Using Neural Networks for Facial Recognition

New Mexico

Supercomputing Challenge

Final Report

April 5, 2017

Team Number: YWiC 1

School Name: YWiC

Team Members:

Abbi Jones-Alberson

John Cooper

Hannah Himelright

Reema Iqbal

Sponsoring Teacher: Analyssa Martinez

Project Mentor: Rebecca Galves

Team number: YWIC 1

School name: YWiC

The project's area of science: Biophysics

The computer language(s) used: Python

Team members grade(s) in school:

Abbi Jones - Alberson 9th

John Cooper 9th

Reem Iqbal 9th

Hannah Himelright 8th

Team members email addresses:

agalberson@gmail.com jfrancec1@gmail.com hannahhimelright@gmail.com

reemmachine20@gmail.com

Attach the final report (team_xxx_report.doc/team_xxx_report.odt/team_xxx_report.pdf, where xxx is your team number)

Attach the source code

an executive summary that is shorter than one page

a statement of the problem that you have investigated

a description of the method you used to solve your problem

a discussion of how you verified and validated your model

the results of your study

the conclusions you reached by analyzing your results

the software, references, tables, and other products of your work

your most significant achievement on the project

an acknowledgment of the people and organizations that helped you

EXECUTIVE SUMMARY

Over the course of this year, we have been working on creating facial recognition software using an Artificial Neural Network. We feel we've made a lot of progress, even though we're not completely done. We have created a program that can process images, and a neural network that can analyze the processed images to produce an output.

There are obviously flaws in our code, and the neural network can't produce an output for all the images we feed it, but we feel pretty accomplished with what we've done. We're not at the point where we can separate twins with our neural network, as we proposed at the beginning of the year, but we've still done some intense coding. We want to come back next year, and continue working on this program.

We weren't able to finish the final report in time, but we wanted to submit what we have done. As well as the executive summary, we've wrote the the introduction, description, the results, and our neural network.

Introduction

All of us can recognize faces. We can pick out our friends in a crowd, and recognize celebrities on TV. But most computer programs can't do what we can. Scientists created neural networks to simulate the human brain. Information is processed in neural networks from node to node until it produces an output; much like how our brain processes information through neurons (thus the name neural network). Using a neural network, we've attempted to create an artificial intelligence that can identify faces within its visual radar.

If security cameras could pick out people in a crowd, like we could, abductees could be found faster, and criminals could be stopped before they followed through with dangerous activities. On June 22, 2015, Facebook created a neural network that could pick out faces using pictures, regardless of the pose they were in. Instead of creating a program that looked at the proportions of natural facial feature (like we attempted to do), their neural network looked at the styles people wore their hair in, and the way they dressed. There artifical intelligence had an accuracy rate of about 83%, and thus, could sort out people very efficiently. But, we used other variables to create our neural network because styles can easily he changed, and we wanted a program that would identify features that people couldn't change, like the distance between two eyes of their eyes. Our accuracy rate probably won't be nearly as high as Facebook's, but we wanted a neural network that would identify criminals in hiding, not sort out regular people.

Reference:
https://www.theatlantic.com/technology/archive/2015/07/how-good-facial-recognition-technology-government-regulation/397289/

http://docs.opencv.org/trunk/d7/d8b/tutorial_py_face_detection.html
https://singhgaganpreet.wordpress.com/tag/explaining-haar-cascade/

