Image Classification

By
Arulselvan Madhavan, Deepen Mehta, Mavez Singh Dabas, Ashutosh Verma
Group Id: 0226, Group Name:Knowledge Miners

Abstract

The purpose of this project is to build an object recognition system that can accurately classify images. We used CIFAR-10 dataset, a benchmark dataset in the field of Computer Vision. We implemented classifiers like K-Nearest Neighbors, Support Vector Machines and Softmax Regression for classifying images. We applied preprocessing techniques like ZCA, Histogram of Oriented Gradients(HOG) and Histogram of HUE value of pixels. We also implemented CNN, which produced best results.

1. Introduction

The primary goal of our project is to apply the knowledge of Data Mining and Machine Learning techniques that we learned in our class, to the field of Computer Vision and build a robust system that can be used for Image classification. Computer Vision has become ubiquitous in our society, with applications in search, image understanding, apps, mapping, medicine, drones, and self-driving cars. Core to many of these applications are visual recognition tasks such as image classification, localization and detection. Recent developments in neural network approaches have greatly advanced the performance of these state-of-the-art visual recognition systems.

2. Problem definition

With high resolution cameras emerging every day and with the exponential increase in the number of cameras out in the world, the task of images classification gets harder and harder everyday. Vision is the richest sense humans have. Emulating such a complex system in computers is hard. Imaging a picture that is poorly illuminated or image the task of recognizing house numbers from a picture. These are hard problems but very useful to human community. Providing computers the ability to see and make sense of what it sees can solve a lot of problems. Image classification, Face detection and object detection are some examples of how teaching computers to see can be of help to humans.

3. Problem Formalization

We approached the problem of Image classification with linear classifiers like KNN, SVM and Softmax. With KNN, we tried to find out if similar images get grouped together in the Euclidean space. With SVM and Softmax classifiers, we decided to train a model that assigns a score to each class for each image. The correct class gets a high score or in other words, there is zero loss for the correct class. All the other classes get a low score, in other words the loss for the incorrect classes are high. By capturing the loss information for images using Cross Entropy loss and hinge loss functions, is relatively sophisticated method for classifying images. Our third and final approach for solving the problem of image

classification is to build a convolutional neural network, which is touted to be the best method for classifying images.

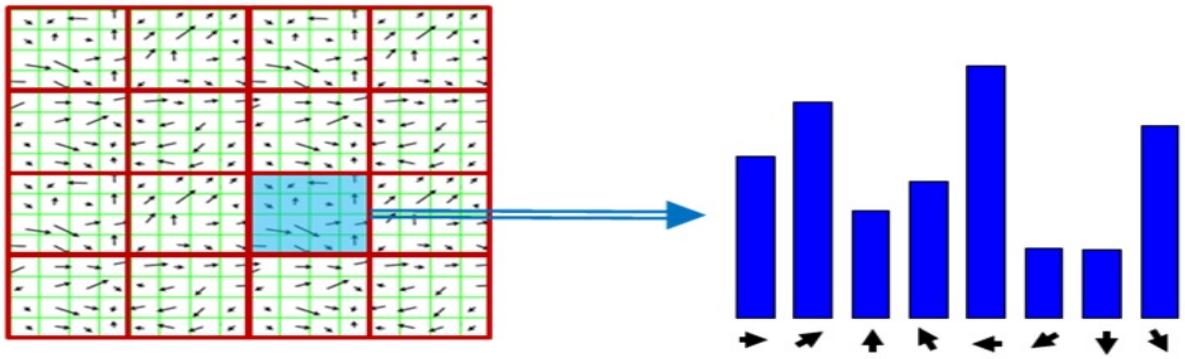
4. Data description

Dataset name: CIFAR-10
Type: Image(Color: RGB)
Image Size: 32x32 pixels. Each pixel has RGB information
Feature Vector: 1 x 32 x 32 x3 = 1x 3072
Location of DataSet: https://www.cs.toronto.edu/~kriz/cifar.html
Description: The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

5. Feature Extraction Techniques

5.1 Histogram of Oriented Gradients:

We used Histogram of oriented gradients to capture the texture of the image. Gradient is defined as the directional change in the intensity or color. An image is of size 32x32 pixels. This image is divided into 4 blocks of size 8x8. For each block we compute the gradient magnitude and gradient orientation as described below.



