Image Classification using Experimentally-driven Ensemble Method

Niloy Gupta
School of Computer Science
Carnegie Mellon University
niloyg@andrew.cmu.edu

Kavya Srinet
School of Computer Science
Carnegie Mellon University
ksrinet@andrew.cmu.edu

Matthew Tay
Electrical and Computer Engineering
Carnegie Mellon University
mhtay@andrew.cmu.edu

ABSTRACT
The CIFAR-10 data-set [1] has been tackled by many researchers using a wide range of image classification techniques. In this paper, we conduct a survey of many of these methods, exploring various baseline methods, conducting parameter tuning, with the final goal of combining them into a strong ensemble method for image classification of the given CIFAR-10 dataset. In doing so, we explore the effect of different types of ensembles on the CIFAR-10 data. In addition, we explore the mutual effect that various machine learning classifiers have on each other, when combined together. The paper is divided into 2 main sub-themes. The first sub-theme discusses individual baseline method types that were explored by each team member, and how the performance of the different types of individual classifiers varied for the CIFAR-10 dataset. The second sub-theme is an exploration of different ensembles of various classifiers as well as the parameter tuning done to achieve the reported performance on the dataset. Through our experiments, we have found that even simple combinations of moderately strong classifiers can perform better than each individual classifier, validating published literature on the subject [2].

Categories and Subject Descriptors
Java, Matlab, Weka, Object-Oriented Programing

General Terms
Image Classification, Ensemble Methods, CIFAR-10, Computer Vision, Feature Extraction, Machine Learning, Parameter Tuning

Keywords
Random Forest, Support Vector Machines, GIST, Principle Component Analysis, K-Means, AdaBoost, Convolution Neural Networks, C4.8 Decision Trees, Ensemble Methods, K-NN, Multilayer Neural Network, HOG Features, Logistic Regression

1. INTRODUCTION
CIFAR-10 is a standard computer vision labeled dataset used for developing and testing image classification methods [1]. The data-set is relatively challenging, with state-of-the art methods in machine learning achieving approximately 80% classification rate. This means that the dataset is well poised to pose a challenge to most algorithms, and is thus able to discriminate effectively between different image classification machine learning algorithms. Inspired by published literature [2] and by the fact that the top performers in the image classifications competitions like ImageNet [10] have used ensembles of some sort, we wanted to firstly, evaluate which classifiers would perform the best on the limited data-set that we were provided and secondly, find means of combining the winning ensembles that would be stronger as a whole than the sum of its parts.

There has been a lot of work done on both the machine learning and feature extraction aspects of image classification [9]. Here we apply some feature extraction and image pre-processing but focus mainly on the machine learning problem of evaluating different classification algorithms.

Each individual on the team focused on different baseline methods for analysis, and collated the results in order to examine which methods performed the best. Our strategy is to quickly and as exhaustively as possible survey the performance of different learning algorithms. We explored methods which include: Random Forest, Nearest Neighbor, Multinomial Logistic Regression, Naive Bayes, Convolution Neural Network, Multi-Layer Neural Networks, K-means with SVMs.

After many iterations of performance evaluation and parameter tuning, we have successfully developed an ensemble of three of the best performing classifiers, which are K-Means with SVM, Random Forests and Spatial Envelope with SVM. This gives an accuracy of 0.5893, a significantly high accuracy given the limited size of the training set, and one that is higher than the any of the individual classifiers that we have considered.

2. BACKGROUND
Here we briefly review some of the most popular classification algorithms used in image classification, and that we have used in our classification.

2.1 KNN
K-Nearest Neighbors (KNN) [11] is a simple but fast method for classification. Every training instance is represented by a set of features. When a new unknown test sample is being classified, the distance of the unknown test sample from all of the other points in the training set is computed. The K nearest training samples are then computed, and the class of the unknown test sample is the class that the majority of the samples in the set of K nearest training samples belong to. KNN gets fast training time since little to no processing is required in the training stage, but in exchange for the fast training time requires a large amount of processing time at testing.

