Distracted Driver Detection :
Image Classification Using Unsupervised Feature Learning

MEI FU, HUI (PHOEBE) LIANG, JUNXU LU, CHIH-CHAN TIEN

UNIVERSITY OF MICHIGAN, ANN ARBOR

Unsupervised feature learning is an important research topic in machine learning as labeled training data is limited most of the time. Recent research has focused on deep neural networks with complex architectures such as convolutional neural network (CNN) for feature learning, which has been widely used in image classification tasks. However, such kind of deep neural networks often require intense data and computational resources for model training. In this report, we implement four different off-the-shelf feature learning algorithms from scratch (Sparse Autoencoder, Sparse Restricted Boltzmann Machine, K-means Clustering, Gaussian Mixtures) on a Kaggle dataset to classify distracted drivers into ten classes. We use the learned features with linear support vector machine (SVM) for prediction on the test data. In combination of convolutional feature extraction, we are able to achieve accuracy close to CNN on the dataset.

Concepts: • **Machine Learning • Unsupervised Feature Learning • Neural Networks • Image Classification**

1 Problem Statement and Motivation

Around one-fifth of motor vehicle accidents were caused by distracted drivers, as reported by the Center for Disease Control and Prevention. Such accidents have led to 425,000 injuries and 3,000 deaths each year. Therefore, it is crucial to detect the behaviors of drivers and prevent them from being distracted to avoid accidents in the first place. Our project aims to solve this problem by classifying images of safe drivers and distracted drivers of different forms.

The paper, *An Analysis of Single-Layer Networks in Unsupervised Feature Learning*, introduces a novel approach to learn features on images using much simpler systems than the current state-of-art deep neural networks. The paper was authored by Adam Coates, Andrew Ng, Honglak Lee and published in 2011 at the International Conference on Artificial Intelligence and Statistics conference (Coates et al., 2011). According to the results reported in the paper, the methods developed by the researchers are able to achieve the same level of accuracy as deep neural networks.

In the paper, several off-the-shelf unsupervised learning algorithms are incorporated to learn features on images. We implement these algorithms from scratch in Python and then feed the learned features and class labels into linear SVM for training and prediction. To compare with the state-of-art convolutional neural network, we also extend beyond the methods described in the paper and use transfer learning to train the images with VGGNet, the pre-trained Convolutional Neural Networks.

To describe the problem mathematically, the problem we are going to solve is multi-class image classification. Formally, a color image with a pixels wide and b pixels high, and with RGB colors (three channels for red, green, and blue) can be represented by a three-dimensional array of real numbers, $x \in R^{a \times b \times 3}$. The goal is therefore to find a function which is able to accurately classify an image into one of the M classes, or, $f : R^{a \times b \times 3} \rightarrow \{1, 2, 3, ..., M\}$ for each image represented by $x \in R^{a \times b \times 3}$.

2 Related Works

2.1 Sparse Autoencoder

An autoencoder neural network is an unsupervised learning algorithm that applies backpropagation, setting the target values to be equal to the inputs, in other words, it is trying to learn an approximation to the identity function, so as to output $\hat{x}$ that is similar to x (Ng et al., 2011). So, to some extend, an autoencoder is learning the features of the input x automatically. By changing the number of nodes in the hidden layer, we can control the size of the features we extract (i.e. how small is the size that we reconstruct x into). Also, by imposing a sparsity constraint on the hidden units, we can force the hidden unit's activations to be mostly near 0, then, the autoencoder will still discover interesting structure in the data, even if the number of hidden units is large. (Ng et al., 2011)

2.2 Sparse Restricted Boltzmann Machine

The Sparse Restricted Boltzmann Machine (RBM) is an energy-based two-layer generative network model. The model is optimized when the energy function is minimized. The Bernoulli RBM model assumes the inputs are either binary values or values between 0 and 1. Hinton proposed a fast learning algorithm Contrastive Divergence (CD) to speed up the sampling learning process (Hinton, 2002), which became a general approach to train the RBM model. The RBM model has many applications including labeled or unlabeled images (Hinton et al., 2006) and user ratings of movies (Salakhutdinov et al., 2007). In this project, we implemented this method to extract the hidden features of the input image.

