DigiCon
Code
Team 9

April 1, 2018

Contents

1 Introduction

All of the code is written in python2.7 with strict linting using pylint.

2 src

2.1 main.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import os
import sys
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
from matplotlib.patches import Polygon
import setupLogging
import window

# Sets up log level based on environment variabel exported by make
def logLevelResolver():
    logLevel = setupLogging.logging.WARNING
    if os.environ.get('logLevel') is None:
        return logLevel
    if os.environ['logLevel'] == 'DEBUG':
        logLevel = setupLogging.logging.DEBUG
    elif os.environ['logLevel'] == 'INFO':
        logLevel = setupLogging.logging.INFO
    return logLevel

# Acquires log level from logLevelResolver
def envHandler():
    logLevel = logLevelResolver()
    return logLevel

# Instantiates the GUI and applies styling.
def run():
    app = window.QtGui.QApplication(sys.argv)
    sshFile = './stylesheet/darkOrange.stylesheet'
    with open(sshFile, 'r') as fh:
        app.setStyleSheet(fh.read())
    app.setStyleSheet(window.qdarkstyle.load_stylesheet_pyqt())
    _GUI = window.Window()
    sys.exit(app.exec_())


if __name__ == '__main__':
    logLevel = envHandler()
    setupLogging.setupLogging(logLevel)
    run()
```

2.2 setupLogging.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import sys
import os
import logging


# Sets up logging based on the input parameter logging. Log level defaults to lo

def setupLogging(logLevel=logging.WARNING):
    logger = logging.getLogger()
    logger.setLevel(logging.DEBUG)
    loggerHandler = logging.StreamHandler(sys.stdout)
    loggerHandler.setLevel(logLevel)
    loggerFormatter = \
        logging.Formatter('%(asctime)s - %(levelname)s - %(message)s')
    loggerHandler.setFormatter(loggerFormatter)
    logger.addHandler(loggerHandler)


if __name__ == '__main__':
    setupLogging()


```

2.3 prescription.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import time
import sys
import requests
import json
import cv2 as cv
import numpy as np
import heapq
from reportlab.pdfgen import canvas
import setupLogging
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
import pickle as pkl
from utils.binary import *
```

```
sys.path.insert(0, '../autocorrect/')
from autocorrect import correctPage
# import autocorrect
'''
The prescription calss holds the prescription image and has all mutators to it
'''
class prescription:

    imagePath = ''
    wordROI = []
    wordROIList = []
    correctedWordROIList = []
    height = 0

    def __init__(self, imagePath):
        self.imagePath = imagePath
        self.c = canvas.Canvas('../temp/output/result.pdf')
        self.pdf = canvas.Canvas('../temp/output/finalResult.pdf')
    # From the detrected words recreates the image with digital version of text.
    def azureCVDispProcessing(self, analysis):
        image_path = self.imagePath
        polygons = [(line['boundingBox'], line['text']) for line in
                    analysis['recognitionResult']['lines']]
        self.azurePolygons = polygons
        img_path = str(image_path)
        img = cv.imread(img_path)
        (height, _width, _channels) = img.shape
        self.height = height
        bg_img = img
        for polygon in polygons:
            vertices = [(polygon[0][i], polygon[0][i + 1]) for i in
                        range(0, len(polygon[0]), 2)]
            cv.fillPoly(bg_img, pts=np.int32([vertices]), color=(255,
                        255, 255))
        self.c.setPageSize((_width, height))
        self.pdf.setPageSize((_width, height))
        cv.imwrite('../temp/bg_img.jpg', bg_img)
        self.c.drawImage('../temp/bg_img.jpg', 0, 0)
        self.pdf.drawImage('../temp/bg_img.jpg', 0, 0)
        for polygon in polygons:
            vertices = [(polygon[0][i], polygon[0][i + 1]) for i in
                        range(0, len(polygon[0]), 2)]
            text = polygon[1]
            min_x = min(vertices, key=lambda t: t[0])[0]
            min_y = min(vertices, key=lambda t: t[1])[1]
            max_x = max(vertices, key=lambda t: t[0])[0]
```

```
64              max_y = max(vertices, key=lambda t: t[1])[1]
65              cv.rectangle(img, (min_x, min_y), (max_x, max_y), (255, 255,
66                           255), cv.cv.CV_FILLED)
67              fontThickness = 2
68              if  ((max_y-min_y)*0.02) < 0.5:
69                  fontThickness = 1
70              cv.putText(
71                  img,
72                  text,
73                  (min_x, (min_y + max_y) / 2),
74                  cv.FONT_HERSHEY_SIMPLEX,
75                  (max_y-min_y)*0.015,
76                  (0, 0, 0),
77                  fontThickness,
78                  cv.CV_AA,
79                  )
80              self.c.setFont('Helvetica', 0.5*(max_y-min_y))
81              self.c.drawString(min_x, height - (min_y + max_y) / 2, text)
82          cvImg = cv.cvtColor(img, cv.cv.CV_BGR2RGB)  # for Qt display
83          setupLogging.logging.debug('Image with ROI saved')
84          self.c.save()
85          return cvImg
86      # Handwriting text detection using state of the art method.
87      def imageAzureHandwriting(self):
88          image_path = self.imagePath
89          subscription_key = '00c800bde4fe46b7b36fc42aba617e6b'
90          assert subscription_key
91          vision_base_url = \
92              'https://westcentralus.api.cognitive.microsoft.com/vision/v1.0/'
93          text_recognition_url = vision_base_url + 'RecognizeText'
94
95          # using image in disk
96
97          image_data = open(image_path, 'rb').read()
98          headers = {'Ocp-Apim-Subscription-Key': subscription_key,
99                     'Content-Type': 'application/octet-stream'}
100         params = {'handwriting': True}
101         response = requests.post(text_recognition_url, headers=headers,
102                                  params=params, data=image_data)
103         response.raise_for_status()
104         _operation_url = response.headers['Operation-Location']
105         analysis = {}
106         while not 'recognitionResult' in analysis:
107             setupLogging.logging.info('Polling azure GET')
108             response_final = \
109                 requests.get(response.headers['Operation-Location'],
```

