Image Classification

Team:

Ahmed Rizk(14)

Yahia Mohamed ELShahawy(87)

# Contents

1. Problem statement
2. Implementation
   1. KNN
   2. SVM
   3. Softmax
   4. Two layer neural network
3. Dataset
4. Results
   1. KNN
   2. SVM
   3. Softmax
   4. Two layer neural network

# Problem statement

In this assignment we practice putting together a simple image classification pipeline, based on the k-Nearest Neighbor or the SVM/Softmax classifier. The goals of this assignment are as follows:

* Understand the basic Image Classification pipeline and the data-driven approach (train/predict stages)
* Understand the train/validation/test splits and the use of validation data for hyper parameter tuning.
* Develop proficiency in writing efficient vectorized code with numpy
* Implement and apply a k-Nearest Neighbor (kNN) classifier.
* Implement and apply a Multiclass Support Vector Machine (SVM) classifier.
* Implement and apply a Softmax classifier.
* Implement and apply a Two layer neural network classifier.
* Understand the differences and tradeoffs between these classifiers.
* Get a basic understanding of performance improvements from using higher-level representations than raw pixels (e.g. color histograms, Histogram of Gradient (HOG) features).

**The image classification pipeline:** Image Classification task is to take an array of pixels that represents a single image and assign a label to it. Our complete pipeline can be formalized as follows:

**Input:** Our input consists of a set of N images, each labeled with one of K different classes. We refer to this data as the training set.

**Learning:** Our task is to use the training set to learn what every one of the classes looks like. We refer to this step as training a classifier, or learning a model.

**Evaluation:** In the end, we evaluate the quality of the classifier by asking it to predict labels for a new set of images that it has never seen before. We will then compare the true labels of these images to the ones predicted by the classifier.

# Implementation

## KNN

The kNN classifier consists of two stages:

* During training, the classifier takes the training data and simply remembers it.
* During testing, kNN classifies every test image by comparing to all training images and transferring the

labels of the k most similar training examples.

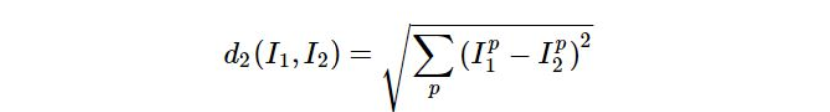
* The value of k is cross-validated.

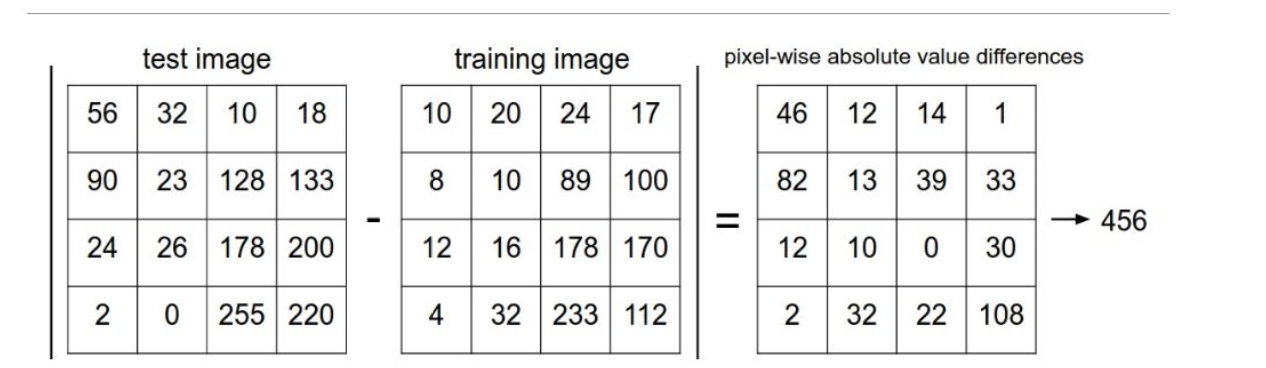
**Pseudocode**

1. Load the training and test data   
2. Choose the value of K   
3. For each point in test data:  
 - find the Euclidean distance to all training data points  
 - store the Euclidean distances in a list and sort it   
 - choose the first k points   
 - assign a class to the test point based on the majority of classes

