Toward a Sequential Approach to Pipelined Image Recognition

Derek Rose, Itamar Arel
Dept. of Electrical Engineering and Computer Science
Machine Intelligence Lab
University of Tennessee
derek@utk.edu, itamar@ieee.org

***Abstract*—This paper introduces a sequentially motivated approach to processing streams of images from datasets with low memory demands. We utilize fuzzy clustering as an incremental dictionary learning scheme and explain how the corresponding membership functions can be subsequently used in encoding features for image patches. We focus on replicating the codebook learning and classification stages from an established visual learning pipeline that has recently shown efficacy on the CIFAR-10 small image dataset. Experiments show that performance near batch oriented learning is achievable by combining naturally online learning mechanisms driven largely by stochastic gradient descent with strictly patch-wise operations. We further detail how backpropagation can be used with a neural network classifier to modify parameters within the pipeline.**

***Index Terms*—sequential learning; image recognition; neural networks**

I. Introduction

There has been a recent surge of interest in multi-stage methods for constructing features or descriptors from images which are good for classifying objects taken from a variety of lighting conditions, scales, and poses. A variant of these multi-stage methods was highlighted in exhaustive experiments by [1], [2] that compared features learned and coded from techniques which stem from deep learning [3], multi-stage architectures [4], and bags of features models [5]. Their results provide both a clear description of an effective pipeline as well as evidence of the value of applying this method to images from challenging datasets, at the time producing state-of-the-art accuracy when classifying images from CIFAR [6] and NORB.

Object recognition using multi-stage architectures pipeline operations such as breaking images down into local parts (or descriptors such as SIFT [7]), learning regularities, building combined features to represent images, and finally classifying each image based on such combined features [1], [8]. Often the methods within each stage treat image data as if it were available to process as a large batch during learning, such as for unsupervised feature learning (sparse coding or k-means) and supervised classification (linear support vector machines). In some applications, this data is only available as a stream of images or may not be stored within working memory for learning. Further, regularities in the images may change during streaming, altering their characteristics and resulting feature distributions. It is thus preferable to sequentially or incrementally process images with online learning algorithms for dictionary building, classification, and any learned pre-processing.

In this paper, we design and evaluate methods for learning stages from the visual pipeline in [1] incrementally, primarily for dictionary learning and classification. Our approach contributes a scheme to modify the feature mapping during classifier training by using a discriminative backpropagation signal in the feature encoding stage. We employ a feedforward neural network for classification and creation of the backpropagation signal, noting the potential for expansion into ensemble learning [9].

II. Description of Stages within the Pipeline

The multi-stage image recognition architecture considered consists of: patch extraction, normalization and whitening, dictionary learning, feature encoding, spatial pooling, and classification. Learning is performed over the dictionary and classification stages which we address sequentially; labels are used at the classifier while unsupervised learning vector quantization is used for building the dictionary. We next detail each of these stages, noting that our contributions lie in the dictionary learning, feature encoding, and classification stages which are discussed more in Sections III,IV.

A. Patch Extraction

Breaking the dataset into patches allows us to learn an overcomplete basis to represent the patch space, where we have potentially more components in the basis (or codebook or dictionary) than dimensions in the patch. In practice, codebooks larger than the dimension of patches have shown to produce excellent results on the CIFAR and MNIST datasets [1], [8], [10]. We work with an image dataset where images stream in which we may extract small patches of raw pixel intensities from. If an image $\mathbf{X}$ is $n \times n \times d$, we densely sample $(n(w-1)+1)^2$ square patches of size $w \times w \times d$ from each image ($d = 1$ for grayscale, $d = 3$ for color). For ease of notation these patches are flattened into vectors $\mathbf{x}^i$ for processing, and for supervised learning stages we assume that each patch retains the label y_j from image $\mathbf{X}_j$. Labels are not necessary for dictionary learning, nor do each of the patches need to be from a coherent stream of images, but may be be

sampled randomly from the dataset. The full dense extraction is only necessary when creating a feature vector to classify $\mathbf{X}$.

