cv.BOWKMeansTrainer(40)

extract\_bow = cv.BOWImgDescriptorExtractor(extract, flann)

bow\_kmeans\_trainer.add(extract.compute(img))

...

voc = bow\_kmeans\_trainer.cluster()

extract\_bow.setVocabulary(voc) 获取视觉单词字典

提取样本集的BOW特征作为训练集

traindata.extend(extract\_bow.compute(img, detect.detect(img))

trainlabels.append(l or -1)

svm = cv.ml.SVM\_create()

svm.train(traindata, trainlabels, ...)

return svm, extract\_bow

svm, extractor = detector()

detect = cv.xfeatures2d.SIFT\_create()

for resized in pyramid(img, scaleFactor):

for (x, y, roi) in sliding\_window(resized, 20, (w, h)):

result = svm.predict(extractor.compute(img, detect.detect(img))

rectangles.append(…)

boxes = nms(rectangles, …)

人脸检测

**Haar分类器：AdaBoosting筛选式级联分类器，每个节点是多个树构成的分类器**，且每个节点的正确识别率很高（如99.9%，也就是很低的错误拒绝率，一般不会把人脸丢掉），但正确拒绝率很低（接近50%,也就是高的错误接收率，很多非人脸不会被检测出来）。优点是：当目标出现频率较低的时候（例如一幅大图里只有一幅小人脸），筛选式级联分类器可以显著地降低计算量，因为大部分被检测的区域都可以很早被筛选掉，迅速判断出此处无人脸。

**在检测人脸的时候，几乎所有的人脸99.9%都被检测出并允许通过，但是50%的非人脸也得以通过。这没关系，因为20个节点使总识别率为0.99920 = 98%, 而错误接收率仅为0.520 = 0.0001%**

非常擅长检测特定视角的刚性物体。

级联分类器 ------弱分类器

-----弱分类器

-----…

-----弱分类器(AdaBoost分类器) --------10个节点左右的决策树

--------10个节点左右的决策树

--------…

**Haar分类器不限于人脸检测，还适用于其他外表有区别的（接近刚性的）物体的检测如正面人脸、车的前部、侧部和后部**都可以用它来检测

**侧脸很难用Haar分类器，因为此分类器使用块特征，Haar小波，侧脸边缘外的背景也会被当作有用信息进行学习**

Haar特征的识别器适用于固有特征如眼睛，嘴，发际线，不适用于树枝或者外形有区别的coffee杯之类

face\_cascade = cv.CascadeClassifier(‘frontalface\_default.xml’)

eye\_cascade = cv.CascadeClassifier(‘haarcascade\_eye.xml’)

faces = face\_cascade.detectMultiScale(gray, 1.3, 5)

for (x, y, w, h) in faces:

img = cv.rectangle(gray, (x, y), (x+w, y+h), (255, 0, 0), 2)

roi\_face = gray[y+y+h, x:x+w]

eye = eye\_cascade.detectMultiScale(roi\_face, …)

for (ex, ey, ew, eh) in eyes:

…

人脸识别

先人脸检测，裁剪灰度帧的区域，将其大小调整为200\*200像素

OpenCV人脸识别方法：Eigenfaces, Fisherfaces, Local Binary Pattern Histogram, deep learning.

Eigenfaces and Fisherfaces的基础是PCA, PCA本质是识别某个训练集上（比如人脸数据库）的主成分，并计算出训练集（图像或帧中检测到的人脸）相对于数据库的发散程度，并输出一个值。该值越小，表明人脸数据库和检测到的人脸之间的差别就越小； 0值表示完全匹配

model = cv.face.createEigenFaceRecognizer()

model.train(X, y)

model.predict(roi\_face)

cv.putText(img, name, (x, y-20), …)

### 目标跟踪

当跟踪所有移动目标时，帧之间的差异会变得有用； 当跟踪视频中移动的手时，基于皮肤颜色的均值漂移方法是最好的解决方案； 当知道跟踪对象的一方面时，模板匹配会是不错的技术。

* **mean-shift**: 在一组数据的密度分布中寻找局部极值的稳定的方法

步骤：

1. 选择搜索窗口
2. 计算窗口（可能带权重）的重心
3. 将窗口的中心设置在计算出的重心处
4. 重复2~3,直到窗口的位置不再变化

收敛的位置在窗口中像素分布的局部最大值（峰值）处

cv.meanShift: 需要首先选择代表物体的特征的分布（例如，颜色+纹理），然后在物体的特征分布上开始mean-shift窗口搜索，最后计算下一帧视频中所选择的特征的分布。从当前的窗口位置开始，mean-shift算法寻找特征分布的新的峰值，它们被假定设置在最初始产生颜色和纹理的物体的中心。这样mean-shift窗口就可以逐帧跟踪物体的运动

均值漂移meanshift是一种目标跟踪算法，**该算法寻找概率函数离散样本的最大密度（例如，感兴趣的图像区域），并且重新计算在下一帧中的最大密度，该算法给出了目标的移动方向**

roi = (x, y, w, h) 初始化跟踪窗口，最好是检测到的ROI

# 图像的颜色分布

roi\_hist = cv.calcHist([hsv\_img], [0], roi, ...)

cv.normalize(roi\_hist, ...)

while True:

ret, frame = cap.read()

# calc histogram back projection直方图反向投影，图像的每个像素属于起初那幅生成直方图的图像的概率。或者说，一幅图像类似于模型图像（产生原始直方图的图像）的概率

dst = cv.calcBackProject([hsv\_img], roi\_hist, ...)

ret, roi = cv.meanShift(dst, roi, ...) 获取新的roi

均值漂移存在一个问题：窗口大小并不与被跟踪帧中的目标大小一起变化

* **camShift**: continuously adaptive meanshift 连续自适应均值漂移 发表于1988

搜索窗口会自我调整尺寸，此算法可以根据物体靠近或远离摄像机时的尺寸而自动调整窗口的尺寸

while True:

ret, frame = cap.read()

# calc histogram back projection直方图反向投影，图像的每个像素属于起初那幅生成直方图的图像的概率。或者说，一幅图像类似于模型图像（产生原始直方图的图像）的概率

dst = cv.calcBackProject([hsv\_img], roi\_hist, ...)

ret, roi = cv.CamShift(dst, roi, ...) 获取新的roi

pts = cv.boxPoints(ret)

pts = np.int0(pts)

img = cv.polylines(frame, [pts]...)