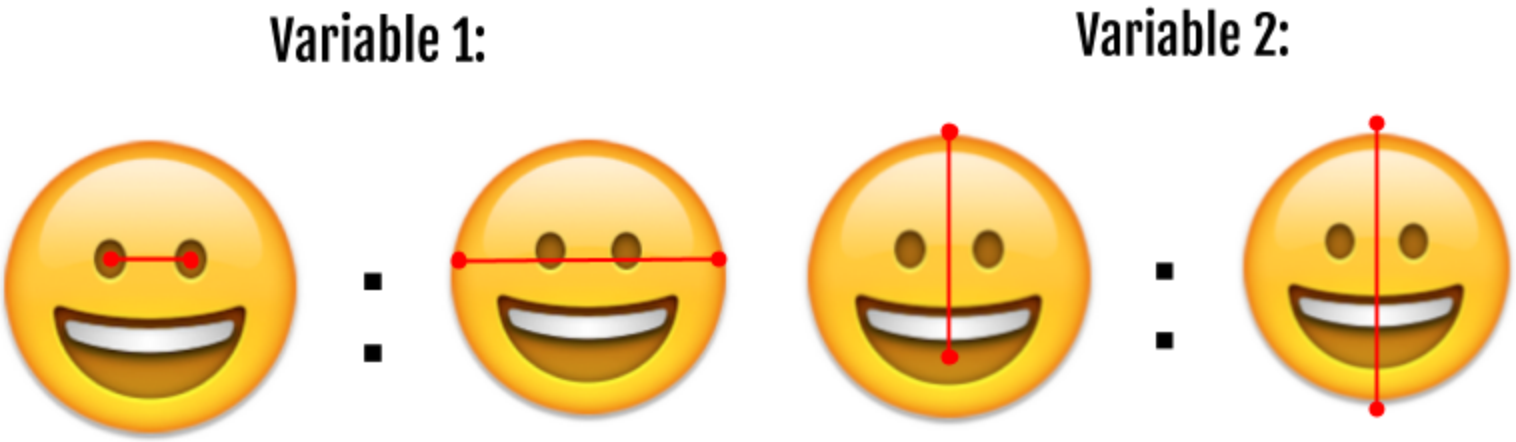
DESCRIPTION

We found that programming our neural network was extraordinarily complicated. To create an effective program, we started by conducting massive research. At the time we weren't sure what software program to use, what variables to use, and how to capture the picture. After researching, we decided to use Python to code our neural network. We considered working with C++ instead, but we favored Python because we were all relatively new to code, and we figured that Python was more simple. We choose our inputs to be the ratio of the distance between both of the eyes compared to the width of the face, and ratio of the distance from the center of the lips to the top of the head, in comparison to the distance from the bottom of the chin, to the top of the head. We coded our neural network using those inputs.

However, not only did we need to program a neural network, we also had to teach the program what an eye was, and where to find a face in a picture. We used Haar Cascades, a downloadable program used to detect faces, for our image processing. Haar Cascades comes with a built-in function called the HaarTraining function. This function teaches the program what a face is by using negative and positive images to train the code. Negative images are images without faces in them. Positive images are pictures of people. In order for an image to be positive, a facial feature must be distinguishable to the program. Coders fed the program what should be positive images, and told the software to categorize the images into either positive or negative (meaning it can't detect a face in the picture) multiple times. Each time the program was told to look for different facial traits. At the end, scientists calculated which traits had the fewest miscalculation errors. That's how the software narrowed down which traits to detect in the program. Haar-Cascades was very convenient, and the program made it possible for us to find the ratios we needed for our neural network.

Overall, we have took big steps towards our goal of creating facial recognition software. Although we weren't able to complete the software, we have made significant progress. Over the course of this year, we have created a neural network, and have found a way to process the images using Haar-Cascades. Our neural network is now producing number outputs for most faces (it can't detect faces in groups, or from a distance but, it's able to detect a good percentage of the images we give it.) We also think there's a hidden fault in the code. Our neural network has been glitching recently. With more time, we can can work towards fixing the

bug. We want to continue working on this project, and we plan on entering in next year's Supercomputing Challenge. In the meantime, we've copied the code we have on this report under *SOFTWARE, REFERENCES, ETC.*



METHOD USED TO SOLVE

Materials: computers, camera

VERIFIED AND VALIDATED

RESULTS

In doing this project we have learned it is very difficult for a computer to recognize a human face and its features straight on and by itself, let alone in a crowd with a strange angle. The computer must first find the face in the image, then locate the features, collect the data, and lastly make sense of it. This is made many times harder when there are multiple faces in an image and some of the faces cannot be clearly seen. ...

