CS 189 Introduction to Machine Learning
Fall 2017 HW13

This homework is due **Friday, December 1 at 10pm.**

1 Getting Started

You may typeset your homework in latex or submit neatly handwritten and scanned solutions. Please make sure to start each question on a new page, as grading (with Gradescope) is much easier that way! Deliverables:

1. Submit a PDF of your writeup to assignment on Gradescope, "HW[n] Write-Up"
2. Submit all code needed to reproduce your results, "HW[n] Code".
3. Submit your test set evaluation results, "HW[n] Test Set".

After you've submitted your homework, be sure to watch out for the self-grade form.

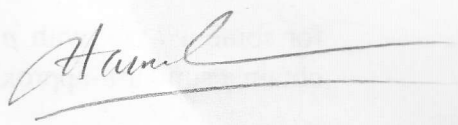
(a) Before you start your homework, write down your team. Who else did you work with on this homework? List names and email addresses. In case of course events, just describe the group. How did you work on this homework? Any comments about the homework?

None - Alone
Comments: check spelling of code. e.g. "accuray" not "accurracy"

(b) Please copy the following statement and sign next to it:

I certify that all solutions are entirely in my words and that I have not looked at another student's solutions. I have credited all external sources in this write up.

I certify that all solutions are entirely in my words and that I have not looked at another student's solutions. I have credited all external sources in this write up.



ATTENTION: THE WHOLE SOLUTION TO HW13 PROB 2 IS BASED ON THE OLD HW13 VERSION. ACCORDING TO THE POLICY, I DO NOT NEED TO ADAPT MY SOLUTION TO THE NEW HW13 VERSION TO GET FULL CREDITS.

Problem #2:

(a) $\sigma(t) = \frac{1}{1+e^{-t}}$

$t \to \infty : e^{-t} \to 0 \Rightarrow \lim_{t\to\infty} \sigma(t) = \frac{1}{1+e^{-t}} = \frac{1}{1+0} = 1$ (1)

$t \to -\infty : e^{-t} \to +\infty \Rightarrow \lim_{t\to-\infty} \sigma(t) = \frac{1}{1+e^{-t}} = \frac{1}{1+\infty} = 0$ (2)

$t_1 > t_2 \Rightarrow -t_1 < -t_2$

$\Rightarrow e^{-t_1} < e^{-t_2}$

$\Rightarrow 1 + e^{-t_1} < 1 + e^{-t_2}$

$\Rightarrow \frac{1}{1+e^{-t_1}} > \frac{1}{1+e^{-t_2}}$

$\sigma(t_1) > \sigma(t_2)$

$\Rightarrow \sigma(t)$ is monotonically increasing (3)

From (1), (2) & (3) $\Rightarrow \sigma(t)$ is bounded by 0 & 1

$\Rightarrow \sigma(t)$ is thresholding function.

$\theta(t) = \text{Relu}(t) - \text{Relu}(t-1) = \max(0, t) - \max(0, t-1)$

3 cases:

1. If $t < 0$: $\theta(t) = 0 - 0 = 0$
2. If $0 \leq t < 1$: $\theta(t) = t - 0 = t$
3. If $t \geq 1$: $\theta(t) = t - (t-1) = 1$

$\Rightarrow t \to \infty$: $\theta(t) \to 1$ $(\theta(t) = 1)$

$t \to -\infty$: $\theta(t) \to 0$

From 3 cases we see that $\theta(t)$ is increasing (not strictly) and bounded by 0 & 1

$\Rightarrow$ $\theta(t)$ is thresholding function

(b) See code attached

(c) We define the set of function :

$$f_k(x) = \tau(\langle w_k, x\rangle + b_k)$$

corresponding to different pairs (w_k, b_k)

Consider the function

$$f^*(x) = \lim_{k\to\infty} f_k(x) = \lim_{k\to\infty} \tau(\langle w_k, x\rangle + b_k)$$

if $k\to\infty$ leads to $\langle w_k, x\rangle + b_k \to \infty$ when $\langle w', x\rangle + b' \geq 0$ and $\langle w_k, x\rangle + b_k \to -\infty$ when $\langle w', x\rangle + b' < 0$

then

$\lim_{k\to\infty} \tau(\langle w_k, x\rangle + b_k) = 1$ for x s.t: $\langle w', x\rangle + b' \geq 0$

$\lim_{k\to\infty} \tau(\langle w_k, x\rangle + b_k) = 0$ for x s.t: $\langle w', x\rangle + b' < 0$

since $\tau(t) \to 1$ for $t\to\infty$ & $\tau(t)\to 0$ for $t\to-\infty$ (definition)

In such case, $f^*(x)$ becomes the step function

Now, our job is to find such w_k, b_k so that it satisfies : $k\to\infty$ leads to $\langle w_k, x\rangle + b_k \to \infty$ if $\langle w', x\rangle + b \geq 0$ and $\langle w_k, x\rangle + b_k \to -\infty$ if $\langle w', x\rangle + b < 0$

We can choose w_k and b_k the scaling-up of w' & b with factor k, i.e $w_k = kw'$, $b_k = kb'$

thus, the closure $cl(\{\tau(\langle w,t\rangle + b) \text{ for some } w, b\})$ include $\lim_{k\to\infty} \tau(\langle w_k, x\rangle + b_k)$ if we choose $w = kw'$, $b = kb'$ and set k as large as possible

This is actually the step function $S(\langle w', x\rangle + b)$, i.e

$$f^*(x) = \lim_{k\to\infty} f_k(x) = \lim_{k\to\infty} \tau(\langle w_k, x\rangle + b_k) = S \ldots$$

Since τ can approach to S as much as we want by scaling up w & b (through k)

$\Rightarrow$ τ is bounded by S

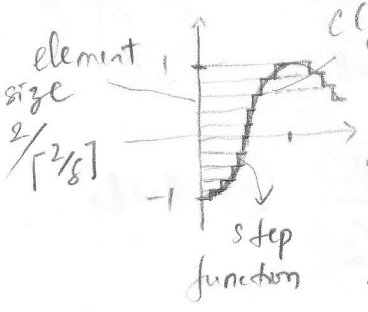
$\Rightarrow$ $cl(\{\tau \text{ for some } w, b\})$ contains S

d.) $c(y') = \cos(\|w\|_1 y')$ where $y' = y / \|w\|_1$,

$c(y')$ has domain $y' \in [-1, 1]$ and range $c(y) \in [-1, 1]$

i.e $c(y')$ is bounded by $[-1, 1]$

Since $c(y')$ is continuous, we can divide the interval $[-1, 1]$ into $\lceil 2/\delta \rceil$ elements, so each element has size $\frac{(1-(-1))}{\lceil 2/\delta \rceil} < \delta$ (see figure)



It does mean, within each element, the jump of $c(y')$ is less than δ.

