```
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
```