Encoding is All You Need: Deep Dictionary Learning Framework for Classification, Using Very Small Data Sets

Alon Shavit, Tal Schiff Levanon

School of Electrical Engineering, Faculty of Engineering, Tel-Aviv University, Ramat Aviv 69978, Israel.
alonshavit@mail.tau.ac.il, talschiff@mail.tau.ac.il

We present a deep dictionary leaning neural network (DDNN) for image classification tasks with very limited data. The proposed architecture is based on encoding each input image using deep dictionary learning. We empirically analyze the effect of tuning several hyper parameters within the proposed framework. In addition, we compare DDNN with standard convolutional neural networks (CNN). We show that for some cases, DDNN achieves superior classification performance, when training is done on small datasets. Code is available at https://github.com/AlonShavit10/DDNN

***Index Terms*—Dictionary Learning, Deep Learning, Small Datasets.**

I. Introduction

Deep learning has become the industry standard when dealing with complicated computer vision tasks. The basic concept in deep neural networks is having multiple hidden layers to create the output. However, its primary limitation is the need of using a lot of training samples in order to achieve satisfying results. A typical human can easily learn to recognize a new letter which he has never seen before, based only on one or two samples. Yet, for most deep learning frameworks it is a very challenging task. Most recent deep neural network (DNN) architectures require thousands of training samples in order to learn the latent space of some category, in a way which allows the networks to provide good classification capability.

Consider, for instance, a rare animal which we want to recognize automatically based on DNN. Clearly, it would be very difficult to create a large train- test data set. One can think of many other examples where the size of the data set is the critical gap between a "good" and "bad" classification algorithm.

Furthermore, modern neural networks for classification tasks are based on convolutional layers, to create convolutional neural networks (CNN). CNN use a kernel (usually of size 3-11), which its parameters (i.e. "weights") are learnt during the training process. Modern CNN are based on a "bank" of kernels (usually known as the "channel depth"), while each kernel performs a convolution process with the previous output layer of the network.

Although CNN achieve state of the art performance on large data sets, it suffers from a major limitation. When using a "kernel bank" for the convolution process, the number of parameters in the networks increases. Hence, the generalization process with limited data might be very challenging, since there are more parameters to learn. Secondly, the floating points operation per second (FLOPS) are being increased, due to the need of performing the convolution operation for each filter. As a result, the run time of the model ("samples per seconds" which could be classified by the NN) decreases, causing a problem for real time applications.

To overcome those limitations, we propose a replacement for the convolutional layers. We show that for some use cases, a dictionary learning framework might be used *instead* of convolutional layers. Using this attitude, we show that for small data sets, the accuracy and the run time performance can be improved.

II. Related work

At the same time deep learning started gaining popularity in the late 90's, dictionary learning concept also rose to popularity. Dictionary learning, previously known as matrix factorization, goal is to learn an empirical basis from the data. It requires decomposing the data matrix to a basis/dictionary matrix and a feature matrix[1]. This method has been widely used for inverse problems in image processing, for example: denoising, image inpainting, or more complex problems such as inverse halftoning and medical image reconstruction.

One advantage of dictionary learning is the ability to replace some of the mathematical transforms such as Discrete Cosine Transform (DCT), wavelet, curvelet, and Gabor that are widely used in image classification problems. Instead, dictionary learning yields not only sparse representation (e.g., curvelet, wavelet, and DCT) but also discriminative information.

Initial techniques in discriminative dictionary learning have proposed naive approaches, which learn specific dictionaries for each class. Later, discriminative penalties are introduced in dictionary learning framework to improve the classification performance. Prior studies on dictionary learning are, generally, "shallow" learning models. Restricted Boltzmann machine (RBM) and an autoencoder (AE) are two commonly used deep learning architectures[1][2]. In dictionary learning, the cost function is usually expressed by the Euclidean distance between the data and the representation given by the learned basis.

Deep learning relays on 3 pillars: stacked autoencoder (SAE), deep belief network (DBN), and convolutional NN (CNN). Deep Belief Network (DBN) is formed by stacking multiple RBMs, one RBM after the other. Similarly, stacked autoencoder (SAE) is created by one AE followed by the other.

The proposed formulation of deep dictionary learning is learning multiple levels of dictionary in a greedy fashion, i.e. the first level learns a standard dictionary and coefficients. In following levels, the coefficients from the previous level act as inputs for dictionary learning. It is not possible to collapse multiple levels dictionaries to a single level, since that dictionary learning is a bi-linear problem. If it has been a linear problem, the structure could be collapsible. Since it is not, the shallow and the deep architectures will not be equivalent.

Since deep dictionary learning is based on the basic concept of dictionary learning, one should understand its basic concept.

A. Dictionary Learning

The popular interpretation to dictionary learning is that it learns a basis, D, for representing the input data Y, by the representation X. This idea is depicted in Fig. 1 left.