Thus if we set the step function at the ends of those elements (as in figure), we can decompose $c(y')$ into the combination of step function w/ jump $\leq \delta$.

e.) we have $c' = \lim_{\Delta z \to 0} \frac{c(z_i) - c(z_{i-1})}{z_i - z_{i-1}}$ $\quad \Delta z = z_i - z_{i-1}$

$$\Rightarrow c' \simeq \frac{c(z_i) - c(z_{i-1})}{\Delta z} = \frac{\Delta c(z)}{\Delta z}$$

$$\Rightarrow \Delta c(z) = c' \Delta z$$

$$\int f(z)\,dz = \lim_{\Delta z \to 0} \sum f(z)\,\Delta z$$

$$\Rightarrow \int f(z)\,dz \simeq \sum f(z)\,\Delta z$$

$$\Rightarrow \sum f(z) \simeq \frac{1}{\Delta z} \int f(z)\,dz$$

now substitute $f(z)$ by $\Delta c(z)$:

$$\sum f(z) = \sum \Delta c(z) \simeq \frac{1}{\Delta z} \int \Delta c(z)\,dz$$

$$= \frac{1}{\Delta z} \int c' \Delta z\,dz$$

$$= \int c'\,dz$$

= take the range from $-1 \to 1$

$$\Rightarrow \sum_i |c(z_i) - c(z_{i-1})| \simeq \int_{-1}^{1} |c|'\,dz = |c|\Big|_{-1}^{1}$$

c is periodic function w/ period $2\pi c / \|w\|_1$

$$\Rightarrow |c|\Big|_{-1}^{1} = \text{range of } c \;*\; \#\text{ of periods over } y' \in [-1,1]$$

$$= (1 - (-1)) * \frac{1 - (-1)}{2\pi c / \|w\|_1} \quad \leftarrow \text{range of } y'$$

$\uparrow$ range of c

$$= 2 * \frac{\|w\|_1}{\pi c} = \frac{\|w\|_1}{\pi/2} < \|w\|_1$$

$$\Rightarrow \sum_i |c(z_i) - c(z_{i-1})| < \|w\|_1$$

explanation:
$c(y') = \cos(\|w\|_1 y')$
$\cos()$ has period 2π
$\Rightarrow \|w\|_1 y' = 2\pi$
$\Rightarrow y' = 2\pi / \|w\|_1$ is the period of $c(y')$

(f)

$$F_{\cos} = \{ f : R^d \to R \text{ s.t. } f(x) = \frac{1}{2\|w\|_1} \cos(\langle w, x \rangle) \text{ for } w \neq 0 \}$$

each $f(x) \in F_{\cos}$ can be decomposed into a combination of step functions scaled up with factor $\frac{1}{2\|w\|_1}$

From 2(c) we have:

$$S \subseteq cl(\{\tau\})$$

$$\Rightarrow c_i S \subseteq cl(\{c_i \tau\})$$

$$\Rightarrow \sum c_i S \subseteq cl(\{\sum c_i \tau\})$$

if we choose $c_i \geq 0$ w/ $\sum_i c_i = 1$

then $\sum c_i \tau$ is the convex hull of $\{\tau\}$, and $cl(\{\sum c_i \tau\})$ is the closed convex hull of $\{\tau\}$, i.e

$$\sum c_i S \subseteq \overline{conv}(\{\tau\})$$

now we show that $f(x) \in F_{\cos}$ can be represented by $\sum c_i S_i$ w/ $c_i \geq 0$ $\sum_i c_i = 1$ and S_i is step function

$$f(x) = \frac{1}{2\|w\|_1} \cos(\langle w, x \rangle)$$

From 2(d) we can write:

$$\cos(\langle w, x \rangle) = \sum_i |c(z_i) - c(z_{i-1})| \, S_i(\langle w_i x \rangle + b)$$

$$\Rightarrow f(x) = \sum_i \frac{|c(z_i) - c(z_{i-1})|}{2\|w\|_1} S_i(\ldots)$$

from 2(e) we know that $\sum_i |c(z_i) - c(z_{i-1})|$ is bounded by $\|w\|_1$

$\Rightarrow \sum_i \frac{|c(z_i) - c(z_{i-1})|}{2\|w\|_1}$ is bounded by $\frac{1}{2}$.

WHY $\frac{1}{2}$ BUT NOT 1 ? BECAUSE THIS SOLUTION IS BASED ON THE OLD WRONG HW #13 VERSION AND ACCORDING TO THE POLICY, I STILL GET FULL CREDITS IF MY REASONING MAKES SENSE WITHOUT ADAPTING TO THE NEW HW VERSION.

(g) $E[\|f_p - f\|^2] = E[\int_{x\in[0,1]^d} (f_p - f)^2 dx$

$= \int E[(f_p - f)^2] dx$

$E[(f_p - f)^2] = Var[f_p - f] + E[f_p - f]^2$.

$= Var[f_p] + (E[f_p] - \underbrace{E[f]}_{0})^2$

$= Var[f_p] + E[f_p]^2$

$= E[f_p^2] = E[\frac{1}{p^2}(\sum_{i=1}^{p} G_i)^2]$

$= \frac{1}{p^2} E[(\sum_{i=1}^{p} G_i)^2 \leq \frac{1}{p^2} \cdot \frac{1}{p} \sum (\underbrace{1+\ldots+1}_{p \text{ terms}})^2$

$= \frac{1}{p^2} \times \frac{1}{p} \times p^2 = \frac{1}{p}$

$\Rightarrow E[(f_p - f)^2] \leq \frac{1}{p}$

$\int_{x\in[0,1]^d} E[(f_p - f)]^2 dx \leq \int_{x\in[0,1]^d} \frac{1}{p} dx = \frac{1}{p} \Rightarrow E[\|f_p - f\|^2] \leq \frac{1}{p}$

(h) Since

$$E[\|f_p - f\|^2] \leq \frac{1}{p} \qquad (1)$$

is true for a convex combination of p randomly chosen threshold functions, if there is no deterministic choice of f_p so that (1) is true → this contradicts with 2(g), i.e. the equation (1) is impossible.