SOFTWARE, REFERENCES, ETC

Image Processing

```
import numpy as np
import cv2
import time
import sys
import matplotlib
import matplotlib.pyplot as plt
import cv2
import sys
import numpy as np
import os
import math
import ANN

data = 1
run = 0
X = np.array(([0,0],[0.5,0.5]), dtype=float)
```

```
Y = np.array(([0],[1]), dtype=float)

while(data):
	response = input("Is there new data? (Y/n)")
	if(response != "n"):
		file = input("What is the file name? (.jpg or likewise included)")
		faceCascade = cv2.CascadeClassifier(sys.argv[1])
		#eyeCascade= cv2.CascadeClassifier(sys.argv[2])

		img = cv2.imread(".\Pictures\%s" % file,0)
		eimg = cv2.medianBlur(img,5)
		cimg = cv2.cvtColor(img,cv2.COLOR_GRAY2BGR)
		ecimg = cv2.cvtColor(eimg,cv2.COLOR_GRAY2BGR)
		gimg = cv2.cvtColor(cimg,cv2.COLOR_BGR2GRAY)
		egimg = cv2.cvtColor(ecimg,cv2.COLOR_BGR2GRAY)

		faces = faceCascade.detectMultiScale(img)
		print(faces)
		faceval = faces[0][2]
		face = 0
		for s in range(1, int(len(faces)/4)):
			if(face < faces[s][2]):
				face = s

		for (x, y, w, h) in faces:
			cv2.rectangle(eimg, (x, y), (x+w, y+h), 255, 2)
			roi = eimg[y:y+h, x:x+w]

			#eyes = eyeCascade.detectMultiScale(roi)
			#for (ex,ey,ew,eh) in eyes:
			#	cv2.rectangle(roi,(ex,ey),(ex+ew,ey+eh), 255, 2)

			#print(faces)
```

```
                        #circles =
cv2.HoughCircles(egimg,cv2.HOUGH_GRADIENT,1,20,param1=50,param2=30,minRadius=0,
maxRadius=0)
                        #print(circles)
                        #circles = np.uint16(np.around(circles))

                        lines = None
                        ret,thresh = cv2.threshold(gimg,127,255,0)
                        im2, contours, hierarchy =
cv2.findContours(thresh,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)

                        #print(eyes.size)
                        #for i in circles[0,:]:
                        #       if i[2] < 10:
                        #               cv2.circle(ecimg,(i[0],i[1]),i[2],(0,255,0),2)
                        #               cv2.circle(ecimg,(i[0],i[1]),2,(0,0,255),3)

                emptyimg = cimg
                grayemptyimg = gimg

                #print("Marker")

                #print (emptyimg.shape[0])
                for k in range(0, int(emptyimg.size/3)):
                        emptyimg[k%emptyimg.shape[0], int(k/emptyimg.shape[0]), 0] = 255
                        emptyimg[k%emptyimg.shape[0], int(k/emptyimg.shape[0]), 1] = 255
                        emptyimg[k%emptyimg.shape[0], int(k/emptyimg.shape[0]), 2] = 0

                #print("Marker")

                cv2.drawContours(emptyimg, contours, -1, (0,255,0), 1)
```

```
                #cv2.imshow("empty", emptyimg)
                #cv2.waitKey(0)
                #cv2.destroyAllWindows()

                #lines = cv2.HoughLines(grayemptyimg, float(1), math.pi/180, 100, 0, 0 )

                #print(contours)

                contour_lengths = [contours[0].size/2]

                for m in range(1, int(len(contours))):
                        contour_lengths.append(contours[m].size/2)

                contour_middles = np.zeros(shape=(len(contours), 2))
                maxLength = 0
                eyes = []
                lips = []

                for n in range(0, int(len(contours))):
                        sumx = 0
                        sumy = 0
                        for o in range(0, int(contours[n].size/2)):
                                sumx = sumx + contours[n][o][0]
