YALE UNIVERSITY, DEPARTMENT OF COMPUTER SCIENCE
CPSC 470/570 Artificial Intelligence
Homework 4: Neural Networks for Image Classification and Visual Recognition
Due: Tuesday, November 7th

Introduction
In this assignment, you will practice putting together a simple image classification pipeline (based on the classifiers), practice writing backpropagation code, and training Neural Networks and Convolutional Neural Networks. This homework is adapted from the Stanford CS class CS231n: Convolutional Neural Networks for Visual Recognition and the challenges created by the Udacity project: The Open Source Self-Driving Car.

The goals of this assignment

- understand the basic Image Classification pipeline and the train/predict stages
- implement and apply k-Nearest Neighbor (kNN), Multiclass Support Vector Machine (SVM), Softmax, Two layer neural network classifiers
- understand Neural Networks and how they are arranged in layered architectures
- understand the architecture of Convolutional Neural Networks
- gain experience with training/detection on real world car driving data

Setup and Requirements: This assignment requires Python 3.5+ with the following packages: numpy, matplotlib, future, scipy, tensorflow (1.3) as well as OpenCV. We recommend that you work remotely on the Zoo using local Anaconda installation with Spyder editor. If you decide to work on you own computer, you may find the following links useful to set up the environment.

For Tensorflow and Tensorflow Object Detection API installation, see

`https://research.googleblog.com/2017/06/supercharge-your-computer-vision-models.html` `https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/installation.md`.

For OpenCV installation, see

`https://www.pyimagesearch.com/2016/10/24/ubuntu-16-04-how-to-install-opencv/` `https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_setup/py_setup_in_windows/py_setup_in_windows.html`.

Evaluation: Your code will be auto graded. Please do not change the names of any provided functions or classes within the code, or you will wreak havoc on the autograder.

Academic Dishonesty: We will be checking your code against other submissions in the class for logical redundancy. If you copy someone else's code and submit it with minor changes, we will know. These cheat detectors are quite hard to fool, so please don't try. We trust you all to submit your own work only; please don't let us down. If you do, we will pursue the strongest consequences available to us.

Getting Help: You are not alone! If you find yourself stuck on something, contact the course staff for help. Office hours and Piazza are there for your support; please use them. If you can't make our office hours, let us know and we will try to accommodate you. We want these projects to be rewarding and instructional, not frustrating and demoralizing. But, we don't know when or how to help unless you ask. Discussion: Please be careful not to post spoilers.

Grading criteria / Late policy: This assignment will be scored out of a total of 100 points: Bonus points will also be awarded on a discretionary basis for good performance. 10% off for each day or part thereof. After three full days, the assignment will be given a score of zero.

Working on the assignment

Make a folder for Homework 4:

```
mkdir /hidden/<YOURPIN>/Homework4/
```

Create a symbolic link to Homework 4 data directory:

```
ln -s /home/classes/cs570/data/hw4/ /hidden/<YOURPIN>/Homework4/data/
```

Copy the homework files to your hidden directory under Homework4 folder:

```
cp -r /home/classes/cs570/assignments/Homework4 /hidden/<YOURPIN>/Homework4
```

Navigate to the homework folder, and list its contents:

```
cd /hidden/<YOURPIN>/Homework4 ls
```

You should see the following files:

- Q1.py - file for Question 1
- Q2.py - file for Question 2
- Q3.py - file for Question 3
- Q4.py - file for Question 4
- Q5.py - file for Question 5
- CPSC470 - data sets, auxiliary files, etc.