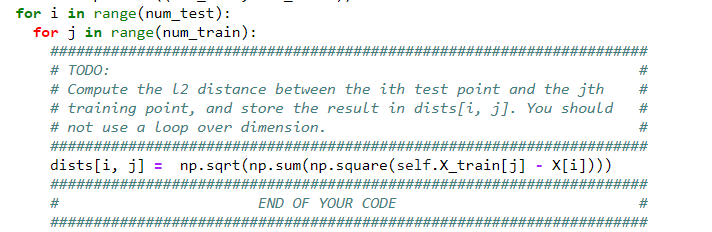
present in the chosen points  
4. End

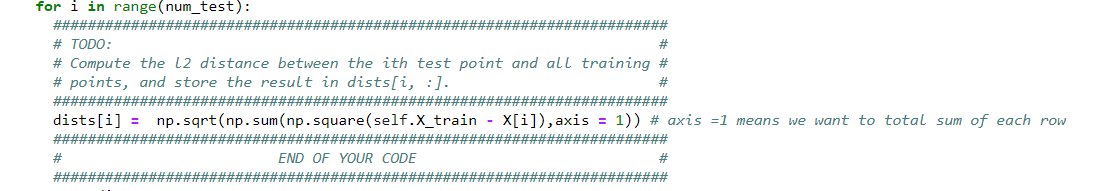
**L2-Distances:**



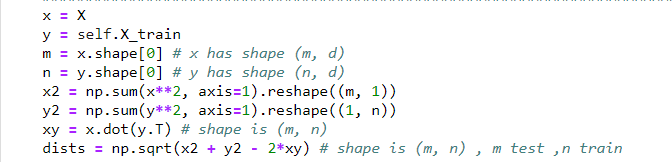


Method 1: using 2 loops



Method 2: using 1 loop

Method3: Full vectorized version



* get the predicted label
* Sort the distances.
* Take the lowest k distances.



* The predicted label is the most common label of the lowest k distances labels.

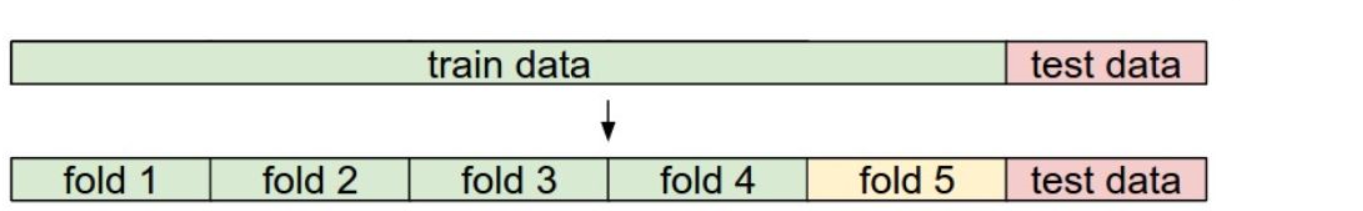


## K-fold cross validation

Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample.

The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. As such, the procedure is often called k-fold cross-validation.

Cross-validation is primarily used in applied machine learning to estimate the skill of a machine learning model on unseen data.

