2012 11th International Conference on Machine Learning and Applications

Deep Structure Learning: Beyond Connectionist Approaches

Ben Mitchell
Department of Computer Science
Johns Hopkins University
Baltimore, MD 21218
Email: ben@cs.jhu.edu

John Sheppard
Department of Computer Science
Montana State University
Bozeman, MT 59717
Email: john.sheppard@cs.montana.edu

Abstract—**Deep structure learning is a promising new area of work in the field of machine learning. Previous work in this area has shown impressive performance, but all of it has used connectionist models. We hope to demonstrate that the utility of deep architectures is not restricted to connectionist models. Our approach is to use simple, non-connectionist dimensionality reduction techniques in conjunction with a deep architecture to examine more precisely the impact of the deep architecture itself. To do this, we use standard PCA as a baseline and compare it with a deep architecture using PCA. We perform several image classification experiments using the features generated by the two techniques, and we conclude that the deep architecture leads to improved classification performance, supporting the deep structure hypothesis.**

I. Introduction

Finding structure in data is one of the most fundamental tasks in many areas of Artificial Intelligence research, including machine learning, statistical pattern recognition, computer vision, data mining, and natural language processing. The name "statistical pattern recognition" makes this connection especially clear, since all structure can be viewed as statistical patterns. Techniques like Bayesian networks try to model these statistical patterns explicitly, but even techniques like connectionist networks can be thought of as implicitly capturing the statistical properties of patterns in data.

The No Free Lunch theorems [17] state that all techniques have the same expected value over all data. This means no technique can be best for all data, making the search for algorithms that consistently perform well seem to be futile. Fortunately, there is significant evidence that real-world data has properties that allow a small class of techniques to perform well on a wide range of problems: humans are able to function in the world, and can make sense of the complex, noisy, and ambiguous input signals the world provides. There is also evidence that the human brain is able to learn to interpret this data using a relatively constrained set of algorithms, and a fairly limited amount of pre-determined structure [15], [11].

The question, then, is what are the properties of real-world data that can be relied upon and exploited to make apparently insoluble learning problems tractable? One hypothesis that has been gaining interest over the last few years is the deep structure hypothesis. This hypothesis essentially states that data of interest has structure at multiple levels of resolution. The applications so far have mostly been to image data, with multiple levels of spatial resolution providing "deep" features [5]. This work has shown great promise for solving "hard" computer vision problems like generalized object recognition.

In *The Need for Biases in Learning Generalizations,* Mitchell states that "progress toward understanding learning mechanisms depends on understanding the sources of, and justification for, various biases." [14] He points out that unbiased learning is useless, and therefore we must consider the bias of a technique to understand when and how it is useful. If deep learning is effective on real-world problems, we must conclude that it has a useful and important bias built in.

Unfortunately, most existing work does not directly explore the deep structure hypothesis, or any other source of bias in deep learning. The learning algorithms are highly complex, and produce results that are difficult to analyze. This is largely due to their connectionist approach; while connectionist techniques often produce good results, they have long been criticized for failing to provide interpretability and separability. This can be thought of as a form of the credit assignment problem: we would like ro analyze why a given architecture produces good results, and tease apart what impact various features of that architecture have on performance.

Our goal in this work is to attempt to separate out some of the properties that existing deep architectures have, and apply them in a simplified, non-connectionist framework. This will allow us to directly explore and test the hypothesized biases that underlie deep structure learning, and see what is required to take advantage of deep structure. While there is some other work in the area of analyzing deep learning [8], to our knowledge no one has attempted to extend deep learning beyond the realm of network-like architectures.

II. Deep Structure Learning

There have been several attempts to exploit deep structure going back many years, including Fukushima's Neocognitron [9], LeCun's Convolutional Networks [13], Behnke's Neural Abstraction Pyramids [4], Hawkins' Hierarchical Temporal Memories [10], Hinton's Deep Belief Networks [12], and Bengio's Stacked Auto-encoders [6]. All of these techniques take a connectionist approach to deep structure learning.