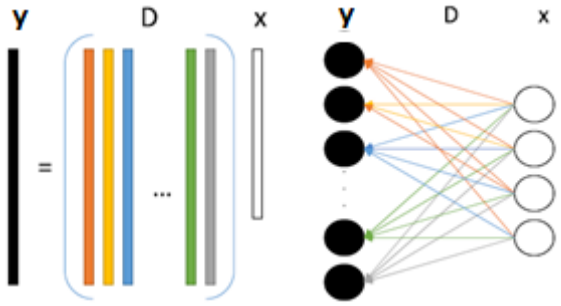

Fig. 1. Dictionary Learning

The columns of D are called "atoms". An alternative interpretation to the scheme is representing the basis (columns) as connections between the input and the representation layer, as presented in Fig. 1 right. To demonstrate the similarity, the color coding is kept intact. Dictionary learning basic for representation does not have constraints on the dictionary atoms or the loading coefficients.

Unlike a NN which is directed from the input to the representation, the dictionary learning network points are reversed—from the representation to the input. This is what is called "synthesis dictionary learning" in signal processing. The dictionary is learned, so that the features (along with the dictionary) can synthesize/generate the data. The basic formulation is

2.1.1. $Y = DX$

The dictionary and the coefficients are learned by minimizing the Euclidean cost function:

2.1.2. $\min_{D,X} \|Y - DX\|_F^2$

This formulation was introduced by Lee and Seung in the late 90's [3].

The most common way to solve problem of this kind is by K-SVD (singular value decomposition) algorithm [4]. If D is a full-rank matrix, an infinite number of solutions are available for the representation problem, hence, constraints on the solution must be set. The solution with the fewest number of non-zero coefficients is certainly an appealing representation. This sparsest representation is the solution of either:

2.1.3. $(P_0) \min_x \|x\|_0 \ s.t. y = Dx$

or

2.1.4. $(P_{0,\epsilon}) \min_x \|x\|_0 \ s.t. \|y - Dx\|_F < \epsilon$

Where $\|\cdot\|_0$ is the ι^0 norm, counting the non-zero entries of a vector. Equivalently, its corresponding unconstrained form is as follows using the Lagrange multipliers,

2.1.5. $(P_{0,\lambda}) \min_x \left[\frac{1}{2}\|y - Dx\|_F^2 + \lambda\|x\|_0\right]$

K-SVD algorithm operates in two stages: at the first stage, it learns the dictionary and in the second stage, it uses the learned dictionary to sparsely represent the data. K-SVD employs the greedy (sub-optimal) orthogonal matching pursuit (OMP) to solve the ι^0-norm minimization problem approximately. Notice, in the second stage, it performs an efficient technique to estimate the atoms one at a time using a rank one update. The major disadvantage of this method is that it is a relatively slow technique owing to its requirement of computing the SVD in every iteration. Since this solution approach does not global optimal [5], we need to use the ι^1 norm instead of ι^0 in order to force the problem to converge. Hence, the new formulation will be:

2.1.6. $(P_{1,\epsilon}) \min_x \|x\|_1 \ s.t. \|y - Dx\|_2 < \epsilon$

Where $\|\cdot\|_1$ is the ι^1 norm and the corresponding unconstrained form is as follows using the Lagrange multipliers,

2.1.7. $(P_{1,\lambda}) \min_x \left[\frac{1}{2}\|y - Dx\|_F^2 + \lambda\|x\|_1\right]$

Notice that as λ goes larger, x tends to be sparser, such that only a few dictionary elements are involved.

B. Deep Dictionary Learning

In order to understand the concept of deep dictionary learning, first a one-layer dictionary (also called "shallow" dictionary learning) is demonstrated and then move up for multiple layers. The schematic diagram for a one-layer dictionary learning is shown in Fig. 2.

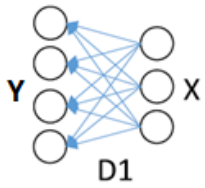

Fig. 2. Schematic diagram for dictionary learning

Dictionary learning follows a synthesis framework in equation 2.

2.2.1. $Y = D_1 X$

The basic approach into a multiple-layers dictionary learning is presented in Fig. 3 by a two-layer dictionary.

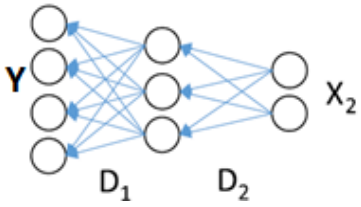

Fig. 3. Schematic diagram for deep dictionary learning

Mathematically, the representation at the second layer can be written as:

2.2.2. $Y = D_1\varphi(D_2X)$

Where φ is the activation function. Since Y can take any real value in the first layer, the activation function is absent there. The main goal of the activation functions is preventing the dictionaries to collapse into a single level dictionary. An important note: learning two-levels of dictionaries along with the coefficients is not the same as learning a single (collapsed) dictionary and its corresponding features. The main different is that single level problem (2.2.1) is a bi-linear problem and (2.2.2) is a tri-linear problem. Therefore, they are not describing the same problem. As a result, it is not possible to achieve the same features from single level dictionary learning and a collapsed two-level dictionary learning, unless the formulation is linear.