```
                                headers=headers)
            analysis = response_final.json()
            time.sleep(1)
        qimg = self.azureCVDispProcessing(analysis=analysis)
        return (qimg, analysis)
    # Denoises the raw input image of the prescription
    def imageDenoising(self, img):
        img = cv.cvtColor(img, cv.COLOR_RGB2GRAY)
        return img
    # Converts the colour imge of the prescription to a black and white image wi
    def imageBinarization(self, img):
        img_sobel = cv.Sobel(img, cv.CV_8U, 1, 0, 3)
        img_threshold = cv.threshold(img_sobel, 0, 255, cv.THRESH_OTSU
                + cv.THRESH_BINARY)[1]
        img_threshold = 255 - img_threshold
        return img_threshold
    # Detects the line of texts in the binarised image
    def imageLOTDetection(self, img):
        return img
    # Draws ROI into the image from the detected ROIs
    def imageWordROIDetection(self, img):
        imageWordROIDetected = cv.cvtColor(img, cv.COLOR_GRAY2RGB)
        for roi in self.wordROI:
            cv.rectangle(imageWordROIDetected, (roi[0], roi[2]),
                         (roi[1], roi[3]), (0, 255, 0), 2)
        return imageWordROIDetected

    def imageNNWordDetection(self, img):
        return img

    def imageWordSpellcorrection(self):
        img_path = str(self.imagePath)
        img = cv.imread(img_path)
        self.wordListCorrected = correctPage(self.wordList, self.wordROIFlag)
        for i in range(len(self.wordListCorrected)):
            min_x, max_x, min_y, max_y = self.wordROI[i]
            text = self.wordListCorrected[i]
            cv.rectangle(img, (min_x, min_y), (max_x, max_y), (255, 255,
                         255), cv.cv.CV_FILLED)
            fontThickness = 2
            if  ((max_y-min_y)*0.02) < 0.5:
                fontThickness = 1
            cv.putText(
                img,
                text+ '{' + str(self.wordROIFlag[i]) + '}',
                (min_x, (min_y + max_y) / 2),
```

```
156                 cv.FONT_HERSHEY_SIMPLEX,
157                 (max_y-min_y)*0.015,
158                 (0, 0, 0),
159                 fontThickness,
160                 cv.CV_AA,
161                 )
162             self.pdf.setFont('Helvetica', 0.5*(max_y-min_y))
163             self.pdf.drawString(min_x, self.height - (min_y + max_y) / 2, text )
164         self.pdf.save()
165         return img
166     # Using trained deep neural network model to detect split characters
167     def charToNN(self, charImg):
168         cvImgResized = cv.resize(255 - charImg, (50, 50)).reshape(1,
169                 2500)
170 
171         lab = LabelEncoder()
172         l = map(chr, list(range(ord('0'), ord('9') + 1))
173                 + list(range(ord('A'), ord('Z') + 1))
174                 + list(range(ord('a'), ord('z') + 1)))
175         lab.fit(l)
176         len(lab.classes_)
177 
178         mlp = pkl.load(open('../classifier/classifier.bin', 'rb'))
179         return (True, 'i', 0.0)
180     def dpEval(self, dpMatrix):
181         return (0.0, 'a')
182     # Using our improved algorithm for splitting handwritten words into chracter
183     def wordImgToNN(self, wordImg):
184         (height, width) = wordImg.shape
185         windowMinSize = 1
186         windowSize = 1
187         windowSizeStep = 1
188         prevX = 0
189         detectedWord = ''
190         detectionArray = [0]
191         for i in range(0, width):
192             if prevX + windowSize > width:
193                 break
194             (detected, charDetected, _) = self.charToNN(wordImg[0:
195                     height, prevX:prevX + windowSize])
196             if detected == True:
197                 prevX = prevX + windowSize
198                 windowSize = windowMinSize
199                 detectedWord += charDetected
200                 detectionArray.append(prevX)
201             else:
```

```
202                 windowSize += windowSizeStep
203         return (detectedWord, detectionArray)
204     # Uses probabilistic model for finding the most probable word represented by
205     def wordTree(
206         self,
207         startPos,
208         prevProb,
209         dpMatrix,
210         heap,
211         maxAggregation=3,
212         ):
213 
214         for i in range(1, maxAggregation + 1):
215             (_detected, detectedChar, detectionProb) = \
216                 dpMatrix[startPos][i]
217             if len(heap[i + startPos]) > 10:
218                 if heapq.nsmallest(1, heap)[0].first > detectionProb \
219                     * prevProb:
220                     return
221             heapq.heappush(heap, (detectionProb, detectedChar))
222     # Used improved=II method for finding the best probabilistic match of an ROI
223     def wordImgToNNTree(self, wordImg):
224         (height, width) = wordImg.shape
225         windowSize = 10
226         maxAggregation = 3
227         nWindows = width / windowSize + 1
228         self.dpMatrix = [[(False, 'a', 0.0) for _x in range(nWindows)]
229                          for _y in range(nWindows)]
230         for i in range(nWindows):
231             for j in range(1, maxAggregation + 1):
232                 imgToTest = wordImg[0:height, i * windowSize:min(width,
233                                     (i + j) * windowSize)]
234                 (_, _wide) = wordImg.shape
235                 if _wide <= 0:
236                     continue
237                 setupLogging.logging.debug(i * windowSize, (i + j)
238                         * windowSize, width)
239                 (detected, detectedChar, detectionProb) = \
240                     self.charToNN(imgToTest)
241                 self.dpMatrix[i][j] = (detected, detectedChar,
242                         detectionProb)
243 
244         heap = [[(0.0, '')] for i in range(nWindows)]
245         self.wordTree(0, 1, self.dpMatrix, heap, 3)
246     # Takes each ROI and prepares it for CNN input and then implements our impro
247     def wordImgToNNDP(self, wordImg):
```