* **运动模板**： motion template

应用在姿态识别中，需要知道物体的轮廓或者轮廓的一部分

* **Kalman滤波器**

预测: Kalman Filter使用由当前点计算的协方差来估计目标的新位置

校正： Kalman Filter记录目标的位置，并为下一次循环计算修正协方差

**Kalman滤波器依赖于线性动态性和Markov独立性**（也就是它假设当前状态只依赖于刚过去的状态而不是所有过去的状态）

Delaunay三角剖分和Voronoi划分

将空间点连接为三角形，使得所有三角形中最小的角最大的一个技术。也就是说Delaunay三角剖分力图避免出现瘦长三角形，同时任何三角形的外接圆都不包含任何其他顶点，这叫外接圆性质。

应用：

通过Delaunay三角剖分，我们可以直接找到一组点的外部轮廓

运用原始点的Delaunay三角测量，你可以快速搜索新点的最近邻居

Voronoi划分是Delaunay三角剖分的对偶图像，任何包含在Voronoi单元中的点都比其他Delaunay点更接近于它们自己的Delaunay点

### 摄像机模型与标定

摄像机标定camera calibration,来矫正（数字方式）因使用透镜而给针孔模型带来的的主要偏差

摄像机标定的过程既给出摄像机几何模型，也给出透镜的畸变模型。

投影变换：空间坐标点投影到摄像机上

“筒形”或“鱼眼”畸变：实际摄像机的透镜总是在成像仪的边缘产生显著的畸变。对某些透镜，光线在远离透镜中心的地方比靠近中心的地方更加弯曲。对常用的普通透镜来说，这种现象更加严重。鱼眼畸变在便宜的网络摄像机中非常厉害，但在高端摄像机中不明显，因为这些透镜系统做了很多消除鱼眼畸变的工作

标定方法：把摄像机对准一个有很多独立可标识点的物体。通过在不同角度观看这个物体。可以利用通过每个图像计算摄像机的相对位置和方向以及摄像机的内参数

OpenCV常用标定物是一个用不同黑白方块构成的棋盘

手持棋盘以各种方向得到的棋盘图像，以确保为完全求解这些图像在整个坐标系（相对于摄像机）的位置和摄像机内参数提供足够的信息，从而可以计算出每个视场的旋转和平移，同时也计算摄像机的内参数（对所有视场不变）

**求解未知量：摄像机4个内参数和5个畸变参数（对所有视场不变），还有6个外参数含三个旋转参数和三个平移参数（依赖于每个视场）？**

为了得到高质量结果，**至少需要10幅7\*8或者更大棋盘的图像（而且只在移动棋盘在不同图像中足够大以从视场图像中得到更加丰富的信息）**

cvCalibrateCamera2

标定摄像机通常是想做两件事：一个是矫正畸变效应，另一个是根据获得的图像重构三维场景

cvUndistort2: 基于摄像机标定时获得的内参数矩阵和畸变系数

### 机器学习

* 统计性机器学习

Mahalanobis距离：多维空间中两点相似性的度量

通过除以协方差来对数据空间进行变换，然后计算距离。如果协方差矩阵是单位矩阵，那么该度量等价于欧氏距离。（测量距离时要考虑数据的协方差）

可以把**Mahalanobis距离看作多维空间中Z-score**的类似物

举例：如果我们以米为单位来测量人的身高，以天为单位测量人的年龄，我们看到身高的范围很小，而年龄的范围很大。通过方差归一化，变量之间的关系便会更加符合实际情况。

1. **决策树**

一般不具有最优性能，但结果容易解释，适应不同的数据类型（包括类别数据、数值数据、未归一化的和混合的数据），能够处理数据丢失，通过分裂的顺序能够给数据特征赋不同的重要性

算法的要点是给树的每个节点定义一个衡量标准：当我们拟合一个函数的时候，我们使用真实值和预测值的差的平方和。当我们分类时，使得当一个节点的大多数值都属于同一类

一旦我们定义了度量，二叉树搜索整个特征向量，搜寻哪个特征和哪个阈值可以正确分类数据或正确拟合数据。

CvDTree ->train(…), predict(…), get\_var\_importance()

CvDTree->save(…), load(…)

在实际数据中的效果可能并不好，需要包含一个附加的步骤来通过修剪树来达到复杂度和性能的平衡

Boosting and random forest内部使用了决策树，所以继承了树的很多有用的性质（能够处理混合数据类型、没有归一化的数据、特征缺失）

1. **Boosting**

多个判别子分类器的组合，最终的分类决策是由各个子分类器的加权组合来决定。**当逐个训练分类器的时候，数据样本的权重被重新分配，使之能够给予分错的数据更多的注意力**。训练过程不停地执行，直到总错误（加权组合所有决策树组成的分类器产生的错误）低于某个已经设置好的阈值。为了达到好的效果，这个方法通常需要很大数据量的训练数据