2.2 Multi-layer Neural Network
Multi-layer neural networks [12] work as an aggregation of activation functions that take a weighted combination of inputs from the previous layer, before applying their activation functions to the weighted inputs and passing the result to the next layer. In this way, the multi-layer neural network work by simulating how individual neurons of the brain react to stimuli and then pass electrical impulses to the next neuron. They are an improvement over the traditional single layer neural network, also known as the perceptron, which was proved in 1969 by Symour et al [12] to be incapable of computing the basic XOR function. Practically, neural networks have been found to perform well for noisy data

[13], and so would be well suited to be tried on the image dataset that we have, since images tend to be corrupted by noise, whether additive isotropic Gaussian or images of other types of distracting objects like background grass and clouds.

2.3 Decision Trees and Random Forests

Decision trees are constructs that learn attributes that would best separate the data, using some objective measure to quantify which separation would be optimal. Often this measure calculated using entropy, by measuring the information gain that would result from splitting the data along using an attribute. Random Forests and other bagged ensembles combine multiple decision trees together in order to reduce the variance of the output and hence increase accuracy.

2.4 Single Layer Networks for Unsupervised Feature Learning

This method [4] has proved to be one of the most successful image classification strategies. Square patches are randomly selected from the training data and K-Means is used to extract features. After which distance from each pixel is computed for each centroid. Distances less than the mean are set to zero. After which each image is split into four quadrants, and the feature values in each quadrant corresponding to each dimension are summed up. Thus each image has a feature vector of length 4*K where K is number of centroids. K is recommended to be 1,500 and patch size 6px X 6px. After which a L2 SVM is trained to predict the classes for the unlabeled images. As per the literature accuracy on CIFAR dataset is 79.6% for K=4000 features.

2.5 Gaussian Kernel SVM

Gaussian Kernel SVMs are known to generalize well on image classification problems when the number of features and dimensions is quite large [13]. It also approximates with run-time and memory complexity that is very nominal. Since an image can have numerous features, using a Gaussian Kernel helps SVM draw a linear decision boundary in a high dimensional space. We, in our experiments used SVM with dimensionality reduced features, i.e. after performing PCA and reducing the dimensions to 300, we used RBF Kernels and when using the GIST techniques, we first scaled down the number of dimensions to a low dimensional space, i.e. reduced the number of dimensions from 3072 to 512 for all images and then used SVM with RBF. Although, as has been shown in [14], RBF Kernel SVMs would perform better for histogram based image comparison, they did perform well enough for our experiments on CIFAR-10 dataset, with dimensionality reduction.

2.6 Naive Bayes

Although it does not seem very intuitive to use Naive Bayes for an image classification problem, we wanted to see the extent to which conditional independence holds true for image features, as compared to text-based classification. Our results showed that the intuition did not work that well, for our dataset, however, there are some extensions to Naive Bayes like Naive Bayes with Nearest Neighbor as covered in [15], that give a performance that is ranked among the top-leading learning based image classifiers.

3. METHODS

We began our search by applying machine-learning techniques sampled from across both supervised and unsupervised learning. The performance of these methods was evaluated using cross-validation on the training dataset. To maximize the different types of algorithms that we could survey, we focused not on implementing the algorithms, but on finding available implementations that we could use for the dataset. We used WEKA for convenience and MATLAB for fast matrix multiplication and transformation operations.

With a final goal of finding an ensemble of our best classifiers, we found that the best performing classifiers in the field [8] were deep belief networks - a form of multi-layer neural network, a careful implementation of a semi-supervised K-Means algorithm, as well as Adaboost and its variants like Stochastic Boost [5] that reduce the run time of Adaboost by stochastic gradient descent.

First we examine classifiers that would help us understand the nature of the dataset in more detail. These were the Naive Bayes and the semi-supervised Seeded K-Means classifiers.

3.1 Naive Bayes classifier

We implemented this using the Naïve Bayes library in Weka, weka.classifiers.bayes.NaiveBayes in Weka.jar 3.6.11.