Let I(i,j) represent the pixel in the i'th row and j'th column in the image,then

$Gradient\ in\ x = \ dx = I(i-1,j) - \ I(i+1,j)$

$Gradient\ in\ y = dy = I(i,j-1) - \ I(i,j+1)$

$Gradient\ Magnitude \ = \sqrt{dx^2 + dy^2}$

$Gradient\ Orientation \ = \ \tan^{-1}(dy/dx)$

We quantized the gradient orientation into 9 histogram bins (Each bin stored orientations of size $\Pi/9$). We concatenated the histogram bins and flattened the blocks (Block width: 8; Block height:8).

5.2 Color histogram

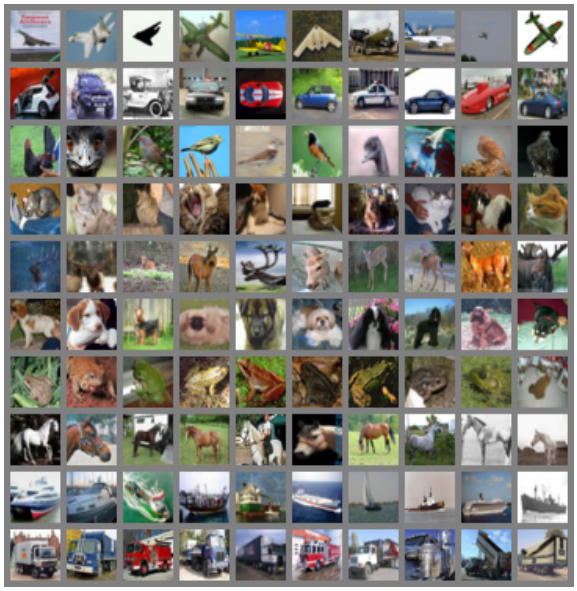
We decided to use the color distribution of images as a feature. We converted 'RGB' colors to HSV(Hue, Saturation, Value) and divided them into 10 different bins. The number of bins is configurable. Then we concatenated the bins and built a feature vector.

$$Feature\ vector = Number\ of\ bins * (block\ width + block\ height) = 9 * (8 + 8) = 144$$

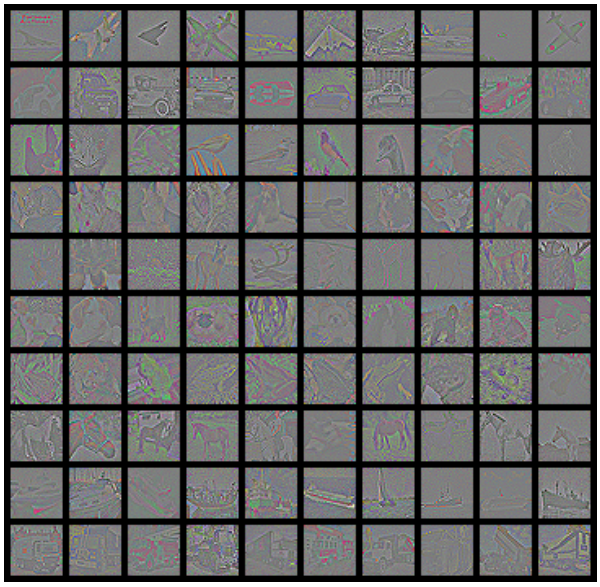
5.3 ZCA whitening - Zero Component Analysis

CIFAR-10 gives us natural color images. The color information may not be very essential in some algorithms. However, if we apply the simplest way to process the data (averaging RGB to grayscale), we will definitely lose information. Moreover, in natural images, color may be related to different objects. For example, flowers tend to have bright warm colors while trucks have relatively cold colors. Hence, we determined not to convert the pixels to grayscale. In addition, the raw data can be redundant, because adjacent pixels have highly correlated RGB values. We want to remove this kind of redundancies while not losing useful information. ZCA whitening provides a great method to preprocess the raw data. This is also a rough model of how biological eyes process the images. We obtained the ZCA-whitened data by following UFLDL tutorial implementation[1].