this consider

$$E(f,p) = \inf \int_{x \in [0,1]^d} (f(x) - h(x))^2 \, dx$$

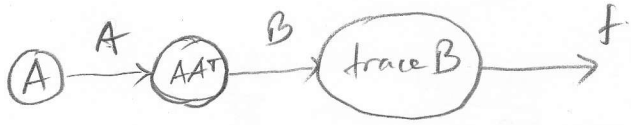
where $h(x) = \sum_1^p c_k \, \sigma(\langle w_k, x \rangle + b_k)$

is such the deterministic choice, and $f(x)$ is any function $f \in F \subseteq \overline{\text{conv}}(\{\pm g\})$ that can be represented by $\sum_1^m c_i \sigma_i$, we have

$$E(f,p) \leq \frac{1}{p}$$

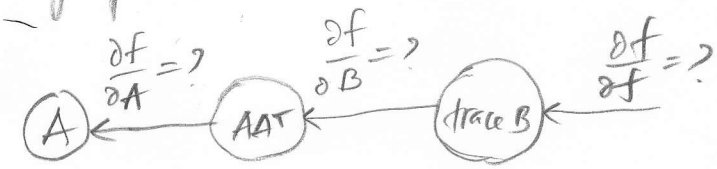
Problem # 4

a.) Given the neural network as below:



given $\sqrt{\text{trace}(AA^T)} = \|A\|_F$

calculate the backward propagation. That is, fill in the ? in the grap below:



Solution:

First: $\frac{\partial f}{\partial f} = 1.$

Second: $\frac{\partial f}{\partial B} = \frac{\partial \text{trace}(AA^T)}{\partial AA^T} = \frac{\partial \sum_{ij}^{n} A_{ij}^2}{\partial AA^T}$

on the diagonal: $AA^T \Rightarrow \sum_j A_{ij}^2 \Rightarrow \frac{\partial f}{\partial B_{ii}} = 1$

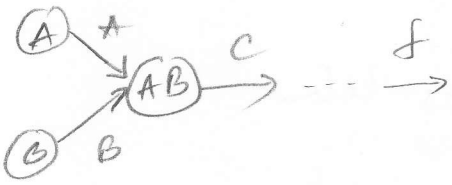
off the diagonal: $AA^T \rightarrow \sum_k A_{ik}A_{jk} \Rightarrow \frac{\partial f}{\partial B_{ij}} = 0$

$\Rightarrow \frac{\partial f}{\partial B} = I$ (identity matrix)

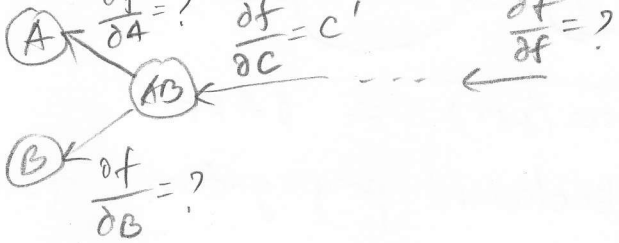
Third: $\frac{\partial f}{\partial A} = \sum_{ij} \underbrace{\frac{\partial f}{\partial B_{ij}}}_{I} \underbrace{\frac{\partial B_{ij}}{\partial A}}_{2A} = 2A$

b.)

Given



Find ? :



Solution

$$\frac{\partial f}{\partial f} = 1$$

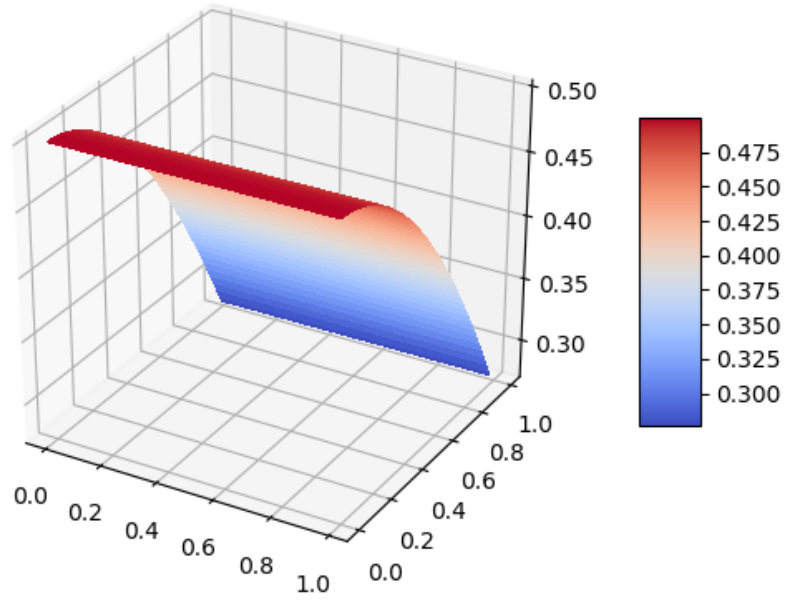
$$\frac{\partial f}{\partial A} = \left[\frac{\partial f}{\partial A_{ij}}\right] = \left[\sum_{k,l} \frac{\partial f}{\partial C_{kl}} \frac{\partial C_{kl}}{\partial A_{ij}}\right]$$

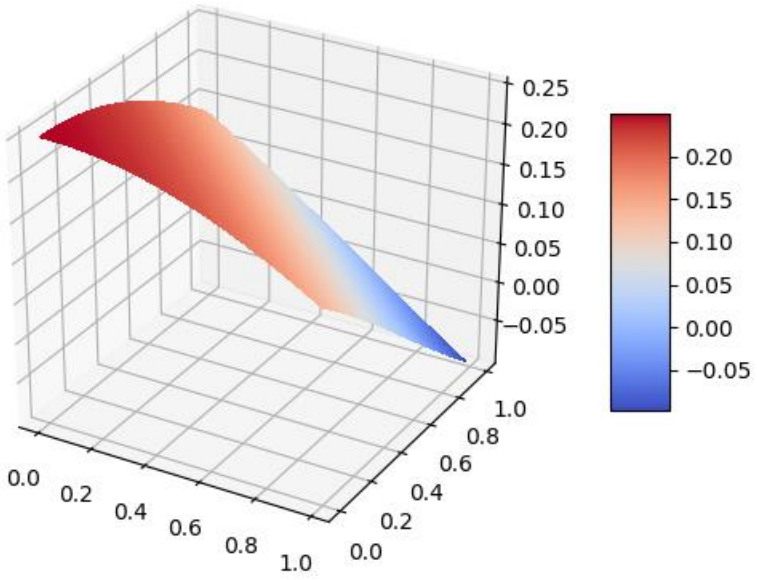
$$= \left[\sum_{l=1}^{p} \frac{\partial f}{\partial C_{il}} \frac{\partial C_{il}}{\partial A_{ij}}\right] = \sum_{l=1}^{p} C'_{il} B_{jl} = C'B^T$$