1. **随机森林**

**由许多决策树组成的森林，在学习过程中，每棵树的每个节点只从特征数据的一个随机子集中选择**。投票结果多为判别结果；平均结果作为回归结果

随机子集的规模一般是特征数量的开方，为了提高鲁棒性，随机森林使用袋外out of bag方法来检验分裂。给定任何节点，训练是发生在一个随机选择然后替换的数据子集上进行的；没有选到的数据被叫做out of bag数据，将用于估计分裂的性能。

随机森林可以用来确定两个数据样本的亲近度（是相似度，不是距离）：计算它们到达同样的叶子的次数。亲近度可以用来检测异常（样本与其他的很不相似），或者用来聚类（把相似的样本聚在一起）

CvRTrees->train(…), predict(…), get\_var\_importance(), get\_proximity(…)

1. **K近邻**
2. **神经网络**

对于字符识别，具有非常不错的性能

def create\_net(hidden=20):

net = cv.ml.ANN\_MLP\_create()

net.setTrainMethod(...)

net.setActivationFunction(...)

net.setlayerSizes(...)

def train(net, samples, epochs):

tr, val, test = wrap\_data()

for x in xrange(epochs):

for img in tr:

data, class = img

net.train(data.ravel(), y)

return net, test

net.predict(sample)

1. **支持向量机SVM:**

将数据投影到高维空间会使数据更容易地线性可分。**当数据有限的时候，该算法可以获得非常好的性能；而boosting and random forest只能在拥有大量训练数据时才有好的效果**

1. **K均值**

使用K个均值来表示数据的分布，和EM区别是K均值的中心不是高斯分布

无法保证找到定位聚类中心的最佳方案，如何解决？如何指定K？

解决方法：基于“解释数据的方差”。在K均值中，每个聚类中心拥有它的数据点，我们计算这些点的方差，最好的聚类在不引起太大的复杂度的情况下使方差达到最小。

令初始聚类中心点不一样，多运行几次，选择方差最小的那个结果

类别数逐渐增加，选择方差最小的那个结果 （一般情况下，总方差会很快下降，直至到达拐点）

将数据乘以逆协方差矩阵，D\* = D Cov-1/2

1. **EM：期望最大化**

拟合N个多维高斯数据，该方法经常用于分科

**EM算法通过迭代先找到给定模型时的最大可能性的猜想，然后调整模型使猜想正确率最大化**

* 贝叶斯网络

贝叶斯网络是因果模型，脸的存在产生了图像中的脸的特征。在使用中，脸是一个隐含变量，通过对输入图像的处理得到的脸部特征，组成了脸的观测证据。这个就是产生式模式，因为脸生成了脸部特征。它的逆过程，我们假设脸是存在的，然后在脸存在的前提下，随机采样生成了哪些特征。

**条件概率 = 可能性\*先验概率/证据**

p(face | LE, RE, N, M, H) = p(LE, RE, N, M, H | face) \*p(face) / p(LE, RE, N, M, H)

朴素贝叶斯分类器

假设特征是高斯分布而且统计上互相独立。在现实中很少见（如：找到一只眼睛常常意味着另一只眼睛在附近）

一般情况下，如果没有很多的数据，简单的模型（朴素贝叶斯）会比很多复杂模型获得很好的性能，因为复杂模型使用了太多的假设，以致产生欠拟合

* 马尔可夫随机场
* 图模型

### 机器学习涉及的重点

无监督的聚类数据经常形成一个特征向量供更高层的有监督的分类器使用

* 判别式算法：即通过给定数据来判断类别P(L|D)。判别式模型在根据给定的数据做出预测上有优势
* 产生式算法：通过给定类别来生成数据的分布P(D|L)。产生式模型则是在为你提供更强大的数据表达式或者有条件地生成新数据时有优势

特征抽取

必须寻找表达**物体固有属性的特征，比如梯度直方图、色彩、或SIFT特征**。如果有背景信息，可能想首先把背景去除，提取出物体；然后进行图像处理（归一化图像、尺度改变、旋转、直方图均衡），计算很多特征。物体的特征向量将与物体的标签对应

特征选择

哪些特征对分类器的准确性有较大贡献？二进制决策树可以解决这个问题：通过在每个节点选择最能够分裂出数据的变量。最上层的变量是最重要的变量

如果没有足够的数据，去除不重要的变量可以提高分类器的准确率

变量重要性技术度量了每个变量对分类器性能的贡献。哪些特征丢弃后使性能下降越多，哪些特征就越重要

预处理特征

预处理每个特征变量，使它们的方差一致。如果特征不相关，这个步骤很重要；如果特征相关，你可以用它们的协方差或平均方差来归一化。**决策树不受方差不一致的影响，若算法以距离度量为准则，就需要预先将方差归一化，可以利用Mahalanobis距离来归一化特征的方差**

基于树的算法（决策树、随机森林和boosting）支持类别变量和数值变量，大多数算法只支持数值输入，使数值输入的算法能够处理类别数据的常用方法是把类别数据表示成one-hot编码

一般分类器的选择?

需要考虑计算速度、数据形式和内存大小。

* 在线用户优先选择建模，所以分类器需要能够快速完成训练。在这种情况下，最邻近算法、朴素贝叶斯和决策树是不错的选择。
* 如果需要考虑内存因素，决策树和神经网络是理想的选择
* 如果不需要很快训练，而需要很快判断，那么神经网络可以满足要求，朴素Bayes and SVM也不错
* 如果不需要训练很快，但是需要精确度很高，可选择boosting and random forest。
* 如果选择的特征比较好，仅仅需要一个简单易懂的分类器，就选择决策树和最邻近算法

若考虑所有的数据分布类型，所有的分类器是一样的。如果给定某个特定的数据分布，或者特定的数据分布，通常存在一个最好的分类器，所以在实际应用中，最好多尝试一下各种不同的分类器。如果训练数据与测试数据的分布相似，性能预测会更精确