```
248             (height, width) = wordImg.shape
249             windowSize = 1
250             maxAggregation = 4
251             maxRows = width / windowSize + 1
252             dpMatrix = [[(False, 'a', 0.0) for _x in range(maxRows)]
253                         for _y in range(maxAggregation)]
254             for i in range(0, maxRows):
255                 x = i * windowSize
256                 for j in range(1, maxAggregation):
257                     if x > width:
258                         continue
259                     if x + windowSize * j > width:
260                         continue
261                     (detected, detectedChar, probChar) = \
262                         self.charToNN(wordImg[0:height, x:x + windowSize
263                                               * j])
264                     dpMatrix[i][j] = (detected, detectedChar, probChar)
265 
266             (detectedWord, detectionArray) = self.dpEval(dpMatrix)
267 
268             return (detectedWord, detectionArray)
269 
270         def wordCorrection(self):
271             pass
272         # Takes roi pilygons and makes a list of ROIs and it's bounding rectangles f
273         def imageWordToList(self, bImg):
274             self.wordROIFlag = []
275             self.wordList = []
276             if len(bImg.shape) == 2:
277                 binarisedImg = cv.cvtColor(bImg, cv.COLOR_GRAY2RGB)
278             else:
279                 binarisedImg = bImg
280 
281             for polygon in self.azurePolygons:
282                 vertices = [(polygon[0][i], polygon[0][i + 1]) for i in
283                             range(0, len(polygon[0]), 2)]
284                 _text = polygon[1]
285                 self.wordList.append(_text)
286                 min_x = min(vertices, key=lambda t: t[0])[0]
287                 min_y = min(vertices, key=lambda t: t[1])[1]
288                 max_x = max(vertices, key=lambda t: t[0])[0]
289                 max_y = max(vertices, key=lambda t: t[1])[1]
290                 self.wordROI.append((min_x, max_x, min_y, max_y))
291                 mid = (min_y+max_y)/2
292                 if(mid < 0.3*self.height):
293                     self.wordROIFlag.append(-1)
```

```
294             elif(mid< 0.56*self.height):
295                 self.wordROIFlag.append(0)
296             else:
297                 self.wordROIFlag.append(1)
298             roi = binarisedImg[min_y:max_y, min_x:max_x]
299             self.wordROIList.append(roi)
300         return self.wordROIList
301
302
303
304
```

2.4 window.py

```
1  #!/usr/bin/python
2  # -*- coding: utf-8 -*-
3  from PyQt4 import QtGui, QtCore
4  from PyQt4.QtGui import *
5  from PyQt4.QtCore import *
6  import prescription
7  import setupLogging
8  import qdarkstyle
9  import os
10 import cv2 as cv
11
12
13 class Window(QtGui.QMainWindow):
14
15     image_path = ''
16     # statusBar = None
17     imageSeq = []
18     currentSeq = -1
19     processingComplete = False
20
21     def __init__(self):
22         super(Window, self).__init__()
23         self.setGeometry(50, 50, 1024, 768)
24         desktop = QtGui.QDesktopWidget()
25         self.screenSize = \
26             desktop.availableGeometry(desktop.primaryScreen())
27         self.setFixedSize(1024, 768)
28         self.setWindowTitle('DigiCon')
29
30         self.lbl = QtGui.QLabel(self)
31         self.setCentralWidget(self.lbl)
32         # Setting up progress bar view
```

```
33          self.lbl.progressBar = QtGui.QProgressBar(self)
34          self.lbl.progressBar.setGeometry(QtCore.QRect(20, 20, 1024, 30))
35          self.lbl.progressBar.setRange(0, 1024)
36          self.lbl.progressBar.setProperty('value', 1)
37          self.lbl.progressBar.move(0, 500)
38          self.lbl.progressBar.setVisible(False)
39          # Setting up file open command
40          openFile = QtGui.QAction('&File', self)
41          openFile.setShortcut('Ctrl+O')
42          openFile.setStatusTip('Open File')
43          openFile.triggered.connect(self.file_open)
44          # Setting up status bar
45          self._statusBar = QStatusBar()
46          self.setStatusBar(self._statusBar)
47          self._statusBar.showMessage('Press N for next/ P for previous')
48          mainMenu = self.menuBar()
49          fileMenu = mainMenu.addMenu('&File')
50          fileMenu.addAction(openFile)
51          self.statusBar().setSizeGripEnabled(False)
52          self.statusBar().setVisible(False)
53
54          self.home()
55      # Sets up home page views and buttons
56      def home(self):
57          # Setting up process button
58          self.process_btn = QtGui.QPushButton('Process', self)
59          self.process_btn.clicked.connect(lambda : self.processImage())
60          self.process_btn.resize(120, 30)
61          self.process_btn.move(452, 540)
62          self.process_btn.setVisible(False)
63          # Sets up file open button
64          self.open_btn = QtGui.QPushButton('Open an image', self)
65          self.open_btn.clicked.connect(lambda : self.file_open())
66          self.open_btn.resize(120, 30)
67          self.open_btn.move(452, 540)
68          # Sets up image preview
69          self.image_btn = QtGui.QPushButton('', self)
70          self.image_btn.setVisible(False)
71          # Render view
72          self.show()
73      # File open handler function
74      def file_open(self):
75          # Gettign the image path and rescaling it to reduce further processing t
76          self.image_path = QtGui.QFileDialog.getOpenFileName(self,
77                  'Open File')
78          bigImage = cv.imread(str(self.image_path))
```