The CIFAR-10 dataset is going to be used for Q1-Q4 (see Figure below). Udacity dataset will be used for Q5. Run the following from the directory:

```
cd CPSC470/datasets
./getdatasets.sh
```

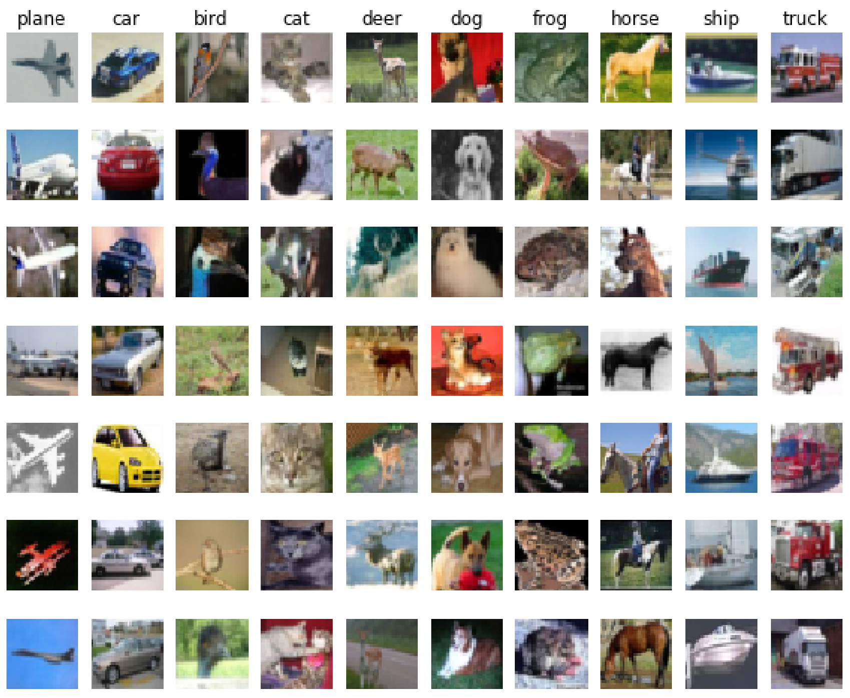
After you have the CIFAR-10 data, you should start the IPython server from the assignment 4 directory.

Files to Edit and Submit You will fill in portions of code during the assignment. You should submit these files with your code and comments. Please do not change the other files in this distribution or submit any of our original files other than these files. Run the following script file to generate an archive with your submission:

```
./collectSubmission.sh
```

Q1: Training and Implementing classifiers - (30 points)

In this exercise you will implement and understand the basic Image Classification pipeline, cross-validation, and gain proficiency in writing efficient, vectorized code. **File *Q1.py* will walk you through how to implement the classifier and testing your implementations.**



Examples of images with labels from the CIFAR dataset.

Q1A: k-Nearest Neighbor (kNN) classifier exercise - **(10 points)**

In this exercise, you are asked to implement the kNN classifier in three ways, i.e. nested for loops, single for loop, and with numpy array. The principle behind nearest neighbor methods is to find a predefined number of training samples closest in distance to the new point and predict the label from these. For more details on k-Nearest Neighbor, see the following links.

`http://scikit-learn.org/stable/modules/neighbors.html#neighbors`
`http://tangra.cs.yale.edu/newaan/index.php/resource/search?q=nearest%20neighbor`

Q1B: Training a Support Vector Machine (SVM) - **(10 points)**

In this exercise you will implement a fully-vectorized loss function for the SVM. Support vector machines are a set of supervised learning methods used for classification, regression and outliers detection. The advantages of support vector machines are:

- Effective in high dimensional spaces.
- Still effective in cases where number of dimensions is greater than the number of samples.
- Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
- Versatile: different kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels.

For more details on Support Vector Machine, see the following links.

`http://scikit-learn.org/stable/modules/svm.html`
`http://tangra.cs.yale.edu/newaan/index.php/resource/search?q=nearest%20neighbor`