                                #sumy = sumy + contours[n][o][1]
                        if(int(maxLength) < int(contour_lengths[n])):
                                maxLength = contour_lengths[n]
                                #print(int(maxLength),int(contour_lengths[n]))
                                maxIndex = n
                        contour_middles[n][0] = sumx[0]/(contours[n].size/2)
                        contour_middles[n][1] = sumx[1]/(contours[n].size/2)
                        if(4*(faces[face][3]**(1/2.0)) < int(contour_lengths[n]) and
int(contour_lengths[n]) < 8*(faces[face][3]**(1/2.0))):
                                #print(faces[0][0] + faces[0][2]/2)
```

```
                                #print(contour_middles[n][0])
                                if(faces[face][1] + 3*faces[face][3]/4 > contour_middles[n][1] and
faces[face][1] + faces[face][3]/4 < contour_middles[n][1]):
                                        if(faces[face][3] + faces[face][1] > contour_middles[n][1]
and contour_middles[n][1] > faces[face][1]):
                                                eyes.append(n)
                        if(4*(faces[face][3]**(1/2.0)) < int(contour_lengths[n]) and
int(contour_lengths[n]) < 10*(faces[face][3]**(1/2.0))):
                                #print(faces[0][0] + faces[0][2]/2)
                                #print(contour_middles[n][0])
                                if(faces[face][1] + faces[face][3] > contour_middles[n][1] and
faces[face][1] + faces[face][3]/2 < contour_middles[n][1]):
                                        lips.append(n)

                #print(contour_lengths)
                #print(contour_middles)
                print(eyes)
                #print(lips)
                for p in contour_middles:
                        cv2.circle(ecimg,(int(p[0]),int(p[1])),5,(0,255,0),2)

cv2.circle(ecimg,(int(contour_middles[maxIndex][0]),int(contour_middles[maxIndex][1])),5,(255,2
55,0),2)
                for q in range(0,len(eyes)):

cv2.circle(ecimg,(int(contour_middles[eyes[q]][0]),int(contour_middles[eyes[q]][1])),5,(0,255,255)
,2)
                for r in range(0,len(lips)):

cv2.circle(ecimg,(int(contour_middles[lips[r]][0]),int(contour_middles[lips[r]][1])),5,(255,0,255),2)
                cv2.drawContours(emptyimg, contours[maxIndex], -1, (255,255,255), 1)
```

```
                eye_ratio =
abs((contour_middles[eyes[1]][0]-contour_middles[eyes[0]][0])/faces[face][2])
                lip_height_ratio = (contour_middles[lips[0]][1]-faces[face][1])/(faces[face][3])

                X[run][0] = int(eye_ratio)
                X[run][1] = int(lip_height_ratio)
                Y[run] = input("The number index for whose face this is:")

                #print(cv2.contourArea(contours[maxIndex]))
                #print(maxLength)
                #print(maxIndex)
                #print(len(contours))
                #print(eyes)
                print(eye_ratio)
                print(lip_height_ratio)
                #for r in range(0,len(eyes)):
                #       print(contour_middles[eyes[r]])
                #       print(contour_lengths[eyes[r]])

                #for l in lines[0,:]:
                #   print (i)
                #   cv2.line( cdst, (l[0], l[1]), (l[2], l[3]), (0,0,255), 3, CV_AA)


                #cv2.imshow("empty", emptyimg)
                """
                cv2.imshow("ecimg", ecimg)
                cv2.imshow('detected circles',ecimg)