978-0-7695-4913-2/12 $26.00 © 2012 IEEE
DOI 10.1109/ICMLA.2012.34

Additionally, they all share the same neuromorphic approach, basing their functionality on the human visual cortex, and they have all been targeted at computer vision problems. In this paper, we will focus on a non-neuromorphic approach, but retain image classification as our benchmark task to allow more direct comparison with previous work.

Until recently, work in deep learning did not receive much attention. In fact, the term "deep learning" did not come into common usage until around 2008. The recent growing interest has been sparked largely by the success of two techniques, Convolutional Networks and Deep Belief Networks.

Convolutional Networks are a deep learning technique introduced by LeCun [13] in an attempt to create a computer vision system that replicates the behavior of the human visual cortex. A Convolutional Network functions by convolving a bank of filters with an image, and then aggregating over local areas to reduce the size of the output. These two steps are alternated until the size of the final output is as small as is desired. For a more detailed explanation of Convolutional Networks, we refer the reader to [13] and [7].

Deep Belief Networks (DBNs) were introduced by Hinton [12] as an alternative to the gradient-based learning done by LeCun. DBNs work by building a hierarchy of Restricted Boltzmann Machines (RBMs), trained using contrastive divergence and then converted into a multilayer neural network. DBN training is conceptually similar to the training of stacked auto-encoders, but does not use standard gradient descent alone to learn the weights of the neural network. For a more detailed description of Deep Belief Networks, see [12] and [7].

III. Deep Feature Extraction

While the results of existing deep learning algorithms are impressive, the complexity of the resulting systems makes it difficult to say which properties of those systems are responsible for the improved performance. Therefore, we have set out to create a simpler form of deep learning algorithm that will allow us to test hypotheses about the existence of deep structure and the utility of various techniques for handling it.

The basic framework we propose for doing this is one of Deep Feature Extraction (DFE). Deep Feature Extraction produces a hierarchy of features representing some data, in which the higher levels correspond to shorter overall description length of that data. For instance, a hierarchy might have a bottom level that was a 4096-dimensional raw image, and a top level that was a 10-dimensional feature vector. Intermediate levels would have intermediate resolutions. The important property of the hierarchy is that each level is created by performing some type of dimensionality reduction or feature extraction on the level below (excepting the bottom level, which is the raw input data). This is unsupervised greedy layer-wise training, because each layer is trained based only on its inputs; the utility of its outputs are not factored in to the training.

This is a fairly broad framework; in fact, traditional Convolutional Networks and DBNs could be described within this framework. In the broadest description, the DFE framework is agnostic to how the dimensionality reduction is performed; the filter bank/softmax approach of a Convolutional Network is a valid, if fairly complex, example. At the far end of the spectrum, the "featurespace projection" could simply be downsampling, in which case the DFE would mimic a standard image pyramid [1]. In this paper, we use a simple but non-trivial dimensionality reduction technique to explore more closely the impact of the hierarchical architecture.