2.3 K-means Clustering

K-means algorithm is a non-probabilistic method of unsupervised learning which groups observation vectors into clusters. Lloyd (1982) proposed the iterated algorithm which ensures the convergence of the local minimum. In the original K-means method, each data point is assigned to one cluster, this is said to be of hard assignment. Fuzzy C-means, proposed by Dunn (1973) and Bezdek (1981), is an extension of the original k-means clustering which allows for each data point to be assigned to be multiple clusters, also referred to as soft assignment. Recently, K-means has been used widely used in computer vision to define image features. For instance, Winn, Criminisi, and Minka (2005) used K-means clustering to select visual words as features of the images. Also, Li and Perona (2005) used K-means clustering to learn the codebook of the images.

2.4 Gaussian Mixture Model

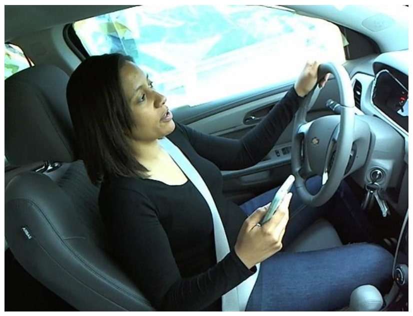
Conceptually, a Gaussian mixture model is a probabilistic model with the assumption that "all the data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters" ("2.1 Gaussian mixture models"). The advantage of Gaussian mixture model is that it doesn't know in advance which distribution the data point comes from and learns the unknown parameters, the means and covariance of each distribution, automatically. As a result, Gaussian mixture model is considered a method for unsupervised learning. Expectation Maximization is an iterative technique often used in Gaussian mixture model to learn the latent variables (Bishop, 2006). In our project, this is essentially soft k-means, which assigns a data point a probability how likely it belongs to one of the clusters/features.

3 Dataset

Our dataset is provided by State Farm on the Kaggle competiton website (The link to download data: https://www.kaggle.com/c/state-farm-distracted-driver-detection). The original dataset contains both training and test datasets. Since we do not have labels for the test data and is limited in computing resources, we use only the labeled training data for the purpose of our project. There is a total of 22,000 labeled images, and each class has about 10% of the total number of images. Additionally, we randomly split the labeled images from all classes into training, validation, and test, with the ratio of 6: 2: 2.

It contains images of drivers with different actions. The actions are divided into safe driving (c0) or 9 distracted behaviors: c1: texting - right, c2: talking on the phone - right, c3: texting - left, c4: talking on the phone - left, c5: operating the radio, c6: drinking, c7: reaching behind, c8: hair and makeup, c9: talking to passenger.

Below are two sample images of the dataset. On the left is an image classified as class 0: "safe driving". On the right the image is classified into class 1 "texting-right".



The paper, *An Analysis of Single-Layer Networks in Unsupervised Feature Learning*, is focused on image classification on the CIFAR-10 and NORB datasets. The CIFAR-10 dataset consists of 60,000 32x32 color images in 10 classes, with 6,000 images per class. The classes are completely mutually exclusive. NORB contains images of 50 toys belonging to 5 generic categories. The objects' images were taken by two cameras under 6 lighting conditions, 9 elevations (30 to 70 degrees every 5 degrees), and 18 azimuths (0 to 340 every 20 degrees).

CIFAR-10 dataset is mainly designed for object classification, and NORB dataset is mainly designed for object recognition. So, most of the images within these two datasets contain an isolated and clear object in the middle of the image. However, the images in our dataset are human driving actions, which may contain more noises. Also, the size of our dataset is relatively smaller than CIFAR-10 and NORB datasets.

4 Methodology

The methods are based on the paper of Coates, Lee, and Ng (2011). We also modify the methods to better fit solving our problem. Below we explain how we implement the methods. There are fours ways of extracting and transforming features from the original feature vectors. The four subsections below illustrates the four methods we use to perform the tasks.

