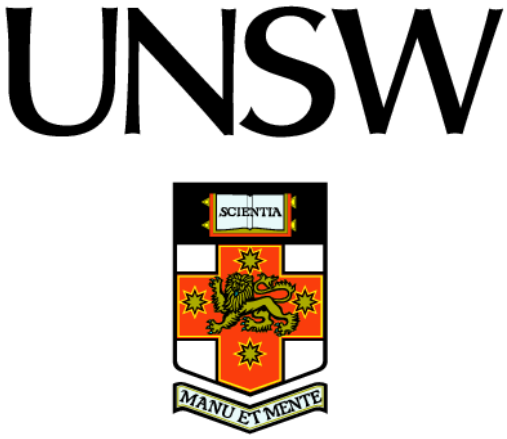

University of New South Wales

Faculty of Computer Science & Engineering

COMP9417 Machine Learning and Data Mining

Assignment 02 report

Project name:

Face Recognition System

Team members:

Qujie Li	**z3429088**
Xin liu	**z5027902**
Shuai Han	**z5038831**
Qianrui Zhao	**z5088359**

Submission date:

04/06/2017

Content

Introduction

Currently machine learning is one of the hottest area, the extension of machine learning is deep learning, the applications are wildly applied in different areas, such as graph recognition, auto-pilot, face recognition.

The goal of this project was to implement a Face Recognition System in real time but deal with photos or videos of relatively lower resolution. The Face Recognition System implemented face detection, face identification and face verification, the general performance measures for face recognition problem.

After though digging into Deep Convolutional Neural Network(CNN), which is wildly used in dealing with audio recognition, computer version. We learned and understand key concepts of Convolution layer, pooling layer, Activation function: sigmoid, Relu, etc. As well as learning and utilizing the basic concepts in Caffe, PyQt, Dlib library, OpenCV.

Face identification is defined as identifying a person based on the image of a face, or compared with all the registered person, one to many matching, since we have already trained the input data for people. It is a general topic, normally includes both face identification and face verification. In our project, face verification is defined as to validate a claimed identity based on the image of a face, or either accepting or rejecting the identity claim, one to one matching.

Implementation

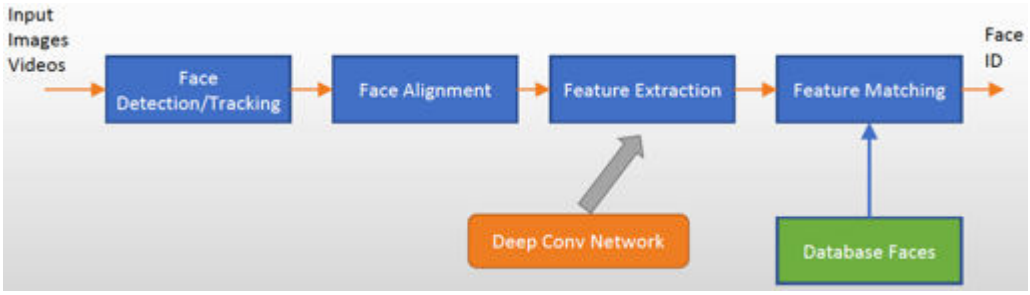

Fig 01. Face recognition framework

Fig 01 shows the overall framework of face recognition. The recognition work basically consists of 5 steps:

Step 01: user inputs imagers or videos, Face Detection module detects the face;

Step 02: in Face Alignment, there is a method face landmark detection can detect the eyes, noses, face etc;

Step 03: utilizing the Deep conv network for feature extraction;

Step 04: feature Matching part, there is a face database trained in previous;

Step 05: in final stage, we need to compare the similarity then output the identity of this face, finally output the FaceID.

Based on the framework, we did a full stack implementation which involves the face detection module, the core training and matching part, in which the model is trained using Caffe, and the GUI developed using PyQt.

In this section each of them is introduced separately. Follows by a summary section which demonstrates how we integrate different modules together.