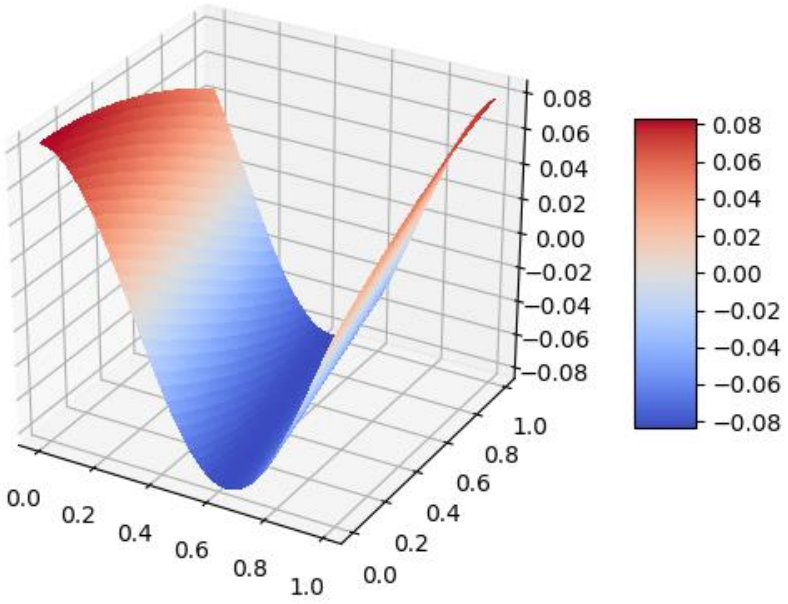
Similarly:

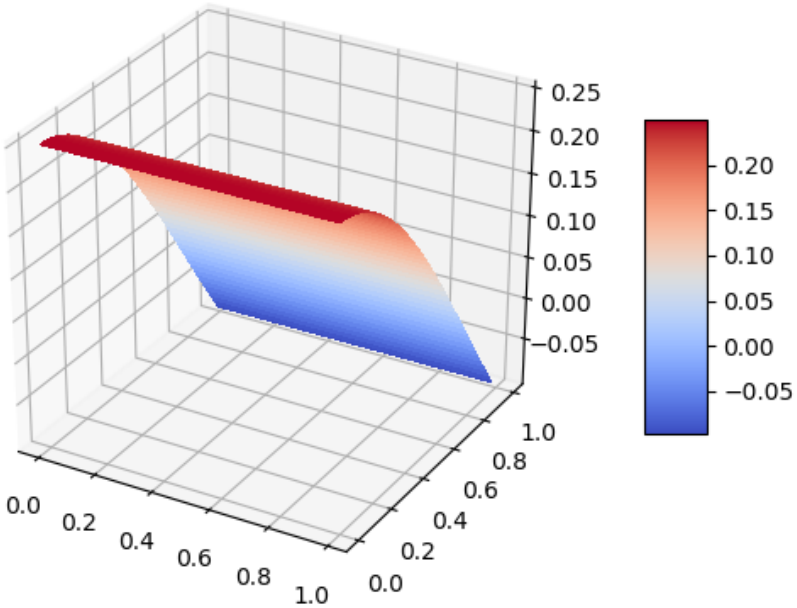
$$\frac{\partial f}{\partial B} = A^T C'$$

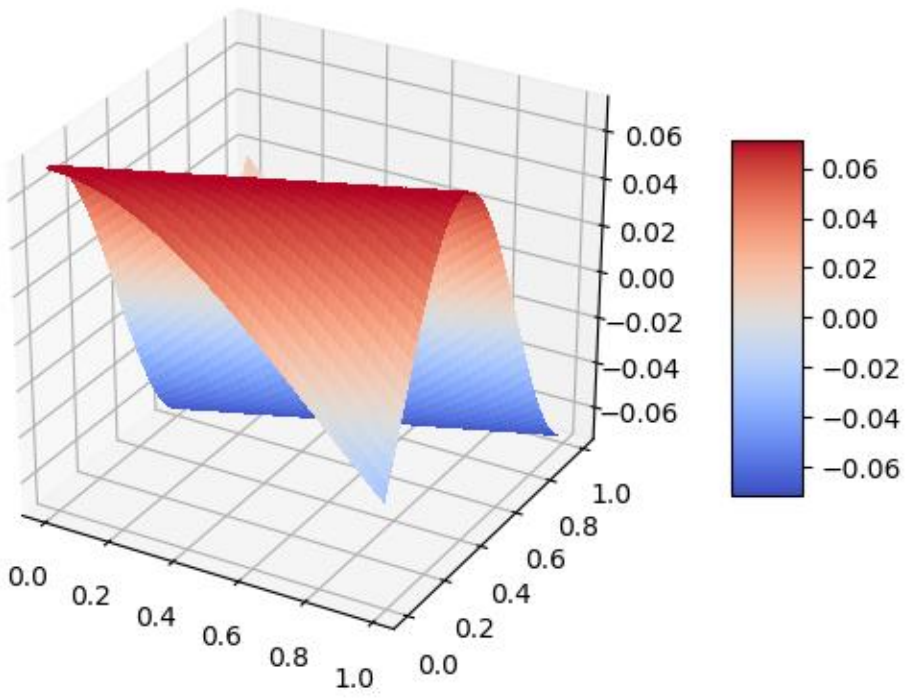
Problem 2(b)


0.50
0.45
0.40
0.35
0.30
0.475
0.450
0.425
0.400
0.375
0.350
0.325
0.300
0.0
0.2
0.4
0.6
0.8
1.0


0.25
0.20
0.15
0.10
0.05
0.00
−0.05
0.20
0.15
0.10
0.05
0.00
−0.05
0.0
0.2
0.4
0.6
0.8
1.0