```
79          (height, width, _) = bigImage.shape
80          rescaledImg = bigImage # cv.resize(bigImage, (768, 768*height/width))
81          if height>3000 or width>3000:
82              if height > width:
83                  rescaledImg = cv.resize(bigImage, (3000*width/height, 3000))
84                  print 'height', rescaledImg.shape
85              if width > height:
86                  rescaledImg = cv.resize(bigImage, (3000, 3000*height/width))
87                  print 'width', rescaledImg.shape
88          else:
89              rescaledImg = bigImage
90              print 'none', rescaledImg.shape
91          (height, width, _) = rescaledImg.shape
92          cv.imwrite('../temp/output/input.jpg', rescaledImg)
93          self.image_path = QtCore.QString('../temp/output/input.jpg')
94          self.prescriptionInstance = \
95              prescription.prescription(str(self.image_path))
96          self.prescriptionInstance.height = height
97          setupLogging.logging.debug('Image path is' + self.image_path)
98          # Triggering some GUI changes on file opening
99          icon = QtGui.QIcon()
100         _inp = QtGui.QPixmap('../temp/output/input.jpg')
101         inp = _inp.scaled(250, 420, QtCore.Qt.KeepAspectRatio)
102         icon.addPixmap(inp)
103         self.image_btn.setIcon(icon)
104         self.image_btn.setIconSize(inp.rect().size())
105         self.image_btn.resize(250, 420)
106         self.image_btn.move(412, 40)
107         self.image_btn.setVisible(True)
108
109         self.open_btn.setVisible(False)
110         self.lbl.progressBar.setVisible(True)
111         self.process_btn.setVisible(True)
112     # Progress update handler
113     def progressBarUpdate(self):
114         self.lbl.progressBar.setValue(self.progressBarCurrent)
115         self.progressBarCurrent += self.progressBarIncrement
116         self.lbl.progressBar.repaint()
117     # The sequence of output handler
118     def imageSeqHandler(self, _cvImg):
119         if len(_cvImg.shape) == 2:
120             cvImg = prescription.cv.cvtColor(_cvImg,
121                     prescription.cv.COLOR_GRAY2RGB)
122         else:
123             cvImg = _cvImg
124         (height, width, channel) = cvImg.shape
```

```
125             bytesPerLine = channel * 3  # Error prone in case of binarized images
126             _qImg = QtGui.QImage(cvImg, width, height, bytesPerLine,
127                                  QtGui.QImage.Format_RGB888)
128             self.imageSeq.append(cvImg)
129         # Hnadles the bookkeeping after each processing step like saving intermediat
130         def processingStepsHandler(self, cvImg):
131             self.imageSeqHandler(cvImg)
132             self.progressBarUpdate()
133         # Top level processing order handler. Calls functions upon the input prescri
134         def processImage(self):
135             self.progressBarIncrement = 1024 / 8
136             self.progressBarCurrent = self.progressBarIncrement
137 
138             virginImg = prescription.cv.imread(str(self.image_path))
139 
140             self.processingStepsHandler(virginImg)
141             denoisedImg = \
142                 self.prescriptionInstance.imageDenoising(virginImg)
143             self.processingStepsHandler(denoisedImg)
144             binarisedImg = \
145                 self.prescriptionInstance.imageBinarization(denoisedImg)
146             self.processingStepsHandler(binarisedImg)
147             (azuredImg, _azureAnalysis) = \
148                 self.prescriptionInstance.imageAzureHandwriting()
149 
150             _wordROIList = \
151                 self.prescriptionInstance.imageWordToList(binarisedImg)
152             wordROIDetectedImg = \
153                 self.prescriptionInstance.imageWordROIDetection(binarisedImg)
154             wordSpellcorrectedImg = \
155                 self.prescriptionInstance.imageWordSpellcorrection()
156             self.processingStepsHandler(wordROIDetectedImg)
157             self.processingStepsHandler(azuredImg)
158             self.prescriptionInstance.wordCorrection()
159             self.processingStepsHandler(wordSpellcorrectedImg)
160 
161             self.processingComplete = True
162             (_height, _width, _) = self.imageSeq[0].shape
163             self.saveIntermediateImgs()
164             self.statusBar().setVisible(True)
165             self.rightKeyHandler()
166             self.adjustSize()
167             self.image_btn.setVisible(False)
168             self.process_btn.setVisible(False)
169             self.lbl.progressBar.setVisible(False)
170         # Display/GUI changes on event handled by this function
```