Two of the challenges of learning multiple levels of dictionaries are the following:

- Studies have proven convergence guarantees for single level dictionary learning. It would be very hard to replicate those proofs for multiple layers.
- The number of parameters required to be solved increases when multiple layers of dictionaries are learned simultaneously. With limited training data, this could lead to over-fitting.

One of the main reasons to use greedy manner in order to learning dictionaries is that it guarantees the convergence of each layer. A layer-wise learning diagram is presented in Fig. 4.

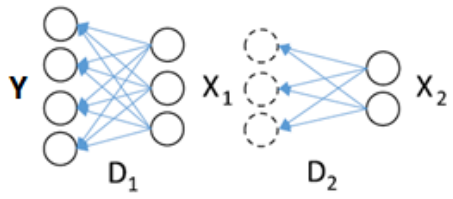

Fig. 4. Greedy layer-wise learning

Extending this idea, a multi-level dictionary learning problem with non-linear activation (φ) can be expressed as,

2.2.3. $Y = D_1\varphi\left(D_2\varphi\left(\dots\varphi(D_NX)\right)\right)$

Ideally, we would have to solve the following problem:

2.2.4. $\min_x \left[\frac{1}{2}\left\|y - D_1\varphi\left(D_2\varphi\left(\dots\varphi(D_NX)\right)\right)\right\|_F^2 + \lambda\|X\|_1\right]$

However, such a problem is highly non-convex and requires solving a huge number of parameters. A greedy approach is applied in order to address these issues, where one layer is learned at a time. Denote $X_1 = \varphi\left(D_2\varphi\left(\dots\varphi(D_NX)\right)\right)$, and substitute it in equation 2.2.3, it can be written as a one-layer dictionary (similar to equation 2.2.1), $Y = D_1X_1$, and can be solved as a single layer dictionary learning. The representation X_1 is not sparse, hence, it can be solved using alternating minimization, proven in [6]. This manner continuous until the penultimate layer. In the last layer, the coefficient X can be sparse. For learning sparse features, one needs to regularize by applying ι^1-norm on the features. It is given by:

2.2.5. $\min_x \left[\frac{1}{2}\|y - D_NX_n\|_F^2 + \lambda\|X_n\|_1\right]$

This too is solved using alternating minimization. Although equation 2.2.5 is not analytic, it can be solved by the Iterative Soft Thresholding Algorithm (ISTA) [7].

C. Least-Angle Regression (LARS)

LARS is an algorithm for fitting linear regression models to high-dimensional data, developed by Bradley Efron, Trevor Hastie, Iain Johnstone and Robert Tibshirani in 2004 [8]. It provides a means of producing an estimate of which variables to include, as well as their coefficients. Instead of giving a vector result, the LARS solution consists of a curve denoting the solution for each value of the ι^1-norm of the parameter vector. The algorithm is similar to forward stepwise regression, but instead of including

variables at each step, the estimated parameters are increased in a direction equiangular to each one's correlations with the residual.

III. METHODS

We introduce each layer of our proposed deep dictionary framework in this section. Further implementation details are given in the supplied code. To develop our solution, we took inspiration from [5]. However, there are some major modifications in our framework. In addition, we want to emphasize that the supplied code is entirely original, and that we have not used any code from the reference articles. In order to demonstrate our learning process, we assume a sample training dataset with 200 grayscale images of shape 28x28 pixels each.

Feature Extraction Layer. We first adopt a feature extraction for each image in the train dataset. We use the SIFT [9] method, which calculates a vector consists of 128 features for each anchor point within the image. The anchor point is taken each ANCHOR_POINTS_STEP pixels. Assuming anchor point each 2 pixels and a 28x28 image, for instance, we get a 196x128 matrix for each image. This process repeats itself for each image.

The final feature extraction matrix for the entire training data set is a concatenation of all the feature matrix from all the training images. Let us call it the “feature extraction matrix”. Hence, for a data set of 200 images, for instance, we get a feature extraction matrix of shape ((196x200) x 128) = (39200 x 128).

First Dictionary Learning Layer. We first learn a dictionary consists of NUM_OF_ATOMS_DICT_1 atoms. Hence, when using 500 atoms, for instance, we get a dictionary matrix of shape 500x128.

The output dictionary D is the solution for the following optimization problem (3.1):

3.1. $$\min_{x} \left[\frac{1}{2}\|y - D_1 x_1\|_2^2 + \alpha\|x_1\|_1\right]$$

Where:

- y is the feature extraction matrix from the previous stage.
- D_1 is the learnt dictionary, with $||D_k||_2 = 1$ for all $0 \leq k \leq t$, where t is the total number of atoms in the first dictionary learning layer.
- x_1 are the dictionary coefficients of the first layer.
- α is the sparsely control parameter. For larger value, x tends to become "more sparse", such that only few dictionary elements are involved in the representation of y (the feature extraction matrix).

To calculate 3.2, we use the algorithm proposed in [10], and its implementation by scikit-learn python package. We used the iterative least angle regression method (LARS [8]) to solve the lasso problem. After completing the calculation, the dictionary D enables sparse representation of the entire training dataset.