Original Image



ZCA Whitened Image

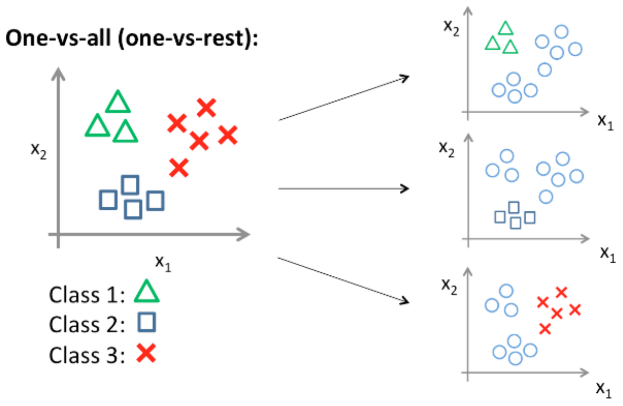


6 Classifiers

6.1 Softmax Regression:(Multi class Logistic Regression)

Basic Idea: Transfer multi-class classification into binary classification problem

We need change multiple classes into two classes, and the idea is to construct several logistic classifier for each class. We set the value of y (label) of one class to 1, and 0 for other classes. Thus, if we have K classes, we build K logistic classifiers and use it for prediction. There is a potential problem that one sample might be classified to several classes or non-class. The solution is to compare all the values of h(x) and classify the sample to the class with the highest value of h(x). The idea is shown in following figure



We use linear function to map the input X (such as image) to label scores y for each class:

Scores = $f(x^{(i)}, W, b) = Wx(i) + b$

Use the largest score for prediction.

Where $x^{(i)}$ *is a vector for all features* $x_j^{(i)}$ *(j=0,1, … , n) for single sample i, and* $x^{(i)}$*is a single column vector of shape [D,1].* ***W*** *is a matrix of shape [K,D][K,D] called* ***weights****,* ***K*** *is the number of categories, and* ***b*** *is a vector of [K,1]called* ***bias vector.***

We use exponential function instead of linear function for normalization. It is reasonable to interpret that the bigger the score of one class is, the even more chance the sample belongs to that category, and the it is better to make derivative strictly increasing (exponential function is an appropriate candidate). Then we normalized the scores by computing the percentage of exponent score of each class in total exponent scores for all classes.

Loss function is as below.

$$L_i = -log\left(\frac{e^{w_{y_j}^T x^{(i)}}}{\sum_{j=1}^{k} e^{w_j^T x^{(i)}}}\right)$$

We maintained the color information and didn't convert the pixels to grayscale, expecting obtaining higher accuracy but we did not. We can compare accuracy of softmax regression with different classifier in Table 1.

6.2 KNN

Idea: Project each image into the Euclidean space and compute the K-nearest neighbors. We used a K=5 and a 5-fold cross validation.

$$d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$$

6.3 SVM

Next we try SVM, which is considered one of the best "out of the box" classifiers. We need to generalize to multiple class case as we have 10 different labels. Our approach here is to transfer multi-class classification into binary classification. We need to change multiple classes into two classes, and the idea is to construct several classifier for each class. We set the value of y (label) of one class to 1, and 0 for other classes. Thus, if we have K classes, we build K SVM and use it for prediction. The idea is the same as to use logistic regression for multi-classification as described above.