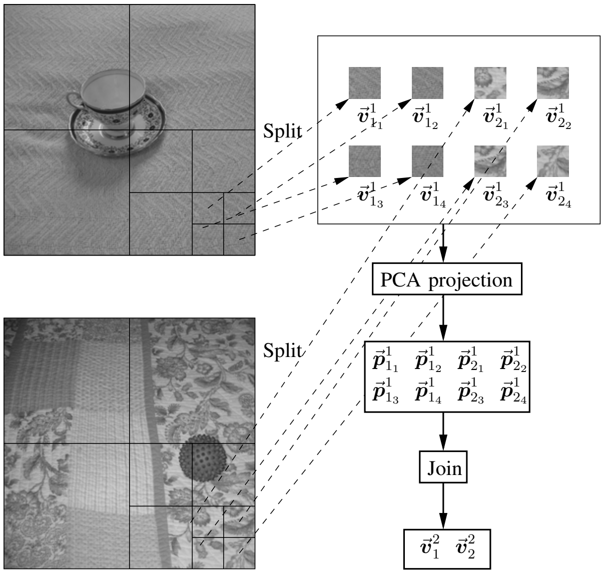
A. Deep PCA

As our basic method of feature extraction, we chose Principal Component Analysis (PCA). We made this choice for several reasons. Firstly, PCA is simple and well understood. PCA is also advantageous because it is deterministic, generates linear encode/decode functions, and is guaranteed to produce an optimal encoding (with respect to minimizing reconstruction error for a length k linear encoding [2]). PCA has no parameters (other than the choice of how many principal components to keep), avoiding a large set of experimental design issues. Finally, PCA has been used successfully for dimensionality reduction in the field of computer vision for many years [16], so there is broad familiarity with the technique. For this paper, we restrict our interest to image data, though there is no reason the technique could not be applied to other types of data, and we hope to do so in future work.

To create a DFE hierarchy for a set of images using PCA, we begin by subdividing the images. In our experiments, we used a quad-tree decomposition to split each image recursively; the bottom level of the quad-tree was a set of non-overlapping 4×4 pixel patches. The set of all 4×4 patches from all images in the training dataset was then used as the input for PCA, and the top k eigenvectors were used as a reduced-dimensionality basis. Each patch was projected into this new basis, and the reduced-dimensionality patches were then joined back together in their original order. The result of this was that each image had its dimensionality multiplied by $^{k}/_{16}$ (Figure 1). These reduced dimensionality images formed the next layer up in the hierarchy after the raw data.

This process was repeated, using the newly created layer as the data for the split-PCA-join process to create the next layer up. At every layer after the first, the dimensionality was reduced by a fixed factor of 4. The process was terminated when the remaining data was too small to split, and the entire image was represented by a single k-dimensional feature vector at the "top" of the hierarchy.

As an illustration, if our original data is a set of m vectors, each length n, then we start with a raw data matrix D_1, which is $m \times n$. In the case of images, this means each row of the matrix is an image. The level one split data matrix, S_1, in our hierarchy is generated by recursively splitting the vectors in D_1 down to $4 \times 4 = 16$ dimensional patches, so it will be a $(m \cdot \frac{n}{16}) \times 16$ matrix. We apply PCA to S_1, extract the top k eigenvectors into F_1, and use F_1 as a basis to project the vectors of S_1 into. Applying the projection to the vectors in S_1 results in P_1, an $(m \cdot \frac{n}{16}) \times k$ matrix. Adjacent vectors in P_1 are then joined (using the inverse of the splitting operator),



Split
$\vec{v}^1_{1_1}$ $\vec{v}^1_{1_2}$ $\vec{v}^1_{2_1}$ $\vec{v}^1_{2_2}$
$\vec{v}^1_{1_3}$ $\vec{v}^1_{1_4}$ $\vec{v}^1_{2_3}$ $\vec{v}^1_{2_4}$
PCA projection
Split
$\vec{p}^1_{1_1}$ $\vec{p}^1_{1_2}$ $\vec{p}^1_{2_1}$ $\vec{p}^1_{2_2}$
$\vec{p}^1_{1_3}$ $\vec{p}^1_{1_4}$ $\vec{p}^1_{2_3}$ $\vec{p}^1_{2_4}$
Join
$\vec{v}^2_1$ $\vec{v}^2_2$

Fig. 1: An example of how Deep PCA works on a pair of images from our dataset. The $\vec{\boldsymbol{v}}^1$ are vectors in S^1, the $\vec{\boldsymbol{v}}^2$ are vectors in S^2, and the $\vec{\boldsymbol{p}}^1$ are vectors in P^1. Note only a small number of the vectors from each of these levels are shown.