First Feature Coding Layer. After learning D_1, each image within the training data set is decomposed into a code, using the following steps:

a. For each descriptor vector within the image, find its coefficients vector c. In order to calculate this, we used the scikit-learn framework.
b. Calculate the anchor representation matrix by $rep\_matrix = c * D1$, when $*$ is the dot-product operation. Note that the dimension of the matrix is the same as the feature extraction matrix (i.e. 196x128)
c. The final coding is then given by $y^1 = sum(rep\_matrix)$, when the sum is done over the anchor dimension. Hence, the output has a shape of (1x128).

The intuition behind this logic is the assumption that the final y^1 coding vector represents the “strongest” parts of the descriptor matrix, in a way which would allow separation by some linear classifier.

Deep Dictionary Learning Layers. To enable deeper layers (i.e. layer index 2,3,4…, q), we apply the optimization problem to learn the n[th] dictionary from the n-1 dictionary. More precisely, for the n[th] dictionary we solve the following (3.2):

$$3.2.\quad \min_{x}\left[\frac{1}{2}\left\|D_{n-1}-D_{n}x_{n}\right\|_{2}^{2}+\alpha\left\|x_{n}\right\|_{1}\right]$$

Where:

- D_n is the n[th] dictionary, n=2...q, where q is the index of the deepest dictionary learning layer. In addition, $||D_{nk}||_2 = 1$ for all $0 \le k \le t_n$, where t_n is the total number of atoms in the n[th] dictionary learning layer.
- x_n are the n[th] dictionary coefficients.
- α is the sparsely control parameter.

Deep Feature Coding Layers. To encode an image using the deep dictionary layers 1…q to the new code representation y^n we propose the following logic:

a. For each deep dictionary D_n, use the very same steps as in the first feature coding layer. The only change is that this time, we use the n[th] dictionary instead of the first dictionary. The output of this stage is the vector $y^{n'}$.
b. The final output, y^n a simple concatenation of all the previous $y^{k'}$ (k = 2...n) and the vector y^1. Note that in this case, the output vector y^n has the shape (1x(128xn))

Fully Connected Layer. In the final layer, we use a simple 10x1 fully connected layer, with a softmax activation function.

The training process is done by simply converting each image in the training set to its unique code y^n by using the described steps. Then, we use a standard optimizer (i.e. Adam) to train the fully connected layer. After training the model, the evaluation is done by converting each test image into its code and then using the fully connected layer to perform the classification.

IV. EXPERIMENTS

A. Data

We used the MNIST [11] dataset, which is a standard dataset for evaluating dictionary learning and convolutional neural networks architectures. While many methods relay on learning at least hundreds or even thousands of samples [1][2][12], since working limited training data might lead to over-fitting , in our work we chose to focus on a case where the data availability is very limited. Hence, we worked with relatively very small data sets for both training and testing.

Moreover, while standard frameworks suggestion to use ~70% of the available dataset for training and ~30% of the data for evaluation, we took another path. In our setup, we use only 2%-16% of the data for training, and the rest for testing. Hence, our testing set during the experiments was 50-500 times bigger compared to the training set. This process simulates a "real life" scenario; the training data is very rare, but the system is expected to be used for a long time, without the ability to perform online learning of any kind.

To achieve the previous assumptions, we created three setups for the data:

Setup Index	Training Samples Per Class	Testing Samples Per Class	Total Training Samples	Total Testing Samples
1	2	100	20	1000
2	5	100	50	1000
3	20	100	200	1000

Table 1: Possible data setups in our experiments

B. Evaluated Architectures

To enable fair evaluation of the proposed dictionary learning framework against conventional convolutional neural networks, we created two setups of networks.

On the standard CNN, we use a grayscale 28x28x1 image as input, followed by a standard convolutional layer (with 8 or 16 channels, 3x3 filters with stride of 3) with a rectified linear unit (ReLU) activation. Then, the output is flattened and passed to a standard

fully connected layer consists of 10 units, with a softmax activation layer to perform the classification.

On the dictionary learning network, we use a **vector** as input, which its length depends on the depth chosen for the dictionary learning coding, as described on the methods session. The input is then being flattened and passed to the same fully connected layer as in the standard convolutional network setup (using the same softmax activation layer). Note that in this case, there is no non-linear activation of any kind.

In both setups, the input is normalized to the range 0-1 in order to ensure dynamic range similarity.

Fig. 5 shows the different architectures which were used during the experiments. The different values of the filters channels 'c' for the CNN are 8 or 16. For $|y^n|$ we used the values 128, 256, 384 and 512 to represent 1,2,3 or 4 deep dictionary learning layers, respectively.

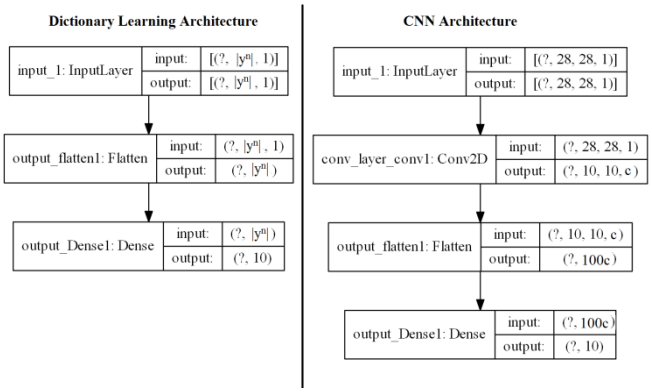

Fig. 5. Experiments setup: dictionary learning vs. CNN architecture.