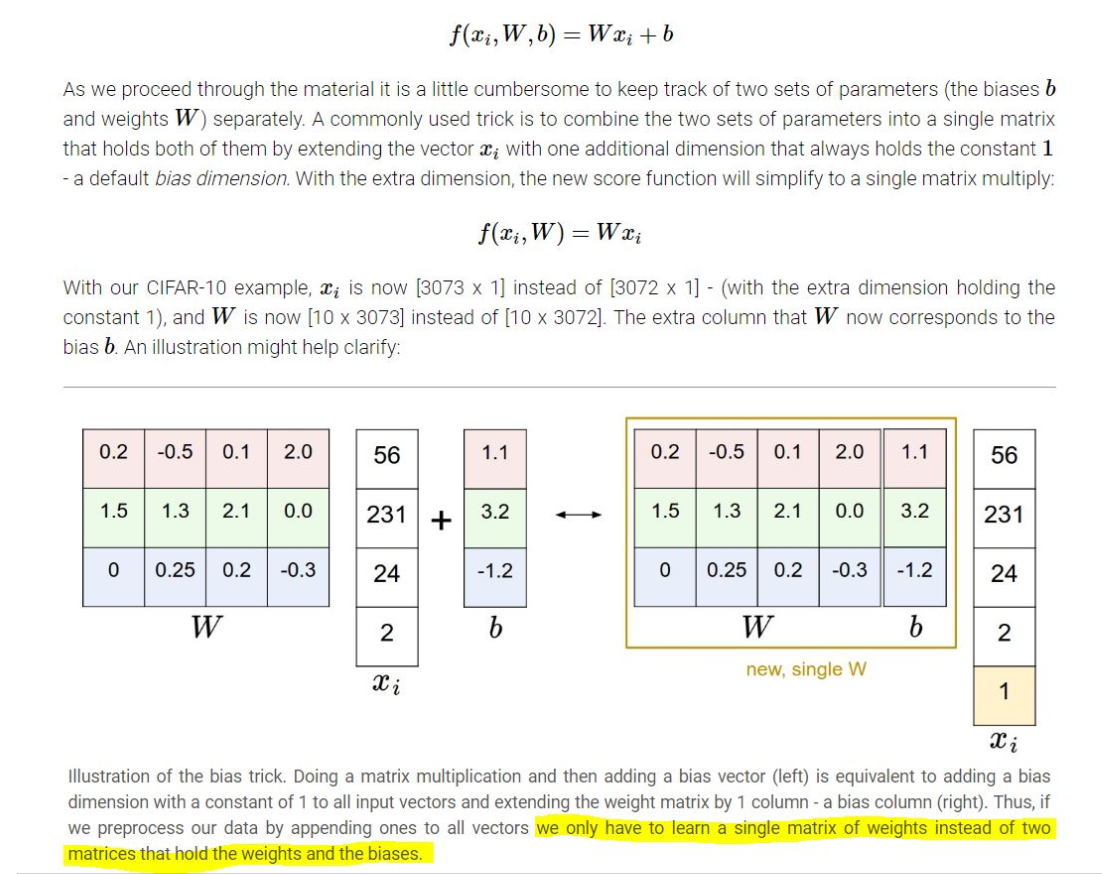


**The general procedure is as follows:**

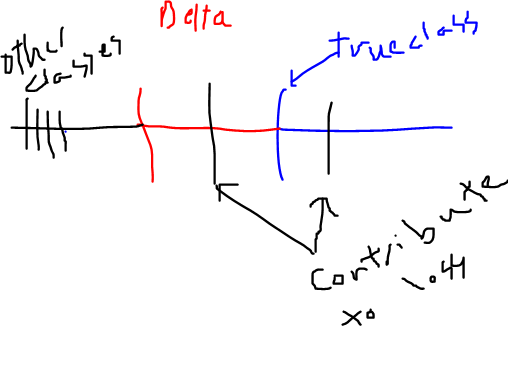
* Shuffle the dataset randomly.
* Split the dataset into k groups
* For each unique group:
  + Take the group as a hold out or test data set
  + Take the remaining groups as a training data set
  + Fit a model on the training set and evaluate it on the test set
  + Retain the evaluation score and discard the model
* Summarize the skill of the model using the sample of model evaluation scores

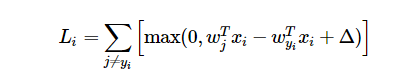
## SVM

**Score**



## Loss



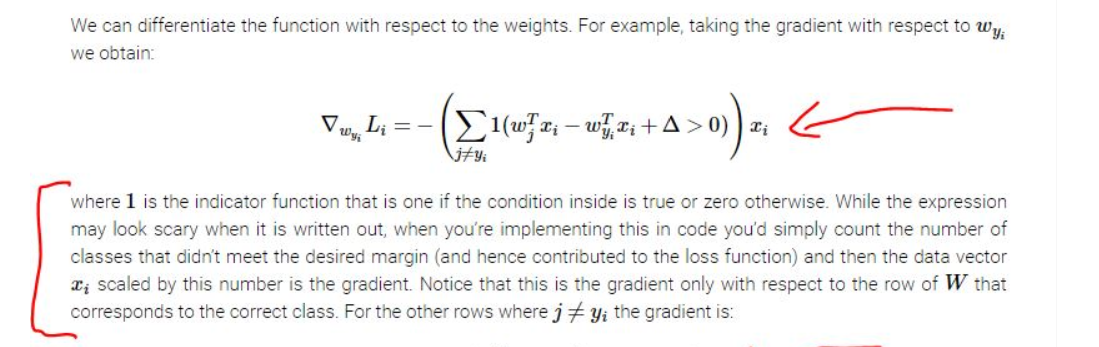


## Computing the gradient

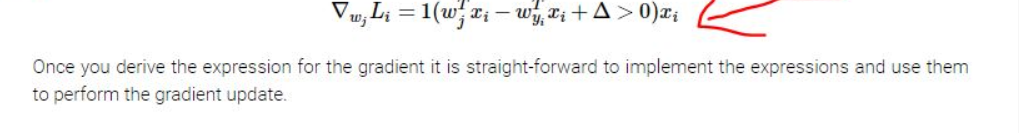
There are two ways to compute the gradient: A slow, approximate but easy way **(numerical gradient)**, and a fast, exact but more error-prone way that requires calculus **(analytic gradient)**.