Face Detection

We use Dlib libray and OpenCV(cv2) in face detection, Dlib is a modern C++ containing machine learning algorithms and tools for creating software in C++. The reason we use this library is it is open source with complete and precise documentation for every class and function, as well as powerful machine learning algorithms.

1. Read Video/Image

After import OpenCV 2, we can read videos or photos, or real time images. Then save to frame. Because every image we read are most likely colourful.

We convert image to Gray, then show the image and release and destroy the openCV read method. Fig 03 shows how to capture a video from a camera.

```
self.FPS = 25

# Note: If self.devideid is string, it loads the video from filesys, if self.
self.cap = cv2.VideoCapture(self.deviceid)
#self.cap.set(cv2.cv.CV_CAP_PROP_FRAME_WIDTH, 640)
#self.cap.set(cv2.cv.CV_CAP_PROP_FRAME_HEIGHT, 320)
```

Fig 02. Code for capturing a video from cameras

According to Fig02, we also can capture the image from camera.

2. Face Detector & Features Extraction

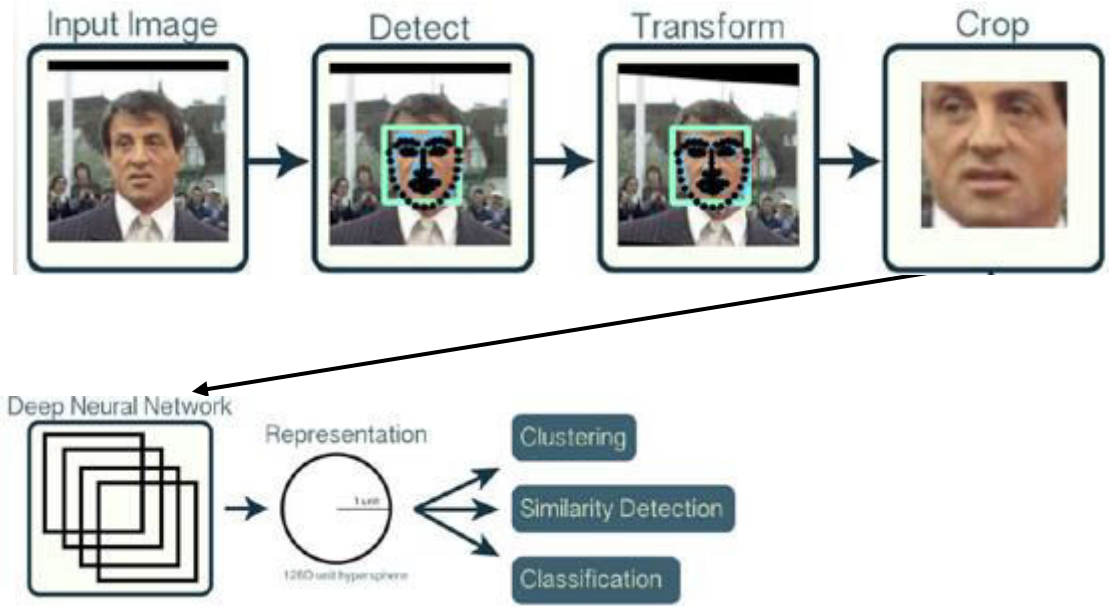

Fig 03. Face detection/ alignment
[Green box : Detector bounding box
Black Dot: Mean fiducial points
Blue Dot: Detected fiducial points]

As shown in Fig 03, given an image, we need to crop person's face, the black dots show face landmark point, after transform, the we crop a person's face, the the cropped face will be used as an input for Deep Neural Network(CNN).

Therefore, in next stage we need to implement the face detection so that it could be used as an input in CNN. This is performed by a Hog detector. The Hog, as shown in Fig 04, is known as Histogram of Oriented gradients, it is also defined as features of faces in our project [4] . HOG face detector using about 3000 images from the LFW dataset and the training took only about 3 minutes. It uses a scanner to scan a 80 * 80 pix region, and check if the scanned area contains a face. If it hits, the image will be transformed into a Hog after some computation.