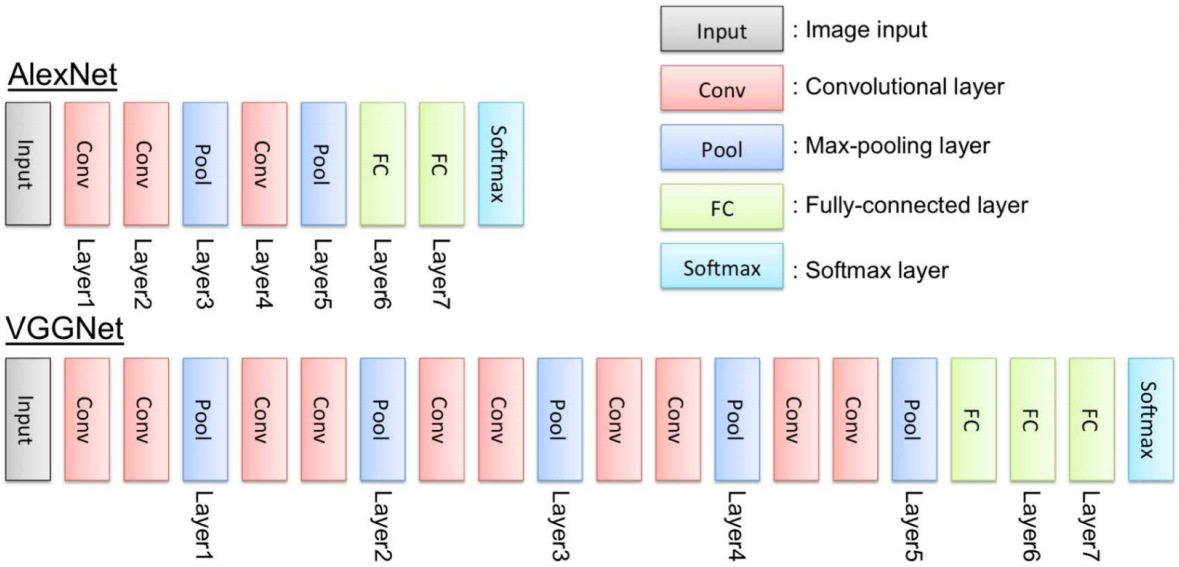
$$L_i = \sum_{j \neq y_i} \max(0, w_j^T x_i - w_{y_i}^T x_i + \Delta)$$

6.4 Results

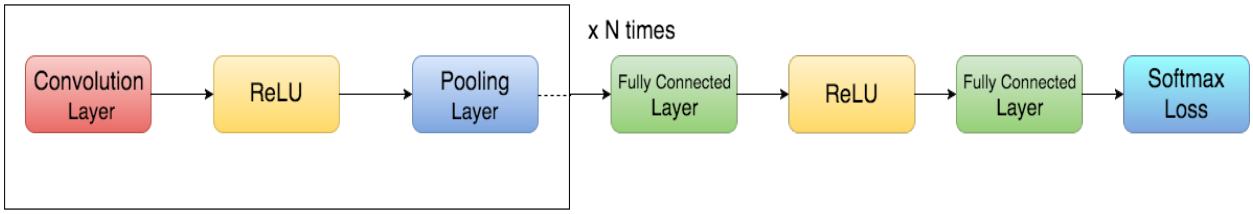
Method	Softmax	KNN	SVM
Original	39.22 %	15.87%	33.84%
ZCA	10.63 %	12.34%	31.60%
Hog	10.77 %	10.12%	21.61%
Histogram	11.46	7.78%	17.48%
Hog and Histogram	10.11 %	6.66%	16.20%

6.5 Convolutional Neural Network

Convolutional Neural Networks have become the de facto technique for image classification, object detection etc., In our project, we analyzed the AlexNet[11] and VGGNet[12] architectures and tried to emulate them in our project. Their architecture is shown below, only to make it easy on the reader to compare the existing architecture with ours.