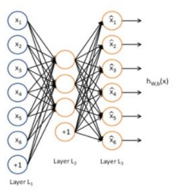
4.1 Data Pre-processing for All Methods

The color image was originally of size 480×640 pixels. We first resize it to be 48×64 pixels. We represent a patch from the image or the whole image as a 1-d array with the length of the array being the product width and height multiplied by 3 as each pixel has three intensities of colors of red, blue, and green. The collections of patches are therefore represented by a matrix. Then we in turn normalize (by demeaning and dividing by the variance) and whiten the data before we further extract features using the four methods as discussed below.

4.2 Sparse Autoencoder

We implemented a single hidden layer sparse autoencoder to extract features with *k* hidden nodes in Layer L_2 as shown in Figure 5-1 (Ng, A., 2011). The goal of this algorithm is to optimize a certain loss function to let the output Layer L_3 be similar to the input Layer L_1 after the process of encoding and decoding. Such a loss function is defined as formula (5-1, 5-2), which constitutes of the mean square error, weight decay, and Kullback–Leibler (KL) divergence.

Fig 4-2: Single hidden layer Sparse Autoencoder



Minimizing the mean square error term enables the input data matrix be similar to the output matrix, and the weight decay term controlled by the hyperparameter λ prevents overfitting. The minimizing of the KL divergence term (sparsity penalty term) controlled by the hyperparameter β makes the average output of the hidden nodes in L_2 be close to a hyperparameter ρ, which is typically a small number close to 0. We optimized the loss function by using the forward and back propagation. The input *X* is a batch of preprocessed images.

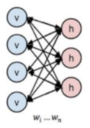
$$J(W,b) = \left[\frac{1}{n}\sum_{i=1}^{n}\left(\frac{1}{2}\|h_{w,b}(x^{(i)}) - y^{(i)}\|^2\right)\right] + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_l+1}(W_{ji}^{(l)})^2 \qquad J_{sparse}(W,b) = J(W,b) + \beta\sum_{j=1}^{s_2}\rho log\frac{\rho}{\hat{\rho}_j} + (1-\rho)log\frac{1-\rho}{1-\hat{\rho}_j} \tag{4-1, 4-2}$$

After training the sparse autoencoder on the training dataset, we got a set of weights (W1, b1, W2, b2). Next, we extracted features from the test dataset using the sparse autoencoder we already trained. Then, we made the prediction of each test data based on the extracted features (i.e. the hidden layer of the sparse autoencoder).

4.3 Sparse Restricted Boltzmann Machine

Different from the Sparse Autoencoder, the Sparse Restricted Boltzmann Machine (RBM) is a generative model with a two-layer stochastic network (one visible layer L_1 and one hidden layer L_2 assuming all nodes are binary). It defines the energy of a model, the joint configuration (*v, h*) of the visible and hidden nodes (Hopfield, 1982), given by (5-3), where v_i, h_j are the binary states of visible node *i* and hidden node *j*, a_i, b_j are their biases and w_{ij} is the weight between them (Hinton, 2010). The network is optimized when the energy reaches minimum.

Fig 4-3: RBM



$$E(\mathbf{v},\mathbf{h}) = -\sum_{i\in\text{visible}} a_i v_i - \sum_{j\in\text{hidden}} b_j h_j - \sum_{i,j} v_i h_j w_{ij} \tag{4-3}$$

In 2002, Hinton proposed a fast learning algorithm Contrastive Divergence (CD) to speed up the sampling (Hinton, 2002), which is the approach that we implemented in this project.

Step 1- Uniformly initialize weights sampled from the range of ± 0.1 * sqrt(6. / (n_visible + n_hidden)) with mean 0 and standard deviation 0.1, and initialize hidden bias and visible bias with 0.

Step 2 - Positive CD Phase: propagate the visible units activation upwards to the hidden nodes with sigmoid function (5-4). The state of the positive hidden nodes is 0 or 1, where 1 represents the positive hidden probability is greater than the value of sampling from a uniform distribution over [0,1).