B. Patch-wise Normalization

Each patch $\mathbf{x}^i$ is preprocessed with a local brightness and contrast normalization as a subtraction of the within-patch mean pixel value and division by the pixel variance. Normalization procedures that weight pixel adjustments based on their position in patch (e.g. with a Gaussian window) were not considered, although this has seen some success by [4] with inspiration from computational neuroscience. This normalization is meant to enforce "local" competition among adjacent feature values and between features at the same location in different patches.

C. Whitening

We next perform whitening, a common pre-processing stage that removes second-order correlation from each dimension of data to reduced redundancy. Natural images have a high degree of correlation between adjacent pixel values as color shifts are usually smooth. Whitening also scales all variances such that the covariance matrix for a transformed set of patches is the identity matrix (sphering). Whitening by principal component analysis [11] is commonly used, although it requires estimating the covariance matrix of the patch dataset as well as performing an eigenvalue decomposition on this covariance matrix. This remains the one stage of our architecture which currently necessarily utilizes the entire dataset for calculating the sample covariance matrix. In practice, each of the learned transform component vectors resembles a shifted single-pixel center-surround filter [11], [12], hinting that this stage can be replaced by simpler methods for natural image scenes. The future research section discusses options for handling this to complete the online system.

We first perform an eigenvalue decomposition on the covariance matrix of a large set of mean centered patches sampled from the full dataset:

$$\mathbf{C} = E\left[\mathbf{x}\mathbf{x}^{\mathrm{T}}\right] = \mathbf{E}\mathbf{D}\mathbf{E} \tag{1}$$

with $\mathbf{E}$ a matrix of orthogonal eigenvalues and $\mathbf{D}$ is a diagonal matrix of corresponding eigenvalues. The whitening transform is performed by multiplying by transform matrix $\mathbf{V}$ such that

$$\mathbf{y} = \mathbf{V}\mathbf{x}. \tag{2}$$

$\mathbf{V} = \mathbf{D}^{1/2}\mathbf{E}^{\mathrm{T}}$ may be chosen such that the transformed data has an identity covariance matrix, i.e. $E\left[\mathbf{y}\mathbf{y}^{\mathrm{T}}\right] = \mathbf{I}$. However, $\mathbf{V}$ may be left multiplied by any orthogonal matrix and still whiten, and the choice of $\mathbf{V} = \mathbf{E}\mathbf{D}^{1/2}\mathbf{E}^{\mathrm{T}}$ is particularly special as it is both the only symmetric whitening matrix and the inverse square root of $\mathbf{C}$. This particular choice of transform is called zero-phase whitening, as when viewed from the perspective of the frequency domain as filters the transform components approximate center-surround filters that are spatially localized [11], [12].

The whitening transform is important for this approach as it enables the use of a simple feature learning algorithm over more complex techniques (k-means vs. sparse-coding) and empirically boosts final classification accuracy regardless of feature learning technique [1]. The codebook learned by clustering zero-phase whitened data approximates edge-filters, which have been shown to be the independent components learned from whitened data for natural images [12], providing an intuition as to why codebooks learned with whitening work so well.

D. Dictionary Learning

We utilize fuzzy algorithms for learning vector quantization (FALVQ) [13], [14] to learn the codebook or dictionary over the image patches extracted from the dataset. FALVQ is a competitive clustering algorithm which converges faster than winner-take-all k-means learning as each presented example contributes a small update for each centroid. Further, in our experiments with small patches the FALVQ algorithm is much more resilient to poor or random initializations; here we seed the algorithm with the first k presented patches. While the advantages seen in this stage diminish with available training data, in a truly sequential learning scheme lower error codebooks that can be reached faster are less likely to disrupt the subsequent training of a classifier. This stage produces a set of k prototypes $\mathbf{v}_i$. Details for this algorithm are presented in Section III.