```
171         def dispalyHandler(self):
172             self._statusBar.showMessage('Press N for next/ P for previous     Showin
173             setupLogging.logging.debug('display handler called')
174 
175             (currentWidth, currentHeight, _) = \
176                 self.imageSeq[self.currentSeq].shape
177             newHeight = int(768 * currentHeight / currentWidth)
178             scaledImage = cv.resize(self.imageSeq[self.currentSeq],
179                                     (newHeight, 768))
180 
181             prescription.cv.imwrite('../temp/disp.jpg', scaledImage)
182             self.lbl.setPixmap(QtGui.QPixmap('../temp/disp.jpg'))
183             self.lbl.repaint()
184             self.setFixedSize(newHeight, 768)
185             self.adjustSize()
186         # P key press event handling helper function
187         def leftKeyHandler(self):
188             if self.processingComplete == False:
189                 return
190             if self.currentSeq == 0:
191                 return
192             self.currentSeq -= 1
193             self.dispalyHandler()
194         # N key press event handling helper function
195         def rightKeyHandler(self):
196             if self.processingComplete == False:
197                 return
198             if self.currentSeq == len(self.imageSeq) - 1:
199                 return
200             self.currentSeq += 1
201             self.dispalyHandler()
202         # Key press event handler function
203         def keyPressEvent(self, event):
204             setupLogging.logging.debug('keyPressEvent happened',
205                                        self.currentSeq, len(self.imageSeq))
206             if event.key() == QtCore.Qt.Key_P:
207                 setupLogging.logging.info('Left key pressed')
208                 self.leftKeyHandler()
209             elif event.key() == QtCore.Qt.Key_N:
210                 setupLogging.logging.info('Right key pressed')
211                 self.rightKeyHandler()
212             event.accept()
213         # Makes a directory if it does not exist
214         def makeDirectoryIfDNE(self, directory):
215             if not os.path.exists(directory):
216                 os.makedirs(directory)
```

```
    # Deletes all files and folders in a directory
    def cleanDirectory(self, directory):
        for the_file in os.listdir(directory):
            file_path = os.path.join(directory, the_file)
            try:
                if os.path.isfile(file_path):
                    os.unlink(file_path)
            except Exception, e:
                setupLogging.logging.warning(e)
    # Saves all intemediate output images for debugging
    def saveIntermediateImgs(self):
        directory = '../temp/output/intermediateImgs/'
        self.makeDirectoryIfDNE(directory)
        self.cleanDirectory(directory)
        i = 0
        for img in self.imageSeq:
            cv.imwrite(directory + str(i) + '.jpg', img)
            i += 1
```

2.5 utils

2.5.1 binary.py

```
import cv2 as cv
import numpy as np

def resize(img):
  # resizing image for standardization
  x,y = img.shape[:2]
  x=float(1200/float(x))
  y=float(1200/float(y))
  res = cv.resize(img,None,fx=float(x), fy=float(y), interpolation = cv.INTER_LI
  return res
# ******************* Image smoothing ***********************
def convolutional_blur(img):
  # simple 2D convolutional image filter / averaging
  kernel = np.ones((3,3),np.float32)/25 #creates a 3X3 kernel of ones
  dst = cv.filter2D(img,-1,kernel)
  return dst
def gaussian_blur(img):
  # gaussian blurring
  gaussian = cv.GaussianBlur(img,(3,3),0)
  return gaussian
def median_blur(img):
  # median blurring- highly effective against salt and pepper noise
  median= cv.medianBlur(img,5)
```

```
    return median
def bilateralFilter(img):
    # Bilateral Filtering- highly effective in noise removal while keeping edges s
    bilateral= cv.bilateralFilter(img,9,75,75)
    return bilateral
def smooth_image(img):
    # blur the image to reduce noise
    dst= median_blur(img)
    dst= gaussian_blur(dst)
    dst= bilateralFilter(dst)
    return dst
# ************************* Binarization ****************
def adaptive_thresholding(img):
    # adaptive mean binary threshold
    th4 = cv.adaptiveThreshold(img,255,cv.ADAPTIVE_THRESH_MEAN_C,cv.THRESH_BINARY,
    th5 = cv.adaptiveThreshold(img,255,cv.ADAPTIVE_THRESH_GAUSSIAN_C,cv.THRESH_BIN
    return th5
def otsu_binarisation(img):
    # global thresholding
    ret1,th1 = cv.threshold(img,127,255,cv.THRESH_BINARY)
    # Otsu's thresholding
    ret2,th2 = cv.threshold(img,0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)
    # Otsu's thresholding after Gaussian filtering
    blur = cv.GaussianBlur(img,(3,3),0)
    ret3,th3 = cv.threshold(blur,0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)
    return th3
def hist_equalise(img):
    eq=cv.equalizeHist(img)
    return eq

def binary(img):
# For handwritten notes the sequence of preprocessing should be as follows:-
# resizing
# Clustering if loading in color else load in grayscale
# Image filtering to reduce noise in grayscale image, the choise of filters depe
# Histogram Equalisation
# Thresholding
# Binarization
# Smoothing ----- this is only necessary for handwritten mode, and not for the s
    # img=resize(img)
    th = img
    th = smooth_image(img)
    th = adaptive_thresholding(th)
    th = otsu_binarisation(th)
    th = smooth_image(th)
    return th
```

2.5.2 call_binary.py