Step 3 - Negative CD Phase: propagate the hidden nodes activation downwards to the visible nodes with sigmoid function (5-5), and calculate the probability of the negative hidden nodes again. The state of the negative hidden nodes is 0 or 1, where 1 represents the negative hidden probability is greater than the value of sampling from a uniform distribution over [0,1).

Step 4 - Update the weights (5-6) and the error. We also added a weight decay term when updating the weights. The error is calculated by taking the sum square of the difference of the input data and p_i.

$$p(h_j = 1) = \sigma(b_j + \sum_i v_i w_{ij}) \qquad p_i = p(v_i = 1) = \sigma(a_i + \sum_j h_j w_{ij}) \qquad \Delta w_{ij} = \epsilon(\langle v_i h_j\rangle_{data} - \langle v_i h_j\rangle_{recon}) \tag{4-4, 4-5, 4-6}$$

4.4 K-means Clustering

For this method, we first apply k-means clustering to learn K clusters. Specifically, for P patches or observations $x_1, x_2, ..., x_i, ..., x_P$, each of which is of N features, or $x_i \in R^N$ for any i, we find K mean vectors or the centroids $\mu_1, \mu_2, ..., \mu_k, ..., \mu_k \in R^N$ and K collections of sets $\widehat{S} = \{S_1, S_2, ..., S_k, ..., S_k\}$ which forms a partition of the patches $\{x_i\}$ to minimize the mean square error, or,

$$\min_{\widehat{S}} \sum_{k=1}^{K} \sum_{x \in S_k} \|x - \mu_k\|^2 \tag{4-7}$$

where the centroid of class k is $\mu_k = \frac{1}{|S_k|} \sum_{x \in S_k} x$, which is the average of the vectors of x in the class of S_k .We use the standard k-means algorithm or the Lloyd's algorithm to find the centroids and the clusters. Specifically, we first initialize the centroids by randomly drawing a number from the Gaussian distribution for every component in every centroid vector. Then for each feature vector of a patch , given all the centroids, we find the centroid which is closest to the feature vector in Euclidean distance. Formally, we assign x to cluster k^* if

$$\|x - \hat{\mu}_{k^*}\|^2 \leq \|x - \hat{\mu}_k\|^2 \text{ for all } k \in \{1,\ 2,\ ...,\ k,\ ...,\ K\}\,. \tag{4-8}$$

Given the assignment of the clusters, we then update the centroids by averaging the features across patches and get

$$\mu_k = \frac{1}{|S_k|} \sum_{x \in S_k} x\,. \tag{4-9}$$

This concludes one iteration. Given the new centroids, we begin the next iteration by assigning the clusters. We repeat the iteration until the loss defined by the mean square error converge.

Given P patches of images, and each patch $x_i \in R^N$ is with N features, and K centroids which we got from applying the k-means algorithm, we derive the K features by using the "triangle activation function." Specifically, we compute the Euclidean distance between the feature vector x and every centroid μ_k , and the distance between x and the centroid k is $\|x_i - \mu_k\|$. For each centroid, we also compute the average distance between it and all the feature vectors, or $\frac{1}{P}\sum_{i=1}^{P} \|x_i - \mu_k\|$. We set the k-th transformed feature of the patch x_i to be $y_{i,k} = \max\{0,\ \frac{1}{P}\sum_{i=1}^{P} \|x_i - \mu_k\| - \|x_i - \mu_k\|\}$. That is, we set the k-th transformed feature of the patch x_i to be the difference between the the average distance across patches and the distance between the patch and the k-th centroid, unless the patch is to far away from the k-th centroid, in which case the transformed feature is set to be 0. For each patch x_i, we collect K transformed features and get a transformed vector $y_i = (y_{i,1},\ y_{i,2}, ...,\ y_{i,k}, ..., y_{i,K})$ of K components for each patch.

4.5 Gaussian Mixture Model

Compared with K-means, Gaussian mixture model takes cluster covariance matrices into consider when assigning probabilities of data points to the clusters. We use the expectation-maximization algorithm, an iterative method, to find the unknown latent variables for the Gaussian mixture model.