诊断机器学习中的问题？

**将机器学习用好，不仅仅是一门技术，更是一门艺术。算法经常有些时候能用，但又不能完全与要求一致**。重要的规律：

* 大量数据比少量数据好
* 好的特征比好的算法更重要
* 如果训练和测试结果都很好，但是算法在实际应用中效果不好，则表明数据集可能是从非实际条件中获得的

解决欠拟合：使用更多的特征有利于拟合，选用一个学习能力更好的拟合算法

解决过拟合：增加训练数据的数量可使得拟合曲线更光滑；减少特征的数量可降低过拟合程度

在实际情况中，我们必须考虑噪声，采样误差和采样错误。测试集或验证集可能并不能精确地反映数据的实际分布。为了更 准确评估分类器性能。我们可以采用交叉验证cross-validation或自抽样法

ROC曲线评估了分类器参数的变化对分类器性能的影响

如何获取负样本？

任何没有我们感兴趣的物体的图像都可以作为负样本。最好从我们需要**测试的数据中选取负样本图像。即如果我们想从在线视频中学习人脸，最好从视频的不包含人脸的帧中获得负样本**

**“好数据”意味着不应该把倾斜的脸和竖直的脸混在一起，解决方法是训练两个分类器，一个用来判断倾斜，一个用来判断竖直。**

“完美分割“圈定物体的矩形边界要保持一致，如果物体边界四处漂移，那么分类器不得不去学习这些变化

**高质量是指已经把所有不需要的变量从数据中除掉。举个例子，如果你学习人脸，需要尽量对齐眼睛（最好加上鼻子和嘴巴）。除非告诉分类器眼睛不可以移动，否则它会认为眼睛可以出现在任意区域内。但是这样是不符合实际情况的，分类器将会无法取得好的效果。一个策略是首先训练一个容易锁定的子集（如眼睛）的级联。然后使用这个级联来寻找眼睛，可以旋转/改变图像大小直至眼睛被对齐。**

## 拓展知识

### 光流跟踪

<https://blog.csdn.net/zouxy09/article/details/8683859>

<http://vision.middlebury.edu/flow/>

光流就是瞬时速率，在时间间隔很小（比如视频的连续前后两帧之间）时，也等同于目标点的位移

光流是由于场景中前景目标本身的移动、相机的运动，或者两者的共同运动所产生的。

研究光流场的目的就是为了从图片序列中近似得到不能直接得到的运动场

那通俗的讲就是通过一个图片序列，把每张图像中每个像素的运动速度和运动方向找出来就是光流场

第t帧的时候A点的位置是(x1, y1)，那么我们在第t+1帧的时候再找到A点，假如它的位置是(x2,y2)，那么我们就可以确定A点的运动了：(ux, vy) = (x2, y2) - (x1,y1)。

那怎么知道第t+1帧的时候A点的位置呢？ 这就存在很多的光流计算方法

基于梯度的方法、基于匹配的方法、基于能量的方法、基于相位的方法。

calcOpticalFlowPyrLK(): 通过金字塔Lucas-Kanade 光流方法计算某些点集的光流（稀疏光流）

calcOpticalFlowFarneback():用Gunnar Farneback 的算法计算稠密光流（即图像上所有像素点的光流都计算出来

CalcOpticalFlowBM(): 通过块匹配的方法来计算光流

CalcOpticalFlowHS(): 用Horn-Schunck 的算法计算稠密光流

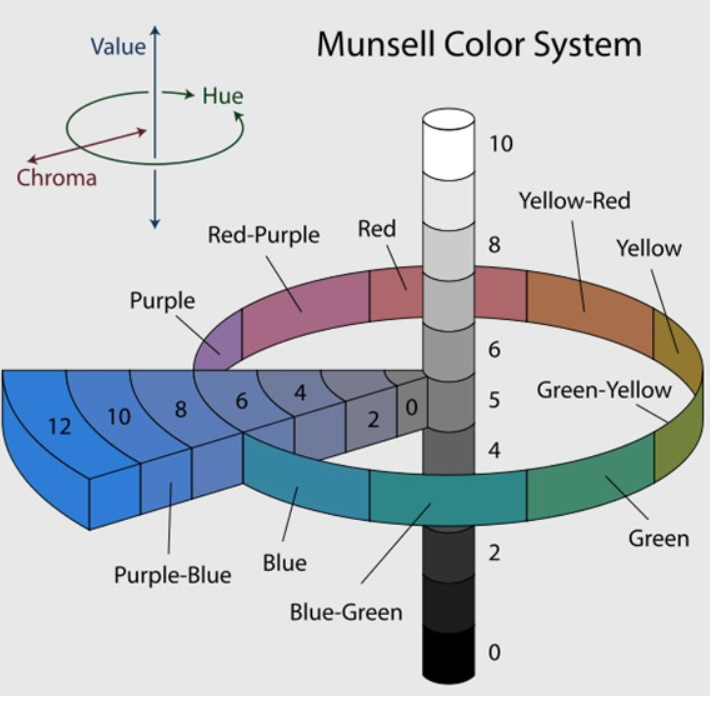
calcOpticalFlowSF():