E. Feature Encoding

The dictionary is used as a reference to create the features which are eventually delivered to a classifier. This stage maps the $w \times w \times d$ raw pixels into a k dimensional vector. From [1] we adopt the triangle threshold encoding scheme as a baseline as it was shown to produce excellent results when compared with sparse coding. The triangle encoding for a patch $\mathbf{x}$ is given by

$$f_i\left(\mathbf{x}\right) = \max\left\{0, \mu\left(\mathbf{d}\right) - d_i\right\} \tag{3}$$

where distance $d_i = \left\|\mathbf{v}_i - \mathbf{x}\right\|_2$ and the mean of all the distances over every dimension is $\mu(\mathbf{d}) = \frac{1}{k}\sum_{i=1}^{k} d_i$. This encoding emphasizes all centroids which are closer to the pattern than on average and creates a degree of sparsity through the zeroing of all features for centroids which are no closer than average. Drawing inspiration from the clustering performed during dictionary learning, we introduce the use of the FALVQ1 membership function for encoding with thresholding, as performed by triangle encoding. This is a relative encoding that produces a membership value $u \in [0, 1]$ as a function of the distances $\mathbf{d}$ above and adjustable parameter vector $\boldsymbol{\alpha}$, which may compress or expand membership values in each dimension. We will later discuss a method for learning the parameters with gradient descent. Thresholding is performed as in the triangle encoding (although flipped as higher memberships imply closer centroids):

$$f_i\left(\mathbf{x}\right) = \max\left\{0, u_i - \mu\left(\mathbf{u}\right)\right\} \tag{4}$$

where $\mathbf{u}$ is now the vector of memberships. In practice, we have found thresholding (and resulting artificial sparsity prior to pooling) is incredibly important for achieving higher accuracy. Further, although it involves a non-linearity, we set the derivative of any $f_i(\mathbf{x}) = 0$ to zero (see Equation 24), allowing us to use the derivative vector for a restricted update in practice. Early tests also indicate that the hard thresholding at zero can be replaced with a smooth logistic thresholding function that remains differentiable while achieving a similar reduction of non-zero features.

F. Pooling

Feature pooling is biologically founded in complex cell functions and is used to instill a degree of invariance of object position (translation) and minor visual changes (which may be noise or sensor distortion) by down-sampling the obtained coded features $f_i(\mathbf{x})$. Pooling also significantly reduces the dimensionality of the features as a practical matter prior to classification. This stage thus builds what are termed "mid-level" features from an assumption of small degrees of spatial invariance for low-level features (coded from patches) [15].

For each codebook dimension, coded features for each patch sampled from the image are arranged spatially corresponding to their image position and a pooling operator is performed over pre-defined pools. Operator functions include max, average, or p-norm and ignore spatial position within the pool. For spatial pyramids, the pool is assumed square with size $p \times p$ [5]. If the coded features in the pool are represented as a vector $\mathbf{y}_i = \left[f_i(\mathbf{x}^1), f_i(\mathbf{x}^2), \ldots, \mathbf{x}^{p*p})\right]$ and $y_{i,j} = f_i(\mathbf{x}^j)$, the average pooling function is

$$f_{avg}(\mathbf{y}_i) = \frac{1}{p^2}\sum_{j=1}^{p^2} y_{i,j} \tag{5}$$

which will be utilized for experiments here as it is both differentiable and works well for this class of problems. If there are P pools, we obtain a resulting feature vector for classification of length $P \times k$.

G. Classification

For the classification stage, it is common to utilize a linear support vector machine (SVM). SVMs don't immediately lend themselves to streams of data, as they require solving an unconstrained optimization problem over all available data points to find the support vectors for the margin. Although recent work has managed to make large datasets tractable for SVMs by processing examples sequentially and keeping a set of examples that form the support vectors [16], here we instead focus on utilizing a neural network for classification due to the ease of sequential training and fixed memory requirements. We note that recent developments in combining multiple networks into an ensemble [9] may aid in overall classification performance and can be performed online, although we focus on classifying with a single network.

Prior to classification, we normalize the data by mean subtracting and dividing by the standard deviation for each dimension. This stage has proven somewhat problematic to perform online, as our estimate of standard deviation learned sequentially is an order of magnitude low due to the infrequency of high input values. A discussion of options for addressing this issue is provided in Section VI.