Step 1 - Initialization: Instead of randomly picking data points as the initial centroids, we run one iteration of k-means to get the initial cluster centroids and use them as the means for the number of clusters determined. We also initialize the covariance matrices of each cluster to be identity matrices.

Step 2 - Expectation: evaluate the probability of each data point to see how likely it belongs to the different cluster. We use the probability density function of a multivariate Gaussian with the initialized means and covariance matrices.

$$\gamma(z_{nk}) = \frac{\pi_k \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}. \tag{4-11}$$

Step 3 - Maximization: update parameters, which include means, covariance matrices, and prior probabilities. The update rules are as follows:

$$\begin{aligned} \boldsymbol{\mu}_k^{\text{new}} &= \frac{1}{N_k}\sum_{n=1}^{N}\gamma(z_{nk})\mathbf{x}_n \\ \boldsymbol{\Sigma}_k^{\text{new}} &= \frac{1}{N_k}\sum_{n=1}^{N}\gamma(z_{nk})\left(\mathbf{x}_n - \boldsymbol{\mu}_k^{\text{new}}\right)\left(\mathbf{x}_n - \boldsymbol{\mu}_k^{\text{new}}\right)^{\text{T}} \\ \pi_k^{\text{new}} &= \frac{N_k}{N} \end{aligned} \tag{4-12}$$

Step 4: Check for convergence of the log-likelihood if the epsilon is smaller than 0.01.

$$\ln p(\mathbf{X}|\boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi}) = \sum_{n=1}^{N} \ln\left\{\sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)\right\} \tag{4-13}$$

Given P patches of images, and each patch $x_i \in R^N$ is with N features, and K mean vectors μ_k and K variance–covariance matrix Σ_k which we got from the Gaussian Mixture Model, we can derive the K features by using the probability density function of the multivariate Gaussian distribution. Specifically, for each patch $x_i \in R^N$, we compute the k-th transformed feature by computing the probability density given the mean being μ_k and the variance–covariance being Σ_k and compute the probability density,

$$y_{i,k} = \frac{1}{\sqrt{|2\pi\Sigma_k|}} \exp(-\tfrac{1}{2}(x_i - \mu_k)^T \Sigma^{-1}(x_i - \mu_k)).$$

Again, we collect the K transformed features and derive, for each patch, a transformed feature vector of dimension K, $y_i = (y_{i,1}, y_{i,2}, ..., y_{i,k}, ..., y_{i,K})$.

4.6 Feature Extraction and Classification

4.6.1 Convolutional Extraction of Patches for K-means and Gaussian Mixture

In the sub-sub-section above, we illustrated how we transform a patch with N features into K features. Below we explain how we apply this transformation in our problem as we extract patches from the images.

Every color image is first resized to n_1 pixels wide, and n_2 high, and we choose $(n_1, n_2) = (48, 64)$. We chose the receptive field size $w = 6$, that is, we extract patches of size 6×6 from every image. The way we extract the patches and features is according to the method called convolutional extraction. When extracting the patches from the image, we also set the stride to be 1, or $s = 1$, that is, each patch is distanced from one another by 1 pixel. Then for each color patch extracted, the number of features is $N = 6 \times 6 \times 3 = 108$. For each patch, we first normalize and whiten the patches. Then we apply the transformation which we introduced in the previous sub-sub-section to get K features. For this project, we tried three different number of K features: 100, 200, and 300.

For each image, there are $\lfloor \frac{n_1}{w+s} \rfloor \times \lfloor \frac{n_2}{w+s} \rfloor$ patches, and each patch is with K transformed features. For each image, we divide the patches into four quadrants, Q_1, Q_2, Q_3, Q_4, and pool all patches in one quadrant by summing up all their transformed feature vectors y_i, or $\hat{y}_q = \sum_{y_i \in Q_q} y_i$, for $q = 1, 2, 3, 4$. We then finally construct the feature vectors of the by stacking the four vectors $\hat{y}_1, \hat{y}_2, \hat{y}_3, \hat{y}_4$ to form the feature vectors of the image, which is of length $4K$.