resulting in S_2, which is $(m \cdot \frac{n}{16 \cdot 4}) \times (4 \cdot k)$. If we continue to recursively join the S^2 data, we will get D_2, which is an $m \times (n \cdot \frac{k}{16})$ matrix. Alternatively, we can simply apply PCA directly to S_2 and avoid some extra split/join operations when building the hierarchy. In general, D_l will be $m \times (n \cdot \frac{k}{16 \cdot 4^l})$. When $16 \cdot 4^l = n$, the hierarchy is complete, giving a top layer with dimensions of $m \times k$. We note that the math only works cleanly for raw data vectors whose length is a power of four; this means we need square images with power of two widths. While there are several ways this constraint could be relaxed (by changing the split/join operation), we leave them for future work.

Pseudocode for this algorithm is given in Algorithm 1. The $SplitQuads$ function (line 3) does a quad-tree style split of an image into four equal-sized sub-images, and the $JoinQuads$ function (line 8) inverts this operation. The function $PCA(M, k)$ (line 6) computes an eigen-decomposition of the covariance of M and then returns the top k eigenvectors. The matrix multiplication in line 7 projects the data into the new eigen-basis. The **for** loop in lines 2–4 does the "recursive" splitting of data, and the **for** loop on lines 5–9 builds the feature hierarchy one level at a time.

Since our goal was to examine the benefits of performing dimensionality reduction in this hierarchical fashion, we used only the top layer as the "output" of the overall "Deep PCA" technique. This allowed us to compare the k-dimensional feature vector produced by Deep PCA directly to the k-dimensional feature vector produced simply by applying PCA directly to the raw image data and projecting into the top-k eigen-basis (which we will call "Flat PCA" for the sake of clarity). We will leave the exploration of using lower layers of the hierarchy as outputs for future work.

Algorithm 1 Deep PCA

```
Require: data matrix $Data$, depth of hierarchy $m$, number of
    eigenvectors to keep $k$
Ensure: featurespace hierarchy $F$, projected data hierarchy $D$
 1: $D_1 = Data$
 2: for $i = 1 \rightarrow m$ do
 3:     $D_1 = SplitQuads(D_1)$
 4: end for
 5: for $i = 1 \rightarrow m$ do
 6:     $F_i = PCA(D_i,\ k)$
 7:     $P = F_i D_i$
 8:     $D_{i+1} = JoinQuads(P)$
 9: end for
10: return $F$, $D$
```

The Deep PCA model has less theoretical power than most previous deep learning models, due to the fact that it is primarily a linear model. In fact, the only non-linearity occurs in the vector split/join operation; there is no other inter-layer processing. This is in contrast to the complex non-linear functions used in other deep models, which generally include things like inter-layer adaptive contrast normalization, soft-max functions, and sigmoid or hyper-tangent input aggregation. Additionally, some deep methods do not restrict themselves to feed-forward operations [4], making the overall behavior of the system that much more complicated.

The inherent power of the repeated non-linear aggregation used in standard deep learning techniques makes it difficult to tell how much of the performance of those techniques is due to the feature hierarchy, and how much is due to the stacked non-linear functions. It was for this reason that we designed Deep PCA to have as little non-linearity as possible. While this likely handicaps its performance in comparison to something like a Convolutional Network, it allows us to examine the effects of the hierarchy much more cleanly than would be possible using non-linear aggregation (or using a non-linear feature extractor at each layer instead of PCA, which would also introduce significant non-linearity into the result).

IV. Experiments

One of the most basic deep-structure hypotheses is that real-world data contains deep structure, and exploiting this structure will yield improved performance on machine learning tasks. To test this hypothesis, we designed experiments to compare the performance of a standard (shallow) feature extractor directly with a deep feature hierarchy using the same extractor. As described, our goal in choosing PCA was to have a well-understood feature extractor that could be used to expose the differences between deep and flat feature extraction; we have no expectation that it will produce optimal classification results. For our purposes here, the differences between deep and flat are more important than the absolute performances. In an application where performance is the primary goal, the best feature extractor available should be used.

