Code gist for Developers

Zca Whitening Opencv

(0.00905299186707 seconds)

rollingstone/gist:425bda3f02aa4b36f43246ff320a280c (python)

```python
import numpy as np
from scipy import linalg
from sklearn.utils import array2d, as_float_array
from sklearn.base import TransformerMixin, BaseEstimator


class ZCA(BaseEstimator, TransformerMixin):

    def __init__(self, regularization=10**-5, copy=False):
        self.regularization = regularization
        self.copy = copy

    def fit(self, X, y=None):
        X = array2d(X)
        X = as_float_array(X, copy = self.copy)
        self.mean_ = np.mean(X, axis=0)
        X -= self.mean_
        sigma = np.dot(X.T,X) / X.shape[1]
        U, S, V = linalg.svd(sigma)
        tmp = np.dot(U, np.diag(1/np.sqrt(S+self.regularization)))
        self.components_ = np.dot(tmp, U.T)
        return self

    def transform(self, X):
        X = array2d(X)
        X_transformed = X - self.mean_
        X_transformed = np.dot(X_transformed, self.components_.T)
        return X_transformed
```

ZCA whitening (

zihaow21/gist:dc206dcbef5168467571f0710022fceb (python)

```python
import numpy as np
from scipy import linalg
from sklearn.utils import array2d, as_float_array
from sklearn.base import TransformerMixin, BaseEstimator


class ZCA(BaseEstimator, TransformerMixin):

    def __init__(self, regularization=10**-5, copy=False):
        self.regularization = regularization
        self.copy = copy

    def fit(self, X, y=None):
        X = array2d(X)
        X = as_float_array(X, copy = self.copy)
        self.mean_ = np.mean(X, axis=0)
        X -= self.mean_
        sigma = np.dot(X.T,X) / X.shape[1]
        U, S, V = linalg.svd(sigma)
        tmp = np.dot(U, np.diag(1/np.sqrt(S+self.regularization)))
        self.components_ = np.dot(tmp, U.T)
        return self

    def transform(self, X):
        X = array2d(X)
        X_transformed = X - self.mean_
        X_transformed = np.dot(X_transformed, self.components_.T)
        return X_transformed
```

ZCA whitening (

duschendestroyer/gist:5170087 (python)

```python
import numpy as np
from scipy import linalg
from sklearn.utils import array2d, as_float_array
from sklearn.base import TransformerMixin, BaseEstimator


class ZCA(BaseEstimator, TransformerMixin):

    def __init__(self, regularization=10**-5, copy=False):
        self.regularization = regularization
        self.copy = copy

    def fit(self, X, y=None):
        X = array2d(X)
        X = as_float_array(X, copy = self.copy)
        self.mean_ = np.mean(X, axis=0)
        X -= self.mean_
        sigma = np.dot(X.T,X) / X.shape[1]
        U, S, V = linalg.svd(sigma)
        tmp = np.dot(U, np.diag(1/np.sqrt(S+self.regularization)))
        self.components_ = np.dot(tmp, U.T)
        return self

    def transform(self, X):
        X = array2d(X)
        X_transformed = X - self.mean_
        X_transformed = np.dot(X_transformed, self.components_.T)
        return X_transformed
```

ZCA whitening (

anwarnunez/gist:3100232 (python)

```python
'''
Implementation of the mahalanobis transform
The random dataset produced with have close to zero correlation,
but nevertheless non-zero correlation (by chance). The transform
will force 0 correlation.
'''

X = np.random.randint(0,10, (49,2)) # Some fake data (same size as real)
xcov = np.cov(X.T)                  # Feature covariance
L, Q = np.linalg.eigh(xcov)         # Eigen decoposition of covariance matrix
Linv = np.diag(1./np.sqrt(L))       # Compute \Sigma^{-1/2}
scov = np.dot(np.dot(Q, Linv), Q.T) # ...
xm = np.mean(X, axis=0)             # Center matrix
assert xm.shape == (2,)             # ...
X -= xm                             # ...
X = np.dot(X, scov)                 # Project the data

# Assert correlation of features is 0
#####################################
corr = np.corrcoef(X.T)
print 'Whitened features correlation:'
print np.round(corr, 4)
assert np.allclose(corr, np.asarray([[1,0],[0,1]]))
```