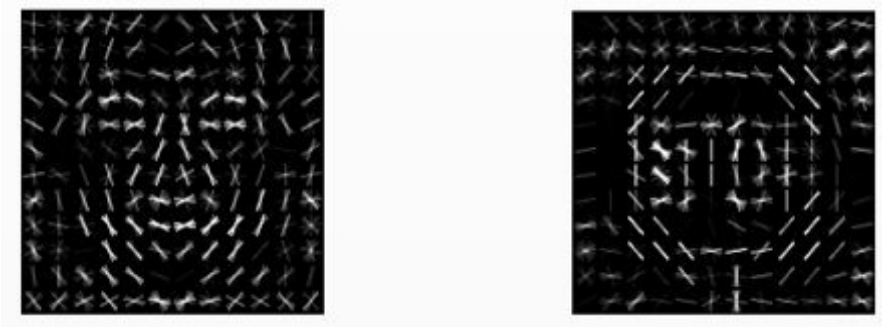

Fig 04. Hog examples

The procedures of using a face detector

1. Load datasets/ images (like load Neo's photo, Res's Photo, Wisdoms' photo)
2. Upsample datasets/ image (Image Pyramid: Zoom image in a specific proportion since the size of image of face could be very small or very big)
3. Flip datasets/ image (increase the size of dataset)
4. Train SVM: input image and face box
5. Test

The code in below show how to use the face detector in project:

In FaceDector.py Module

```
import dlib    # import dlib first
```

In Face_detector Class,

```
    self.face_detector = dlib.get_frontal_face_detector()
    self.ldmark_detector = dlib.shape_predictor('./dlib_model/shape_predictor_68_face_landmarks.dat')
```

Initialize a new detector, then read the image, the second parameter is set to 0. Although the result is not always good enough, but speed is actually quite good.

As you can see the code, dets stores all of data of detected face image, as we know, we could have detected more than one face images, then we could simply print the square for the position of face image.

For dlib landmark detection, we get the shape_predictor_68_face_landmarks.data file from: http://dlib.net/files/shape_predictor_68_face_landmarks.dat.bz2

```
for k, d in enumerate(dets):
    #print("Detection {}: left: {} Top: {} Right: {} Bottom: {}".format(
    #    k, d.left(), d.top(), d.right(), d.bottom()
    #))
    #self.textBrowser.append("Detection {}: left: {} Top: {} Right: {} Bottom: {}".form
    #    k, d.left(), d.top(), d.right(), d.bottom()))

    # ldmark detection
    landmarks = []
    if self.ldmarking:
        shape = self.ldmark_detector(img, d)
        landmarks = [(shape.part(i).x, shape.part(i).y) for i in range(68)]
        eye_l = np.mean(landmarks[36:42], axis=0)
        eye_r = np.mean(landmarks[42:48], axis=0)
        #print eye_l, eye_r
        #self.textBrowser.append('eye_l {}:{}'.format(eye_l[0], eye_r[1]))
    crop_face = np.copy(img[max(0, d.top()):d.bottom(), max(0,d.left()):d.right(), :])
```

Fig 05. Code for detecting faces

According to Fig 05, from the previous detected image data, shape is used to draw facial landmarks with the predictor class, then get eyes landmarks, finally crop the face image.

3. Sliding windows for scanning faces in an image

Fig 06. First sliding window of scanning an image

Fig 07. A series of sliding windows during scanning an image

According to Figure 6 and Figure 7, we can see, scanner start scanning the image from top left, row by row, if a face image is detected, then store crop the face image.

we can get features as well as known representation, Now we get features, we can do clustering, similarity detection and classification.