```
import cv2
from utils.binary import *
import sys

img = cv2.imread(sys.argv[1],0)
img_gray = img
img_sobel = cv.Sobel(img_gray, cv.CV_8U, 1, 0, 3)
img_threshold = cv.threshold(img_sobel, 0, 255, cv.THRESH_OTSU+cv.THRESH_BINARY)
r1 = binary(img)
r = cv2.threshold(img, 0, 255,cv2.THRESH_BINARY | cv2.THRESH_OTSU)[1]
cv2.namedWindow("ddf",cv2.WINDOW_NORMAL)
cv2.imshow("ddf",r)
cv2.namedWindow("ddf2",cv2.WINDOW_NORMAL)
cv2.imshow("ddf2",r1)
cv2.namedWindow("ddf2er",cv2.WINDOW_NORMAL)
cv2.imshow("ddf2er",img_threshold)

cv2.waitKey(0)
```

3 classifier

3.1 classifier.py

```
import numpy as np
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from matplotlib import pyplot as plt
import cv2 as cv
import pickle as pkl
import gzip

f = gzip.open("fetr1.txt.gz")

a = np.ndarray([55*62, 50*50])
b = np.ndarray([55*62], dtype=int)
for i, lin in enumerate(f):
    lin = lin.strip().split()
    b[i] = lin[0]
    for j in range(2500):
        a[i][j] = lin[1+j]
# Split the dataset into test and training set
xtr, xte, ytr, yte = train_test_split(a, b, test_size=.05)
```

```
# Scaling the dataset to managable size
s = StandardScaler()
s.fit(xtr)
xtrn = s.transform(xtr)
xten = s.transform(xte)
# Encoding the probabilites to the corresponding character
lab = LabelEncoder()
l = map(chr, list(range(ord('0'), ord('9')+1))+list(range(ord('A'), ord('Z')+1))
lab.fit(l)
len(lab.classes_)
# Defining the classifier and training
mlp = MLPClassifier(solver='lbfgs', max_iter=1000, hidden_layer_sizes=(100))
mlp.fit(xtrn, ytr)
pkl.dump((mlp, s, lab), open('classifier.bin', 'wb'))
# Testing for presicting accuracy ~ 73%
sorted(mlp.predict_proba(xten)[0])
np.sum(mlp.predict(xten)==yte)/float(len(yte))
np.sum(mlp.predict(xtrn)==ytr), len(ytr)

for i, j in zip(mlp.predict(xten), yte):
    print lab.inverse_transform(int(i)), lab.inverse_transform(j)

lab.inverse_transform(mlp.predict(s.transform(cv.resize(255-cv.imread('p.png', 0

for i,j in enumerate(mlp.predict_proba(s.transform(cv.resize(255-cv.imread('p.pn
    print lab.inverse_transform(i), j

for i,j in enumerate(mlp.predict_proba(xten)[0]):
    print lab.inverse_transform(i), j

data = np.zeros([50, 50])
n = 3
xte[0][1000]
for i in range(50):
    for j in range(50):
        data[i][j] = (cv.resize(255-cv.imread('p.png', 0), (50, 50)).reshape(1,
plt.imshow(data, interpolation='nearest')

if True:
    for i in range(50):
        for j in range(50):
            data[i][j] = a[lab.transform(['p'])[0]*55+5][i*50+j]
    plt.imshow(data, interpolation='nearest')

cv.imshow('N', cv.resize(255-cv.imread('N.png', 0), (50, 50)))

```

```
if True:
    import gzip
    import shutil
    with open('classifier/fetr1.txt', 'rb') as f_in, gzip.open('classifier/fetr1
        shutil.copyfileobj(f_in, f_out)
```

4 autocorrect

4.1 word.py

```
# Python 3 Spelling Corrector
#
# Copyright 2014 Jonas McCallum.
# Updated for Python 3, based on Peter Norvig's
# 2007 version: http://norvig.com/spell-correct.html
#
# Open source, MIT license
# http://www.opensource.org/licenses/mit-license.php
"""
Word based methods and functions

Author: Jonas McCallum
https://github.com/foobarmus/autocorrect

"""
from autocorrect.utils import concat
from autocorrect.nlp_parser import NLP_WORDS, MED_WORDS
from autocorrect.word_lists import LOWERCASE, MIXED_CASE
from autocorrect.word_lists import LOWERED, CASE_MAPPED, MEDICINE, SYMPTOMS, ENG

ALPHABET = 'abcdefghijklmnopqrstuvwxyz'
KNOWN_WORDS = LOWERCASE | LOWERED | NLP_WORDS

class Word(object):
    """container for word-based methods"""

    def __init__(self, word):
        """
        Generate slices to assist with typo
        definitions.

        'the' => (('', 'the'), ('t', 'he'),
                  ('th', 'e'), ('the', ''))

        """
```

```
        word_ = word.lower()
        slice_range = range(len(word_) + 1)
        self.slices = tuple((word_[:i], word_[i:])
                            for i in slice_range)
        self.word = word

    def _deletes(self):
        """th"""
        return {concat(a, b[1:])
                for a, b in self.slices[:-1]}

    def _transposes(self):
        """teh"""
        return {concat(a, reversed(b[:2]), b[2:])
                for a, b in self.slices[:-2]}

    def _replaces(self):
        """tge"""
        return {concat(a, c, b[1:])
                for a, b in self.slices[:-1]
                for c in ALPHABET}

    def _inserts(self):
        """thwe"""
        return {concat(a, c, b)
                for a, b in self.slices
                for c in ALPHABET}

    def typos(self):
        """letter combinations one typo away from word"""
        return (self._deletes() | self._transposes() |
                self._replaces() | self._inserts())

    def double_typos(self):
        """letter combinations two typos away from word"""
        return {e2 for e1 in self.typos()
                for e2 in Word(e1).typos()}

    def triple_typos(self):
        return {e3 for e2 in self.double_typos()
                for e3 in Word(e2).typos()}

def isEnglish(words):
    return set(words) & ENGLISH

def isMedicine(words):
```

```
    return (set(words) & MEDICINE) | (set(words) & MED_WORDS)

def isSymptom(words):
    return set(words) & SYMPTOMS

def common(words):
    """{'the', 'teh'} => {'the'}"""
    return set(words) & NLP_WORDS

def exact(words):
    """{'Snog', 'snog', 'Snoddy'} => {'Snoddy'}"""
    return set(words) & MIXED_CASE

def known(words):
    """{'Gazpacho', 'gazzpacho'} => {'gazpacho'}"""
    return {w.lower() for w in words} & KNOWN_WORDS

def known_as_lower(words):
    """{'Natasha', 'Bob'} => {'bob'}"""
    return {w.lower() for w in words} & LOWERCASE

def get_case(word, correction):
    """
    Best guess of intended case.