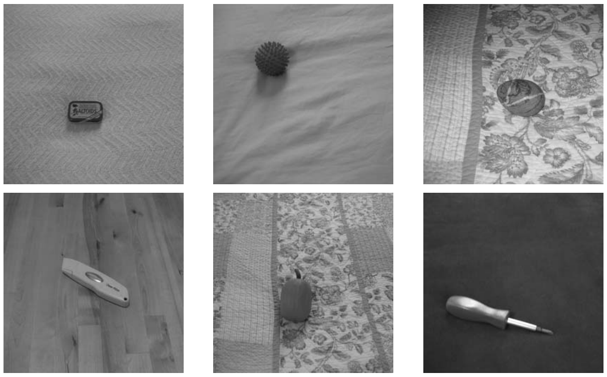


Fig. 2: Some example images from our data set.

We chose an image classification task because this is the type of task that has been used in most of the deep structure literature. Since previous work has shown improved performance on these tasks using deep architectures, we expect that this type of data should have deep structure that can be exploited.

We began with a dataset consisting of 600 greyscale images. The images were pictures of 10 different objects taken against 5 different backgrounds. Multiple images were taken of each object/background pair, and the camera was moved slightly between images so that no two were alike in the position and scale of the foreground object (see Figure 2 for example images). We performed our experiments on three different versions of the dataset, each of which was a different resolution. The resolutions used were 512×512, 256×256, and 128×128 pixels. We created this novel data set so that we could have natural images (i.e. not artificially generated or composited) in multiple resolutions, with multiple images of each object. In the future, we hope to find other data sets that we can use to test our algorithms, but many existing image data sets have only low (and often variable) resolutions, which make deeper hierarchies less interesting.

For each image width, we did experiments using both 5×2 cross-validation, and 10-fold cross-validation. Five-by-two has some nice theoretical properties, but due to the relatively small size of our dataset, the accuracy achievable by 10-fold cross-validation was higher. We report the results for both methods.

For each experiment, a dataset was split into "train" and "test" sets using one of the validation methods, and the training set was used to generate two feature spaces. The first used Flat PCA to generate 16 features, and the second used Deep PCA to generate 16 features. The dimensionality of the resultant feature space was the same for both techniques. Due to the length of our data vectors, operations on the full covariance matrix proved intractable, so we used an iterative PCA algorithm [3] to generate only the first 16 eigenvectors.

The training data was then projected into both feature spaces, and the projected training data was used to train two standard classifiers. Once the classifiers were trained, the testing data was projected into each feature space and presented to the corresponding classifier to evaluate its performance.

TABLE I: Classification accuracy for different experiments. Each score is averaged over the samples created by the indicated validation technique. Bold numbers indicate that the advantage a technique showed was significant ($p \geq .95$ using two-sided paired Wilcoxon).

Width	Validation	Classifier	Flat	Deep
128	10-fold	KNN	52.56%	53.20%
128	10-fold	SVM	45.40%	**48.09**%
128	5x2	KNN	43.84%	**44.93**%
128	5x2	SVM	37.77%	**39.63**%
256	10-fold	KNN	51.26%	52.08%
256	10-fold	SVM	45.04%	46.71%
256	5x2	KNN	42.33%	**43.54**%
256	5x2	SVM	36.28%	37.83%
512	10-fold	KNN	50.83%	52.60%
512	10-fold	SVM	43.59%	46.57%
512	5x2	KNN	43.87%	**45.03**%
512	5x2	SVM	36.61%	**38.47**%

TABLE II: Percentage of validation runs in which one technique outperformed the other. In cases where performance was the same, no winner is listed. The margin is the amount that the winning technique won by, averaged over the instances in which that technique won.