III. Codebook Creation: Fuzzy Algorithms for Learning Vector Quantization

We next present the algorithm detailed in [13], [14]. Assume that a set of feature vectors $\mathbf{x} \in \mathcal{X} \subset \mathbb{R}^n$ are to be represented using a set of k prototypes $\mathcal{V} = \{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k\}$ with a loss function

$$L_x = L_x(\mathbf{v}_r) = \sum_{r=1}^{k} u_r(\mathbf{x}) \left\|\mathbf{x} - \mathbf{v}_r\right\|, \quad r = 1, 2, \ldots, k. \tag{6}$$

to be minimized instantaneously, as minimizing the expectation over all $\mathbf{x}$ is a problem for gradient descent. If the closest "winning" prototype is $\mathbf{v}_i$ then the membership mapping is

$$u_r(\mathbf{x}) = u_{ir} = \begin{cases} 1 & if\, r = i \\ u\left(\frac{\|\mathbf{x}-\mathbf{v}_i\|^2}{\|\mathbf{x}-\mathbf{v}_r\|^2}\right) & if\, r \neq i \end{cases}. \tag{7}$$

To obtain relative contribution of each prototype, the membership function $u(\cdot)$ should be of the form $u(z) = zp(z)$ where $z = \frac{\|\mathbf{x}-\mathbf{v}_i\|^2}{\|\mathbf{x}-\mathbf{v}_r\|^2}$. Admissible choices of function $p(z)$ must be differentiable and satisfy several conditions as outlined in [13] that ensure appropriate membership functions for fair competition among prototypes. Three families of functions for $p(z)$ were originally proposed; here we focus on the first, FALVQ1, with membership function

$$u(z) = \frac{z}{1+\alpha z}. \tag{8}$$

The interference function $w(z)$ defines how much non-winning prototypes affect the winning prototype update with

$$\Delta\mathbf{v}_i = \eta(\mathbf{x} - \mathbf{v}_i)\left(1 + \sum_{r\neq i}^{k} w_{ir}\right) \tag{9}$$

where

$$w_{ir} = w(z) = p(z) + zp'(z) = u'(z) \tag{10}$$

and $\eta \in [0, 1]$ is an adjustable learning rate parameter. We utilize a learning rate which decays every iteration. The interference function $n(z)$ defines how much the winning prototype affects the non-winning prototypes with the update rule for $\mathbf{v}_j \neq \mathbf{v}_i$:

$$\Delta\mathbf{v}_i = \eta(\mathbf{x} - \mathbf{v}_i)\, n_{ij} \tag{11}$$

where

$$n_{ij} = n(z) = -z^2 p'(z) = u(z) - zu'(z). \tag{12}$$

For FALVQ1, this implies $w(z) = (1+\alpha z)^{-2}$ and $n(z) = \alpha z^2 (1+\alpha z)^{-2}$. The slightly modified sequential algorithm as used is summarized in Algorithm 1.

Select: k, $\eta_0 \in [0,1]$, $\eta_{decay} < 1$, η_{final} stopping rate, initial codebook $\mathcal{V}_0 = \{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k\}$
$step \Leftarrow 0$
$\eta \Leftarrow \eta_0$
while $\eta > \eta_{final}$ and inputs $\mathbf{x}$ available **do**
$step \Leftarrow step + 1$
$\eta \Leftarrow \eta * \eta_{decay}$
$i \Leftarrow \arg\min_j \|\mathbf{x} - \mathbf{v}_j\|^2$ {find winning prototype}
$u_{ir} \Leftarrow u \left(\|\mathbf{x} - \mathbf{v}_i\|^2 / \|\mathbf{x} - \mathbf{v}_r\|^2 \right) \forall r \neq i$
$w_{ir} \Leftarrow w \left(\|\mathbf{x} - \mathbf{v}_i\|^2 / \|\mathbf{x} - \mathbf{v}_r\|^2 \right) \forall r \neq i$
$n_{ir} \Leftarrow n \left(\|\mathbf{x} - \mathbf{v}_i\|^2 / \|\mathbf{x} - \mathbf{v}_r\|^2 \right) \forall r \neq i$
$v_i \Leftarrow v_i + \eta (\mathbf{x} - \mathbf{v}_i) \left(1 + \sum_{r \neq i} w_{ir} \right)$ {update winning prototype}
for all $v_j \neq v_i$ **do**
$v_j =\Leftarrow v_j + \eta (\mathbf{x} - \mathbf{v}_j) n_{ij}$ {update non-winning prototypes}
end for
end while

