

# Image Classification

Megan Schoendorf (megan2@stanford.edu)

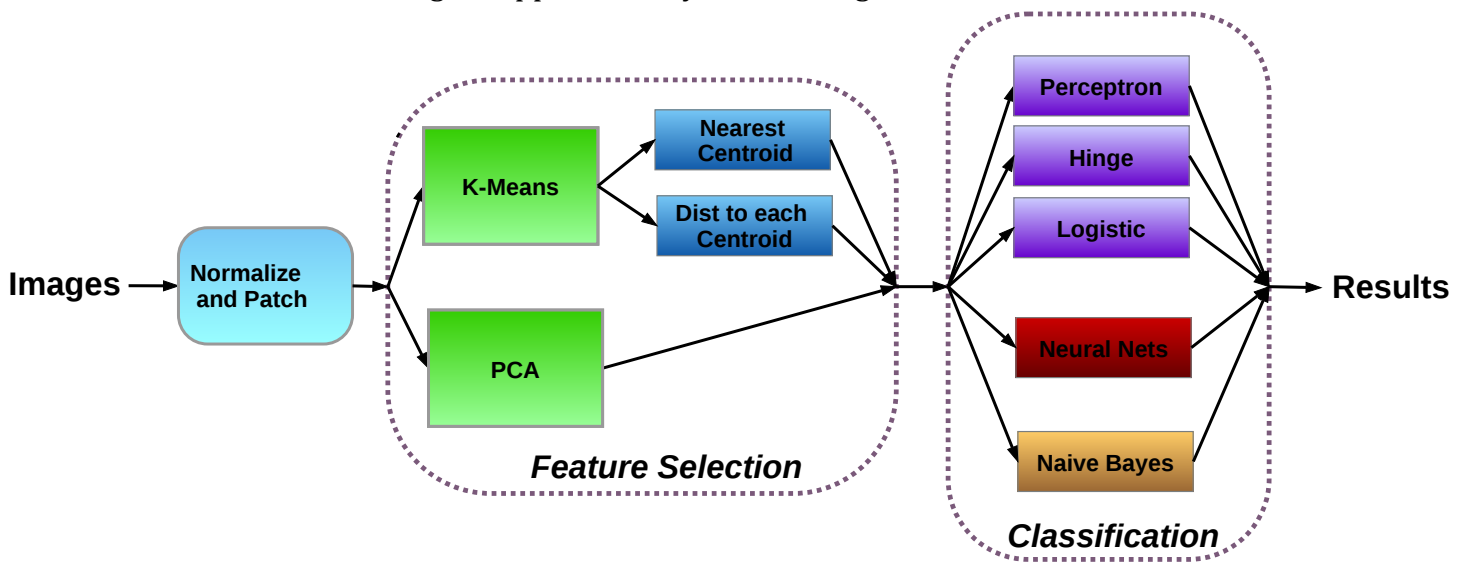
Christian Elder (celder@stanford.edu)

## Objective:

We explore several algorithms to classify images in the CIFAR-10 data set. There are approximately 50,000 labeled, 32x32-pixel, RGB images. Each has one of 10 labels: cat, dog, horse, etc. depending on the subject of the image.

## Summary: Methods and Results

After normalizing the data to have zero mean and unit variance, we used PCA or K-Means for feature extraction, followed by Gradient Descent, Neural Networks, or Naïve Bayes for classification. We found the best combination was K-Means for feature extraction and Gradient Descent with a logistic loss function for classification, which gave approximately 60% testing error with all 10 labels.



## Data Preprocessing

We initially experimented with compressing the RGB values of each pixel to single luminance value with the following equation:

$$y = .2126R + .7512G + .0722B$$

However, this step did not improve testing error, so we removed it.