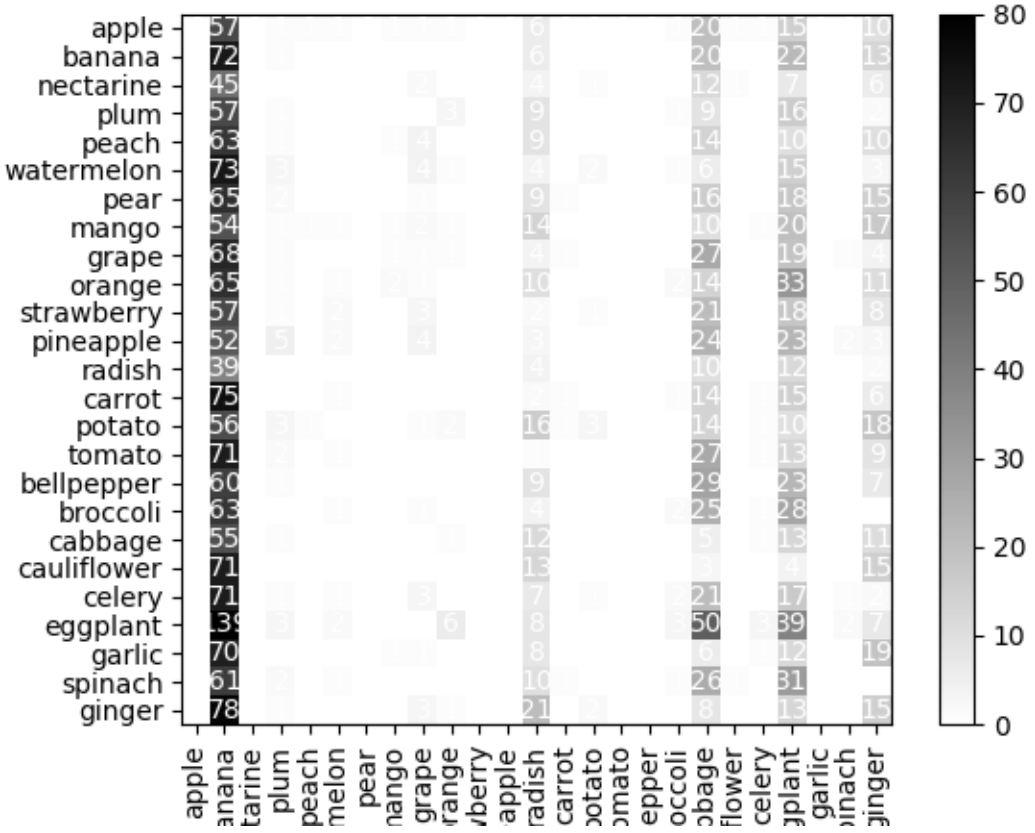

0.08
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
−0.08
0.0
0.2
0.4
0.6
0.8
1.0


0.25
0.20
0.15
0.10
0.05
0.00
−0.05
0.0
0.2
0.4
0.6
0.8
1.0


0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06

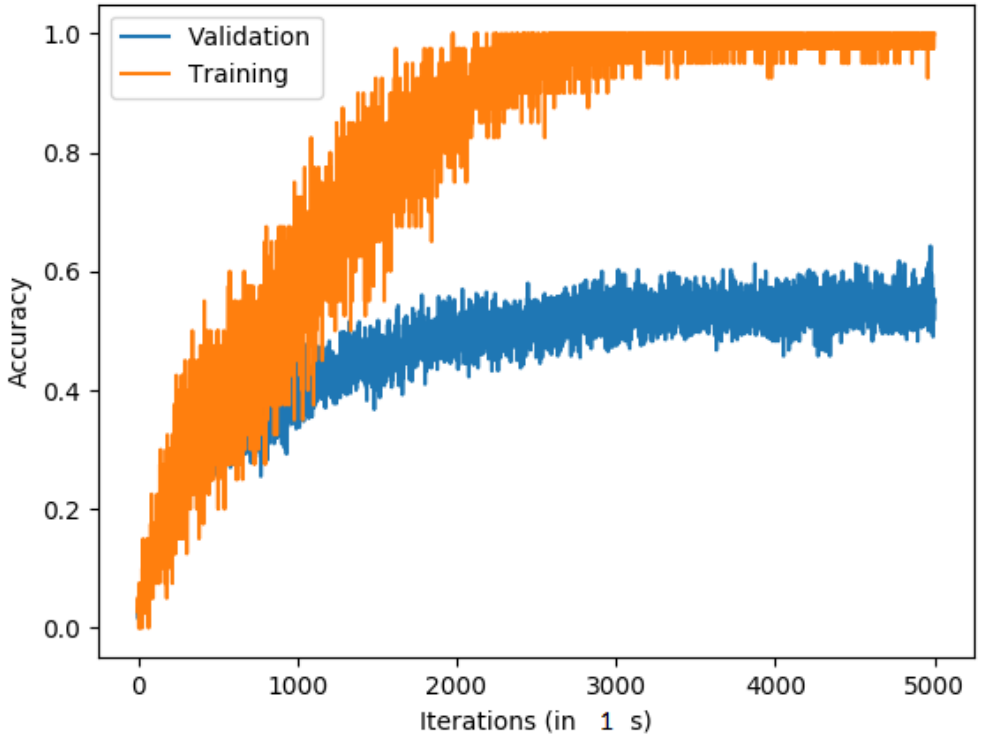
Problem 3

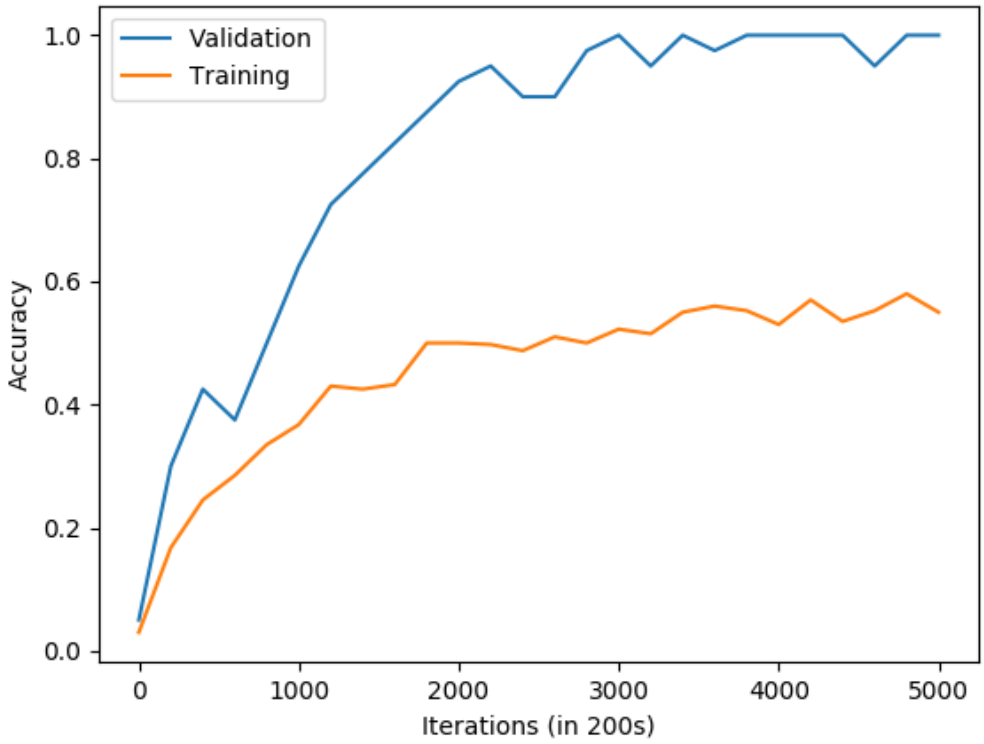
(a) See code attached



(b) See code attached

(c) See code attached


Validation
Training
Accuracy
1.0
0.8
0.6
0.4
0.2
0.0
0
1000
2000
3000
4000
5000
Iterations (in 1 s)


Validation
Training
Accuracy
1.0
0.8
0.6
0.4
0.2
0.0
0
1000
2000
3000
4000
5000
Iterations (in 200s)