4.6.2 The linear Classification Model

Before feeding the data into classification model, we also standardize each image by subtracting the mean and dividing the standard deviation. The classification model we used is soft-margin SVM. Given a test data, the input of this model is an array of the features, and the output is the most likely category of it.

To tune the hyperparameters in linear SVM, we tried four different values of the penalty parameter C of the error term, which include [1, 5, 10, 50]. The higher the penalty parameter is, the harder the margin is. We use sklearn.svm.LinearSVC library for the prediction. The LinearSVC library implements “one-vs-the-rest” multi-class modeling, thus training 10 different models based on the number of classes you have.

5 Evaluation

5.1 Baseline Accuracy

The baseline model is a VGG-19 model using 5 convolutional blocks and 3 fully connected layers at the end. Each convolutional block uses relu activation function and includes a max-pooling layer. The structure of the convolutional layers is 2 layers (12 nodes) x 2 layers (32 nodes) x 4 layers (64 nodes) x 4 layers (128 nodes) x 4 (128 layers), taking an input shape of 3 x 48 x 48. The 0.5 drop-out layer and normalization layer are added before each fully connected layers. It achieved 82.3% accuracy on the testset using 7 hours 45 minutes running on the google cloud.

5.2 Classification Accuracies Based on the Sparse Autoencoder Model

As discussed in 5.5.1 Features Extraction (a) The Model of the Sparse Autoencoder, we built classification model based on the extracted features of the images. We built two sparse autoencoders, the hidden layers are 12 x 16 and 24 x 32 respectively, and we tried different number of epochs (5 and 10). The classification model we used is soft-margin Support Vector Machine and we tried different values of tuning parameter C to find the best classifier. By comparing the results of different models (different in parameters of sparse autoencoder, soft-margin SVM and whitening) we found that, the best model combination is: 1) sparse autoencoder with 24 x 32 nodes in the hidden layer; 2) training 5 epochs; 3) whitening; 4) soft-margin SVM with C=5.

Table of Classification Accuracies Based on the Sparse Autoencoder Model

Feature Size (# nodes in hidden layer)	Corresponding Parameters of Sparse Auto-encoder	Whitening or Not	C (SVM)	Accuracies
12 x 16	**epoches=5,** batch_size=100, alpha=0.1, beta=6, Lambda=0.001, rho=0.05	TRUE	10	training: 56.92% **test: 56.34%**
			5	training: 59.88% **test: 58.64%**
			1	training: 55.52% **test: 53.82%**
		FALSE	10	training: 51.38% **test: 49.54%**
			5	training: 52.55% **test: 51.53%**
			1	training: 49.31% **test: 47.71%**
24 x 32	**epoches=5,** batch_size=100, alpha=0.1, beta=6, Lambda=0.001, rho=0.05	TRUE	10	training: 80.80% **test: 79.38%**
			5	training: 82.90% **test: 82.25%**
			1	training: 80.43% **test: 79.38%**
		FALSE	10	training: 79.72% **test: 77.06%**
			5	training: 79.77% **test: 78.15%**
			1	training: 80.69% **test: 79.58%**
24 x 32	**epoches=10**, batch_size=100, alpha=0.1, beta=6, Lambda=0.001, rho=0.05	TRUE	10	training: 67.58% **test: 65.91%**
			5	training: 62.07% **test: 60.96%**
			1	training: 63.68% **test: 62.23%**
		FALSE	10	training: 55.50% **test: 52.78%**
			5	training: 48.82% **test: 47.05%**
			1	training: 55.50% **test: 52.78%**

5.3 Classification Accuracies - Sparse Restricted Boltzmann Machine

Similar to the Sparse Autoencoder, we built classification model based on the extracted features of the images. We trained two types of RBM model, one with 12x16 hidden nodes, and one with 24x32 hidden nodes. The epochs, alpha, and lambda are tuned based on the learning curve. We saved the weights for these two models using the best hyperparameters. When training the linear SVM classifier, we compared different values of C, whether whitened or not, and with two sizes of the hidden features. We found that the best model combination is: 1) sparse RBM with 24 x 32 nodes in the hidden layer; 2) training 10 epochs; 3) non-whitening; 4) soft-margin SVM with C=50.