Algorithm 1: FALVQ Learning

IV. Utilizing Supervised Backpropagation for Encoding Adjustment

We propose a methodology for utilizing a supervised signal obtained from the neural network to modify the encoding performed by tuning the fuzzy membership parameter. The concept of using discriminative learning for adapting a dictionary was performed by [15] in the context of sparse coding, although we note that we do not consider modifying the dictionary here. This could be done by altering the membership parameter and continuing to train the dictionary. Further, the discriminative update could be applied to any parameter within the pipeline as long as we may smoothly differentiate the input to the network with respect to that parameter. This does restrict the choice of pooling function, although in our trials the non-linear max operation was less useful than averaging. When using a neural network, the computation of the input gradient for updates is not costly as the network already computes most of the gradient information during standard backpropagation training. The remainder is calculated with the derivative of the feature values prior to pooling.

A. Derivation of Update Rule

First assume that we are interested in adjusting input vector $\mathbf{x}$ to reduce the network error J. It is helpful to borrow notation from gradient descent backpropagation which adjusts network weights $\mathbf{w}$ to minimize error. As we shall detail, several of these calculations may be reused. In general, for no particular layer, the gradient descent adjustment to the network weights is given by

$$\Delta \mathbf{w} = -\eta \frac{\partial J}{\partial \mathbf{w}} = -\eta \frac{\partial J}{\partial \mathbf{net}} \frac{\partial \mathbf{net}}{\partial \mathbf{w}}, \tag{13}$$

where $\mathbf{net}$ is a vector of activation inputs and we define the error as

$$J(\mathbf{w}) = \frac{1}{2} \sum_k^{N_O} (t_k - y_k)^2 . \tag{14}$$

We may define sensitivities in a two layer network for N_O output layer units indexed on k and N_H hidden layer units indexed on j as follows:

$$\delta_k^{(2)} = -\frac{\partial J}{\partial net_k^{(2)}} = (t_k - y_k) f_O' \left(net_k^{(2)} \right) \tag{15}$$

and

$$\delta_j^{(1)} = -\frac{\partial J}{\partial net_j^{(1)}} = f_H' \left(net_j^{(1)} \right) \sum_k^{N_O} w_{jk}^{(2)} \delta_k^{(2)}. \tag{16}$$

This allows us to express the updates for input to hidden weights $\mathbf{w}^{(1)}$:

$$\Delta w_{ij}^{(1)} = -\eta \frac{\partial J}{\partial w_{ij}^{(1)}} = -\eta \frac{\partial J}{\partial net_j^{(1)}} \frac{\partial net_j^{(1)}}{\partial w_{ij}^{(1)}} = \eta \delta_j^{(1)} \frac{\partial net_j^{(1)}}{\partial w_{ij}^{(1)}} \tag{17}$$

As a straightforward extension of this idea, we may adjust an input x_i to reduce the error by the same gradient descent rule using

$$\begin{aligned} \Delta x_i &= -\eta \frac{\partial J}{\partial x_i} = -\eta \frac{\partial J}{\partial \mathbf{net}^{(1)}} \frac{\partial \mathbf{net}^{(1)}}{\partial x_i} \\ &= -\eta \frac{\partial J}{\partial \mathbf{net}^{(1)}} \begin{bmatrix} w_{i1} & \cdots & w_{ij} & \cdots & w_{iN_H} \end{bmatrix}^T \\ &= \eta \sum_j^{N_H} \delta_j^{(1)} w_{ij} = \eta {\boldsymbol{\delta}^{(1)}}^T {\mathbf{w}_i^{(1)}}^T \end{aligned} \tag{18}$$

as $net_j^{(1)} = \sum_i^{N_I} w_{ij}^{(1)} x_i$ with N_I inputs. For a vector of inputs, the update on $\mathbf{x}$ becomes:

$$\Delta \mathbf{x} = -\eta \frac{\partial J}{\partial \mathbf{x}} = -\eta \frac{\partial J}{\partial \mathbf{net}^{(1)}} \frac{\partial \mathbf{net}^{(1)}}{\partial \mathbf{x}} = \eta {\boldsymbol{\delta}^{(1)}}^T {\mathbf{w}^{(1)}}^T \tag{19}$$