C. Training Setup & Framework Hyper-Parameters

For both architectures, we used the same constant learning rate (0.01), with the Adam [13] optimizer. When learning the dictionary learning coding, we had some hyper parameters which we tuned in order to test their effect on the classification results:

a. *Alpha* – controls the sparsity of the dictionary. Larger alpha yields more sparse coefficients vector representation.

b. *Number of deep layers* – the total number *n* of dictionary layers which are used to create the representation y^n.

c. *Number of atoms for each layer* – number of atoms on each dictionary learning layer

The possible values of the parameters during our experiments are shown on Table 2:

	Alpha	**Deep Dictionary Layers**	**Atoms in Each Layer**
Possible Values	0.25 ,1,2,4,8	1,2,3,4	[250,125,60,30], [500,250,125,60], [1000,500,250,125]

Table 2: Hyper parameters values used for the experiments

D. Evaluation Methods

In order to evaluate the performance of the suggested method over the test images, we trained both architectures (CNN and our suggested dictionary learning framework) on the same training data. Then, we compared the classification accuracy on the test images. In addition, we created a confusion matrix for each architecture, over the train and the test data, separately. Furthermore, the loss and the accuracy performance of the training process, as function of the training epoch, were recorded to enable comparison between both training processes.

E. Quantitate Evaluation

In this section, we evaluate several properties of the suggested deep dictionary learning neural network (**DDNN**) architecture.

CNN vs. DDNN*:* As a primary base line, let us compare the performance of DDNN and CNN on different sizes of data sets. Table 3 shows the best results which were obtained on each dataset.

		Train Accuracy		Test Accuracy	
Total Train Samples	**Total Test Samples**	CNN	Dictionary Learning	CNN	Dictionary Learning
20	**1000**	100%(8) 100%(16)	100%	43%(8) 44%(16)	49%
50	**1000**	100%(8) 100%(16)	94%	62%(8) 65%(16)	58%
200	**1000**	100%(8) 100%(16)	84%	72%(8) 77%(16)	74%

Table 3: Comparison of train & test results between CNN and DDNN. For CNN, the number in the parenthesis stands for the number of channels in the convolution kernel

It can be seen that on the smallest dataset (only two samples for each class) the DDNN yields better results compared to CNN (49% against 44%). However, for larger datasets, CNN provides better results.

In addition, note that for the training set, the CNN always provides 100% accuracy for small datasets, while DDNN narrows the gap between the train and the test data sets. For real applications, this difference is a major advantage of the DDNN, since it enables better understanding of the expected performance even during training time.

Let up further investigate this important difference; Using 200 train samples, for instance, we can compare the MNIST confusion matrix between the DDNN and the CNN (8 channels). The results for this setting are given on Fig. 6

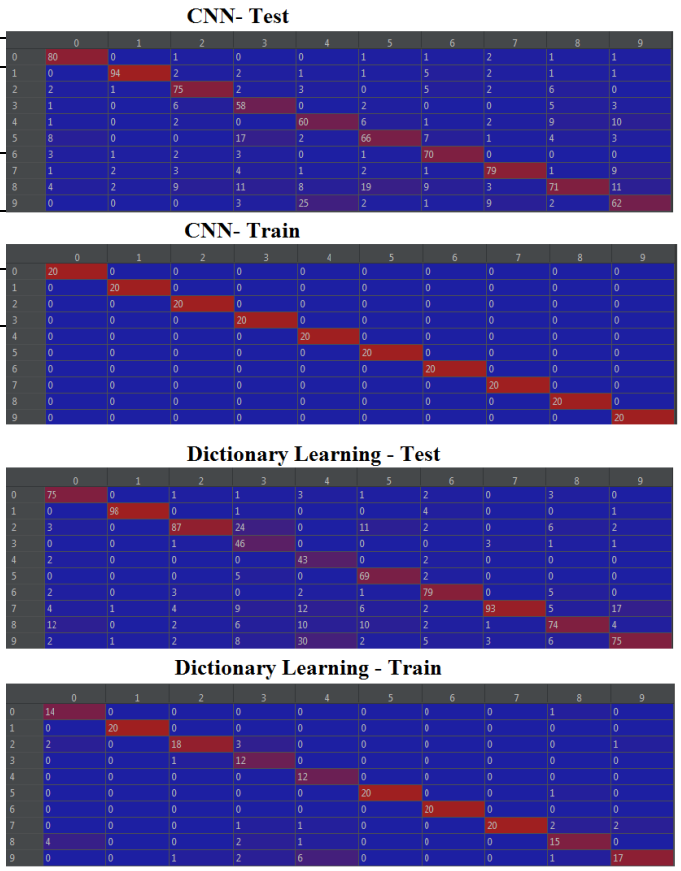

Fig. 6. Confusion matrix for CNN and deep dictionary leaning method

In CNN architecture, there is no way to estimate the failure cases when looking at the test data set, since a perfect accuracy is achieved on the train data. On the other hand, in DDNN, the failure cases of the test set can be predicted. For example, looking at number '4' on the train dataset, the most confused number for the network is '9'. This same phenomenon happens when looking at the test results as well. This "failure cases" prediction capability is an essential component in many real-life prediction systems.