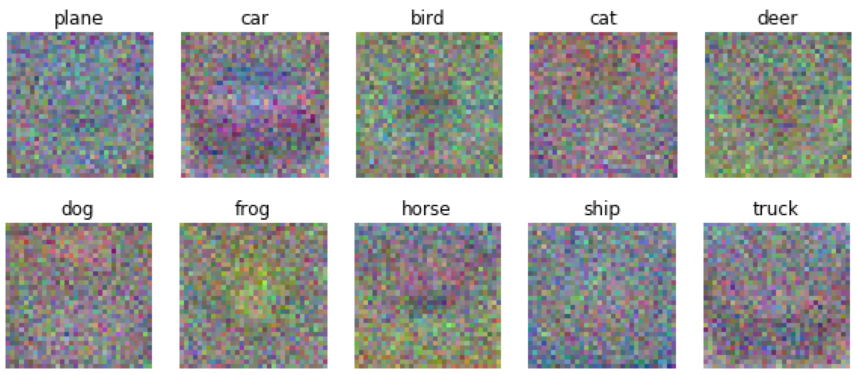
Q1C: Implement a Softmax classifier - **(10 points)**

This exercise is analogous to the SVM exercise. You will implement a fully-vectorized loss function for the Softmax classifier. More details can be found on Wikipedia.

`https://en.wikipedia.org/wiki/Softmax_function`

Q2: Two-Layer Neural Network - **(15 points)**

In this exercise, **File *Q2.py* will walk you through developmet of a neural network with fully-connected layers to perform classification, and test it out on the CIFAR-10 dataset.** To train the network you will use stochastic gradient descent (SGD) **(10 points)**, similar to the SVM and Softmax classifiers.



Visualizations of example learned weights for each class for the CIFAR dataset.

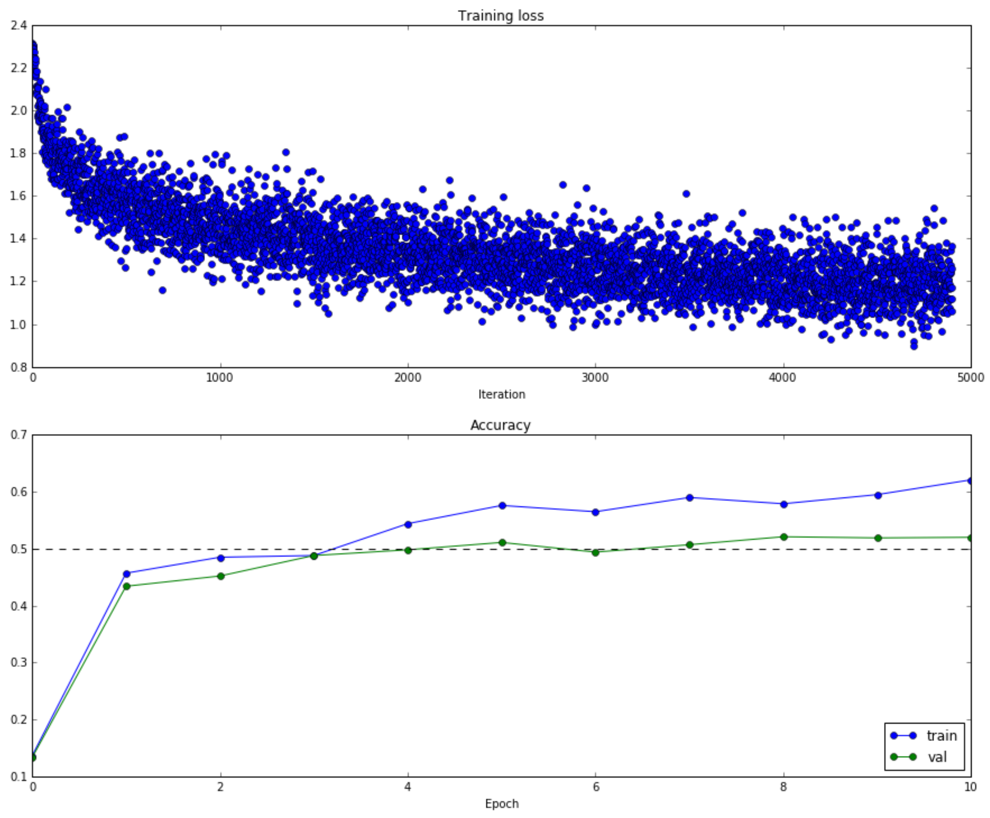
Here, you will get a lot of practice on how tuning of the hyperparameters such hidden layer size, learning rate, number of training epochs, and regularization strength affect the final performance of the neural network **(5 points)**.