Table of Classification Accuracies - Sparse Restricted Boltzmann Machine

Feature Size (# nodes in hidden layer)	Corresponding Parameters of Sparse RBM	Whitening or Not	C (SVM)	Accuracies
12 x 16	epochs=50, alpha=0.02, Lambda=0.0002	TRUE	50	training: 66.36% **test: 63.28%**
			10	training: 46.84% **test: 45.11%**
			5	training: 26.45% **test: 26.31%**
			1	training: 12.73% **test: 12.40%**
		FALSE	50	training: 87.70% **test: 87.07%**
			10	training: 72.23% **test: 71.64%**
			5	training: 60.83% **test: 59.26%**
			1	training: 27.53% **test: 27.38%**
24 x 32	epochs=10, alpha=0. 02, Lambda=0.0002	TRUE	50	training: 95.64% **test: 93.20%**
			10	training: 88.62% **test: 86.78%**
			5	training: 77.15% **test: 75.34%**
			1	training: 18.20% **test: 18.24%**
		FALSE	50	training: 97.82% **test: 96.90%**
			10	training: 93.20% **test: 92.82%**
			5	training: 87.81% **test: 87.25%**
			1	training: 57.34% **test: 58.04%**

5.4 Classification Accuracies - Gaussian Mixture Model and K-means Clustering

Table of Test Accuracy - Gaussian Mixture Model

Features Sets (# of centroids)	Whitening or Not	C Penalty (LinearSVM)	Accuracies
100	TRUE	1	Validation: 65.41%, Test: 65.73%
		5	Validation: 63.07%, Test: 63.12%
		10	Validation: 61.45%, Test: 61.32%
		50	Validation: 53.76%, Test: 53.02%
200	TRUE	1	Validation: 78.87%, Test: 79.38%
		5	Validation: 77.03%, Test: 77.77%
		10	Validation: 76.51%, Test: 77.26%
		50	Validation: 74.02%, Test: 76.37%
300	TRUE	1	Validation: 81.10%, Test: 80.91%
		5	Validation: 78.90%, Test: 79.26%
		10	Validation: 78.90%, Test: 78.68%
		50	Validation: 78.73%, Test: 78.99%
300	FALSE	1	Validation: 85.93%, Test: 85.89%
		5	Validation: 84.14%, Test: 84.01%
		10	Validation: 83.72%, Test: 83.86%
		50	Validation: 83.78%, Test: 83.86%

Table of Test Accuracy - K-means Clustering

Features Sets (# of centroids)	Whitening or Not	C Penalty (LinearSVM)	Accuracies
100	TRUE	1	Validation: 93.37%, Test: 93.18%
		5	Validation: 91.97%, Test: 91.84%
		10	Validation: 90.86%, Test: 90.90%
		50	Validation: 90.89%, Test: 90.32%
100	FALSE	1	Validation: 90.72%, Test: 90.59%
		5	Validation: 89.60%, Test: 89.50%
		10	Validation: 88.32%, Test: 88.72%
		50	Validation: 84.53%, Test: 84.28%
200	TRUE	1	Validation: 96.96%, Test: 96.79%
		5	Validation: 96.74%, Test: 96.79%
		10	Validation: 96.74%, Test: 96.79%
		50	Validation: 96.74%, Test: 96.79%
200	FALSE	1	Validation: 97.05%, Test: 97.06%
		5	Validation: 96.46%, Test: 96.83%
		10	Validation: 96.46%, Test: 96.83%
		50	Validation: 96.46%, Test: 96.83%
300	TRUE	1	Validation: 98.05%, Test: 97.37%
		5	Validation: 98.05%, Test: 97.37%
		10	Validation: 98.05%, Test: 97.37%
		50	Validation: 98.05%, Test: 97.37%
300	FALSE	1	Validation: 82.77%, Test: 83.50%
		5	Validation: 83.03%, Test: 83.34%
		10	Validation: 79.79%, Test: 80.53%
		50	Validation: 74.08%, Test: 76.52%