If we are instead interested in modifying a parameter, α, that affects the value of each input x_i (i.e. $x = f(\alpha, \ldots)$), we must extend the chain rule with the update:

$$\Delta \alpha = -\eta \frac{\partial J}{\partial \alpha} = -\eta \frac{\partial J}{\partial \mathbf{x}} \frac{\partial \mathbf{x}}{\partial \alpha} \tag{20}$$

In the case of an input created from pooling membership values, this chain rule can be extended further:

$$\begin{aligned} \Delta \alpha_i &= -\eta \frac{\partial J}{\partial \mathbf{x}} \frac{\partial \mathbf{x}}{\partial \alpha_i} \\ &= \eta {\boldsymbol{\delta}^{(1)}}^T {\mathbf{w}^{(1)}}^T \begin{bmatrix} \frac{\partial x_1}{\partial \alpha_i} & \cdots & \frac{\partial x_i}{\partial \alpha_i} & \cdots & \frac{\partial x_{N_I}}{\partial \alpha_i} \end{bmatrix}^T \end{aligned} \tag{21}$$

where

$$x_i = \sum_p^{|pool|} u_p \tag{22}$$

with u_p as the membership value that contributes to the pooled input x_i and

$$\frac{\partial x_i}{\partial \alpha_i} = \frac{\partial \sum_p^{|pool|} u_p}{\partial \alpha_i} = \sum_p^{|pool|} \frac{\partial u_p}{\partial \alpha_i}. \tag{23}$$

When encoding using the FALVQ1 membership function, the computation of the membership derivative is straightforward:

$$\frac{\partial}{\partial \alpha_i} u(z_i) = \frac{-x_i^2}{(1+\alpha_i x_i)^2} = -u(z_i)^2. \tag{24}$$

V. Testing Procedure

Most of the parameters selected by [1] through cross-validation are kept here for comparison. We extract square patches from the CIFAR-10 small image dataset [6] with $w = 6$, where each image is $32 \times 32 \times 3$ resulting in 27×27 patches. After patch normalization and whitening (which was computed using PCA over the available training data) we use the FALVQ1 algorithm as presented above for learning a dictionary of $k = 400$ with k patches presented each iteration and an initial learning rate of $\frac{1}{k}$ that decays every iteration by a factor of $2e^{-4}$. Membership parameter $\alpha = 0.1$ was chosen during codebook learning (from recommendations in [13]) and we halt updates when the learning rate drops below $1e^{-6}$; in practice these choices are flexible. Although [1] was able to achieve better results using larger codebook sizes, we are interested in relative performance and have run our tests with $k = 400$ based on memory and computation limits. We use a pool size of $p = 14$ for four pooled quadrants over the coded features and a resulting feature vector of dimension 1600 for classification. Code released by Coates et. al. for [1] was used as a starting point for these trials. During dictionary training we treat CIFAR-10 as a stream of unlabeled images sampled at random and learn from a random subset of patches within each mini-batch of ten images. During classifier training, we extract a dense feature vector using the learned mappings.

For classification, we perform a sweep over regularization parameter C for the linear SVM to obtain the best performance for reporting (cross-validation would be necessary in practice). For the neural network, we use 128 hidden neurons in a single hidden layer with random small initial weights. We use an initial stepsize of $1/128$ with stochastic meta descent and targets $t \in \{-1, 1\}$. When looking into tuning encoding parameters, we initially zero α in Equation 8. This choice stems from results obtained with SVM sweeps. A small learning rate $\eta = 5e^{-4}$ is used to adjust α and we cycle learning α for 10 mini-batches after training the network for 100 mini-batches. We reserve ten thousand (20%) of the training examples from CIFAR-10 to perform periodic validation checks on our network.