                cv2.imshow("gimg", gimg)
                cv2.imshow("eimg", eimg)
                """
                #cv2.waitKey(0)
                #cv2.destroyAllWindows()
```

```
                """

                img = cv2.imread(sys.argv[3])
                print(img.shape)
                checkimg = np.ndarray(shape=(img.shape[0],img.shape[1]), dtype=int)
                for i in range(0, img.shape[0]*img.shape[1]):
                        checkimg[i] = img[i, 0] + img[i, 1] + img[i, 2]
                """
                run = run + 1
        else:
                data = 0

N1 = ANN.Neural_Network(Lambda = 0.5)
T1 = ANN.trainer(N1)
T1.train(X,Y)

imageYN = input("Is there an image(1) or are there just numbers?(2):")

if(imageYN != "2"):
                checkFaceName = input("What is the file name for the face to by propogated
through the network?")
                faceCascade = cv2.CascadeClassifier(sys.argv[1])
                #eyeCascade= cv2.CascadeClassifier(sys.argv[2])

                img = cv2.imread(".\Pictures\%s" % checkFaceName,0)
                eimg = cv2.medianBlur(img,5)
                cimg = cv2.cvtColor(img,cv2.COLOR_GRAY2BGR)
                ecimg = cv2.cvtColor(eimg,cv2.COLOR_GRAY2BGR)
                gimg = cv2.cvtColor(cimg,cv2.COLOR_BGR2GRAY)
                egimg = cv2.cvtColor(ecimg,cv2.COLOR_BGR2GRAY)

                faces = faceCascade.detectMultiScale(img)
```

```
		print(faces)
		faceval = faces[0][2]
		face = 0
		for s in range(1, int(len(faces)/4)):
			if(face < faces[s][2]):
				face = s

		for (x, y, w, h) in faces:
			cv2.rectangle(eimg, (x, y), (x+w, y+h), 255, 2)
			roi = eimg[y:y+h, x:x+w]

			#eyes = eyeCascade.detectMultiScale(roi)
			#for (ex,ey,ew,eh) in eyes:
			#	cv2.rectangle(roi,(ex,ey),(ex+ew,ey+eh), 255, 2)

			#print(faces)

			#circles =
cv2.HoughCircles(egimg,cv2.HOUGH_GRADIENT,1,20,param1=50,param2=30,minRadius=0,
maxRadius=0)
			#print(circles)
			#circles = np.uint16(np.around(circles))

			lines = None
			ret,thresh = cv2.threshold(gimg,127,255,0)
			im2, contours, hierarchy =
cv2.findContours(thresh,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)

			#print(eyes.size)
			#for i in circles[0,:]:
			#	if i[2] < 10:
			#		cv2.circle(ecimg,(i[0],i[1]),i[2],(0,255,0),2)
			#		cv2.circle(ecimg,(i[0],i[1]),2,(0,0,255),3)
```

```
emptyimg = cimg
grayemptyimg = gimg

#print("Marker")

#print (emptyimg.shape[0])
for k in range(0, int(emptyimg.size/3)):
        emptyimg[k%emptyimg.shape[0], int(k/emptyimg.shape[0]), 0] = 255
        emptyimg[k%emptyimg.shape[0], int(k/emptyimg.shape[0]), 1] = 255
        emptyimg[k%emptyimg.shape[0], int(k/emptyimg.shape[0]), 2] = 0

#print("Marker")

cv2.drawContours(emptyimg, contours, -1, (0,255,0), 1)

#cv2.imshow("empty", emptyimg)
#cv2.waitKey(0)
#cv2.destroyAllWindows()

#lines = cv2.HoughLines(grayemptyimg, float(1), math.pi/180, 100, 0, 0 )

#print(contours)

contour_lengths = [contours[0].size/2]

for m in range(1, int(len(contours))):
        contour_lengths.append(contours[m].size/2)

contour_middles = np.zeros(shape=(len(contours), 2))
maxLength = 0
eyes = []
lips = []
```

```
		for n in range(0, int(len(contours))):
			sumx = 0
			sumy = 0
			for o in range(0, int(contours[n].size/2)):
				sumx = sumx + contours[n][o][0]
				#sumy = sumy + contours[n][o][1]
			if(int(maxLength) < int(contour_lengths[n])):
				maxLength = contour_lengths[n]
				#print(int(maxLength),int(contour_lengths[n]))
				maxIndex = n