**Computing the gradient analytically with Calculus:**

That form calculates gradient for the true class:



This form for the other classes, we would put 1\*xi if that class contributed to the loss



## Regularization

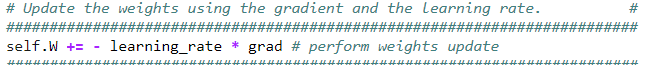
Used to prevent overfitting

**L2 regularization:**

For every weight w in the network, we add the term 0.5λw2 to the objective, where λ is the regularization strength.

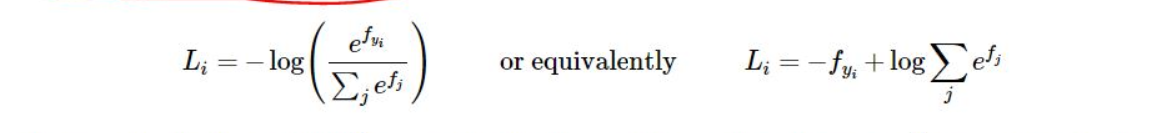
## Update

* Here we will notice that the true class previously was a -1\*count of class contributing in loss\*X
* That means that dw is -ve and when applying the following update we will get higher W for the true class
* That will give us higher score since score of class j f(j) = W[j]xi



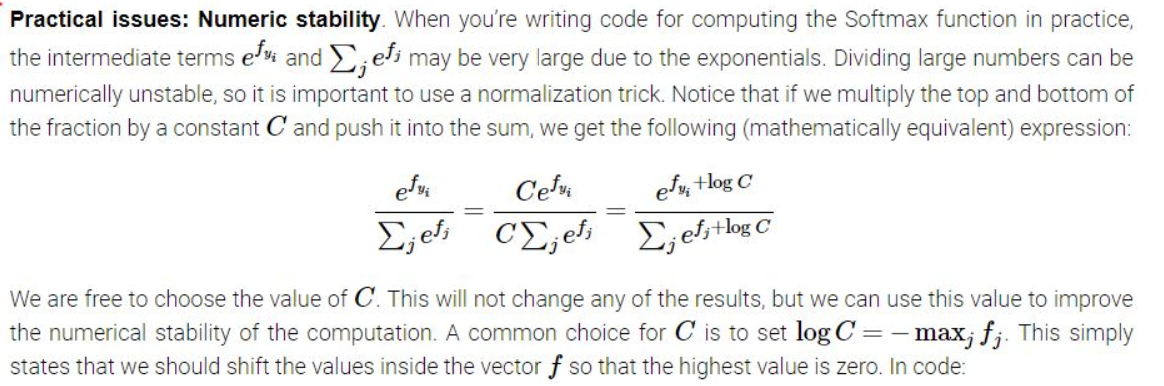
## Softmax

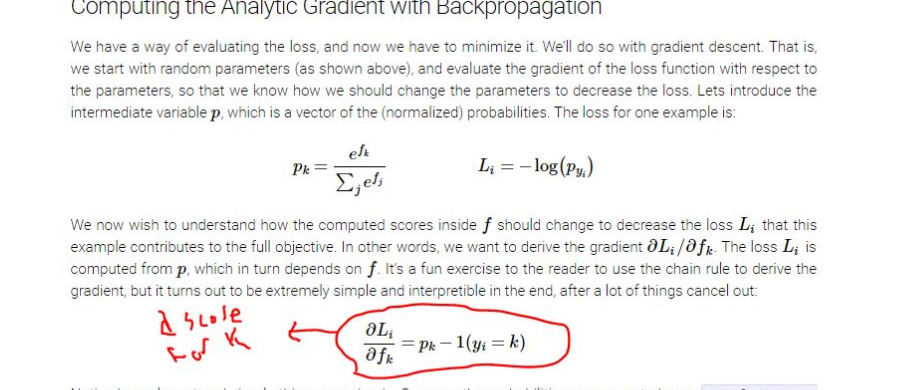
## Loss



Here the f denotes class score, so f\_yi is true class score if the less than all other classes scores fj