(d) See code attached

Image 0

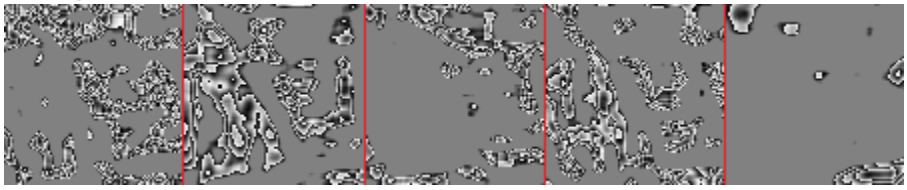


Image 10

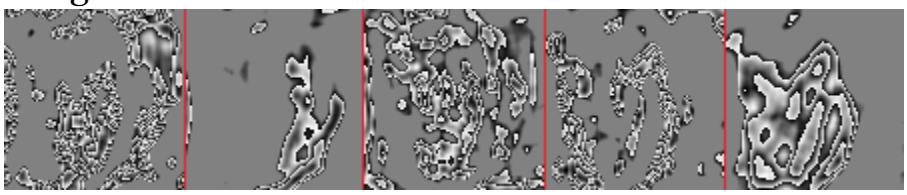
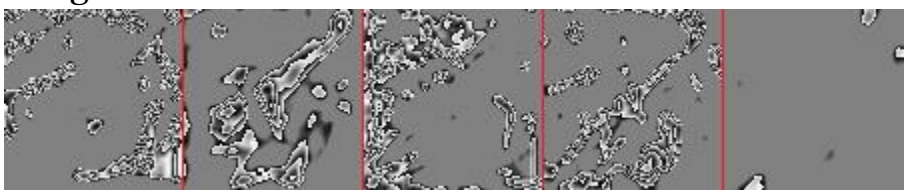
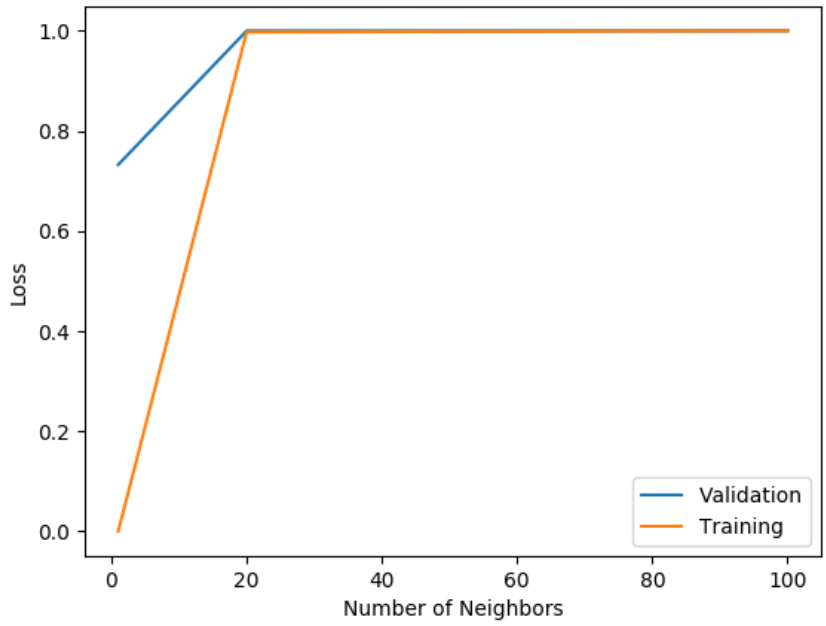


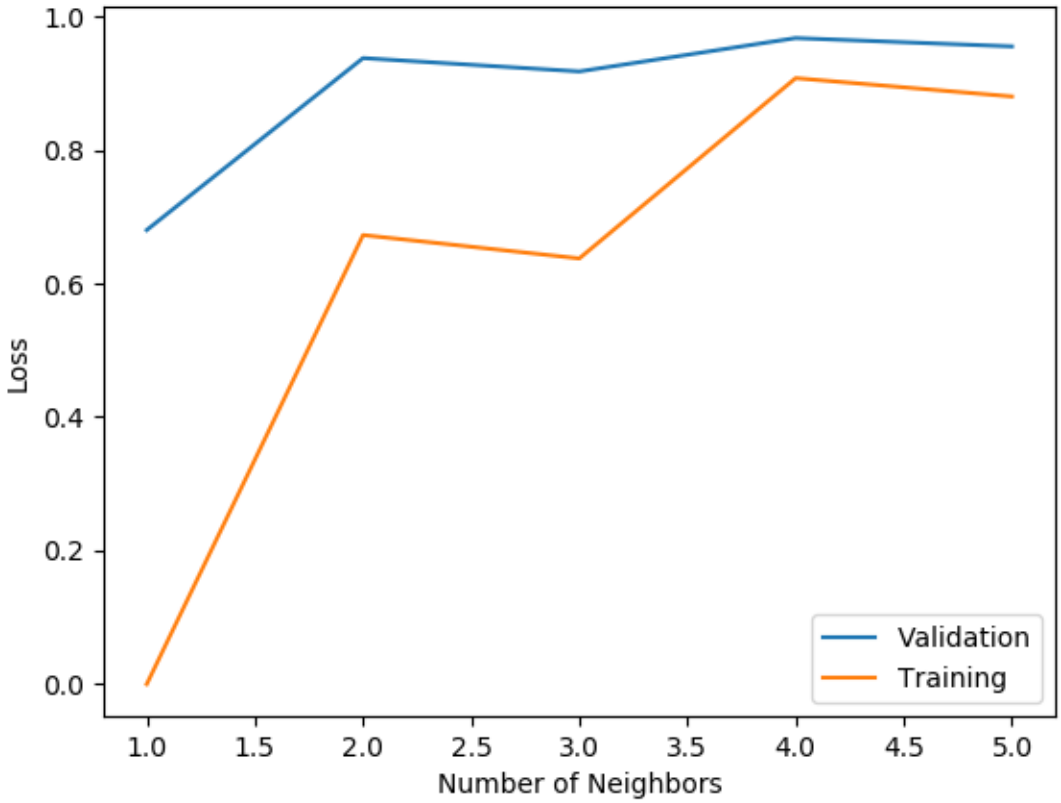
Image 100



(e) See code attached:
K = [1, 20, 100]



K = [1, 2, 3, 4, 5]