稠密光流需要使用某种插值方法在比较容易跟踪的像素之间进行插值以解决那些运动不明确的像素，所以它的计算开销是相当大的。

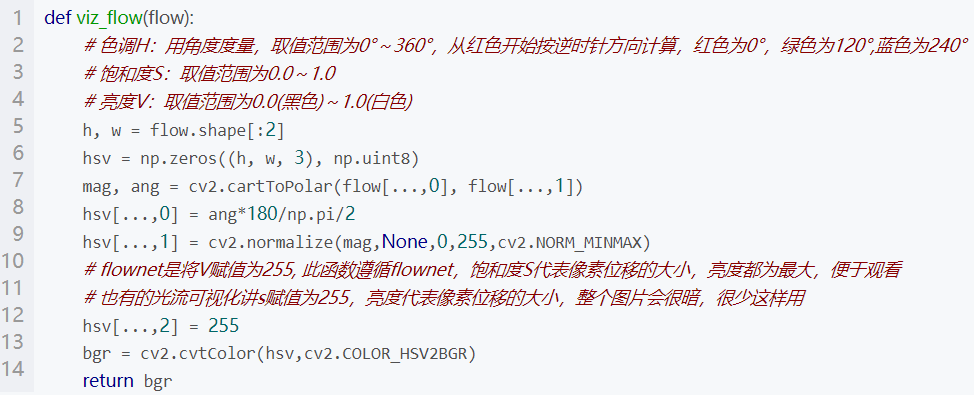
稀疏光流在计算时需要在被跟踪之前指定一组点（容易跟踪的点，例如角点），因此在使用LK方法之前我们需要配合使用cvGoodFeatureToTrack()来寻找角点，然后利用金字塔LK光流算法，对运动进行跟踪。但个人感觉，对于少纹理的目标，例如人手，LK稀疏光流就比较容易跟丢。

光流场可视化

光流场是图片中每个像素都有一个x方向和y方向的位移，所以在上面那些光流计算结束后得到的光流flow是个和原来图像大小相等的双通道图像。不同颜色表示不同的运动方向，深浅表示运动的速度。

将x和y转为极坐标，夹角(actan2(y,x))代表方向，极径(x和y的平方和开根号)代表位移大小，用hsv的图像表示。上图的光流可以看到，红色的人在往右边动，那个蓝色的东西在往左上动

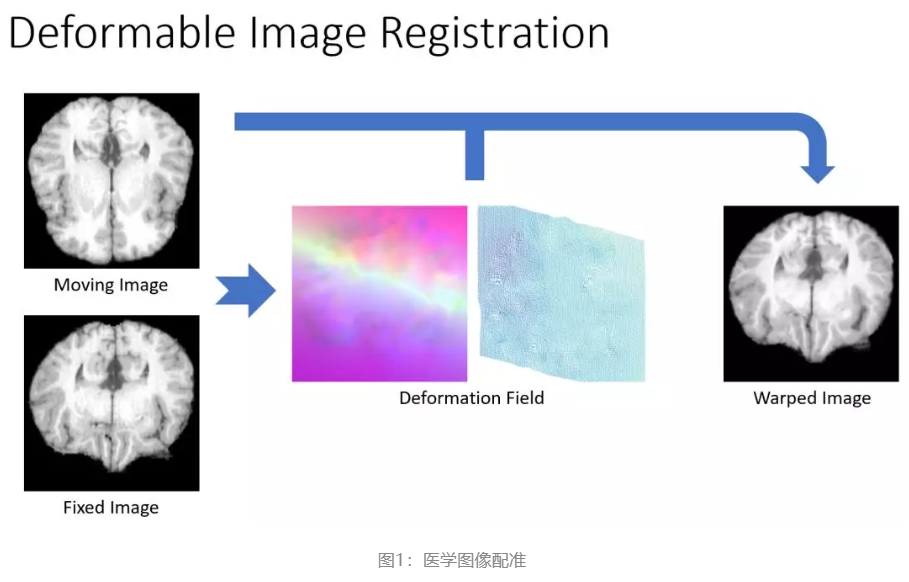


### 医学图像配准

医学图像配准即分别给定一张运动的和固定的 3D 医学图像，希望将运动图像（moving image）配准到固定图像（fixed image）。

图像可能来自相同或不同个体的三维脑 MRI 的二维切片。通过预测非线性变形场，我们可以将运动图像变形为变形图像（warped image）

[ICCV 2019] Recursive Cascaded Networks for Unsupervised Medical Image Registration



# MMCV

<https://mmcv.readthedocs.io/en/latest/>

$pip install mmcv

Import mmcv

1. 图像及视频处理函数

**Read video**

video = mmcv.VideoReader('test.mp4')

# obtain basic information

print(len(video))

print(video.width, video.height, video.resolution, video.fps)

# iterate over all frames

for frame in video:

print(frame.shape)

To convert a video to images or generate a video from a image directory.

# split a video into frames and save to a folder

video = mmcv.VideoReader('test.mp4')

video.cvt2frames('out\_dir')

# generate video from frames

mmcv.frames2video('out\_dir', 'test.avi')

# cut a video clip

mmcv.cut\_video('test.mp4', 'clip1.mp4', start=3, end=10, vcodec='h264')

# join a list of video clips

mmcv.concat\_video(['clip1.mp4', 'clip2.mp4'], 'joined.mp4', log\_level='quiet')

# resize a video with the specified size

mmcv.resize\_video('test.mp4', 'resized1.mp4', (360, 240))

# resize a video with a scaling ratio of 2

mmcv.resize\_video('test.mp4', 'resized2.mp4', ratio=2)

# image + flow -> next image

img1 = mmcv.imread('img1.jpg')

flow = mmcv.flowread('flow.flo')

warpped\_img2 = mmcv.flow\_warp(img1, flow)

how to calculate flow?