    manchester => manchester
    chilton => Chilton
    AAvTech => AAvTech
    THe => The
    imho => IMHO

    """
    if word.istitle():
        return correction.title()
    if word.isupper():
        return correction.upper()
    if correction == word and not word.islower():
        return word
    if len(word) > 2 and word[:2].isupper():
        return correction.title()
    if not known_as_lower([correction]): #expensive
        try:
            return CASE_MAPPED[correction]
        except KeyError:
            pass
    return correction
```

4.2 word_lists.py

```
# Python 3 Spelling Corrector
#
# Copyright 2014 Jonas McCallum.
# Updated for Python 3, based on Peter Norvig's
# 2007 version: http://norvig.com/spell-correct.html
#
# Open source, MIT license
# http://www.opensource.org/licenses/mit-license.php
"""
Word lists for case sensitive/insensitive lookups

Author: Jonas McCallum
https://github.com/foobarmus/autocorrect

"""
from autocorrect.utils import words_from_archive

# en_US_GB_CA is a superset of US, GB and CA
# spellings (color, colour, etc). It contains
# roughly half a million words. For this
# example, imagine it's just seven words...
#
# we (lower)
# flew (lower)
# to (lower)
# Abu (mixed)
# Dhabi (mixed)
# via (lower)
# Colombo (mixed)

LOWERCASE = words_from_archive('en_US_GB_CA_lower.txt')
# {'we', 'flew', 'to', 'via'}

CASE_MAPPED = words_from_archive('en_US_GB_CA_mixed.txt',
                                 map_case=True)

MEDICINE = words_from_archive('Medicines.txt')
SYMPTOMS = words_from_archive('Symptoms.txt')
ENGLISH = words_from_archive('english.txt')
#  {abu': 'Abu',
#  'dhabi': 'Dhabi',
#  'colombo': 'Colombo'}
#
# Note that en_US_GB_CA_mixed.txt also contains
```

```
# acronyms/mixed case variants of common words,
# so in reality, CASE_MAPPED also contains:
#
# {'to': 'TO',
#  'via': 'Via'}

MIXED_CASE = set(CASE_MAPPED.values())
# {'Abu', 'Dhabi', 'Colombo'}

LOWERED = set(CASE_MAPPED.keys())
# {'abu', 'dhabi', 'colombo'}
```

4.3 utils.py

```
# Python 3 Spelling Corrector
#
# Copyright 2014 Jonas McCallum.
# Updated for Python 3, based on Peter Norvig's
# 2007 version: http://norvig.com/spell-correct.html
#
# Open source, MIT license
# http://www.opensource.org/licenses/mit-license.php
"""
File reader, concat function and dict wrapper

Author: Jonas McCallum
https://github.com/foobarmus/autocorrect

"""
import re, os, tarfile
from contextlib import closing
from itertools import chain

PATH = os.path.abspath(os.path.dirname(__file__))
BZ2 = 'words.bz2'
RE = '[A-Za-z]+'

def words_from_archive(filename, include_dups=False, map_case=False):
    """extract words from a text file in the archive"""
    bz2 = os.path.join(PATH, BZ2)
    tar_path = '{}/{}'.format('words', filename)
    with closing(tarfile.open(bz2, 'r:bz2')) as t:
        with closing(t.extractfile(tar_path)) as f:
            words = re.findall(RE, f.read().decode(encoding='utf-8'))
            # words = words.encode('utf-8')
    if include_dups:
```

```
        return words
    elif map_case:
        return {w.lower():w for w in words}
    else:
        return set(words)

def concat(*args):
    """reversed('th'), 'e' => 'hte'"""
    try:
        return ''.join(args)
    except TypeError:
        return ''.join(chain.from_iterable(args))

class Zero(dict):
    """dict with a zero default"""

    def __getitem__(self, key):
        return self.get(key)

    def get(self, key):
        try:
            return super(Zero, self).__getitem__(key)
        except KeyError:
            return 0

zero_default_dict = Zero
```

4.4 nlp_parser.py

```
# Python 3 Spelling Corrector
#
# Copyright 2014 Jonas McCallum.
# Updated for Python 3, based on Peter Norvig's
# 2007 version: http://norvig.com/spell-correct.html
#
# Open source, MIT license
# http://www.opensource.org/licenses/mit-license.php
"""
NLP parser

Author: Jonas McCallum
https://github.com/foobarmus/autocorrect

"""
from autocorrect.utils import words_from_archive, zero_default_dict

```

```
def parse(lang_sample):
    """tally word popularity using novel extracts, etc"""
    words = words_from_archive(lang_sample, include_dups=True)
    counts = zero_default_dict()
    for word in words:
        counts[word] += 1
    return set(words), counts