			contour_middles[n][0] = sumx[0]/(contours[n].size/2)
			contour_middles[n][1] = sumx[1]/(contours[n].size/2)
			if(4*(faces[face][3]**(1/2.0)) < int(contour_lengths[n]) and
int(contour_lengths[n]) < 8*(faces[face][3]**(1/2.0))):
				#print(faces[0][0] + faces[0][2]/2)
				#print(contour_middles[n][0])
				if(faces[face][1] + 3*faces[face][3]/4 > contour_middles[n][1] and
faces[face][1] + faces[face][3]/4 < contour_middles[n][1]):
					if(faces[face][3] + faces[face][1] > contour_middles[n][1]
and contour_middles[n][1] > faces[face][1]):
						eyes.append(n)
			if(4*(faces[face][3]**(1/2.0)) < int(contour_lengths[n]) and
int(contour_lengths[n]) < 10*(faces[face][3]**(1/2.0))):
				#print(faces[0][0] + faces[0][2]/2)
				#print(contour_middles[n][0])
				if(faces[face][1] + faces[face][3] > contour_middles[n][1] and
faces[face][1] + faces[face][3]/2 < contour_middles[n][1]):
					lips.append(n)

		#print(contour_lengths)
		#print(contour_middles)
		print(eyes)
```

```
                #print(lips)
                for p in contour_middles:
                        cv2.circle(ecimg,(int(p[0]),int(p[1])),5,(0,255,0),2)

cv2.circle(ecimg,(int(contour_middles[maxIndex][0]),int(contour_middles[maxIndex][1])),5,(255,2
55,0),2)
                for q in range(0,len(eyes)):

cv2.circle(ecimg,(int(contour_middles[eyes[q]][0]),int(contour_middles[eyes[q]][1])),5,(0,255,255)
,2)
                for r in range(0,len(lips)):

cv2.circle(ecimg,(int(contour_middles[lips[r]][0]),int(contour_middles[lips[r]][1])),5,(255,0,255),2)
                cv2.drawContours(emptyimg, contours[maxIndex], -1, (255,255,255), 1)

                eye_ratio =
(contour_middles[eyes[1]][0]-contour_middles[eyes[0]][0])/faces[face][2]
                lip_height_ratio = (contour_middles[lips[0]][1]-faces[face][1])/(faces[face][3])

                #X[run][0] = int(eye_ratio)
                #X[run][1] = int(lip_height_ratio)
                #Y[run] = input("The number index for whose face this is:")

                #print(cv2.contourArea(contours[maxIndex]))
                #print(maxLength)
                #print(maxIndex)
                #print(len(contours))
                #print(eyes)
                print(eye_ratio)
                print(lip_height_ratio)
                #for r in range(0,len(eyes)):
                #       print(contour_middles[eyes[r]])
                #       print(contour_lengths[eyes[r]])
```

```
                #for l in lines[0,:]:
                #    print (i)
                #    cv2.line( cdst, (l[0], l[1]), (l[2], l[3]), (0,0,255), 3, CV_AA)



                #cv2.imshow("empty", emptyimg)
                """
                cv2.imshow("ecimg", ecimg)
                cv2.imshow('detected circles',ecimg)
                cv2.imshow("gimg", gimg)
                cv2.imshow("eimg", eimg)
                """
                #cv2.waitKey(0)
                #cv2.destroyAllWindows()

                """

                img = cv2.imread(sys.argv[3])
                print(img.shape)
                checkimg = np.ndarray(shape=(img.shape[0],img.shape[1]), dtype=int)
                for i in range(0, img.shape[0]*img.shape[1]):
                        checkimg[i] = img[i, 0] + img[i, 1] + img[i, 2]
                """
                go = False
else:
        eye_ratio = float(input("What is the eye ratio?"))
        lip_height_ratio = float(input("What is the lip-height ratio?"))

testFaceMatrix = [eye_ratio, lip_height_ratio]

print(N1.forward(testFaceMatrix))
```