6.5.1 Our Architecture



6.5.2 Implementation

6.5.2.1 Convolution Layer

We implemented our own convolution layer. Though our implementation works, it was not efficient. For an efficient implementation of the convolution layer, we used the convolution methods in the pydeeplearn[13] library. We tried different filter sizes (3,5,7) and we tried different number of filters (16,32,64,128). We tried varying the number of filters in each layer and the size of the filters. The results of these experiments are presented in the table 6.3.2.5. The weights of the filters were initialized randomly with zero mean and unit variance (The variance can be configured). The bias vectors were initialized with zeros.

6.5.2.2 Activation Functions

We implemented two different activation functions. Rectifier Linear Unit(ReLU) and TanH activation function. In our final architecture we decided to use ReLU activation function, as the VGGNet and AlexNet papers used them and proved that they work.

6.5.2.3 Pooling Layer

We implemented max-pooling in our project. We always tried to keep the pooling window height, pooling window width and stride length equal as this avoids overlap of the pooling windows. Throughout our project, we used a fixed pooling height,pooling width and stride length of 2.

6.5.2.4 Fully Connected Layers

In our implementation we have 2 fully connected layers in the end. They are fixed. Even if there are no convolution layers, there will be 2 fully connected layers. The number of hidden layers is a configurable parameter. We achieved our best results when we used hidden layers with 500 parameters. These fully connected layers are also called as output layers as these layers prepare the input data for computation of softmax classification scores.

6.5.2.5 Gradient Update Functions

We tried two different gradient update functions.

1. Mini batch Stochastic Gradient Descent
2. Adam

Even though, we were getting good results with Mini batch SGD, we decided to use 'Adam' to see if it helps us to converge faster. We found the best results for learning rate of 0.001. We decayed our learning rate after each epoch. For Adam, we maintained the first order moment and second order moment during iterations and computed the next update using the first order moment and second order moment.

6.3.2.6 Softmax Loss

The final piece of our CNN architecture is the loss function. We decided to use the softmax loss function that we wrote for our softmax regression. Details of this loss function are mentioned in the section on softmax regression.

7. Experiments Design and Evaluation:

7.1 Setup for evaluation

- Training Data: 49000 images
- Validation Data: 1000 images
- Test Data: 10000 images
- Mini-batch gradient descent with batch size(SGD, Adam): 50
- Learning Rate: 0.001 (Decayed by 0.95 after each epoch)
- L2 Regularization Strength: 0.05
- Epochs Count: 20

- Calculate Validation accuracy after each epoch. The weights that give the best accuracy is selected for the test dataset.
- The weights of all the convolutional layers and fully connected layers are initialized with unit normal distribution.
- The bias used by the convolution and fully connected layers are initialized with zeros.

7.2 AWS machines

We used AWS EC2 instances with configuration m3.2xlarge and m4.4xlarge to run our neural network.

7.3 Evaluation

7.3.1 Three Layer CNN (1 Conv Layer - 1 FC layer - 1 FC layer)

The log files are in the log file directory of the source code that we submitted.

Configurations				Accuracy		
Convolutional Layer1		FC Layer 2	FC Layer 3			
Filter Size	Filter Count	Hidden Layer Size	Hidden Layer Size	Training Set	Validation Set	Test Set
3	16	500	500	60.00%	58.10%	57.37%
	32	500	500	62.23%	57.2%	57.09%
	64	500	500	63.90%	55.80%	56.71%
5	16	500	500	61.34%	57.10%	55.29%
	32	500	500	59.4%	54.37%	53.35%
	64	500	500	58.3%	53.38%	52.44%
7	16	500	500	56.12%	54.23%	51.33%
	32	500	500	55.31%	52.12%	49.2%
	64	500	500	54.32%	48.03%	47.77%

7.3.2 Four Layer CNN (2 Conv Layer - 1 FC layer - 1 FC layer)

Configurations					Accuracy		
Filter Size	Conv Layer1	Conv Layer2	FC Layer3	FC Layer4			
	Filter Count	Filter Count	Hidden Layer Size	Hidden Layer Size	Train Set	Validation Set	Test Set
3	16	32	500	500	81.8%	68.8%	69.71%
	32	64	500	500	69.70%	61.70%	63.24%
5	16	32	500	500	75.3%	66.34%	66.8%