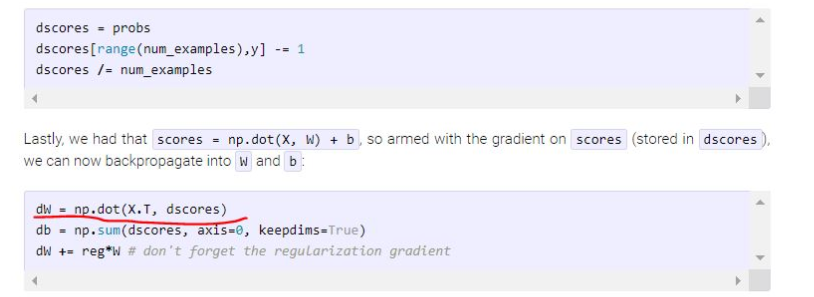
Then the Li would accumulate higher loss since log becomes -ve



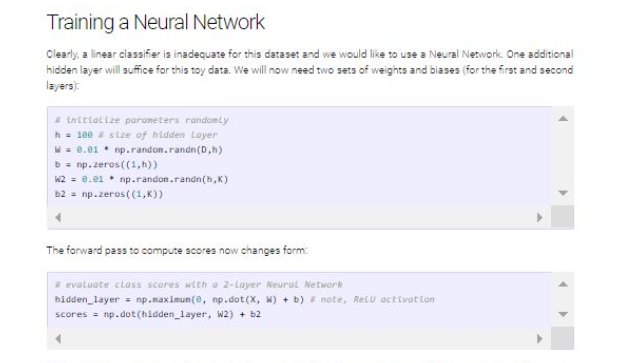


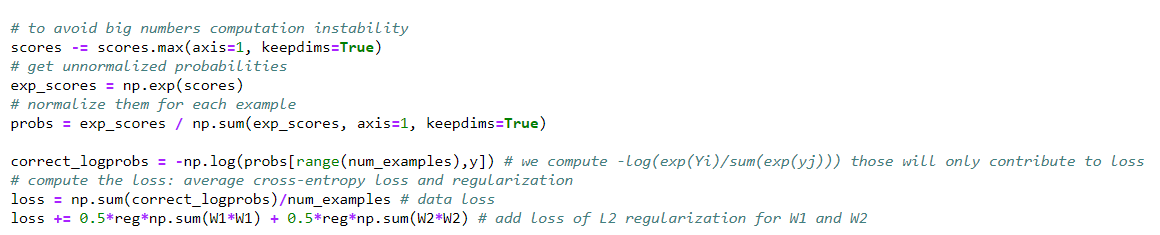
The previous partial derivative would be called dscores then to calculate the gradient

We simply get dot product X to dscores to get dW

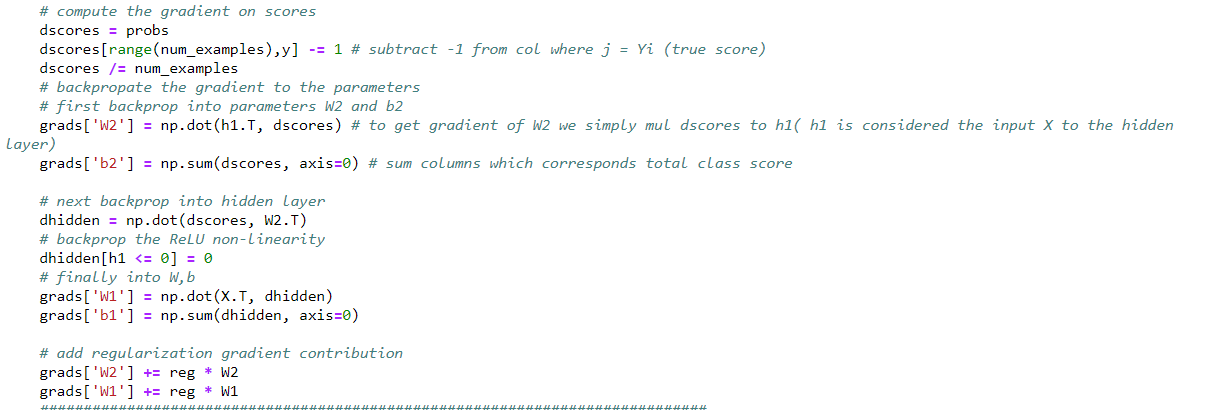


## Two layer neural network

* Forward pass
* Loss (softmax loss)