1.0
0.8
0.6
0.4
0.2
0.0
Loss
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Number of Neighbors
Validation
Training

```
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
from matplotlib import cm

fig = plt.figure()
ax = fig.gca(projection='3d')

w1 = np.array([0, 1])
w2 = np.array([1, 1])
w3 = np.array([5, 1])
w4 = np.array([0, -2])
w5 = np.array([-2, 5])

x1 = np.arange(0, 1, 0.01)
x2 = np.arange(0, 1, 0.01)
X1, X2 = np.meshgrid(x1, x2)

f1 = 1 / (2*np.sum(abs(w4))) * np.cos( w4[0] * X1 + w4[1] * X2 )
print(X1)
print(X2)
print(f1)

surf = ax.plot_surface(X1, X2, f1, cmap=cm.coolwarm,
                       linewidth=0, antialiased=False)

fig.colorbar(surf, shrink=0.5, aspect=5)

plt.show()


############################


import numpy as np
import tensorflow as tf
#import yolo.config_card as cfg

import IPython

slim = tf.contrib.slim


class CNN(object):

    def __init__(self,classes,image_size):
        '''
        Initializes the size of the network
        '''

        self.classes = classes
        self.num_class = len(self.classes)
        self.image_size = image_size

        self.output_size = self.num_class
        self.batch_size = 40

        self.images = tf.placeholder(tf.float32, [None,

self.image_size,self.image_size,3], name='images')


        self.logits = self.build_network(self.images,
```

```
num_outputs=self.output_size)

        self.labels = tf.placeholder(tf.float32, [None, self.num_class])

        self.loss_layer(self.logits, self.labels)
        self.total_loss = tf.losses.get_total_loss()
        tf.summary.scalar('total_loss', self.total_loss)

    def build_network(self,
                      images,
                      num_outputs,
                      scope='yolo'):

        with tf.variable_scope(scope):
            with slim.arg_scope([slim.conv2d, slim.fully_connected],


weights_initializer=tf.truncated_normal_initializer(0.0, 0.01),
                                weights_regularizer=slim.l2_regularizer(0.0005)):

                '''
                Fill in network architecutre here
                Network should start out with the images function
                Then it should return net
                '''
                net = slim.conv2d(images, 5, [15, 15], scope="conv_0")
                self.response_map_1 = net
                net = slim.max_pool2d(net, [3, 3], scope="pool")
                self.response_map_2 = net
                net = slim.flatten(net, scope="flat_1")
                net = slim.fully_connected(net, 512, scope="fc_2")
                self.response_map_3 = net
                net = slim.fully_connected(net, 25, scope="fc_3")
                self.response_map_4 = net


        return net



    def get_acc(self,y_,y_out):

        '''
        Fill in a way to compute accurracy given two tensorflows arrays
        y_ (the true label) and y_out (the predict label)
        '''

        cp = tf.equal(tf.argmax(y_out,1), tf.argmax(y_,1))

        ac = tf.reduce_mean(tf.cast(cp, tf.float32))

        return ac

    def loss_layer(self, predicts, classes, scope='loss_layer'):
        '''
        The loss layer of the network, which is written for you.
        You need to fill in get_accuracy to report the performance
        '''
        with tf.variable_scope(scope):

            self.class_loss = tf.reduce_mean
```

```
(tf.nn.softmax_cross_entropy_with_logits(labels = classes,logits = predicts))

            self.accurracy = self.get_acc(classes,predicts)



############################


import os
import numpy as np
from numpy.random import random
import cv2

import copy
import glob

import _pickle as pickle
import IPython



class data_manager(object):
    def __init__(self,classes,image_size,compute_features = None, compute_label =

None):

        #Batch Size for training
        self.batch_size = 40
        #Batch size for test, more samples to increase accuracy
        self.val_batch_size = 400

        self.classes = classes
        self.num_class = len(self.classes)
        self.image_size = image_size



        self.class_to_ind = dict(zip(self.classes, range(len(self.classes))))



        self.cursor = 0
        self.t_cursor = 0
        self.epoch = 1

        self.recent_batch = []

        if compute_features == None:
            self.compute_feature = self.compute_features_baseline

        else:
            self.compute_feature = compute_features

        if compute_label == None:
            self.compute_label = self.compute_label_baseline
        else:
            self.compute_label = compute_label


        self.load_train_set()
        self.load_validation_set()
```

```
    def get_train_batch(self):
        '''

        Compute a training batch for the neural network
        The batch size should be size 40

        '''
        train_batch = []
        for i in range(self.batch_size):
            index = int( random() * len(self.train_data) )
            train_batch.append(self.train_data[index])

        return train_batch



    def get_empty_state(self):
        images = np.zeros((self.batch_size, self.image_size,self.image_size,3))
        return images

    def get_empty_label(self):
        labels = np.zeros((self.batch_size, self.num_class))
        return labels

    def get_empty_state_val(self):
        images = np.zeros((self.val_batch_size,

self.image_size,self.image_size,3))
        return images

    def get_empty_label_val(self):
        labels = np.zeros((self.val_batch_size, self.num_class))
        return labels



    def get_validation_batch(self):

        '''
        Compute a training batch for the neural network

        The batch size should be size 400

        '''
        #FILL IN
        val_batch = []
        for i in range(self.val_batch_size):
            index = int( random() * len(self.val_data) )
            val_batch.append(self.val_data[index])

        return val_batch



    def compute_features_baseline(self, image):
        '''
        computes the featurized on the images. In this case this corresponds
        to rescaling and standardizing.
        '''
```

```
        image = cv2.resize(image, (self.image_size, self.image_size))
        image = (image / 255.0) * 2.0 - 1.0

        return image


    def compute_label_baseline(self,label):
        '''
        Compute one-hot labels given the class size
        '''

        one_hot = np.zeros(self.num_class)

        idx = self.classes.index(label)

        one_hot[idx] = 1.0

        return one_hot


    def load_set(self,set_name):
        '''
        Given a string which is either 'val' or 'train', the function should load
all the
        data into an
        '''

        data = []
        data_paths = glob.glob(set_name+'/*.png')

        count = 0


        for datum_path in data_paths:

            label_idx = datum_path.find('_')


            label = datum_path[len(set_name)+1:label_idx]

            if self.classes.count(label) > 0:

                img = cv2.imread(datum_path)

                label_vec = self.compute_label(label)

                features = self.compute_feature(img)


                data.append({'c_img': img, 'label': label_vec, 'features':
features})

        np.random.shuffle(data)
        return data


    def load_train_set(self):
        '''
        Loads the train set
```

```
        '''

        self.train_data = self.load_set('train')


    def load_validation_set(self):
        '''
        Loads the validation set
        '''

        self.val_data = self.load_set('val')

#########################


import tensorflow as tf
import datetime
import os
import sys
import argparse
import numpy as np

slim = tf.contrib.slim


class Solver(object):

    def __init__(self, net, data):


        self.net = net
        self.data = data

        self.max_iter = 3000
        self.summary_iter = 200




        self.learning_rate = 0.1

        self.saver = tf.train.Saver()

        self.summary_op = tf.summary.merge_all()