7.3.3 Five Layer CNN (3 Conv Layer - 1 FC layer - 1 FC layer)

Configurations						Accuracy		
Filter Size	Conv Layer1	Conv Layer2	Conv Layer3	FC Layer4	FC Layer5			
	Filter Count	Filter Count	Filter Count	Hidden Layer Size	Hidden Layer Size	Train Set	Validation Set	Test Set
3	16	32	64	500	500	88.5%	74.6%	73.96%
	32	64	128	500	500	87.2%	73.22%	70.31%
5	16	32	64	500	500	86.31%	70.21	68.32%

7.3.4 Six Layer CNN (4 Conv Layer - 1 FC layer - 1 FC layer)

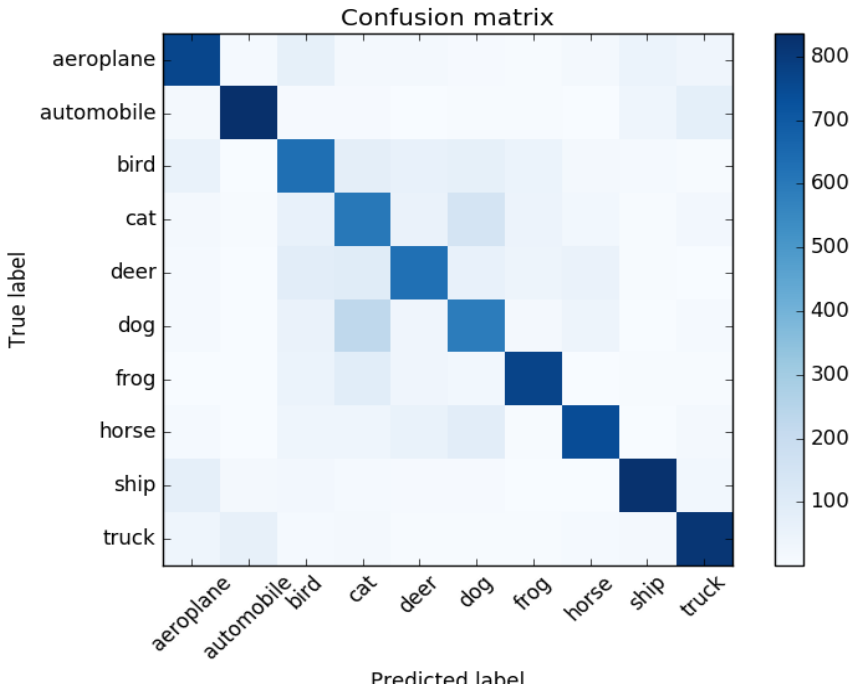
Configurations							Accuracy		
Filter Size	Conv Layer1	Conv Layer2	Conv Layer3	Conv Layer4	FC Layer5	FC Layer6			
	Filter Count	Filter Count	Filter Count	Filter Count	Hidden Layer Size	Hidden Layer Size	Train Set	Validation Set	Test Set
3	8	16	32	64	500	500	80.5	69.1	68.82
	16	32	64	128	500	500	65.3	70.8	69.23
	32	32	64	128	500	500	98.6	72.00	68.89
5	16	32	64	128	500	500	98.3	72.96	72.1

7.3.5 Seven Layer CNN(5 Conv Layers - 1 FC layer - 1 FC layer)

Configurations								Accuracy		
Filter Size	Conv Layer1	Conv Layer2	Conv Layer3	Conv Layer4	Conv Layer5	FC Layer5	FC Layer6			
	Filter Count	Filter Count	Filter Count	Filter Count	Filter Count	Hidden Layer Size	Hidden Layer Size	Train Set	Validation Set	Test Set
3	8	16	32	64	128	500	500	92.4%	63.9%	67.12%
	16	32	64	128	256	500	500	98.6%	73.9%	71.69%

Analysis of Results

From the results in section 7.3.1, we concluded that increasing the filter size did not improve the accuracy. The intuitive reasoning behind low filter size providing high accuracy is, low filters depend on their immediate neighbors to produce the convolution layer output. High filter size means we are including more neighbors to produce our convolution output. From the results in section 7.3.2,7.3.3,7.3.4 and 7.3.5, it is clear that increasing the number of layers increases the accuracy of classification.