whitening transform

andreanidouglas/opencv (shell)

```shell
#!/bin/sh

if [ $(id -u) -ne 0 ]; then
    echo "Please run as: sudo sh $0"
    exit
fi

#update system
apt-get update && apt-get upgrade -y

#install dependencies
```

```
apt-get -y install build-essential python-dev python-numpy libtbb2 libtbb-dev libjpe
apt-get install cmake git libgtk2.0-dev pkg-config libavcodec-dev libavformat-dev li
apt-get remove libopencv
apt-get remove opencv


mkdir opencv
cd opencv

git clone https://github.com/Itseez/opencv.git
git clone https://github.com/Itseez/opencv_contrib.git

cd opencv
git checkout 3.1.0
cd ../opencv_contrib
git checkout 3.1.0
cd ..

mkdir release
cd release

cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/usr/local -D OPENCV_EXTRA

make
make install
```

rgerhardt/Opencv (python)

```python
#!/usr/bin/env python

from collections import OrderedDict

class OpenCvGuiFunc(object):

    tags = []
    variables = OrderedDict()
#     __features = {}

    def __init__(self):
        self.features = {}

    def clear_all_info(self):
        self.features.clear()

    def get_info(self, name=None):
        if not name:
            return self.features
        return self.features.get(name, [])

#    @staticmethod
#    def clearAllInfo():
#        """ Clear all stored info
#        """
#        OpenCvGuiFunc.__features = {}


 #   @staticmethod
 #   def get_info(name=None):
 #       if not name:
 #           return OpenCvGuiFunc.__features
 #
 #       return OpenCvGuiFunc.__features.get(name, [])

    def add_pipeline_info(self, features):
        for feature in features:
            if features: #check if it is empty
                self.features.update(feature) #merge

    def add_info(self, name, info):
        """ Adds info to the pipeline run

        :param name: string to identify what's the info's name. If there's already s
            info with that name in the pipeline, both infos will be associated with
            name
        :param info: Any object to be assigned as info for the given name
        """
        if name not in self.features:
            self.features[name] = []
        self.features[name].extend(info)
```

```
52
53
54      def apply(self, img):
55          raise NotImplementedError("apply is not implemented")
56
57      def on_video_start(self):
58          """ Method that should be called whenever a video starts
59
60          Functions can use this to set up any video-based algorithms
61          """
62          pass
63
64      def on_video_stop(self):
65          """ Method that should be called whenever a video stops
66
67          Functions can use this to set up any video-based algorithms
68          """
69          pass
70
71      def original_image(self):
72          """ Method that returns the original image of a pipeline run
73          The runner in possession of an OpenCvGuiFunc is responsible for making
74          sure this method works
75
76          :returns: image
77          """
78          return self._original_img
79
80      def get_param(self, name):
81          """ Method that returns a kw parameter passed to `runner.run`
82          The runner in possession of an OpenCvGuiFunc is responsible for making
83          sure this method works
84          """
85          return self._params.get(name)
```

eickenberg/convolutional_zca.py (python)

```
1   # Disclaimer: This doesn't seem to work 100% yet, but almost ;)
2
3   # Convolutional ZCA
4   # When images are too large in amount of pixels to be able to determine the
5   # principal components of an image batch, one can suppose translation
6   # invariance of the eigen-structure and do the ZCA in a convolutional manner
7
8   import theano
9   import theano.tensor as T
10  import numpy as np
11  from sklearn.feature_extraction.image import extract_patches
12  from sklearn.decomposition import PCA
13
14
15  def convolutional_zca(X, patch_size=(8, 8), step_size=(2, 2)):
16      """Perform convolutional ZCA
17
18      Parameters
19      ==========
20      X: ndarray, shape= (height, width, n_channels)
21      """
22
23      n_imgs, h, w, n_channels = X.shape
24      if len(patch_size) == 2:
25          patch_size = patch_size + (n_channels,)
26
27      if len(step_size) == 2:
28          step_size = step_size + (1,)
29
30      patches = extract_patches(X,
31                                (1,) + patch_size,
32                                (1,) + step_size).reshape((-1,) + patch_size)
33      pca = PCA()
34      pca.fit(patches.reshape(patches.shape[0], -1))
35
36      # Transpose the components into theano convolution filter type
37      components = theano.shared(pca.components_.reshape(
38              (-1,) + patch_size).transpose(0, 3, 1, 2).astype(X.dtype))
39      whitening_factors = T.addbroadcast(
40          theano.shared(1. / np.sqrt(pca.explained_variance_).astype(X.dtype).reshape(
41      componentsT = components.dimshuffle((1, 0, 2, 3))[:, :, ::-1, ::-1]
42
```

```
    input_images = T.tensor4(dtype=X.dtype)
    conv_whitening = T.nnet.conv2d(
        T.nnet.conv2d(input_images.dimshuffle((0, 3, 1, 2)),
                      components, border_mode='full') * whitening_factors,
        componentsT)

    f_whitening = theano.function([input_images], conv_whitening)

    return f_whitening(X)


if __name__ == "__main__":

    CIFAR_DIR = '/home/me/data/datasets/cifar-10-batches-py'
    CIFAR_FILE = 'data_batch_2'

    n_images = 1000

    import os
    import pickle

    cifar = pickle.load(open(os.path.join(CIFAR_DIR, CIFAR_FILE)))
    images = cifar['data'][:n_images].reshape(
        -1, 3, 32, 32).transpose(0, 2, 3, 1)

    whitened = convolutional_zca(images.astype(np.float32))
```