Instead, we calculated the per-pixel mean and variance and removed them so that our data had zero-mean and unit-variance. This preprocessing had a small positive impact in our testing error. In the following equations,  $x_j^{(i)}$  is the jth pixel of the ith image.

$$1. \mu = \frac{1}{m} \sum_{i=1}^m x_j^{(i)} \quad 2. x^{(i)} = x^{(i)} - \mu \quad 3. \sigma_j^2 = \frac{1}{m} \sum_{i=1}^m x_j^{(i)^2} \quad 4. x_j^{(i)} = x_j^{(i)} / \sigma$$

Finally, we “patched” the image, dividing the image up into patches of size 1,2,4,8,16, or 32 pixels on a side, stacked those patches into a vector, and lined those vectors up into a matrix of size (patch size, number of patches). This is the data that we used for feature extraction.

## Feature Extraction: K-Means and PCA

### PCA

For PCA, instead of using the “patched” data, we stacked the images into vectors and created the matrix  $X$  with each column corresponding to one image. We then used PCA to reduce the dimensionality by creating a feature vector,  $y$ :

$$y^{(i)} = \begin{bmatrix} u_1^T x^{(i)} \\ \vdots \\ u_k^T x^{(i)} \end{bmatrix}$$

where  $u_i$  is the  $i$ -th principle component of the covariance matrix of the data. We used the number of eigenvalues greater than 0 as our value  $k$ . We experimented with using this method before, after, and instead of K-means.

### K-Means

K-means clustering is an unsupervised learning algorithm that groups the data into clusters about  $k$  centroids, minimizing the distortion function:

$$J(c, \mu) = \sum_{i=1}^m \|x^{(i)} - \mu_{c(i)}\|^2$$

where  $\mu_{c(i)}$  is the cluster centroid to which the  $i$ -th sample has been assigned. We accomplished this by repeatedly assigning the points to the nearest cluster and moving the clusters to the centroids of those assigned points. For the final feature extraction step, we had two methods: in the first we created a vector of size (number of patches \* number of clusters, 1) where each entry was the distance between a patch and a cluster. In the second, we used 0s and 1s to indicate, for each patch, which cluster was closest. We tuned the patch size, the number of clusters, and the number of iterations to optimize the testing error.

## Classifiers

### Gradient Descent with Regularization: Perceptron, Hinge, and Logistic

Image classification was initially attempted using a One vs. All classifier trained using stochastic gradient descent with one of three loss functions: Perceptron, Hinge, or Logistic. For each label, a binary classifier is trained, and an image is classified as the label corresponding to the largest “score” ( $\theta^T x$ ). To train, we are using the standard stochastic gradient descent update:

$$\theta_j := \theta_j + \alpha * [(y^{(i)} - h_{\theta}(x^{(i)})) * x_j^{(i)}]$$

where  $h_{\theta}$  is the loss function. The parameters of gradient descent include the number of training iterations and the learning step,  $\alpha$ . After each iteration through all the data, we added a regularization step:

$$\text{if } \|\theta\| > B: \theta = \frac{B}{\|\theta\|}$$

### Naïve Bayes

We paired this algorithm with K-Means and the nearest centroid indicator for feature extraction so that the feature vector would  $\in \{0,1\}$ . We calculated the conditional probabilities of  $p(x_j|y)$  and the margin  $p(y)$ :

$$p(x_j = 1|y = label_i) = \frac{\sum_{i=1}^m 1\{x_j^{(i)} = 1 \wedge y^{(i)} = label_i\} + 1}{\sum_{i=1}^m 1\{y^{(i)} = label_i\} + k}$$

$$p(y = label_i) = \frac{\sum_{i=1}^m 1\{y^{(i)} = label_i\}}{m}$$

We modeled X as binomial, so to calculate the  $p(x|y)$ :

$$p(x|y = label_i) = \prod_{j=1}^n p(x_j = 1|y = label_i) * x_j + p(x_j = 0|y = label_i) * (1 - x_j)$$

And finally, for the prediction, we choose the label that gave the maximum likelihood estimate:

$$Prediction = \operatorname{argmax}_i p(x|y = label_i) * p(y = label_i)$$

## Neural Networks

A neural network with one hidden layer was also implemented for comparison with the previous classification algorithms. Once the feature vector is extracted from the input data, it is fed into the neural network, which propagates the data through the hidden layer nodes to the output nodes via the following equation:

$$node_{i,out} := g\left(\sum_j W_{i,j} * node_{j,out}\right)$$

where  $W_{i,j}$  is the weight associated with the edge connecting  $node_j$  from the previous layer to  $node_i$  of this layer, and  $g$  is the activation function of  $node_i$ . We used the sigmoid function for  $g$ .