CNN training model

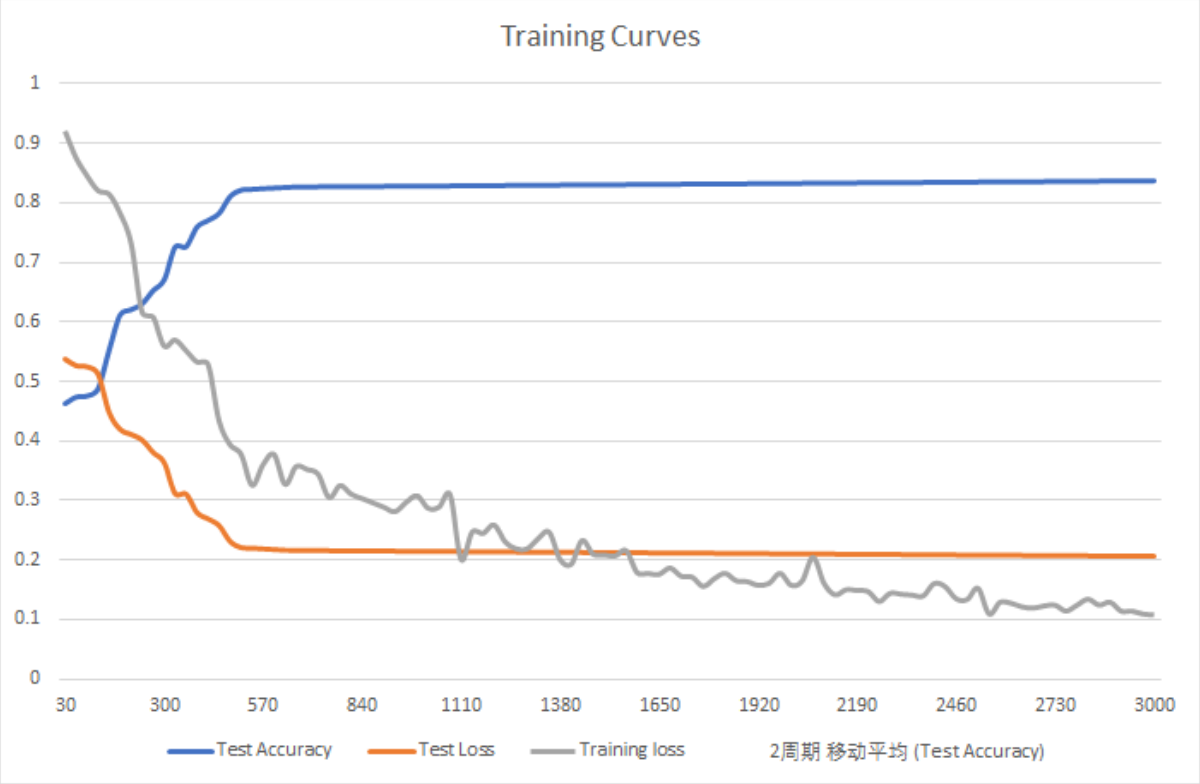

Fig 08. Training curve of our model

According to Fig 08, as the iterations increasing, the test accuracy increase, on the contrary training Loss will decrees, it is deceasing while the test accuracy is increasing. When Test Lost and Test accuracy tends to constant, the model is finished.

1. How we using the model

Use pre-trained model for prediction:

1. Load model
2. Load mean and configure pre-processing
3. Reshape input layer and set batch size (each time maybe 1 image for processing)
4. Load image in the data layer
5. Forwarding (load image in the nets until the last layer)
6. Predict result

2. Use pre-trained model for prediction

Load model

```
# load face model
caffemodel = './deep_model/VGG_FACE.caffemodel'
deploy_file = './deep_model/VGG_FACE_deploy.prototxt'
mean_file = None
self.net = Deep_net(caffemodel, deploy_file, mean_file, gpu=True)
```

Fig 09. Code for loading Caffe [5] model
This is for face cognition model training, same idea for gender recognition

As you can see from Caffe load CNN model first [2] [3]

```
def get_transformer(self, deploy_file, mean_file=None):
    """
    Returns an instance of caffe.io.Transformer
    :param deploy_file: path to a .prototxt file
    :param mean_file:   path to a .binaryproto file (default=None)
    :return: caffe.io.Transformer
    """
    network = caffe_pb2.NetParameter()
```

Fig 10. Code for loading mean and configuring pre-processing

As you can see from Fig 10, in the caffe_net.py module, after load the model, get_transformer method will load input and configure pre-processing.

And the rest of the processing from step 3 to step 6 are all defined and finished in caffe_net.py module.