This classifier was chosen to see whether we could assume that the pixels were conditionally independent of one another given the class. This would impact our choice of feature representation as highly correlated pixels would suggest that applying Principle Component Analysis (PCA) would yield principle axes on which most of the data could be represented, and not lead to large reconstruction errors when projecting the data unto the new space spanned by the Principle Components.

The accuracy on the test data was 0.298. (Training Time: 3 seconds).This is on par with many of the classifiers we had trained in Milestone 1, and thus agrees with our intuition that suggests that Naive Bayes should not work well, since the strong assumption of conditional independence in Naive Bayes would not be as true for pixels of an image that would look almost the same as its neighboring pixels, unlike different words in certain types of text classification. From this result, we can more safely use PCA whenever we need to reduce the computation time of the algorithm, unless the algorithm itself depends on having a very feature rich space, like Adaboost.

3.2 Semi-supervised Learning: “Seeded” K-Means

The intuition behind using a semi-supervised learning approach is that we have a set of labelled training examples and test data which is unlabelled. The seeds for the K-Means are the training examples. In Seeded K Means we recalculate the centroids of only the unlabelled data. Training time was 20s.

The accuracy of this method was 0.229 while after PCA decomposition the accuracy on the test set was 0.235. We believe the poor results can be attributed to the fact that we assumed that each class of images would have distinct features compared to other class of images.

Next we examined techniques that other researchers have found to perform well. These are the multilayer neural networks, a combination of K-Means with other algorithms, as well as Adaboost.

3.3 Multilayer Perceptron

Deep belief networks, a variant of multilayer neural networks, had achieved state of the art performance and would likely be a strong classifier to use in a final ensemble.

We first used Principal Component Analysis for dimensionality reduction to get the important 300 features from the set of all available features using pca function in Matlab. We then used the neural network learner of Weka, basically weka.filter.functions.MultiLayerPerceptron on the dimensionally

reduced data. We used a learning rate of 0.005, a momentum of 0.79, and all other default parameters.
The training took about 1100 seconds, and resulted into an accuracy of 36% on the 15000 test instances. We observed that we got errors in images that had edges that were a little obscured.

Since we observed that the edges being distorted might lead to errors in classification, we experimented with different forms of feature transformation and representation, like GIST and HOG. The experiments are reflected below.

3.4 GIST with Gaussian SVM

We used the spatial envelope approach as proposed in [6], where we transform the image to a very low dimensional space. Here in this approach, we maintain a set of perceptual dimensions (naturalness, openness, roughness, expansion, ruggedness etc.) that represent the dominant spatial structure of an image, we used the Matlab code for GIST given on http://people.csail.mit.edu/torralba/code/spatialenvelope/ , for this. Thus GIST helps us do feature extraction , and reduces our dimensions from 3072 to 512. Using GIST we first perform dimensionality reduction and then we used LibSVM, to train an SVM with RBF Kernels, with C = 100 and g = 0.8 for this. This resulted into an accuracy of 0.55943. To understand the impact of GIST, and how well it performs, we experimented with various different values of C and g, as shown in the table below:

Table 1. Parameter Selection Search for SVM

C	g	Accuracy
8	0.8	0.55448
10	0.6	0.55124
10	0.8	0.55390
100	0.06	0.53591
100	0.8	0.55943
100	0.9	0.55790
140	0.8	0.55562

Thus we observed, beyond a certain optimal combination of C and G, the accuracy would drop. This happened for images that had blurred edges.

3.5 Neural Networks with HOG Features

We used Matlab and Matlab's Computer Vision System Toolbox, to first extract features of each image in the cell size 4x4 and 8x8 by directly using extractHOGFeatures. This helped us extract features that encode local shape information from withing the colored images that have been provided to us. We figured that since, scarcity of training data, was making neural networks do worse on our dataset, we would try and augment the training data, by adding these new features, hence combining the above mentioned features, with the training data. We now used Weka to run neural networks on this, with two hidden layers, each with 120 nodes. This resulted into an accuracy of 39%.