The weights are trained by propagating the error at the output nodes back through the network. Weights between the output layer and the hidden layer are updated via the following equations:

$$W_{i,j} := W_{i,j} + \alpha * node_{j,out} * \Delta_i$$

$$\Delta_i := (y^{(i)} - node_{i,out}) * g'\left(\sum_j W_{i,j} * node_{j,out}\right)$$

The weights between hidden layers and the input layer are updated via:

$$W_{j,k} := W_{j,k} + \alpha * node_{k,out} * \Delta_j$$

$$\Delta_j := g'\left(\sum_k W_{j,k} * node_{k,out}\right) * \sum_i W_{i,j} * \Delta_i$$

The parameters of neural nets include the number of training iterations and the learning step,  $\alpha$ .

## Procedure and Results

We found the best combination of algorithms were feature extraction by K-Means with distance to centroid and classification by gradient descent with a logistic loss function. In our milestone, we showed the effects of turning the K-Means parameters and found the best were 1x1 pixel patch sizes, 20 centroids, 20 K-means iterations, learning rate of 1 and no regularization. However, because we implemented all of these algorithms ourselves, speed and memory were an

impediment to running the tests with more than 8x8 pixel patches and 10,000 total images.

Figures 1 and 2 below were created using 500 images per label and the average of 5 runs per test. Figure 1 gives a comparison of the different classification algorithms, showing that logistic performs the best for the majority of numbers of labels. Figure 2 compares different feature extraction algorithms: none at all (just vectorize the image), PCA, and K-Means with distance to centroid. We tested with both logistic and perceptron loss functions and found that in both cases, K-means performed better than the other two feature extractors. For figure 3, we found a learning curve using all ten labels. As expected, the testing error decreases with number of training samples. Figure 4 shows the effect of training iterations of gradient descent on the training and testing error.