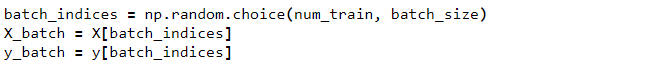


* Backward pass
* H1 is considered input of the hidden layer
* After getting probs at last layer
* Get dscore by subtracting -1
* Then gradient of last weights dw2 = h1(input of last layer)\* dscores
* Then to get the props of the hidden layer we multiply dscores to w2
* Then we get it’s dscores (dhidden) by applying relU
* Then to get gradient of W1 >> dw1 = X\*dhidden

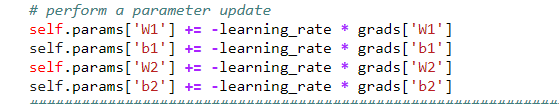


* Train

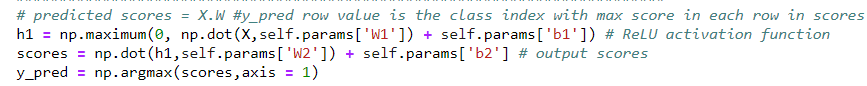
Create a random minibatch of training data and labels



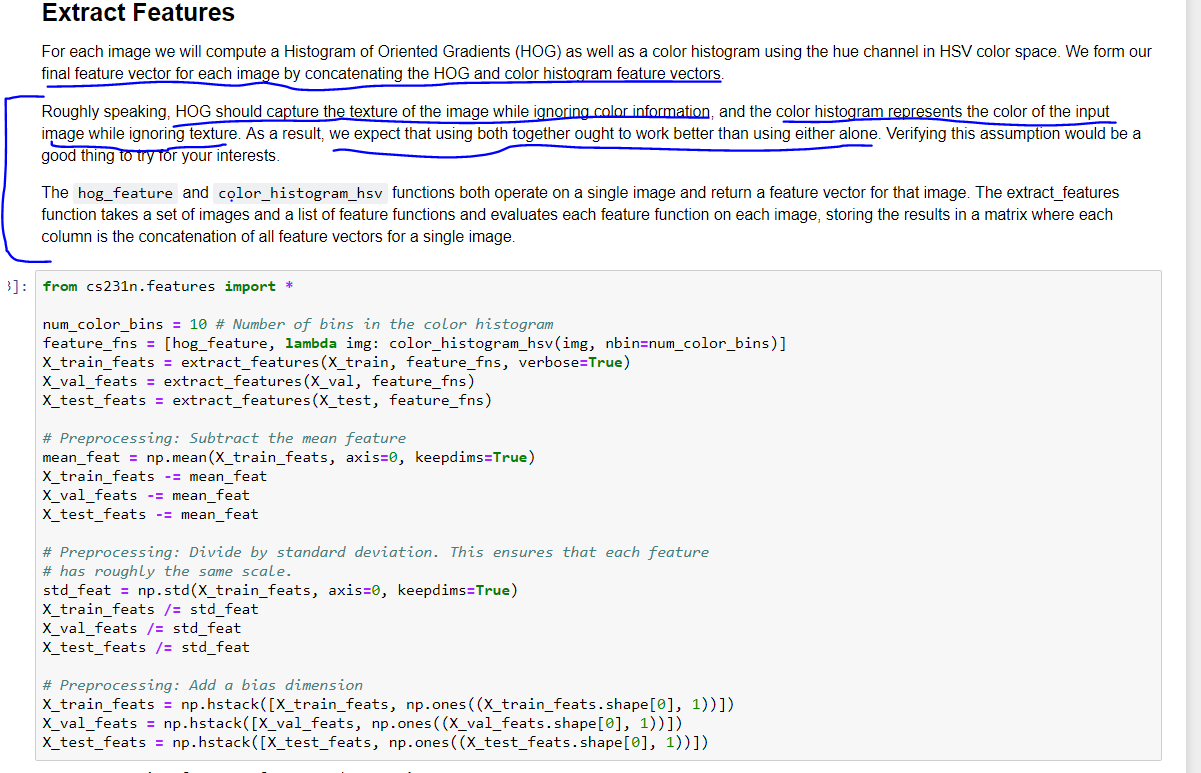
Use the gradients in the grads dictionary to update the parameters of the network