ANN (Artificial Neural Network)

```
import numpy as np
import pylab
import mahotas as mh

class Neural_Network(object):
    def __init__(self, Lambda=0):
        #Define Hyperparameters
        self.inputLayerSize = 2
        self.outputLayerSize = 1
        self.hiddenLayerSize = 3

        #Weights (parameters)
        self.W1 = np.random.randn(self.inputLayerSize,self.hiddenLayerSize)
        self.W2 = np.random.randn(self.hiddenLayerSize,self.outputLayerSize)

        #Regularization Parameter:
        self.Lambda = Lambda

    def forward(self, X):
        #Propogate inputs though network
        self.z2 = np.dot(X, self.W1)
        self.a2 = self.sigmoid(self.z2)
        self.z3 = np.dot(self.a2, self.W2)
        yHat = self.sigmoid(self.z3)
        return yHat

    def sigmoid(self, z):
        #Apply sigmoid activation function to scalar, vector, or matrix
        return 1/(1+np.exp(-z))

    def sigmoidPrime(self,z):
```

```
        #Gradient of sigmoid
        return np.exp(-z)/((1+np.exp(-z))**2)

    def costFunction(self, X, y):
        #Compute cost for given X,y, use weights already stored in class.
        self.yHat = self.forward(X)
        J = 0.5*sum((y-self.yHat)**2)/X.shape[0] +
(self.Lambda/2)*(np.sum(self.W1**2)+np.sum(self.W2**2))
        return J

    def costFunctionPrime(self, X, y):
        #Compute derivative with respect to W and W2 for a given X and y:
        self.yHat = self.forward(X)

        delta3 = np.multiply(-(y-self.yHat), self.sigmoidPrime(self.z3))
        #Add gradient of regularization term:
        dJdW2 = np.dot(self.a2.T, delta3)/X.shape[0] + self.Lambda*self.W2

        delta2 = np.dot(delta3, self.W2.T)*self.sigmoidPrime(self.z2)
        #Add gradient of regularization term:
        dJdW1 = np.dot(X.T, delta2)/X.shape[0] + self.Lambda*self.W1

        return dJdW1, dJdW2

    #Helper functions for interacting with other methods/classes
    def getParams(self):
        #Get W1 and W2 Rolled into vector:
        params = np.concatenate((self.W1.ravel(), self.W2.ravel()))
        return params

    def setParams(self, params):
        #Set W1 and W2 using single parameter vector:
        W1_start = 0
```

```
        W1_end = self.hiddenLayerSize*self.inputLayerSize
        self.W1 = np.reshape(params[W1_start:W1_end], \
                             (self.inputLayerSize, self.hiddenLayerSize))
        W2_end = W1_end + self.hiddenLayerSize*self.outputLayerSize
        self.W2 = np.reshape(params[W1_end:W2_end], \
                             (self.hiddenLayerSize, self.outputLayerSize))

    def computeGradients(self, X, y):
        dJdW1, dJdW2 = self.costFunctionPrime(X, y)
        return np.concatenate((dJdW1.ravel(), dJdW2.ravel()))

def computeNumericalGradient(N, X, y):
        paramsInitial = N.getParams()
        numgrad = np.zeros(paramsInitial.shape)
        perturb = np.zeros(paramsInitial.shape)
        e = 1e-4

        for p in range(len(paramsInitial)):
            #Set perturbation vector
            perturb[p] = e
            N.setParams(paramsInitial + perturb)
            loss2 = N.costFunction(X, y)

            N.setParams(paramsInitial - perturb)
            loss1 = N.costFunction(X, y)

            #Compute Numerical Gradient
            numgrad[p] = (loss2 - loss1) / (2*e)

            #Return the value we changed to zero:
            perturb[p] = 0

        #Return Params to original value:
```

```
        N.setParams(paramsInitial)

        return numgrad

from scipy import optimize


class trainer(object):
    def __init__(self, N):
        #Make Local reference to network:
        self.N = N

    def callbackF(self, params):
        self.N.setParams(params)
        self.J.append(self.N.costFunction(self.X, self.y))

    def costFunctionWrapper(self, params, X, y):
        self.N.setParams(params)
        cost = self.N.costFunction(X, y)
        grad = self.N.computeGradients(X,y)
        return cost, grad

    def train(self, X, y):
        #Make an internal variable for the callback function:
        self.X = X
        self.y = y

        #Make empty list to store costs:
        self.J = []

        params0 = self.N.getParams()

        options = {'maxiter': 200, 'disp' : True}
```

```
        _res = optimize.minimize(self.costFunctionWrapper, params0, jac=True, method='BFGS', \
                                 args=(X, y), options=options, callback=self.callbackF)

        self.N.setParams(_res.x)
        self.optimizationResults = _res
#Credit to Welsh Labs for the help
```

MOST SIGNIFICANT ACHIEVEMENT

ACKNOWLEDGEMENT

We would like to thank YWiC at NMSU and Dr. Cooper for advising and assisting us.