Width	Validation	Classifier	Deep:Flat Wins	Margin
128	10-fold	KNN	50%:20%	2.94%:4.08%
128	10-fold	SVM	90%:10%	3.14%:1.66%
128	5x2	KNN	60%:20%	1.93%:0.33%
128	5x2	SVM	80%:10%	2.45%:1.00%
256	10-fold	KNN	50%:30%	3.24%:2.70%
256	10-fold	SVM	60%:40%	6.03%:4.93%
256	5x2	KNN	70%:20%	1.87%:0.49%
256	5x2	SVM	70%:30%	2.82%:1.43%
512	10-fold	KNN	50%:30%	4.90%:2.18%
512	10-fold	SVM	50%:30%	8.17%:3.81%
512	5x2	KNN	80%:20%	1.57%:0.50%
512	5x2	SVM	80%:20%	2.53%:0.83%
Average			65.83%:23.33%	3.47%:2.00%

We used two classifiers, a simple Nearest Neighbor classifier and a Support Vector Machine. As with the choice of feature extractor, we chose simple, widely-used, deterministic classification algorithms. While we performed a few experiments to make sure we had reasonable parameters for the SVM (i.e. kernel type, degree, etc.), we make no claim that these classifiers will yield the highest possible performance on the task. Again, the goal was to use simple algorithms to make the difference between the deep and flat feature extraction as clear as possible.

Finally, we performed experiments on a modified version of the dataset, in which a random permutation was applied to the feature vectors. This meant that the pixels of an image were re-ordered randomly (but consistently across all the images in the data set). This permutation effectively erases any local structure in the data, while preserving global statistical properties. This was done to test whether the deep architecture was truly making use of local structure or not.

V. Results and Discussion

We ran both 10-fold and 5×2 cross-validation in combination with each image size and classifier. The results of

TABLE III: Mean squared reconstruction error for the different techniques and image sizes. The results are averaged over all data sets with the indicated resolution.

Width	Flat MSE	Deep MSE	Flat Accuracy	Deep Accuracy
128	6.53	6.96	44.89%	46.47%
256	14.08	13.90	43.73%	45.04%
512	29.92	29.51	43.73%	45.67%

TABLE IV: Classification error on randomly permuted images. Results reported using 5x2 validation and the nearest neighbor classifier; results for other methods were similar.

Width	Flat Original	Flat Permuted	Deep Original	Deep Permuted
128	43.84%	44.07%	44.93%	35.89%
256	42.33%	41.02%	43.54%	31.11%
512	43.87%	43.25%	45.03%	28.59%

these experiments are summarized in Table I by giving the mean accuracy achieved by each group of 10 experiments (one per validation fold). As can be seen in this table, Deep PCA achieves a higher mean accuracy than Flat PCA in all cases; the overall mean improvement achieved by Deep PCA is 1.16%. While this difference is small, it is highly significant; a two-sided paired Wilcoxon test yields a p-value of $p = 1.86 \times 10^{-7}$ for the null hypothesis that the two methods produce equivalent results.

The absolute values of the accuracies are low in all cases, though well above random chance for a 10 class problem. This seems to be due largely to the difficulty of the problem; as a baseline, we performed experiments using standard Convolutional Networks, and were unable to obtain accuracy above 48%. It is possible that with more parameter tuning we could do slightly better, but the difference is minimal. Our technique offers competative performance despite using a simple, non-connectionist architecture. Additionally, it is far more computationally efficient; the Convolutional Networks took several orders of magnitude longer to train.

Additionally, while object recognition is known to be a hard problem, it is likely that we could achieve better results using something other than PCA to do feature extraction, since PCA tends to work best for vision problems after lots of pre-processing (e.g. see the original Eigenfaces work [16]). Most work with Convolutional Networks also does more preprocessing than we used here; in particular, local contrastive normalization is standard, and would likely improve the performance of either technique. As previously stated, we wanted a simple, general algorithm for our feature extractor, with as little preprocessing as possible. Our goal was to examine the role of deep structure in learning, not create a state-of-the-art classifier system.