VI. Notes on normalization

As was discussed in Section II-G, pooling often creates inputs with very large variance ranges that make online normalization difficult. This is problematic when performing classification with a neural network due to the sensitivity of the initial weights to normalized data. We found that imposing a minimum variance (0.1) and adjusting up from this with online learning performed on par to using the true sample variance, likely as this minimum imposed an additional degree of compression of inputs up from the true variance. Alternatively, a small batch of samples can also be used to learn an estimated mean and standard deviation by averaging over each dimension. This average can be applied to each dimension for the remainder of training, although we found this hinders the best attainable performance. For consistency, we report results from normalization with the actual sample standard deviation calculated beforehand, assuming the encoding parameter $\alpha = 0$.

After investigating mean square error increases during parameter learning we discovered that online learning for normalization of the mean or standard deviation of inputs must be discontinued if network training has concluded as modifying the encoding parameter changes the statistics of the inputs to the network. The normalization attempts to compensate for these changes and subsequently modifies the mapping the network has learned.

VII. Preliminary Results

Table I contains a summary of results comparing the proposed fuzzy encoding and triangle encoding. It also compares dictionary learning (FALVQ) with batch k-means as well as SVM classification to an online neural network. For CIFAR-10, the dictionary learning may be converted from k-means to FALVQ without a loss in performance. Although we consistently get roughly 1% less error with the triangle encoding than the proposed fuzzy encoding with a linear SVM, we do not see the same discrepancy when training with a neural network, with the gap falling to 0.5%. This may be due to the difficulty in normalizing data appropriately that we discuss above, although the inherently scaled fuzzy encoding may be easier to pre-process.

Modifying the encoding parameter through backpropagation fails to increase performance reliably; however, we note that triangle encoding results may indicate that the network has reached its upper bound on performance for this number of features without more sophisticated pre-processing and the tuning parameter may be fighting this restriction. While attempting to verify the derivative signal's applicability with thresholding and normalization, we performed trials with fixed normalization constants ($\mu = 0.06, \sigma = 0.045$) and constant shifting with thresholding (0.5) during feature encoding. These tests were more informative on how tuning the encoding parameters could be beneficial for classification accuracy. In some trials, after stopping network learning based on validation checks, we saw a nearly immediate 2% jump in accuracy from modifying α: validation set accuracy rose from 62.8% to 65.2%. We also saw jumps of similar magnitude when placing an artificial bound on the maximum post-normalized features (i.e. setting values outside the top 5% of the normal curve to 1.645). The low initial maximum accuracy reduces the impact

Table I
CIFAR-10 TEST SET ACCURACY COMPARISON (AVERAGED OVER 5 TRIALS). NETWORK COLUMNS ALSO CONTAIN AVERAGE NUMBER OF DATASET PRESENTATIONS BEFORE VALIDATION STOP.

Encoding \ Training Details	**Triangle**	**Fuzzy**
Batch k-means, SVM, Full CIFAR-10 (as in [1])	73.61% [0.08]	72.28% [0.28]
FALVQ, SVM, Full CIFAR-10	73.6% [0.21]	72.52% [0.28]
FALVQ, SVM, 80% CIFAR-10	73.2% [0.29]	72.22% [0.14]
FALVQ, Online NN, 80% CIFAR-10	71.35% [0.31], 69	70.84% [0.2], 41
FALVQ, Online NN, Adjusted α, 80% CIFAR-10	–	70.98% [0.36], 59

of these results, but does indicate that discriminative tuning can yield worthwhile improvements.

We found that simultaneous learning of both the encoding parameters and network weights does not perform well in practice. Validation performance for simultaneous learning lags behind that of a network which uses a fixed $(\alpha = 0)$ parameter, and this discrepancy persists when convergence is indicated by mean-square error readings or validation accuracy. This hints at problems when using a neural network to learn over an adapting input space.