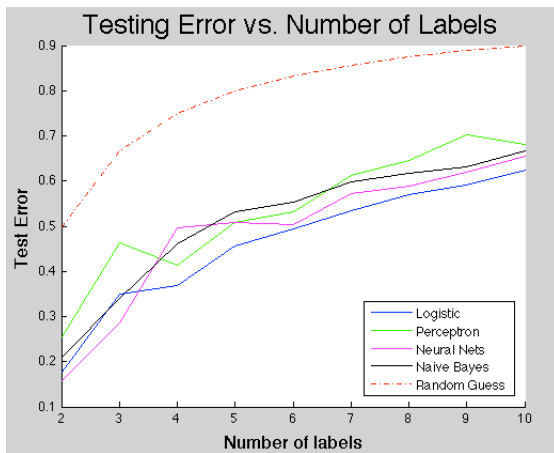


Figure 1 - Using K-Means feature extraction

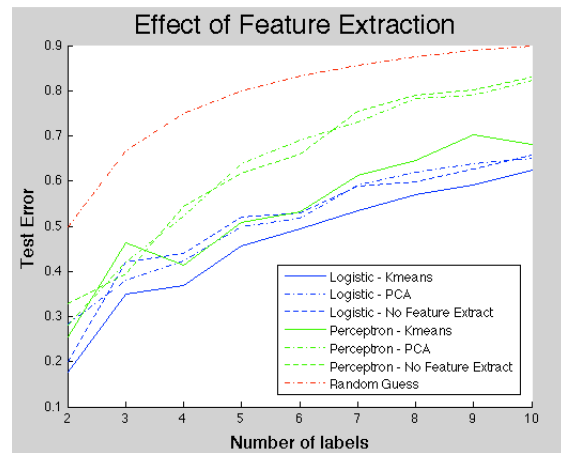


Figure 2

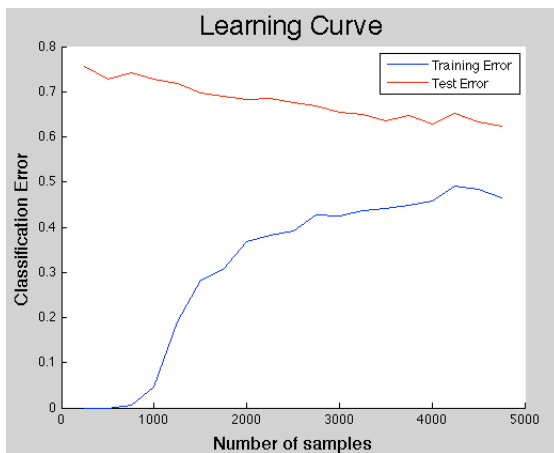


Figure 3 – Using K-Means feature extraction and gradient descent with logistic loss function, 10 labels

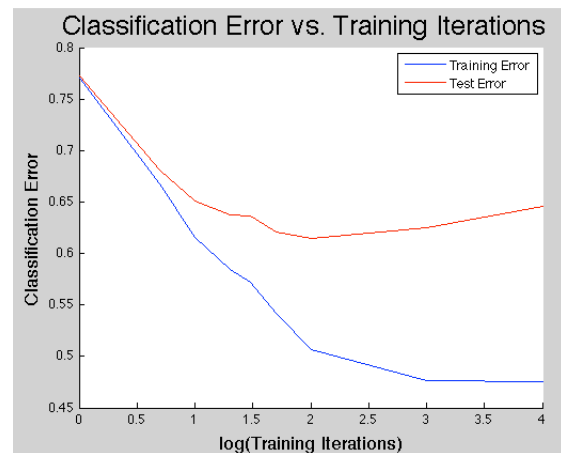


Figure 4 – Using K-Means feature extraction and gradient descent with logistic loss function, 10 labels

## Discussion

In our initial data from the milestone, we found that with proper parameters, all algorithms would yield a <10% training error, yet the testing error never went below 60%. We suspected over-fitting, and explored a few solutions including regularization, early stopping, and feature reduction. Regularization and PCA both hurt our classification error (see figure 2 for PCA results), but tuning the number of training iterations proved to be very important. In particular, the number of training iterations until convergence changed drastically with the number of training samples and finding the perfect number proved challenging.

The two parameters that had the largest effect on our classification error were training iterations and the feature extraction method. It is interesting to note that once all the parameters were turned, all of the classification algorithms performed roughly equivalently. This seems to suggest that it was not the classification algorithm that needed optimizing, but the feature extraction algorithm. Varying the parameters of the feature extraction algorithm had a much larger effect on the testing error. For K-Means, the number of centroids and patch size together determines the dimensionality of the feature vector to which each input patch is mapped. The number of centroids chosen must be high enough to distinguish dissimilar images, but low enough to avoid over-fitting the data or slowing down the algorithm too much.

The importance of differing feature extraction is corroborated by several papers, which espouse classification accuracy upwards of 80% [1]. One feature selection technique, made public by the originator of the CIFAR-10 dataset, involves training a neural network variant, Convolutional Neural Networks [2]. However, CNNs used for this purpose tend to be several layers deep, requiring computational power beyond our capabilities. For reference, the state-of-the-art CIFAR-10 classifier incorporates a four-layer CNN feature extractor which requires industrial-grade GPUs for reasonable training time [3]. As such, exploring more exotic and computationally-expensive feature extraction methods was deemed beyond the scope of this project.

## References

- [1] Benenson, Rodrigo. "Classification Dataset Results: CIFAR-10." Last modified November 30, 2013.  
[http://rodrigob.github.io/are\\_we\\_there\\_yet/build/classification\\_datasets\\_results.html#43494641522d3130](http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html#43494641522d3130).
- [2] Krizhevsky, Alex, Ilya Sutskever, and Geoff Hinton. "Imagenet classification with deep convolutional neural networks." In *Advances in Neural Information Processing Systems* 25, pp. 1106-1114. 2012.
- [3] Wan, Li, Matthew Zeiler, Sixin Zhang, Yann L. Cun, and Rob Fergus. "Regularization of neural networks using dropconnect." In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pp. 1058-1066. 2013.