# show image with bounding boxes

img = np.random.rand(100, 100, 3)

bboxes = np.array([[0, 0, 50, 50], [20, 20, 60, 60]])

mmcv.imshow\_bboxes(img, bboxes)

1. Utils

# 处理python文件

test.py

a = 1

b = {'b1': [0, 1, 2], 'b2': None}

c = (1, 2)

d = 'string'

cfg = Config.fromfile('test.py')

assert cfg.a == 1

assert cfg.b.b1 == [0, 1, 2]

cfg.c = None

assert cfg.c == None

# 进度条

Prog\_bar = mmcv.ProgressBar(len(sequence))

For frame\_idx, iter in enumerate(sequence):

…

Prog\_bar.update()

**《计算机视觉--模型，学习和推理》**

先验分布：观测数据x之前，对全局状态y有个先验的估计P(y)

后验分布：观测数据之后，更新对全局状态的估计P(y | x)

如何学习模型 的参数来拟合训练数据？

最大似然法：假设数据点从分布中独立抽样，点的集合的函数=数据点似然的乘积

最大后验法：引入参数的先验信息 + 最大似然法

Bayers法：与ML and MAP的点估计不同，认为参数是个概率

p(t | x) = p(x | t) p(t) / p(x)

p(x\* | x) = 对p(x\* | t) p(t | x) dt求积分，即在每个可能参数值上作无穷加权和

随机森林等效于Bayes法的近似：

随机森林是随机树的集合，每个随机树都使用一个不同的随机选取的函数集合，然后通过对这些树预测的p(y|x)进行平均，就可以获得一个更稳定的分类器。因为通过对不同参数集的每个预测值进行加权求和，所以类似于Bayes法的一个近似

判别模型：全局状态可能性模型 P(y | x)

假定全局状态y分布p(y), 并将全局状态分布参数设为数据x的函数

举例：

二分类，假定全局状态分布是伯努力分布：p(y) = lambdax(1-lambda)1-x

参数lambda用观测数据x的函数表示

若逻辑回归： lambda = sigmoid(XTW)

ML: W = argmax( lambday1(1-lambda)1-y1...lambdayn(1-lambda)1-yn )

多分类，假定全局状态分布是多类分布：p(y) = Cat(lambda)

参数[lambda1, lambda2, ...lambdaN]用观测数据x的函数表示

若逻辑回归： lambdai = softmax

无法求得闭式解，需要借助非线性优化算法

回归，假定高斯分布: p(y) = Norm(u, s)

参数(u, s)用观测数据x的函数表示

若线性回归：u = XTW, s假定球状对角协方差

则p(y | X, W, s) = Norm(XTW, s)

可以用ML, MAP, Bayes法优化求解参数

推理：p(y | x) 将新数据和参数代入表达式计算

生成模型：数据可能性模型P(x | y)

假定数据x分布p(x), 并将数据分布参数设为全局状态y的函数

举例：

假定数据分布是高斯分布：p(x) = Norm(u, s)

参数(u, s)用全局状态y的函数表示

若线性拟合：u = YTW, s假定球状对角协方差

则似然函数p(x | Y, W, s) = Norm(YTW, s)

二分类: 已知训练样本{(x, y), ...}拟合参数{(u0, s0), (u1,s1)}

取Y=1时的观测数据估计高斯分布的均值和方差(u1,s1)

取Y=0时的观测数据估计高斯分布的均值和方差(u0, s0)

推理：p(y | x) = p(x|y)p(y) / p(x)

分别计算

p(y=1 | x) = p(x|y=1)p(y) /p(x) 新数据和对应的参数

p(y=0 | x) = p(x|y=0)p(y) /p(x)

非线性回归和分类：将数据样本-->非线性变换-->然后在变换后的数据进行估计参数

非线性优化算法

W = argminf(w) f(.)代价函数或者目标函数

线性搜索：

1. 基于函数的局部特征选择一个搜索方向s
2. 沿着选择好的方向进行搜索找到最小值，即lambda = argminf(w + lambda\*s)
3. 更新w = w + lambda\*s

梯度下降法：选择下降速度最快的方向

w = w + lr\* (-f’) = w - lr\*f’

牛顿迭代法：不仅仅考虑函数的梯度，还考虑梯度如何变化

如果二阶导数的值很小，那么梯度变化缓慢，因此它完全趋缓并成为一个最小值可能将需要一段时间，因此移动一个较大的距离是安全的

如果二阶导数的值很大，那么梯度变化快，最小值可能就在附近，应当移动一个较小的距离

w = w - lr \* f’’\* f’

缺点：hessian矩阵求逆计算量大，若hessian矩阵不是正定(正定说明是凸函数，负定说明凹函数），那么可能不是下降的方向（这种情况很常见）

还有拟牛顿法，高斯-牛顿法，共轭梯度法等

应用及Tools

## OCR

Optical Character Recognition (OCR)即光学字符辨识是把打印文本转换成一个数字表示的过程。它有各种各样的实际应用--从数字化印刷书籍、创建收据的电子记录，到车牌识别甚至破解基于图像的验证码。

Tesseract

场景文字检测

Text Detection + Text Recognition

<https://github.com/MhLiao/TextBoxes_plusplus>

plate recognition + OCR



Object detector or pose estimation，不仅仅用于人脸，脸姿势，也可用于训练别的目标

人脸检测及识别

数据集

LFW (labeled faces in the wild)

<http://vis-www.cs.umass.edu/lfw/>

training(13233 images) - validation(5749 people) - testing(1680 people with two or more images)

FaceScrub

<http://vintage.winklerbros.net/facescrub.html>

100k Face images of 530 people

人脸检测

Dlib C++ library -- histogram-of-oriented-gradient (HOG) based object detectors

<https://github.com/davisking/dlib>

优点：

1. the HOG trainer uses dlib's structural SVM based training algorithm which enables it to **train on all the sub-windows in every image**. This means you don't have to perform any tedious subsampling or "hard negative mining". It also means you often **don't need that much training data**. In particular, the example program that trains a face detector takes in only 4 images, containing a total of 18 faces. That is sufficient to produce the HOG detector used above
2. **trained** a HOG face detector using about **3000 images** from the labeled faces in the wild dataset and the training took only about **3 minutes**.
3. Fewer false alarms than V-J face detector

Opencv -- Cascaded haar object detector

缺点：take hours or days to train and requires you to fiddle with false negative rates and all kinds of spurious parameters