VIII. FUTURE WORK AND CONCLUSIONS

In this paper, we have shown the viability of replacing both large batch oriented dictionary learning as well as classification stages with online components that provide little to no degradation in performance as demonstrated on the CIFAR-10 dataset. These components have a fixed memory usage and are able to process as few as one sample per iteration, allowing them to be used in constrained memory applications, for large datasets, or for datasets where the feature representation is high dimensional. As shown by [1], expanding the number of dimensions even beyond the dimension of the original image has proven to work well, and for datasets with larger images the required memory for learning quickly grows. These components also work well when the entire dataset is not available, such as learning on streams. We have further introduced a method for adjusting the feature encoding process with discriminative guidance, which has opened questions on how we may tune pipelines like these during training and what considerations should be made, particularly for normalization of features for classification. Additional work is ongoing to realize greater gains in performance and we plan to investigate pre-processing of pooled features for classification as such features often have outliers that make scaling difficult for neural networks. Verifying that tuning performance scales for larger dictionaries (and/or additional pools) is also planned; backpropagation could become cumbersome for large numbers of inputs. We further intend to apply smarter step sizes to the encoding parameter adjustment and look into other parameters in the model that may be learned through supervised tuning.

In online systems or systems with large image patches, whitening with PCA may be impractical or impossible due to the intractability of solving the eigenvalue decomposition of large matrices or the difficulty in estimating the covariance matrix of the pixel values from the image stream. For the future, we plan to consider replacing offline whitening with online PCA or whitening filters as outlined in [11]. Our initial work on replacing PCA with candid covariance-free incremental principal component analysis [17] is promising and recreates the center-surround filter shape, however noise in these learned components produces a noticeable degradation in the learned dictionary.

REFERENCES

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

[2] A. Coates and A. Y. Ng, "The importance of encoding versus training with sparse coding and vector quantization," in *Proceedings of the 28th International Conference on Machine Learning*, 2011.

[3] Y. Bengio, "Learning deep architectures for AI," *Foundations and Trends in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[4] K. Jarrett, K. Kavukcuoglu, M. A. Ranzato, and Y. LeCun, "What is the best multi-stage architecture for object recognition?" in *Proceedings of International Conference on Computer Vision (ICCV'09)*, 2009.

[5] S. Lazebnik, C. Schmid, and J. Ponce, "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories," in *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, vol. 2, 2006, pp. 2169–2178.

[6] A. Krizhevsky, "Learning multiple layers of features from tiny images," *Master's thesis, Department of Computer Science, University of Toronto*, 2009.

[7] Y. L. Boureau, F. Bach, Y. LeCun, and J. Ponce, "Learning mid-level features for recognition," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, 2010, pp. 2559–2566.

[8] Y. Jia, H. Chang, and T. Darrel, "Beyond spatial pyramids: Receptive field learning for pooled image features," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, 2012.

[9] Y. Liu and X. Yao, "Ensemble learning via negative correlation," *Neural Networks*, vol. 12, no. 10, pp. 1399–1404, 1999.

[10] Q. V. Le, A. Karpenko, J. Ngiam, and A. Y. Ng, "ICA with reconstruction cost for efficient overcomplete feature learning," in *NIPS*, 2011.

[11] A. Hyvärinen, J. Hurri, and P. O. Hoyer, *Natural Image Statistics — A probabilistic approach to early computational vision*. Springer-Verlag, 2009.

[12] A. J. Bell and T. J. Sejnowski, "The "independent components" of natural scenes are edge filters," *Vision research*, vol. 37, no. 23, pp. 3327–3338, 1997.

[13] N. B. Karayiannis, "A methodology for constructing fuzzy algorithms for learning vector quantization," *Neural Networks, IEEE Transactions on*, vol. 8, no. 3, pp. 505–518, 1997.

[14] N. B. Karayiannis and P. I. Pai, "Fuzzy algorithms for learning vector quantization," *Neural Networks, IEEE Transactions on*, vol. 7, no. 5, pp. 1196–1211, 1996.

[15] Y. L. Boureau, J. Ponce, and Y. LeCun, "A theoretical analysis of feature pooling in visual recognition," in *Proceedings of the 27th International Conference on Machine Learning*, 2010, pp. 111–118.

[16] A. Bordes, S. Ertekin, J. Weston, and L. Bottou, "Fast kernel classifiers with online and active learning," *The Journal of Machine Learning Research*, vol. 6, pp. 1579–1619, 2005.

[17] J. Weng, Y. Zhang, and W. S. Hwang, "Candid covariance-free incremental principal component analysis," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 25, no. 8, pp. 1034–1040, 2003.