* Predict



# Feature Extraction:

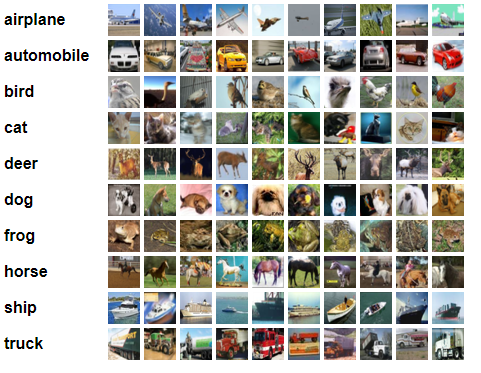


# Dataset

### The CIFAR-10 dataset

The CIFAR-10 dataset consists of 60000 32x32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

Here are the classes in the dataset, as well as 10 random images from each:

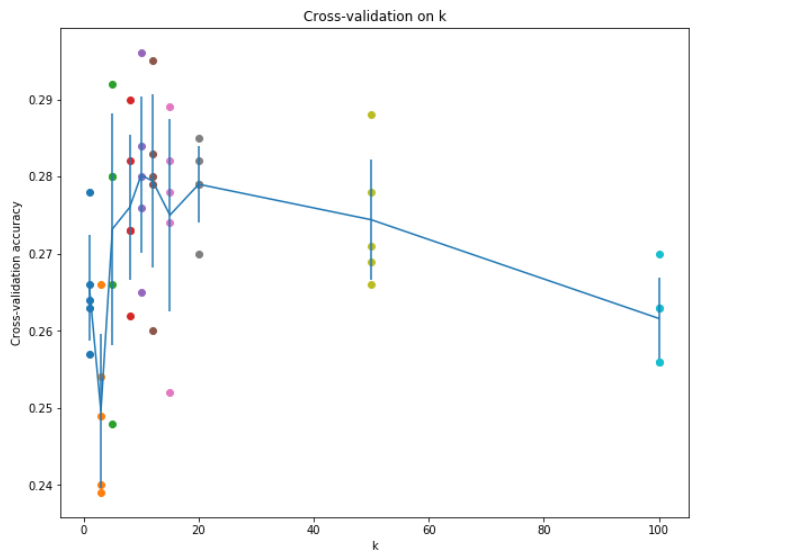


# Results

## KNN (Goal test score is 28%)

**Validation score**



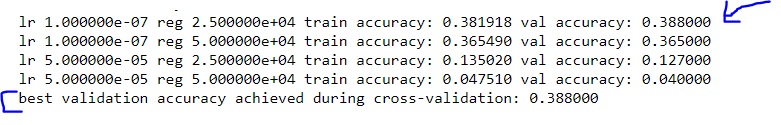


**Test score**



* 1. **SVM (goal was val\_score > 38%)**

**Validation score (we got 38.8%)**

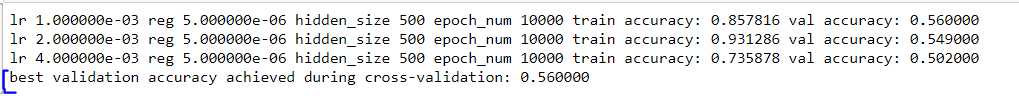


**Test score (37.5%)**



* 1. **Neural Net (Goal was val\_score > 48% .. the best was 52%)**

**Validation score (we got 56%)**

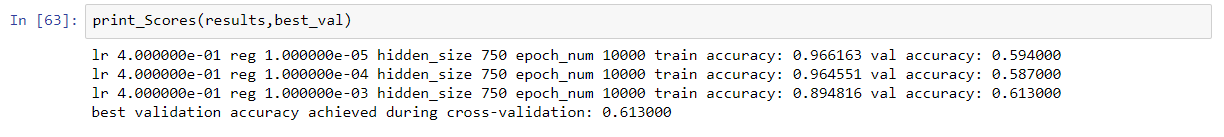


**Test score ( we got 55.3%)**



* 1. **Feature Extraction(Goal was to achieve val\_score>55% .. the best was 60%)**

**Validation score (we got 61.3%)**



**Test score (60.2%)**