NLP_WORDS, NLP_COUNTS = parse('big.txt')
MED_WORDS, MED_COUNTS = parse('medCorpus.txt')
# parse('big.txt')
```

4.5 autocorrect.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
"""
Spell function

"""
from itertools import chain
from itertools import izip
from nlp_parser import NLP_COUNTS, MED_COUNTS
from word import Word, common, exact, known, get_case, \
    isMedicine, isSymptom, isEnglish


def spell(word):
    """most likely correction for everything up to a double typo"""

    w = Word(word)
    candidates = common([word]) or exact([word]) or known([word]) \
        or known(w.typos()) or common(w.double_typos())

    if len(candidates) is 0:
        return -1
    correction = max(candidates, key=NLP_COUNTS.get)
    print candidates
    return get_case(word, correction)


def spellMed(word):
    """returns list of words with exact match or up to double typos,
     searching in medical dictionary"""
    w = Word(word)

```

```
    candidates = isMedicine([word])

    if len(candidates) != 0:
        for x in candidates:
            x = x.encode('utf-8')
        return max(candidates, key=MED_COUNTS.get).encode('utf-8')

    candidates = isMedicine(w.typos())

    if len(candidates) is not 0:
        for x in candidates:
            x = x.encode('utf-8')
        return max(candidates, key=MED_COUNTS.get).encode('utf-8')

    candidates = isMedicine(w.double_typos())

    if len(candidates) is not 0:
        for x in candidates:
            x = x.encode('utf-8')
        return max(candidates, key=MED_COUNTS.get).encode('utf-8')

    # candidates = (isMedicine([word]) or isMedicine(w.typos()) or isMedicine(w.

    if len(candidates) is 0:
        return -1


def spellSymp(word):
    """returns list of words with exact match or up to double typos,
     searching in symptoms dictionary"""

    w = Word(word)

    candidates = isSymptom([word])

    if len(candidates) is not 0:
        return candidates

    candidates = isSymptom(w.typos())

    if len(candidates) is not 0:
        return candidates

    candidates = isSymptom(w.double_typos())

    if len(candidates) is not 0:
```

```
79          return candidates
80
81      # candidates = (isMedicine([word]) or isMedicine(w.typos()) or isMedicine(w.
82
83      if len(candidates) is 0:
84          return -1
85
86
87  def spellEnglish(word):
88      """returns list of words with exact match or up to double typos,
89       searching in Englsh dictionary"""
90
91
92      w = Word(word)
93
94      candidates = isEnglish([word])
95
96      if len(candidates) is not 0:
97          return max(candidates, key=NLP_COUNTS.get).encode('utf-8')
98
99      candidates = isEnglish(w.typos())
100
101     if len(candidates) is not 0:
102         return max(candidates, key=NLP_COUNTS.get).encode('utf-8')
103
104     candidates = isEnglish(w.double_typos())
105
106     if len(candidates) is not 0:
107         return max(candidates, key=NLP_COUNTS.get).encode('utf-8')
108
109     # candidates = (isMedicine([word]) or isMedicine(w.typos()) or isMedicine(w.
110
111     if len(candidates) is 0:
112         return -1
113
114
115
116
117 def concatSlash(*args):
118     """returns single string comprised of iterable args concatenated with a slas
119
120     try:
121         return '/'.join(args)
122     except TypeError:
123         return '/'.join(chain.from_iterable(args))
124
```

```
125
126 def findWord(wordlist, flag):
127     """The function takes a list of probable words and a flag as input.
128     (flag = 1 -> Medicine, flag = 0 -> Symptom)
129     Returns a list of most probable autocorrected words with preference given
130     to the appropriate dictionary"""
131     if flag is 0:
132         for word in wordlist:
133             result = spellSymp(word.encode('utf-8'))
134 
135             # print result, word
136 
137             if result is not -1:
138                 if word in result:
139                     return word.encode('utf-8')
140 
141         for word in wordlist:
142             result = spellEnglish(word.encode('utf-8'))
143 
144             # print result, word
145 
146             if result is not -1:
147                 if word in result:
148                     return word.encode('utf-8')
149 
150         for word in wordlist:
151             result = spellSymp(word.encode('utf-8'))
152             if result is not -1:
153                 return result
154 
155         for word in wordlist:
156             result = spellEnglish(word.encode('utf-8'))
157             if result is not -1:
158                 return result
159 
160         return wordlist[0].encode('utf-8')
161     elif flag is 1:
162 
163         for word in wordlist:  # exact match with medicine
164             result = spellMed(word.encode('utf-8'))
165             if result is not -1:
166                 if word in result:
167                     return word.encode('utf-8')
168 
169         for word in wordlist:  # search in english
170             result = spellEnglish(word.encode('utf-8'))
```

```
171             if result is not -1:
172                 if word in result:
173                     return word.encode('utf-8')
174 
175         for word in wordlist:  # match with medicine
176             result = spellMed(word.encode('utf-8'))
177             if result is not -1:
178                 return result
179 
180         for word in wordlist:
181             result = spellEnglish(word.encode('utf-8'))
182             if result is not -1:
183                 return result
184 
185         return wordlist[0].encode('utf-8')
186 
187 
188 def correctWord(wordlist, flag):
189     """concatenates and returns the output of findWord into a single string"""
190 
191     if len(wordlist) < 3 or flag == -1:
192         return wordlist
193     c = []
194     c.append(wordlist)
195     a = findWord(c, 1)
196     b = concatSlash(a)
197     return b.encode('utf-8')
198 
199 
200 def correctSent(sentence, flag):
201     """Calls correctWord for each word in sentence"""
202     wordlist = sentence.split()
203     result = ''
204     for word in wordlist:
205         result += correctWord(word, flag) + ' '
206 
207     return result
208 
209 def correctPage(sentenceList, flagList):
210     """Calls correctSent with appropriate flag value for each sentence in senten
211     page = []
212     for sentence, flag in izip(sentenceList, flagList):
213         page.append(correctSent(sentence, flag))
214 
215     return page
```