3.6 Neural Network learner

Here, we did not do any pre-processing as such and used the DeepLearning Toolbox for Matlab, to use the neural network learner. Here we tried to do a lot of experiments with the number of nodes in the input layer, hidden layer and output layer. For 2000 nodes in the input layer, 90 in the hidden layer and 8 in the output layer, we got an accuracy of 28%.

Moving back to the list of well-established techniques, we continued to try to do parameter tuning and customization of the best-performing techniques on the CIFAR-10 dataset. Here are our findings.

3.7 K-Means reduction with various classification algorithms

We experiment with the method discussed in the paper by Coates, Lee, Ng [3]. The authors apply K-Means on randomly chosen patches on the training set and extract the centroids. The distance between each patch and the centroids are computed and normalized. Values less than the mean of the feature matrix is set to 0. This ensures that only relevant features are considered in computation. The resultant feature vectors for each patch become the training features. The same distance computation is done for the test data. The feature matrix is fed into a L2 SVM for training and testes on the test feature matrix. Using a cost C=50, we get 84.5% training accuracy and test accuracy 0.55905. The paper mentions an accuracy of ~79%. This discrepancy can be attributed to the difference in size of the training set.

We tried replacing L2 SVM with other classification methods such Seeded K Means, K-NN, Logistic Regression. Results indicate the L2 SVM gives the best accuracy.

Table 2. Classification Algorithm Paired with KMeans Performance

Classification Algorithm (used after feature KMeans extraction)	Accuracy on Test Data
L2 SVM C=50	0.56933
K-NN K=7	0.38419
Seeded K Means	0.32576
Multinomial Kernel Logistic Regression	0.49105
Random Forest (numTrees:50,maxDepth=20)	0.10895

We then try an estimate the parameter cost C, which gives the best results. From the plot we can observe that the train accuracy falls with the increase in Cost. However, we observe no significant fluctuation in the test accuracy. The test accuracy remained around ~55%.

Figure 1. Training Accuracy Vs. SVM Cost

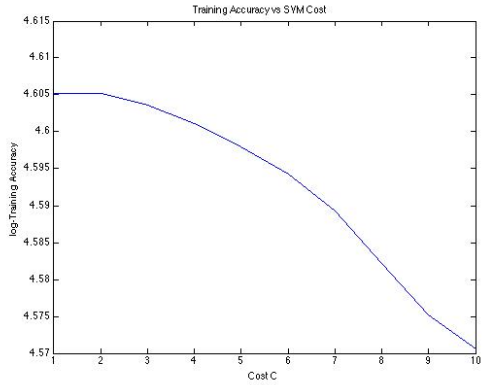
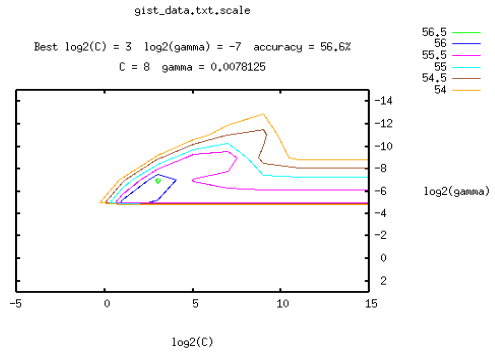


Figure 2. Local Optimums search for SVM parameter Selection



We further investigate the impact of the patch size and number of K-mean centroids (clusters). The paper [2] recommends using a patch size of (6px x 6px) and number of clusters K = 1500. We varied the patch size within 4,6,8 and number of clusters K = [1200,1300,1500,1600]. There was no change in training accuracy however the test accuracy dropped marginally for lower values of K. The patch size did not have any measurable effect. This could be attributed to the small size of the training and testing data.

3.8 PCA with Guassian SVM

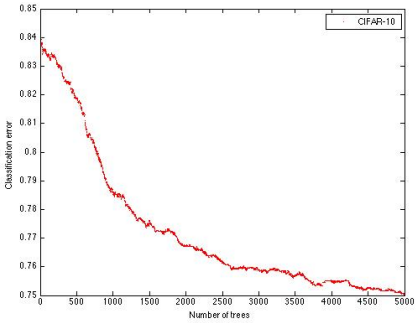
The Data parsed through principle component analysis while maintaining 95% variance. Training time using the WEKA GUI interface was approximately 1 hour. This produced 208 principle components, which is a reduced dimensionality size, thereby reducing the amount of data required.