Another interesting property is the partial independency between the CNN and the DDNN results. Take a look at the predictions of number '3', for instance. While the most confusing number for CNN is '5', the DDNN architecture is most confused by '2'. This observation can further be used in order to design a combined classification system, which wisely fuses the results from both frameworks. We believe that this "fused" result might provide better

performance compared to the usage of each individual network (DDNN/CNN) separately.

Number of deep dictionary layers: Let us demonstrate the advantage of using our suggested DDNN; In order to evaluate the contribution of using deep dictionary learning layers, we compare the test accuracy over 200 training samples data set (and 1000 samples for testing). Fig. 7 shows the results of this experiment as a function of the epoch. Note that for this example and for the further similar graphs, the only layer which is being trained on each epoch, according to our DDNN framework, is the fully connected layer.

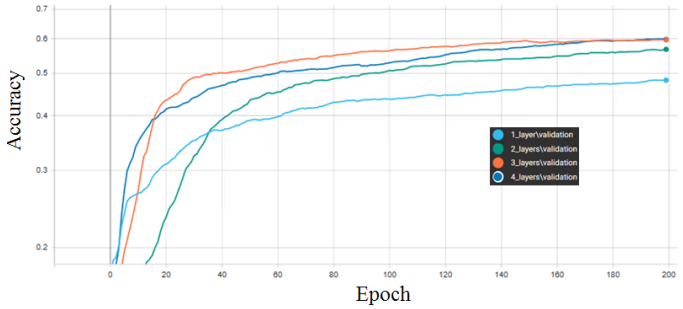

Fig. 7. Test accuracy as a function of training epoch, for different number of layers in the deep dictionary learning framework

From this graph, the advantage of our DDNN can be easily seen; The network with only 1 layer has yielded the lowest accuracy score. Next, the network with 2 layers achieved ~15% more accuracy performance. Next, the 3- and 4-layers networks seem to supply nearly the same (and highest) accuracy results. The 4-layer configuration has still the advantage of higher "accuracy to epoch" (which is the graph slop) over the first 20 epochs. This characteristic can also be when looking at the loss vs. epoch graph, as shown on Fig. 8.

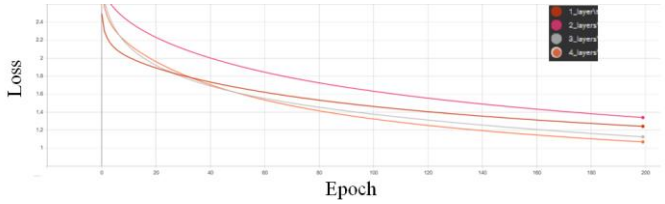

Fig. 8. Train loss as a function of training epoch, for different number of layers in the deep dictionary learning framework

Number of atoms in each dictionary: We investigated the effect of changing the number of the dictionary atoms within each layer. As a reminder, on our DDNN we reduce the number of atoms by a factor of 0.5 on each deeper layer.

To study the effect of this property, we created a 4 layers DDNN architecture. Then, we used three different dictionaries collection for the 4 layers. The training process used 20 samples as train set and 1000 samples as the test dataset.

The results of this experiment, for both train and test datasets, are shown on Fig. 9. It can be seen that the best results were obtained when using the med-value of 500 atoms for both train and test scenarios; on this configuration, the network has the fastest convergence on the train data set, while providing the best accuracy results on the test data set.

This last observation might be seemed a bit weird - why haven't we gotten the best results for the configuration with the highest number of atoms? However, this behavior can be explained. When using atoms, the generalization capability might be harmed; In other words, higher dictionary dimension does not improve the ability to classify correctly since it does not add useful information which could be used on the fully connected training process. Even more generally - let us think of infinite size dictionary. In this case, it does not code the descriptor values at all, since it can just store them. This lack of coding capability when increasing the dictionary size is a basic and important characteristic of the DDNN framework. Hence, it should be taken into consideration when tuning the hyper parameters for DDNN based classification framework.