Draft of a convolutional ZCA

adelnizamutdinov/MainActivity.java (gradle)

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 25
    buildToolsVersion "25.0.3"

    defaultConfig {
        applicationId "adeln.opencv"
        minSdkVersion 21
        targetSdkVersion 25
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-
        }
    }
}

repositories {
    // opencv
    maven {
        url "http://dl.bintray.com/steveliles/maven"
    }
}

dependencies {
    compile "org.opencv:OpenCV-Android:3.1.0"
    compile group: 'com.squareup.okio', name: 'okio', version: '1.12.0'
}
```

opencv

75u2u/mosaic.cpp (c)

```
#include <vector>
#include <string>
#include <iostream>
#include <opencv2/opencv.hpp>

```

```
cv::CascadeClassifier classifier;
const char * const file = "C:/Program Files/OpenCV/sources/data/haarcascades/haarcas
cv::Size min_face;

//! initialize
void init(int w, int h)
{
    std::cout << "frame=" << w << 'x' << h << std::endl;

    if (!classifier.load(file)) {
        std::cout << "fail to load: " << file << std::endl;
        std::exit(1);
    }
    const double scale = 0.10;
    min_face = cv::Size(w * scale, h * scale);
    std::cout << "min_face=" << min_face << std::endl;
}

//! process image frame
cv::Mat process(cv::Mat& frame)
{
    std::vector<cv::Rect> faces;
    classifier.detectMultiScale(frame, faces, 1.1, 3, 0, min_face);
    for (const auto& r : faces) {
        // mosaic filter
        const double f = 0.05;
        cv::Mat tmp;
        cv::resize(frame(r), tmp, {}, f, f);
        cv::resize(tmp, frame(r), r.size(), 0, 0, cv::INTER_NEAREST);
        // draw area border
        cv::rectangle(frame, r.tl(), r.br(), { 0, 0, 0 });
    }
    return frame;
}

int main(void) {
    const char * const window = "Capture (Press ESC to exit)";
    const double scale = 0.75;
    const int delay = 32;   // [msec]

    cv::VideoCapture cap(0);
    if (!cap.isOpened()) {
        std::cerr << "fail to open cv::VideoCapture" << std::endl;
        return 2;
    }

    const double width = cap.get(cv::CAP_PROP_FRAME_WIDTH) * scale;
    const double height = cap.get(cv::CAP_PROP_FRAME_HEIGHT) * scale;
    cap.set(cv::CAP_PROP_FRAME_WIDTH, width);
    cap.set(cv::CAP_PROP_FRAME_HEIGHT, height);
    init(width, height);

    cv::namedWindow(window);
    cv::Mat frame;
    do {
        cap >> frame;
        cv::imshow(window, process(frame));
    } while (cv::waitKey(delay) != '\x1b');

    return 0;
}
```

opencv

ksoda/mask.py (python)

```
import cv2, matplotlib
import numpy as np
import matplotlib.pyplot as plt

bgr_img = cv2.imread('images/landscape.jpg')
img = cv2.cvtColor(bgr_img, cv2.COLOR_BGR2HSV)
blue_min = np.array([100, 0, 0], np.uint8)
blue_max = np.array([140, 255, 255], np.uint8)
wh_min = np.array([0, 0, 180], np.uint8)
wh_max = np.array([180, 60, 255], np.uint8)
blue = cv2.inRange(img, blue_min, blue_max)
white = cv2.inRange(img, wh_min, wh_max)
inv_mask = cv2.bitwise_or(white, blue)
mask = cv2.bitwise_not(inv_mask)
```

```
plt.imshow(mask, cmap=plt.get_cmap('gray'))
mask_rgb = cv2.cvtColor(mask, cv2.COLOR_GRAY2RGB)
masked_rgb = cv2.bitwise_and(cv2.cvtColor(bgr_img, cv2.COLOR_BGR2RGB), mask_rgb)
plt.imshow(masked_rgb)
```

OpenCV