OpenFace (Free and open source **face recognition** with deep neural networks)

[2015]FaceNet --A Unified Embdeding for Face Recognition and Clustering

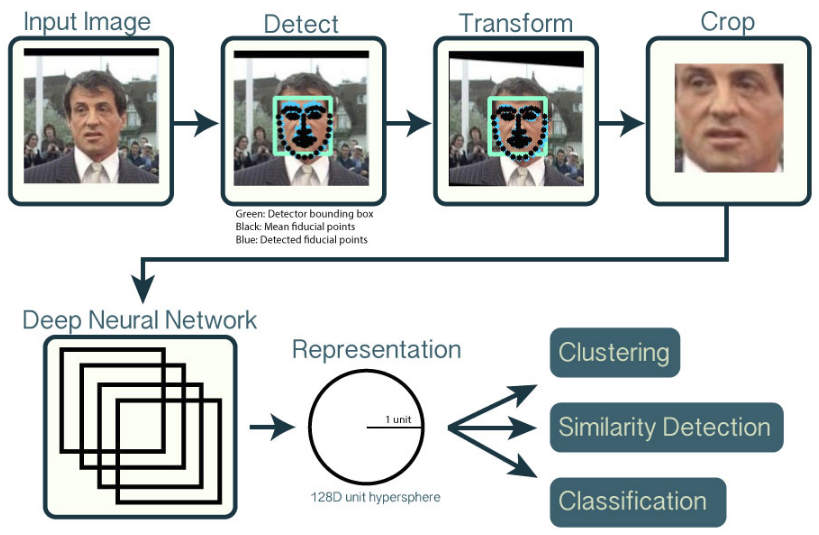
<http://cmusatyalab.github.io/openface/>

<https://github.com/cmusatyalab/openface>

<https://github.com/davidsandberg/facenet> tensorflow

Accuracies from research papers have just begun to surpass human accuracies on some benchmarks.

Workflow:



1. **Detect faces** with a pre-trained models from dlib or OpenCV.
2. **Transform the face** for the neural network. This repository uses dlib's real-time pose estimation with OpenCV's affine transformation to try to make the eyes and bottom lip appear in the same location on each image.
3. Use a deep neural network to **represent (or embed) the face on a 128-dimensional** unit hypersphere. The embedding is a generic representation for anybody's face. Unlike other face representations, this embedding has the nice property that a larger distance between two face embeddings means that the faces are likely not of the same person. This property makes clustering, similarity detection, and classification tasks easier than other face recognition techniques where the Euclidean distance between features is not meaningful.
4. Apply your favorite **clustering** or **classification** techniques to the features to complete your recognition task. See below for our examples for classification and **similarity detection**, including an online web demo.

客户端代码：

import openface

align = openface.AlignDlib(args.dlibFacePredictor)

net = openface.TorchNeuralNet(args.networkModel, args.imgDim, cuda=args.cuda)

bb = align.getLargestFaceBoundingBox(img)

alignedFace = align.align(args.imgDim, img, bb,

landmarkIndices=openface.AlignDlib.OUTER\_EYES\_AND\_NOSE)

rep1 = net.forward(alignedFace)

# `rep2` obtained similarly.

d = rep1 - rep2

distance = np.dot(d, d)

VGG Face Descriptor

<http://www.robots.ox.ac.uk/~vgg/software/vgg_face/>

<https://github.com/AlfredXiangWu/face_verification_experiment>

Face Pose Estimation

[2014]One Millisecond Face Alignment with an Ensemble of Regression Trees

give it an image of someone's face it will add this kind of annotation

Image Inpainting (图像修复)

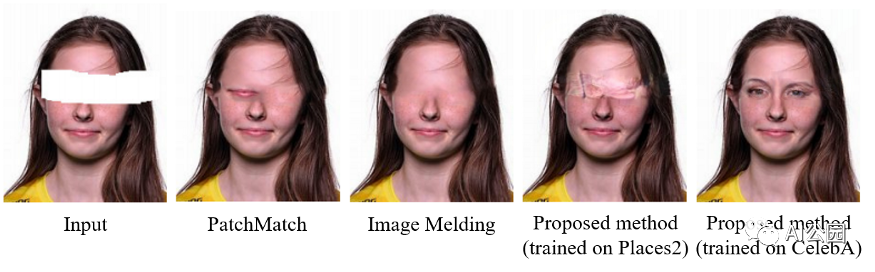


Figure. 举例说明生成新片段的重要性，我们只能生成之前在训练中看到的内容

1. 传统方法

给出一个有一些缺失区域的图像，最典型的传统方法填充缺失区域是复制粘贴。

主要思想是从图像本身或一个包含数百万张图像的大数据集中寻找最相似的图像补丁，然后将它们粘贴到缺失的区域。然而，搜索算法可能是耗时的，它涉及到手工设计距离的度量方法。在通用化和效率方面仍有改进的空间。