Future Work

We would like to add a batch normalization layer after each convolutional layer, as this will ensure that the input data will have a normal distribution before entering the next layer of the neural networks. There

are several benchmark datasets in image recognition including MNIST, NIST, CIFAR-10, CIFAR-100, ImageNet etc. We will further explore more larger datasets with more categories. Also, we will turn to GPU to train our models, which is promising to greatly shorten the training time. Furthermore, there are a large pool of Convolutional Neural Networks as well as other Neural Networks and involved features can be added to both increase the accuracy and efficiency. We are to learn more varieties in designing Neural Networks.

Conclusion

We first tried out KNN, Softmax Regression and SVM and gained a general view of the image recognition problem. We tried out feature extraction techniques like Histogram of Oriented Gradients and Histogram of color bins. They It was non-linearly separable and suffered overfitting. Then we designed our own Convolutional Neural Networks and improved the performance greatly.

References Note:

[1]: Softmax - UFLDL	[12]: VGGNet Paper
[2]: Softmax Implementation Sample	[13]: pydeeplearn
[3]: CIFAR-10 Site	[14]: Paper on Adam
[4]: HOG	[15]: Geoff Hinton's lecture on Optimization algorithmshttp://keras.io/layers/convolutional/
[5]: HOG-paper	[16]: Paper comparing AlexNet and VGGNet
[6]:Image gradient	[17]: FC Layers
[7]:Gradient Orientation	[18]: Deep learning tutorial
[8]:HOG_Explained	[19]: LeNet
[9]:Linear_Classifiers Overview	[20]: Convolution Layer implementation
[10]:ZCA_Explained	[21]: Theano - Conv Layers
[11]:AlexNet architecture	[22]: Image Processing Basics

Task Distribution Form

Task	People
Data Loading	Arul
Data Preprocessing	Deepen
Zero Component Analysis	Deepen
Feature Extraction - HOG and Histogram	Arul
Softmax	Deepen
SVM	Mavez, Ashutosh

KNN	Mavez, Ashutosh
Convolution Layer	Arul
Activation Functions - Rectifier Linear Unit , Leaky ReLU and TanH activation function	Arul
Pooling layer	Ashutosh
Gradient Descent (SGD & Adam)	Arul
CNN - Architecture design	Arul
Evaluation	Deepen, Ashutosh
AWS instances	Deepen, Ashutosh
Presentation	Deepen, Ashutosh, Mavez
Report	Deepen, Ashutosh

Claim for Additional Points

- We picked up a domain that was not taught in class. We motivated ourselves to look beyond the class and complete the project.
- We did not use any deep learning framework. We designed the entire architecture on our own. Designing an architecture that holds all the layers of the CNN was the toughest task in our project.
- Convolution Layer – We designed a naïve layer. It works but we are using an efficient implementation from pydeeplearn[13] library in our actual code.
- Activation Layers – ReLU,TanH and Leaky ReLU – We designed these functions
- Pooling Layer - We designed a naïve layer. It works but we are using an efficient implementation in our actual code. It works but we are using an efficient implementation from pydeeplearn[13] library in our actual code.
- Gradient Update – Adam Update – We used a reference(mentioned in the code file) to design our own Adam function.

Source Code Repository

Github Link: https://github.com/Arulselvanmadhavan/ConvNets.git
Data Link:
https://drive.google.com/a/husky.neu.edu/folderview?id=0B1hBKt9aABf9MFM0Sko1enYwU3M&usp=sharing