For more information about SGD please follow:

`http://tangra.cs.yale.edu/newaan/index.php/resource/search?q=Stochastic%20gradient%20descent`

Q3: Fully-connected Neural Network **(15 points)**

In the previous exercise you implemented a fully-connected two-layer neural network on CIFAR-10. The implementation was simple but not very modular since the loss and gradient were computed in a single monolithic function. This is manageable for a simple two-layer network, but would become impractical as we move to bigger models. **Open *Q3.py* and follow the instructions.** This question will introduce you to modular layer design, and then use those layers to implement fully-connected networks of arbitrary depth. To optimize these models you will implement several popular update rules **(10 points)**.

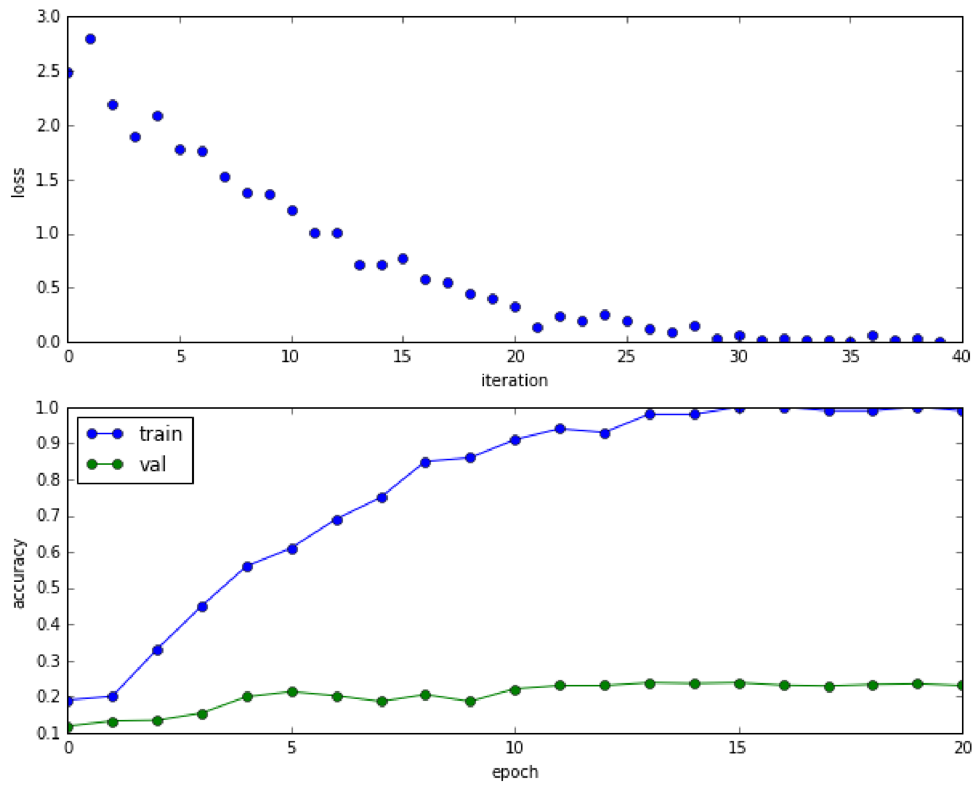


Examples of training loss and accuracy as functions of iterations and epochs.

In Q2, the logic for training models was coupled to the models themselves. Following a more modular design, for this assignment we have split the logic for training models into a separate class: Solver. By using the Solver instance, you will first train a Two-Layer Neural Network following by a fully-connected network with an arbitrary number of hidden layers. You will again tune the hyperparameters and should be able to achieve 100% training accuracy within 20 epochs **(5 points)**.