Now we got the model for detect the face and gender.
Use pre-trained model for fine-tuning

1. Convert data
2. Define net (as prototxt)
3. Define solver (as prototxt)
4. Train (with pertained weight)

We have the model, next step for Feature matching in face recongnition

Feature matching in face recognition

Distance computation in python

We use Cosine distance to computer the similarity of two different images:

$$1 - \frac{u \cdot v}{\|u\|_2 \|v\|_2}$$

We can directly import:
sklearn.metrics.pairwise.cosine_similarity
Sklearn.metrics.pairwise.cosine_distence

PyQt GUI

In our project, PyQt4 is used in GUI development. PyQt [1] is a GUI widgets toolkit, it is a python interface for Qt, one of the most powerful, and popular cross-platform GUI library, most importly it is easy to learn and implement.

The advantage of PyQt is, unlike a console mode application, PyQt is executed in a sequential manner, a GUI based application is event driven. Functions or methods are executed in responded to user's actions like clicking on a button, selecting an item from a collection or mouse click.

So we use PyQt to construct the user interface, such as checkbox, LCD, catch recognition results signals.

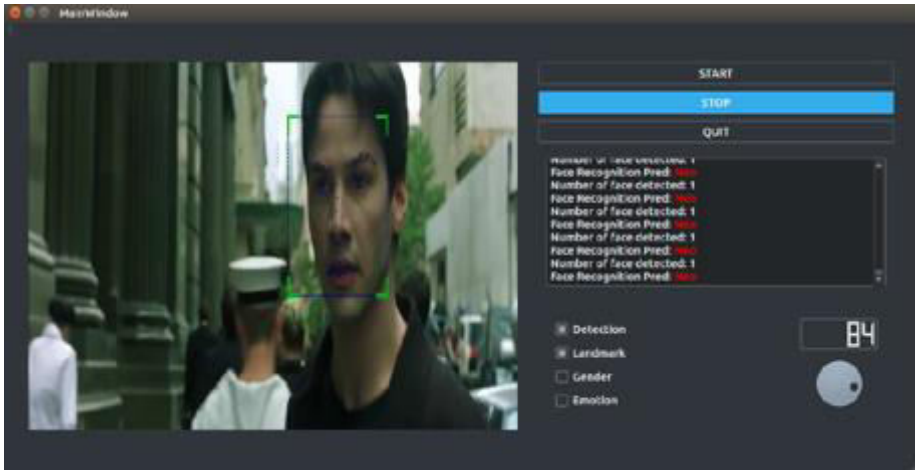

Fig 11. PyQt GUI in our project

The GUI is expected to offer a more intuitive experience when users using our face recognition system. As you can see in Fig 11, the interface of our project is pretty simple, just three buttons and output window which could be operated by users without a usage instruction.