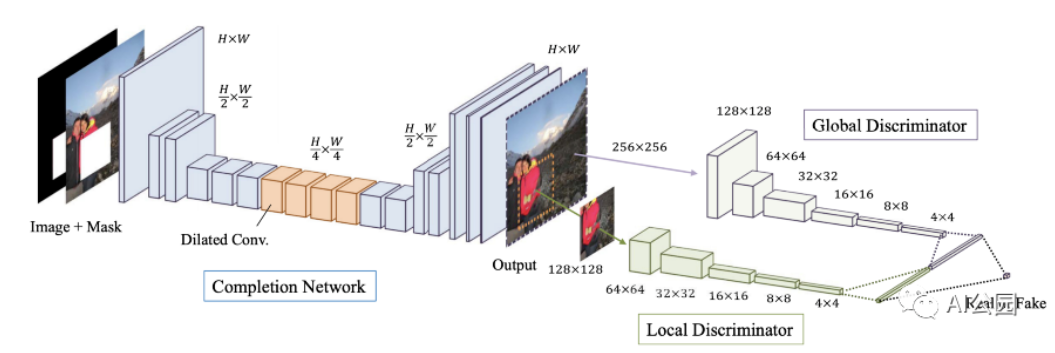
* 多尺度patch

基于patch的方法，一个很大的假设是我们相信我们可以在缺失区域之外找到相似的patch，这些相似的补丁将有助于填充缺失区域。这个假设对于自然场景可能是正确的，因为天空和草坪在一个图像中可能有许多相似的patch。如果缺失区域之外没有任何类似的patch，就像图1中所示的人脸图像修复的情况。在这种情况下，我们找不到眼睛的patch来填补相应的缺失部分。因此，鲁棒的修复算法应该能够生成新的片段。

1. 所有基于深度学习的修复算法

都使用生成对抗网络(GANs)来产生视觉上吸引人的结果。这是相当主观的！典型的GAN由一个生成器和一个鉴别器组成。生成器负责填补图像中缺失的部分，鉴别器负责区分已填充图像和真实图像。我们将随机地将填充的图像或真实的图像输入识别器来欺骗它。最终，如果鉴别器不能判断图像是被生成器填充的还是真实的图像，生成器就能以良好的视觉质量填充缺失的部分!

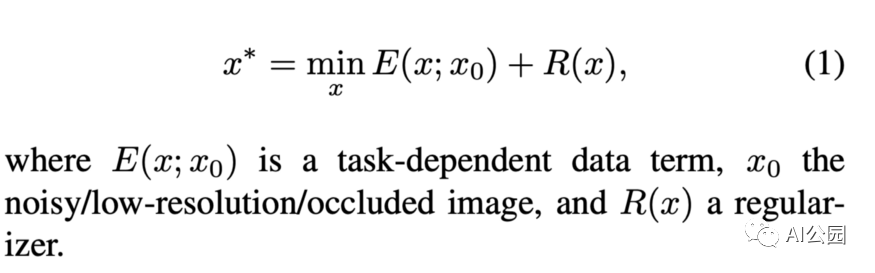
* [2017] Globally and Locally Consistent Image Completion



* [2017] Deep Image Prior

研究人员发现深度CNN有一个有趣的特性 —— 随机初始化的网络比纯噪声能更快的拟合自然图像。换句话说，CNN对自然图像有自然的“优先”偏好，可以利用这一点在没有任何数据的情况下去除图像中的人工痕迹！

为了对图像去噪(去除水印，修复等)，随机初始化一个CNN并将图像输入到模型中(input=image, output=image，就像一个自动编码器)。不出所料，模型逐渐达到零训练损失(参数数量>>图像中的像素数量)。然而，当训练被适当地提前停止时，网络产生了去噪的图像。



we fit a generator network to a single degraded image. In this scheme, the network weights serve

as a parametrization of the restored image. The weights are randomly initialized.

we cast reconstruction as a conditional image generation problem and show that the only information required to solve it is contained in the single degraded input image and the handcrafted structure of the network used for reconstruction.

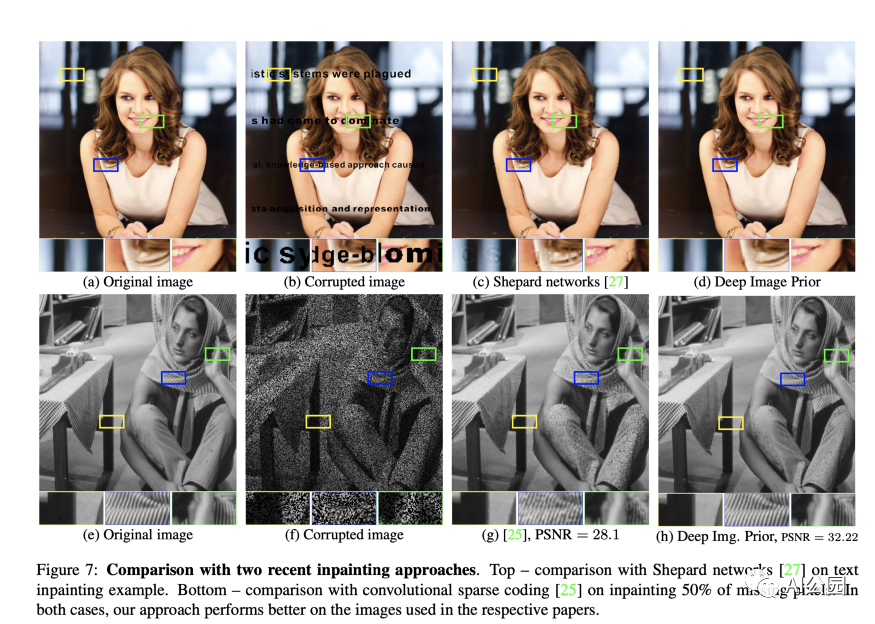


Image Super-resolution (图像超分辨率)

Image Denoising (图像降噪)

# Interview

## How to detect line and circle in image?

## 如何评价两张图像的相似度？

* 平均均方误差
* Correlation Coefficient

ccoeff(roi of reference, compare frame i) = max( ccoeff(roi of ref, roi of compared frame i))

ccoeff(roi\_ref, roi) =

* Structural Similarity Index

SSIM(reference, compare frame i) = mean (SSIM(x, y))

compare two windows centered in (x, y)

from skiimage.metrics import structural\_similarity as compare\_ssim, ssim

score = ssim(img0, img1)

(score, diff) = compare\_ssim(img0, img1)

* Siamese Network and Triplet loss

图示

描述已自动生成