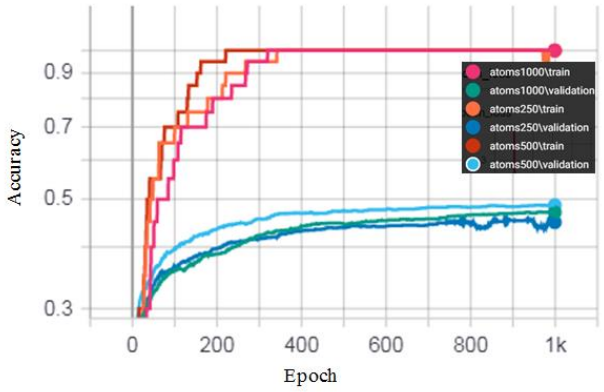

Fig. 9. Accuracy for different number of atoms in the first dictionary layer, for a framework with 4 layers. The atoms in each layer is half relative to the previous layer

Sparsity parameter: We investigated the effect of changing the parameter 'alpha' when creating the dictionary for each layer within the DDNN framework. We used 1000 atoms for the first layer and, as usual, reduced this number by a factor of 0.5 for each of the next 3 layers. For training each dictionary, we used the LARS method for a period of 3000 iterations. Fig. 10 shows the test accuracy (over 1000 images database), as a function of the epoch, for 5 different values of alpha. For this experiment, we used 20 images as training set.

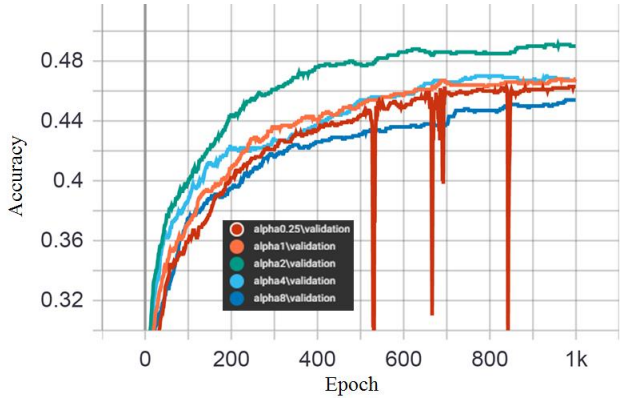

Fig. 10. Test Accuracy different values of alpha, as a function of training epoch

The results are very interesting; As we explained in previous sections, bigger alpha stands for more "sparse" solution. But what is the "optimum sparsity"? Should we use "low- sparsity" or "high sparsity" dictionaries? Fig. 11 demonstrates the answer to those questions, based on the final state of our experiment.

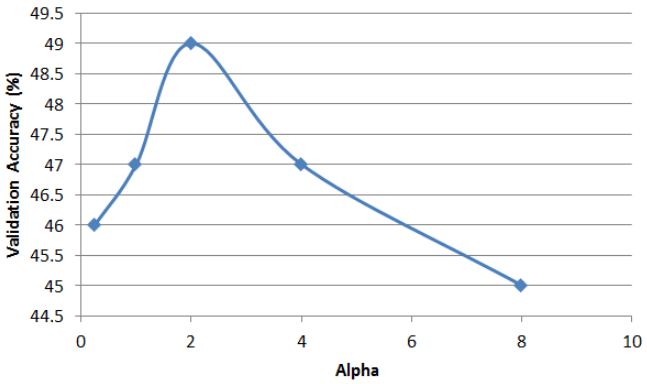

Fig. 11. Test Accuracy as a function of alpha

It can be seen that the best performance is obtained when using an "intermediate" value for alpha, which generates "intermediate sparsity".

The intuition for this observation can be given by thinking on the edge cases of this problem for a dictionary consists of infinite number of atoms; for "zero spaced" dictionary, no coding capability is created on this dictionary, since all the input data can be coded using the infinite number of atoms. Hence, the "latent space" is exactly the same as the input. On the other hand, for the "very spaced" configuration, saying only 1 coefficient is not zero, only one feature is created. Hence, any linear classifier would struggle to linearly separate classes from each other.

F. Resources Evaluation (Flops density)

In order to estimate the capability of using a machine learning model in real time scenario, a common benchmark is analyzing the floating-point operations per second, usually referred as "FLOPS". In many cases, the more FLOPS a model has, the more time it takes to process an input to create the required output. Hence, a model with a small number of FLOPS is preferred for real time applications. Fig. 12 compares the number of flops for 6 different architectures: 2 CNN (with 8 or 16 channels) and 4 DDNN (with 1-4 deep dictionary layers).

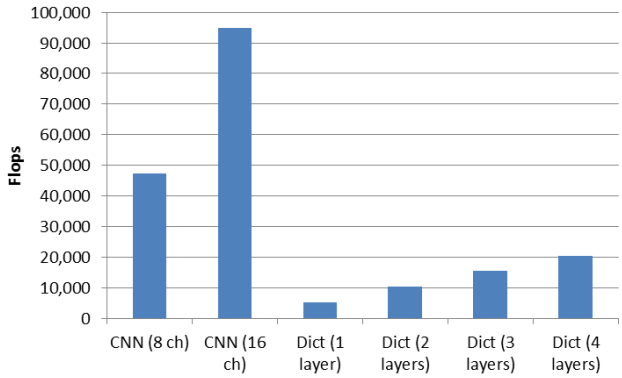

Fig. 12. Flops (Floating Point Operation per Second) for CNN and dictionary learning architectures

It can be seen that DDNN uses much less FLOPS comparing to CNN, due to the absence of the convolutional layer. In addition, Table 4 shows that ratio between the output accuracy to the number of FLOPS in DDNN (which we call "accuracy density") is about 2-4 times better in DDNN compared to CNN. However, note that in our analysis we count only the number of FLOPS which are involves on the network itself, while calculations for coding process of an input into its *y*' representation were ignored. Further research should analyze this property to provide full understanding of its influence.