Integration of all modules

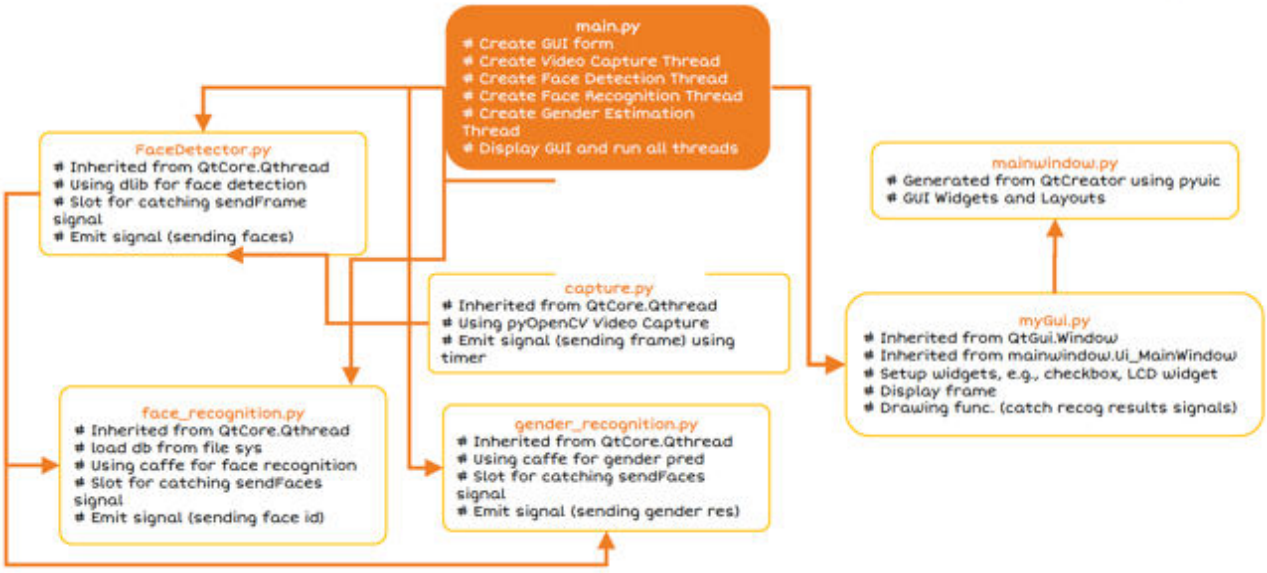

Fig 12. Integration of all modules

As showed in Fig 12.

1. in main.py module we create GUI form, and need multi-tread to deal with different tasks. As the goal of this project is for face recognition, so we need to a simple GUI for input video and image, then show the gender and identity.
2. Same as we need to module to deal with the videos or images, so all of tasks that related to video/image processing in capture.py
3. For all jobs of face detection in FaceDetector.py module.
4. face_recognition.py module to crop all possible human face.
5. Gender_recognition.py module to recognize the gender of person.
6. The User interface is in myGui.py module and mainwindow.py module.

Experimentation

Our face recognition system is tested by input an image and examine the feedback from GUI. The feed back contains the number of verified faces, and provides match results for the detected faces. A correct result is showed in more than 90% of the tests, which shows that our system provides good performance in terms of correctness and robustness.

Moreover, a short video is provided in order to show how our system works. The video can be found here:

https://www.youtube.com/watch?v=494OZOdDXLQ

Besides, some test data could be downloaded via Internet, which could be used as input in our system:

http://dlib.net/files/shape_predictor_68_face_landmarks.dat.bz2

Conclusion

Overall, in this project, from the initial idea until we started design the Face recognition framework, there are mainly four parts for the framework, Face/Detection to read the input, detect the face, then Face Alignment, there is a method face landmark detection can detect the eyes, noses, face etc. After previous stage, utilizing the Deep conv network(Caffe) for feature extraction, then final stage for face matching which is implemented by trained different people's face.

Then we draw first version UML diagram, the modules that we need, the functionality of each module, although we modified the UML diagram many times. We decided use opencv2 for reading input videos/images, dlib library for landmark detection, Caffe for data training(CNN), use pre-trained Caffe model for prediction and fine-turning (mainly for face recognition and gender recognition), at final stage by using cosine distance for distance computation for feature matching in face recongnition. PyQt GUI implement GUI user interface, import GUI from qtcreator.

In this project, we re-designed project UML several times, read documentation of Caffe, PyQt, dlib libray, deeply understand how CNN works, although it is perfect, we did learn and finish this great team project.

References

[1] Getting Started with PyQt
https://wiki.python.org/moin/PyQt/Tutorials

[2] Convolutional Neural Networks (LeNet). (2017, May 5). Retrieved June 4, 2017, from http://deeplearning.net/tutorial/lenet.html

[3] Geitgey, A. (2016, June 13). Machine Learning is Fun! Part 3: Deep Learning and Convolutional Neural Networks. Retrieved June 4, 2017, from https://medium.com/@ageitgey/machine-learning-is-fun-part-3-deep-learning-and-convolutional-neural-networks-f40359318721

[4] Mallick, S. (2016, December 6). Histogram of Oriented Gradients. Retrieved June 4, 2017, from http://www.learnopencv.com/histogram-of-oriented-gradients/