Looking at the p-values for each experiment individually (highlights in the last column of Table I), we see that the 5×2 cross-validation gives much better significance. In fact, in all but one of the 5×2 experiments, Deep PCA was better by a statistically significant margin ($p \geq 0.95$). In the one instance where it failed to meet this significance, it was only off by about 2% ($p = 0.93$). The 10-fold cross-validation runs, on the other hand, had poor significance results, but higher absolute performance scores. This result should not be surprising; the 10-fold method has more training data, so it can achieve better performance, but much less testing data, which handicaps its ability to produce a wide and consistent margin.

While Deep PCA is better on average for every cross-validation run, it was not always better for every single training fold. Table II shows how often each algorithm beat the other during each 10-experiment validation run, as well as the margin of that victory. In cases where the two algorithms tied, neither was counted as winning. We note that Deep PCA not only wins more frequently, but when it wins it does so by a larger margin.

Due the relatively small data sets, we saw a large variance between different test/train splits; there could be as much as a 15% difference in accuracy (the same for both techniques) between the different splits of a single 10-fold cross-validation run. This behavior suggests that the number of training samples was a limiting factor in the final performance of the classifiers. Additionally, the average accuracy achieved during the 10-fold cross-validation experiments was around 10% higher than that achieved during the 5×2 cross-validation experiments, which lends support to this hypothesis. The difference between the classification accuracy achieved by the "flat" and "deep" methods was rarely more than a few percent, but it also showed a very small variance, proving to be quite stable across all the different experiments. Thus, we expect that a larger data set would show improved accuracies for both flat and deep methods, but we do not expect the difference between flat and deep would be impacted greatly.

Table III shows the average mean-squared reconstruction error (MSE) that results from projecting into and then out of each feature space, along with the average accuracy for each method. These results do not show any significant difference in MSE between the flat and deep techniques. As expected, increasing the length of the raw data vectors while leaving the dimensionality of the generated feature space fixed leads to higher MSE. Higher classification accuracy without higher MSE suggests that the deep technique is doing a better job of keeping "meaningful" features.

Table IV shows the results of randomly permuting the data before applying the learning process. The fact that permutation makes no significant difference for Flat PCA is exactly what we would expect; since PCA works by looking an global statistical properties, the order in which the features appear makes no difference in the projected data. In the case of the deep technique, however, there is a significant difference; performance on the permuted dataset is far worse than on the unmodified images. This suggests that two of our original hypotheses hold.

First, it suggests that one of the major biases of the deep technique is an expectation of local structure. And second, it suggests that our images have local structure that fits this expectation reasonably well. In both cases, the support is the fact that random permutation destroys local structure, but leaves the global statistical properties of the data unchanged. If permutation hurts the performance, then this can only mean that our data started out with useful local structure, and that the deep technique was exploiting this structure; otherwise, the performance would not have been impacted. If the improved performance of the deep technique on the original data was due only to the non-linearity involved in the split/join operation, we would expect that the performance on the permuted data would not have been impacted, so we can conclude that the increased performance was likely due to the exploitation of deep structure, and not just the non-linearity.

VI. Conclusions and Future Work

The central result of this work is that deep architectures can yield improved results even without connectionist models, and that this performance seems to be due to a bias that assumes the presence of deep local structure. While many authors have claimed that deep techniques can learn abstract features, it has never been demonstrated that this property holds even without complex connectionist models. We have demonstrated that this property is, at least in part, created directly by the structure of the deep feature hierarchy, and not just by the interactions of non-linearities in a multi-layer connectionist network. Both our deep and flat techniques have the same length output, meaning that neither one has an information-theoretic advantage in its representational power, and removing the local structure from the data via random permutation results in a significant performance loss for the deep technique only.