Architecture	Number of Flops	Average Test Accuracy	Accuracy Density (1000*Accuracy/ Flops)	Relative accuracy density
CNN (8 ch.)	47,422	59%	1.24	0.43
CNN (16 ch.)	94,766	62%	0.65	0.22
DDNN (4 layers)	20,557	60%	2.91	1.0

Table 4: Comparison of resources between CNN and the proposed dictionary learning framework

V. CONCLUSION

We have addressed the problem of image classification, when only very few samples are given for the training process. We reviewed several articles which deal with deep dictionary learning problem, showing that despite its high relevance, this problem has not been widely addressed.

In this work, we proposed a deep dictionary learning framework, which can be easily integrated into any neural network architecture. The framework has been quantitatively evaluated on the publicly available MNIST data set.

We analyzed the influence of several hyper parameters on the classification performance and demonstrated the advantage of using several deep dictionary layers.

Furthermore, our deep dictionary learning neural network (DDNN) architecture has been compared against conventional convolutional neural network (CNN). We showed that for very small datasets, DDNN is able to achieve equal or superior performance, making it a considerable choice when dealing with data availability problem. In addition, we analyzed the resources consumption of both solutions, showing that DDNN might have an advantage over CNN in computational complexity for real time applications.

Further work includes several possible extensions. First, the proposed method should be tested on other data sets to fully evaluate its performance. In addition, several dictionary learning algorithms can be compared against each other, to find the one which best fits the proposed architecture. Finally, the effect of replacing the final fully connected layer with another linear classifier (i.e. SVM) can be analyzed.

References

[1] S. Tariyal, A. Majumdar, R. Singh, and M. Vatsa, "Deep Dictionary Learning," *IEEE Access*, vol. 4, no. Dl, pp. 10096–10109, 2016, doi: 10.1109/ACCESS.2016.2611583.

[2] and A. M. Vanika Singhal, Hemant K. Aggarwal, Snigdha Tariyal, "Discriminative Robust Deep Dictionary Learning for Hyperspectral Image Classification," *IEEE Trans. Geosci. Remote Sens.*, vol. 55, no. 09, pp. 5274–5283, 2017, doi: 10.1109/JSTARS.2018.2877769.

[3] D. D. Lee and H. S. Seung, "Learning the Parts of Objects by Non-Negative Matrix Factorization," *Nature*, vol. 401, no. 6755, pp. 788–791, 1999, doi: 10.1038/44565.

[4] M. Aharon, M. Elad, and A. Bruckstein, "K-SVD: An algorithm for designing overcomplete dictionaries for sparse representation," *IEEE Trans. Signal Process.*, vol. 54, no. 11, pp. 4311–4322, 2006, doi: 10.1109/TSP.2006.881199.

[5] H. Tang, H. Liu, W. Xiao, and N. Sebe, "When Dictionary Learning Meets Deep Learning: Deep Dictionary Learning and Coding Network for Image Recognition with Limited Data," *IEEE Trans. Neural Networks Learn. Syst.*, pp. 1–13, 2020, doi: 10.1109/tnnls.2020.2997289.

[6] A. Makhzani and B. Frey, "k-Sparse autoencoders," *2nd Int. Conf. Learn. Represent. ICLR 2014 - Conf. Track Proc.*, 2014.

[7] I. Daubechies, M. Defrise, and C. De Mol, "An iterative thresholding algorithm for linear inverse problems with a sparsity constraint," *Commun. Pure Appl. Math.*, vol. 57, no. 11, pp. 1413–1457, 2004, doi: 10.1002/cpa.20042.

[8] B. EFRON, T. HASTIE, I. JOHNSTONE, and A. R. TIBSHIRANI, "Least Angle Regression," *Ann. Stat.*, vol. 32, no. 2, pp. 407–499, 2004, doi: 10.1109/glocom.1988.25879.

[9] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vis.*, vol. 60, no. 2, pp. 91–110, 2004, doi: 10.1023/B:VISI.0000029664.99615.94.

[10] J. Mairal, F. Bach, J. Ponce, and G. Sapiro, "Online dictionary learning for sparse coding," *Proc. 26th Int. Conf. Mach. Learn. ICML 2009*, pp. 689–696, 2009.

[11] L. Deng, "The MNIST Database of Handwritten Digit Images for Machine Learning Research [best of The Web]," *IEEE Signal Process. Mag.*, no. November, pp. 141–142, 2012.

[12] V. Singhal and A. Majumdar, "Noisy deep dictionary learning," *ACM Int. Conf. Proceeding Ser.*, vol. Part F1302, no. 1, 2017, doi: 10.1145/3041823.3041826.

[13] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," *3rd Int. Conf. Learn. Represent. ICLR 2015 - Conf. Track Proc.*, pp. 1–15, 2015.