A linear SVM was trained using the largest 200 principle components, using a Gaussian Kernel (Radial Basis Function). The principle parameters that can be adjusted are Gamma and C. A large value of Gamma makes a broad radial basis function, smoothing out the density estimation, increasing bias, and reducing variance. Smaller values of Gamma increase the variance, while decreasing the bias. On the other hand, C is the coefficient for the Slack. This means that the smaller the value of C, the more tolerance the final Support Vector Machine Classifier has for errors, and the SVM will tend to look for larger margin separations, even if this results in classifying more points. Despite knowing what these parameters mean, it is difficult and non-obvious to select the values of Gamma and C that will result in the decision boundary that will perform the best on the held out Test Dataset. The way in which we can select these dataset is via plotting the performance of the classifier on a held out cross-evaluation dataset. However this is also tedious to do by hand, and an exhaustive enumeration of all possible combinations, even within a small grid interval, is difficult. Initially we wrote a simple program that searched for all possible combinations of C and Gamma within an interval, trying to find the parameters that will result in the best cross-evaluation performance. However, this took in excess of 40 minutes just for a small search space. Instead, later on, we relied on tools like easy.py in the libSVM library, that carries out search pattern of the cost function using chosen scattered samples of C and Gamma. A contour plot of an example output of the search pattern is included below.

3.9 Adaboost with Trees

We used Matlab's inbuilt implementation of Adaboost on the training dataset, and achieved a 5-fold cross validation accuracy of 25%. We note that classification error drops with increasing number of weak learners.

Figure 3. Adaboost Cost vs Number of Weak Learners



4. ENSEMBLE LEARNING SYSTEM

After various experimentation as discussed above, we develop an ensemble learning system that uses the best performing methods that we found experimentally. Primarily we use the results from various K-Means with L2SVM parameter variations and Gist with SVM, combining it with a mixture of moderately performing classifiers such as Random Forest, Kernel Multinomial Logistic Regression.

The ensemble system uses a biased voting strategy where the most common class label predicted by each classifier is considered as the final predicted class label. However, if there is a tie, we default to the label predicted by the strongest individual classifier.

After using this ensemble classifier, we observer dramatic improvements in performance. The best combination gave a classification accuracy of 0.5965 on the test data set. We summarize the results of the various ensemble methods below.

Table 3. Training Accuracy Vs. SVM Cost

Ensemble Composition	Test Accuracy
KMeans-SVM + GIST-Gaussian SVM + MLR + Random Forest	58.933%
KMeans-SVM + GIST-Gaussian SVM + MLR + (PCA+MLR)	49.105%
KMeans(K=1600,P=6) - SVM+KMeans(K=1800,P=6)-SVM+ GIST-Gaussian SVM	58.152%
KMeans(K=1600,P=6) - SVM(C=2)+KMeans(K=1800,P=6)-SVM(C=40)+ GIST-Gaussian SVM	59.695%

5. RESULTS

In this section we summarize the best performance of all individual classifiers which were tested. Machines:

M1: 2.5 GHz Intel Core i5 Macbook Pro with 4GB RAM

M2: 1.5 GHz Intel core i5 Macbook Air with 4GB RAM

M3: 2.6 GHz Intel core i5 Macbook pro with 8GB of RAM

Table 4. Results of several classifiers

Classifier	Package/Tool	Machine	Training Time	Train Accuracy	Test Accuracy
Random Forest (trees: 500, depth: 50)	Weka 3.6.11 weka.classifiers.trees.RandomForest	M1	32s	100%	40.343%
Seeded K-Means	MATLAB Code Link:https://github.com/ImageClassification/CIFAR-10/blob/master/SeededKMeans.m	M1	19s	70%	23.5%
K-Means with L2 SVM	Third party library: http://www.cs.stanford.edu/~acoates/papers/kmeans_demo.tgz	M1	~15mins	100%	56.933%
GIST with Radial Kernel SVM	Third party library: http://people.csail.mit.edu/torralba/code/spatialenvelope/	M2	20 minutes	95%	55.943%
Adaboost with 5000 decision Trees	MATLAB Code Link:https://github.com/ImageClassification/CIFAR-10/blob/master/run_adaboost.m	M3	12.9 hours	95%	25.1%
MultiLayer Neural Network	Using Weka GUI	M2	30 minutes	91%	36%
NeuralNetwork with extractHOG Features	Matlab library	M2	20 minutes	93.5%	39%
Naive Bayes	Weka inbuilt NaiveBayes library	M2	20 seconds	94%	29.8%

The following table illustrates the improvement on baseline methods over two Kaggle runs:

Ensemble Composition	Test Accuracy
KMeans-SVM	0.55905
GIST-Gaussian SVM	0.55562
KMeans(K=1600,P=6) - SVM(C=2)+KMeans(K=1800,P=6)-SVM(C=40)+ GIST-Gaussian SVM	59.695%

The individual parameter tuning experiments are discussed in detail under each method under the methods section. The appropriate methodology is discussed in detail, which can make replication of our results convenient.

6. CONCLUSION

From our extensive survey and analysis we discovered that classifier performance generally improves with data-set size. This was observed even in state-of-the-art methods like deep learning methods which have been proved to perform well in image classification problems as per literature [4], we did not achieve a performance more than ~20%. This could be because of the limited training data set. For methods like AdaBoost that relied on a large number of weak classifiers, the need for a large feature rich data-set is more critical. In literature, AdaBoost had given results of about ~60% [3], however the available training data in published works was 50,000, over 10 times the data available in our experiments. With respect to using K-Means with L2 SVM we observed an accuracy of ~55%, which is contradictory to the 79.2% in [4]. With greater amounts of training data, we could probably increase the accuracy. Otherwise a linear, simpler classifier would work better.

We have also found that in machine learning, despite the abundance of tools and methods, not all methods perform well on a given dataset. One must often look into the types of assumptions that the technique assumes and whether the dataset follows these assumptions. We saw this when Multinomial Kernel Logistic Regression performed better than Random Forest and Linear SVM, and Random Forest in turn performed better than Naive Bayes. Here Naive Bayes did not perform well, because image pixels values are highly dependent on their neighboring pixel values. It is to be noted that after performing dimensionality reduction with Principal Component Analysis we observed a drop in performance with multinomial logistic regression.

Even within the methods established to perform the best for a given problem or dataset, we have found that extensive parameter tuning is a non-trivial and challenging problem to execute. Simply

by changing a parameter for the map size from 24 by 24 to 32 by 32 for the KMeans Algorithm, we were able to increase the accuracy rate on the held out dataset by over 1.5 percent. The idea that parameter tuning can be crucial to the performance of the network was proved for unsupervised learning algorithms in Ng et al [4], where it was shown that small stride size and whitening was critical to performance. An example of this is the Deep Convolution Neural Networks that we have tried to apply to the data-set. Despite state-of-the-art performance that some researchers have achieved using deep belief networks and convolutional neural networks, we have gotten only a fraction of the that performance. Among other reasons, the parameters involved in every layer of the network, like step size, momentum, output map size, as well as the type of layer - jitter, maximum pooling or linear regression and so on, can have a large impact on the final output accuracy.

Finally, we note that we have been successful in creating an ensemble that improves on our baseline. After combining and ensemble of the above-mentioned techniques we observe a high score of 0.59695. Our best guess is that combining very different moderately performing baseline methods gives a strong accuracy, and thus far our results have validated that hypothesis.

7. References and Citations

[1] "10-601 Fall 2014 Project - Object Recognition." *Data* -. Web. 6 Dec. 2014. <https://inclass.kaggle.com/c/10-601-fall-2014-project-object-recognition/data>.

[2] Web. 9 Dec. 2014. <http://arxiv.org/pdf/1106.0257.pdf>.

[3] "Weka 3: Data Mining Software in Java." *Weka 3*. Web. 9 Dec. 2014. <http://www.cs.waikato.ac.nz/ml/weka/>.