[5] Bourez, C. (2015, September 4). Deep learning tutorial on Caffe technology : basic commands, Python and C code. Retrieved June 4, 2017, from http://christopher5106.github.io/deep/learning/2015/09/04/Deep-learning-tutorial-on-Caffe-Technology.html

Appendix – Test result

NO.	Test Accuracy	Test Loss	Training loss
30	0.462	0.538	0.920
60	0.473	0.527	0.875
90	0.475	0.525	0.845
120	0.487	0.513	0.820
150	0.552	0.448	0.814
180	0.611	0.420	0.782
210	0.620	0.411	0.732
240	0.630	0.402	0.616
270	0.652	0.381	0.608
300	0.670	0.364	0.560

330	0.725	0.311	0.570
360	0.726	0.310	0.552
390	0.759	0.279	0.533
420	0.770	0.269	0.529
450	0.782	0.257	0.434
480	0.810	0.231	0.394
510	0.821	0.220	0.378
540	0.822	0.219	0.326
570	0.823	0.218	0.361
600	0.824	0.217	0.378
630	0.826	0.216	0.328
660	0.826	0.215	0.358
690	0.826	0.215	0.353
720	0.827	0.215	0.345
750	0.827	0.215	0.306
780	0.827	0.215	0.326
810	0.827	0.214	0.311
840	0.827	0.214	0.304
870	0.827	0.214	0.297
900	0.827	0.214	0.289
930	0.827	0.214	0.282
960	0.828	0.214	0.298
990	0.828	0.214	0.309
1020	0.828	0.214	0.288
1050	0.828	0.213	0.290
1080	0.828	0.213	0.311
1110	0.828	0.213	0.202
1140	0.828	0.213	0.248
1170	0.828	0.213	0.246
1200	0.829	0.213	0.260
1230	0.829	0.213	0.232
1260	0.829	0.213	0.220
1290	0.829	0.212	0.220
1320	0.829	0.212	0.237
1350	0.829	0.212	0.248
1380	0.829	0.212	0.202
1410	0.830	0.212	0.194
1440	0.830	0.212	0.234
1470	0.830	0.212	0.211
1500	0.830	0.212	0.209
1530	0.830	0.211	0.208
1560	0.830	0.211	0.216
1590	0.830	0.211	0.180
1620	0.830	0.211	0.178
1650	0.831	0.211	0.177
1680	0.831	0.211	0.188
1710	0.831	0.211	0.174
1740	0.831	0.211	0.172
1770	0.831	0.210	0.157
1800	0.831	0.210	0.169
1830	0.831	0.210	0.178

1860	0.832	0.210	0.166
1890	0.832	0.210	0.165
1920	0.832	0.210	0.158
1950	0.832	0.210	0.162
1980	0.832	0.210	0.178
2010	0.832	0.209	0.159
2040	0.832	0.209	0.165
2070	0.832	0.209	0.205
2100	0.833	0.209	0.161
2130	0.833	0.209	0.142
2160	0.833	0.209	0.151
2190	0.833	0.209	0.150
2220	0.833	0.209	0.148
2250	0.833	0.208	0.131
2280	0.833	0.208	0.145
2310	0.834	0.208	0.143
2340	0.834	0.208	0.142
2370	0.834	0.208	0.140
2400	0.834	0.208	0.161
2430	0.834	0.208	0.156
2460	0.834	0.208	0.136
2490	0.834	0.207	0.134
2520	0.834	0.207	0.153
2550	0.835	0.207	0.110
2580	0.835	0.207	0.130
2610	0.835	0.207	0.128
2640	0.835	0.207	0.122
2670	0.835	0.207	0.120
2700	0.835	0.207	0.124
2730	0.835	0.206	0.125
2760	0.836	0.206	0.115
2790	0.836	0.206	0.125
2820	0.836	0.206	0.135
2850	0.836	0.206	0.125
2880	0.836	0.206	0.130
2910	0.836	0.206	0.115
2940	0.836	0.205	0.115
2970	0.836	0.205	0.110
3000	0.837	0.205	0.109