6 Conclusions

Recent research has focused on deep neural networks with complex architectures for feature learning. In this project, we focused on using simpler neural network structures to extract features, and achieved close results as deep neural networks. Among the methods we tried, K-means Clustering has the best performance (the highest test accuracy is 97.37%), other methods also achieved decent results on test dataset: Sparse Restricted Boltzmann Machine (96.90%), Gaussian Mixture Model (85.89%), Sparse Autoencoder (82.25%). Also, whitening improves the classification performance in some of the methods. From the results we can see that, deep neural networks are not irreplaceable in feature extraction. We learned the strengths of properly extracting features using unsupervised methods before conducting the supervised learning and the weaknesses are that it requires several preprocessing steps such as triangle activation function on the data to get accurate results.

In this project, we successfully implements several data preprocessing and feature extraction methods from scratch. However, we only used linear soft-margin SVM for the classification and prediction. In order to improve the performance of our models, more classification methods need to be considered.

7 Description of Individual Effort

Mei Fu: Implement Sparse Restricted Boltzmann Machine and part of Sparse Autoencoder model from scratch. Train VGG19 comparison model. Write sections of the report based on implemented model.

Hui (Phoebe) Liang: Research and identify machine learning papers. Implement Gaussian Mixture Model from scratch and preprocess and split data for the team to use. Write sections of the report based on implemented model.

Junxu Lu: Implement part of Sparse Autoencoder model from scratch. Implement soft-margin SVM for classification based on Sparse Autoencoder model. Write corresponding sections in the report.

Chih-Chan Tien: Implement K-means from scratch. Implement data preprocessing and convolutional extraction. Implement SVM for classification. Write sections of the report based on implemented model.

REFERENCES

2.1 Gaussian mixture models. (n.d.). Retrieved December 17, 2017, from http://scikit-learn.org/stable/modules/mixture.html

A Beginner's Tutorial for Restricted Boltzmann Machines. Retrieved on December. 2017, from https://deeplearning4j.org/restrictedboltzmannmachine.

Bezdek, J.C. (1981). Pattern Recognition with Fuzzy Objective Function Algorithms. Plenum Press.

Bishop, C. M. (2006). Pattern Recognition and Machine Learning (Information Science and Statistics). Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Coates, A., Ng, A., & Lee, H. (2011, June). An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics* (pp. 215-223).

Dunn, J.C. (1973). A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters. J. Cybernet. 3, 32–57.

Erhan, D., Bengio, Y., Courville, A., Manzagol, P. A., Vincent, P., & Bengio, S. (2010). Why does unsupervised pre-training help deep learning?. *Journal of Machine Learning Research*, *11*(Feb), 625-660.

Hinton, G. E. (2006). Training products of experts by minimizing contrastive divergence. *Training*, *14*(8).

Hinton, G. (2010). A practical guide to training restricted Boltzmann machines. *Momentum*, *9*(1), 926.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79:2554–2558.

Huang, F. J., & LeCun, Y. (n.d.). THE NORB DATASET, V1.0. Retrieved December 17, 2017, from https://cs.nyu.edu/~ylclab/data/norb-v1.0/

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (pp. 1097-1105).

Li, Fei-Fei and P. Perona (2005). A Bayesian hierarchical model for learning natural scene categories. Computer Vision and Pattern Recognition.

Lloyd, S. (1982). Least squares quantization in PCM. IEEE Trans. Inform. Theory 28, ,129–137.

Ng, A. (2011). Sparse autoencoder. CS294A Lecture notes, 72(2011), 1-19.

Salakhutdinov, R. R., Mnih, A., and Hinton, G. E. (2007). Restricted Boltzmann machines for collaborative filtering. In Ghahramani, Z., editor, *Proceedings of the International Conference on Machine Learning*, 24(2007), 791–798.

Winn, J. , A. Criminisi, and T. Minka (2005). Object categorization by learned universal visual dictionary. International Conference on Computer Vision

The CIFAR-10 dataset. (n.d.). Retrieved December 17, 2017, from https://www.cs.toronto.edu/~kriz/cifar.html