        self.global_step = tf.get_variable(
            'global_step', [], initializer=tf.constant_initializer(0),

trainable=False)
        '''
        Tensorflow is told to use a gradient descent optimizer
        In the function optimize you will iteratively apply this on batches of

data
        '''
        self.train_step = tf.train.MomentumOptimizer(.003, .9)
        self.train = self.train_step.minimize(self.net.class_loss)


        self.saver = tf.train.Saver()
        self.sess = tf.Session()
        self.sess.run(tf.global_variables_initializer())
```

```
    def optimize(self):

        self.train_losses = []
        self.test_losses = []

        '''
        Performs the training of the network.
        Implement SGD using the data manager to compute the batches
        Make sure to record the training and test loss through out the process
        '''
        f = open("accuracy.txt", "w")

        for i in range(self.max_iter):
            print("Iter " + str(i) + ": ", end="")
            train_batch = self.data.get_train_batch()
            train_images = np.array( [ train_batch[j]["features"] for j in range

(self.data.batch_size) ] )
            train_labels = np.array( [ train_batch[j]["label"] for j in range

(self.data.batch_size) ] )
            self.sess.run(self.train, feed_dict={self.net.images: train_images,

self.net.labels: train_labels})
            train_accuracy = self.sess.run(self.net.accurracy,
                feed_dict={self.net.images: train_images, self.net.labels:

train_labels})
            self.train_losses.append(train_accuracy)

            val_batch = self.data.get_validation_batch()
            val_images = np.array( [ val_batch[j]["features"] for j in range

(self.data.val_batch_size) ] )
            val_labels = np.array( [ val_batch[j]["label"] for j in range

(self.data.val_batch_size) ] )
            prediction = self.sess.run(self.net.logits, feed_dict=

{self.net.images: val_images})
            val_accuracy = self.sess.run(self.net.get_acc(val_labels,

prediction),
                feed_dict={self.net.images: val_images})
            print(train_accuracy, val_accuracy)
            self.test_losses.append(val_accuracy)

            f.write(str(i) + " " + str(train_accuracy) + " " + str(val_accuracy)

+ "\n")

        # self.saver.save(self.sess, "my-model", global_step=5000)

        # with open("accuracy.txt", "w") as f:
        #     for i, train, val in enumerate(zip(self.train_losses,

self.test_losses)):
        #         f.write(str(i) + " " + str(train) + " " + str(val) + "\n")
```

```
###################################

from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import random
import cv2
import IPython
import numpy as np



class Viz_Feat(object):


    def __init__(self,val_data,train_data, class_labels,sess):

        self.val_data = val_data
        self.train_data = train_data
        self.CLASS_LABELS = class_labels
        self.sess = sess




    def vizualize_features(self,net):

        images = [0,10,100]
        '''
        Compute the response map for the index images
        '''
        for i in images:
            features = np.array( [self.val_data[i]["features"] for _ in range(1)

] )
            feature_map_1 = self.sess.run(net.response_map_1, feed_dict=

{net.images: features})

            s = feature_map_1.shape[1]
            image = np.zeros( [s, s * 5, 3] )
            for j in range(5):
                image[:, j*s : (j+1)*s, :] = self.revert_image(feature_map_1[0,

:, :, j])
                plt.imshow(image)
                plt.imsave("image_" + str(i) + "_response_map_1.png", image)

            feature_map_2 = self.sess.run(net.response_map_2, feed_dict=

{net.images: features})
            s = feature_map_2.shape[1]
            image = np.zeros( [s, s * 5, 3] )
            for j in range(5):
                image[:, j*s : (j+1)*s, :] = self.revert_image(feature_map_2[0,

:, :, j])
                plt.imshow(image)
                plt.imsave("image_" + str(i) + "_response_map_2.png", image)
```

```
    def revert_image(self,img):
        '''
        Used to revert images back to a form that can be easily visualized
        '''

        img = (img+1.0)/2.0*255.0

        img = np.array(img,dtype=int)

        blank_img = np.zeros([img.shape[0],img.shape[1],3])

        blank_img[:,:,0] = img
        blank_img[:,:,1] = img
        blank_img[:,:,2] = img

        img = blank_img.astype("uint8")

        return img


######################


import IPython
from numpy.random import uniform
import random
import time

import numpy as np
import glob
import os

import matplotlib.pyplot as plt


import sys

from  sklearn.neighbors import KNeighborsClassifier



class NN():


        def __init__(self,train_data,val_data,n_neighbors=5):

                self.train_data = train_data
                self.val_data = val_data

                self.sample_size = 400

                self.model = KNeighborsClassifier(n_neighbors=n_neighbors)


        def train_model(self):

                '''
                Train Nearest Neighbors model
                '''
                X_train = np.array( [ np.copy(self.train_data[i]

["features"]).flatten() for i in range(len(self.train_data)) ] )
```

```
                y_train = np.array( [ self.train_data[i]["label"] for i in range
(len(self.train_data)) ], dtype="uint8" )
                zero = np.zeros( [1, 25], dtype="uint8")

                for i in range(len(y_train)):
                        if np.array_equal(y_train[i], zero):
                                print("eureka")

                self.model.fit(X_train, y_train)



        def get_validation_error(self):

                '''
                Compute validation error. Please only compute the error on the
sample_size number
                over randomly selected data points. To save computation.
                '''
                X_val_sampled = []
                y_val_sampled = []
                for i in range(self.sample_size):
                        index = random.randint(0, len(self.val_data) - 1)
                        X_val_sampled.append( np.copy(self.val_data[index]
["features"]).flatten() )
                        y_val_sampled.append( self.val_data[index]["label"] )

                X_val_sampled = np.array(X_val_sampled)
                y_val_sampled = np.array(y_val_sampled, dtype="uint8")

                y_predicted = self.model.predict(X_val_sampled)
                count = 0
                for i in range(self.sample_size):
                        if not np.array_equal(y_predicted[i], y_val_sampled[i]):
                                count += 1

                print("Val error: " + str(count / self.sample_size))
                return count / self.sample_size



        def get_train_error(self):

                '''
                Compute train error. Please only compute the error on the
sample_size number
                over randomly selected data points. To save computation.
                '''

                X_train_sampled = []
                y_train_sampled = []
                for i in range(self.sample_size):
                        index = random.randint(0, len(self.train_data) - 1)
                        X_train_sampled.append( np.copy(self.train_data[index]
["features"]).flatten() )
                        y_train_sampled.append( self.train_data[index]["label"] )
```

```
                X_train_sampled = np.array(X_train_sampled)
                y_train_sampled = np.array(y_train_sampled, dtype="uint8")

                y_predicted = self.model.predict(X_train_sampled)
                count = 0
                for i in range(self.sample_size):
                        if not np.array_equal(y_predicted[i], y_train_sampled
[i]):
                                count += 1

                print("Train error: " + str(count / self.sample_size))
                return count / self.sample_size
```