Q4: Convolutional Networks **(15 points)**

So far we have worked with deep fully-connected networks, using them to explore different optimization strategies and network architectures. Fully-connected networks are a good testbed for experimentation because they are very computationally efficient, but in practice all state-of-the-art results use convolutional networks instead. In this question (**corresponding file *Q4.py***) first you will implement several layer types that are used in convolutional networks. You will then use these layers to train a convolutional network on the CIFAR-10 dataset **(10 points)**.



Examples of training loss and accuracy achievable with convolutional networks.

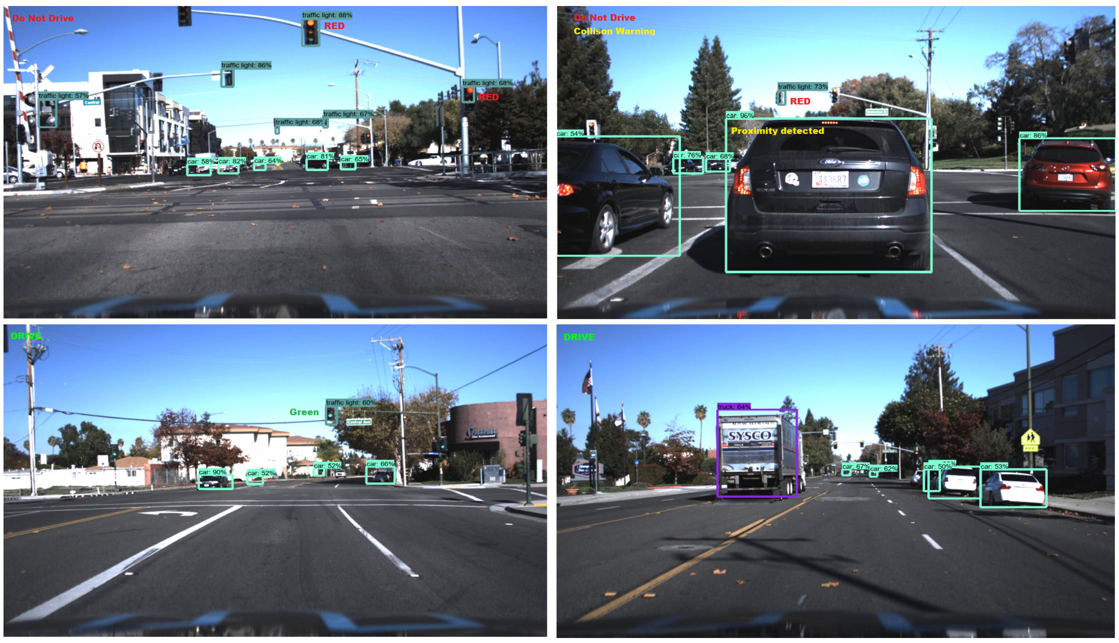
You will experiment and try to get the best performance that you can by tuning filter size, number of filters and implementation of alternative convolutional networks architectures **(5 points)**.

Q5: Determining Traffic Signals and Proximity warnings with TensorFlow and OpenCV **(25 points)**

For this last part (***Q5.py* will guide you through it**), you will be working with two popular and powerful frameworks: TensorFlow (Deep Learning) and OpenCV (Computer Vision). You will be using the recently released TensorFlow Object Detection API to detect cars, buses, trucks and traffic lights.

Tensorflow Object Detection API provides access to the pre-trained models for out-of-the-box inference purposes, see the following link:

(`https://research.googleblog.com/2017/06/supercharge-your-computer-vision-models.html`)



Examples of analysis for different driving situations.

There, you will learn how the frameworks work, culminating in the development of algorithms for further image analysis with the aim to detect traffic signal lights **(10 points)** and display proximity warnings **(10 points)** (see Figure above for examples). The code for this question will generate a video file that you will submit along with a brief description **(5 points)** outlining your strategy for **Drive/No Drive** decision making in the autonomously driving car.

Please Note: The videos for top performers will be shown in the class and will earn you extra credit (up to 10 points, not exceeding total of 100 points for the entire assignment).