[4] A. Coates, H. Lee, and A. Y. Ng, "An analysis of single-layer networks in unsupervised feature learning," in AISTATS 14, 2011.

[5] Web. 6 Dec. 2014. <http://www.jdl.ac.cn/doc/2011/20141319501391488_2013-icip_pang-stochastic boosting for large-scale image classification.pdf>.

[7] Web. 8 Dec. 2014. <http://www.jmlr.org/proceedings/papers/v15/coates11a/coates11a.pdf>.

[6] Oliva, Aude, and Antonio Torralba. "Modeling the shape of the scene: A holistic representation of the spatial envelope." *International journal of computer vision*42.3 (2001): 145-175.

[8] "What Is the Class of This Image ?" *Classification Datasets Results*. Web. 6 Dec. 2014. <http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html>.

[9] "ECCV-2010 Tutorial: Feature Learning for Image Classification." *ECCV 2010 Tutorial on Feature Learning*. Web. 9 Dec. 2014. <http://ufldl.stanford.edu/eccv10-tutorial/>.

[10] "What I Learned from Competing against a ConvNet on ImageNet." *What I Learned from Competing against a ConvNet on ImageNet*. Web. 9 Dec. 2014. <http://karpathy.github.io/2014/09/02/what-i-learned-from-competing-against-a-convnet-on-imagenet/>.

[11] "K-nearest Neighbors Algorithm." *Wikipedia*. Wikimedia Foundation, 12 May 2014. Web. 9 Dec. 2014. <http://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm>.

[12] "Artificial Neural Network." *Wikipedia*. Wikimedia Foundation, 12 Aug. 2014. Web. 9 Dec. 2014. <http://en.wikipedia.org/wiki/Artificial_neural_network>.

[13] Maji, Subhransu, Alexander C. Berg, and Jitendra Malik. "Efficient classification for additive kernel SVMs." *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 35.1 (2013): 66-77.

[14] Chapelle, Olivier, Patrick Haffner, and Vladimir N. Vapnik. "Support vector machines for histogram-based image classification." *Neural Networks, IEEE Transactions on* 10.5 (1999): 1055-1064.

[15] Boiman, Oren, Eli Shechtman, and Michal Irani. "In defense of nearest-neighbor based image classification." *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*. IEEE, 2008.

APPENDIX

A. Task Division

Each team member explored a different baseline as per interest. The work was divided as follows:

Kavya worked on implementing and exploring various dimensionality reduction techniques like spatial envelope using GIST, variations of PCA, extracting HOG features, to try and extract the most influential and prominent features, that can retain the information and reduce the dimensions. After figuring out these three approaches, she experimented with different classifiers on dimensionally reduced data. Using SVM with RBF Kernels gave satisfactory results to an extent. She also implemented PCA with Multilayer Perceptron and Naive Bayes to explore the application of Naive Bayes to image classification. An interesting observation was that Naive Bayes combined with Nearest Neighbor can be used for image classification [15]

Matthew worked on Voting Algorithm, AdaBoosting Weak classifiers, Gradient Descent, Convolutional Neural Nets

Niloy worked on developing the seeded-Kmeans algorithm. He also tried out Random Forests. His primary focus was understanding and customizing the Unsupervised Feature Learning using KMeans algorithm implemented in [4].

As a team we worked on building the ensemble voting system and trying out various cocktails of classification algorithms to boost our accuracy and rank on Kaggle.

B. Key Learning

We learnt the importance of combining various baseline methods to develop a strong classifier for a given machine learning task. Importantly exploring various research papers helped us build upon the foundations which we learnt in class. Trying out tools like MATLAB, WEKA, Theano gave us a flavour of developing and evaluating algorithms using different platforms.

In retrospect, had we done this project again we should have more thorough literature survey and experimented the baseline methods earlier. Also, we should have looked at parallelising various algorithms. For example, AdaBoost and Neural Network algorithms were computationally expensive. We could have used GPUs for running our algorithm and Hadoop for many of the image pre-processing steps