Another potential advantage of deep methods, and one that is not generally emphasized in the literature, is that in cases where less abstract features are required, a lower level of the hierarchy can be used as the "output" layer, easily providing a range of feature representations at varying levels of abstraction. We have made no use of this property in these experiments, but it is easy to imagine circumstances under which this would be a desirable property. For example, in an image classification or retrieval setting, it would be useful to be able to specify a sub-region of an image, and ask for any image with a similar sub-region to be returned. Many image retrieval systems try to incorporate some type of "region of interest," but they tend to use a brute-force approach; in a deep system, this ability would fall out naturally.

As another example, it might be the case that we would want to include a "level of abstraction" in our query; by performing classification/retrieval at the top level of a DFE hierarchy, we can expect results that are broadly similar, while performing the same operation using a lower level would give us a much more narrow similarity set. We view the exploration of these properties as a promising direction for future work.

As with any results based on real-world data, it is difficult to know how well those results will generalize to different kinds of data. In the future, we intend to apply DFE not only to other image data sets, both natural and synthetic, but also to non-image data. The basic deep-learning hypothesis suggests that other types of data should be amenable to DFE; after all, humans are able to process data other than static images. In particular, we are interested in seeing how the ideas of deep feature extraction can be applied to time-series data.

We also expect that the use of feature extractors other than PCA should be able to improve absolute performance, and plan to do empirical testing to discover how great this impact is. While absolute performance was not our main interest in this work, it will be of prime importance in real world applications of DFE. The feature extractor is also the dominant factor in the overall algorithmic complexity; the overall runtime of our experiments scaled approximately linearly with the depth of the feature hierarchy, but not all feature extractors would be so well behaved.

We set out to explore the properties of deep learning hierarchies by starting with as simple a hierarchy as we could create. Even in this highly simplified hierarchy, we see some benefits from deep learning on a real-world image classification task. These results, while interesting, are only the beginning of a full exploration of how and why deep learning works. Both theoretical analysis and experimental exploration are needed to understand what gives deep learning hierarchies their power, what types of data they are appropriate for, and how to best design hierarchies for particular tasks.

References

[1] E. Adelson, C. Anderson, J. Bergen, P. Burt, and J. Ogden. Pyramid Methods in Image Processing. *RCA Engineer*, 29-6, 1984.

[2] Ethem Alpaydin. *Introduction to Machine Learning*. MIT Press, 2004.

[3] M. Andrecut. Parallel gpu implementation of iterative pca algorithms. *Journal of Computational Biology*, 16(11), 2008.

[4] S. Behnke. *Hierarchical Neural Networks for Image Interpretation*. Springer, 2003.

[5] Y. Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127, 2009. Also published as a book. Now Publishers, 2009.

[6] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep belief networks. *Advances in Neural Information Processing Systeml 19 (NIPS '06)*, pages 153–160, 2007.

[7] Y. Bengio and Y. LeCun. Scaling learning algorithms towards AI. *Large-Scale Kernel Machines*, 2007.

[8] D. Erhan, Y. Bengio, A. Courville, P. Manzagol, and P. Vincent. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11:625–660, 2010.

[9] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, 1980.

[10] D. George and J. Hawkins. Invariant Pattern Recognition using Bayesian Inference on Hierarchical Sequences. *Proc. of International Joint Conference on Nerual Networks*, 2005.

[11] J. Hawkins and S. Blakeslee. *On Intelligence*. Owl Books, Henry Holt and Company, 2004.

[12] G. E. Hinton, S. Osindero, and Y. W. Teh. A fast learning algortihm for deep belief nets. *Neural Computation*, 18:1527–1554, 2006.

[13] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[14] T. Mitchell. The need for biases in learning generalizations. *Technical Report CBM-TR-117*, 1980.

[15] C. Shaw and J. McEachern. Toward a theory of neuroplasticity. *Psychology Press*, 2001.

[16] M. Turk and A. Pentland. Face recognition using eigenfaces. *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 586–591, 1991.

[17] D. H. Wolpert and W. G. Macready. No free lunch theorems for search. *Technical Report SFI-TR-95-02-010*, 1995.