Modern Approaches in Deep Learning for SAR ATR

Michael Wilmanski, Chris Kreucher and Jim Lauer
Integrity Applications Incorporated, Ann Arbor MI, 48108

ABSTRACT

Recent breakthroughs in computational capabilities and optimization algorithms have enabled a new class of signal processing approaches based on deep neural networks (DNNs). These algorithms have been extremely successful in the classification of natural images, audio, and text data. In particular, a special type of DNNs, called convolutional neural networks (CNNs) have recently shown superior performance for object recognition in image processing applications. This paper discusses modern training approaches adopted from the image processing literature and shows how those approaches enable significantly improved performance for synthetic aperture radar (SAR) automatic target recognition (ATR). In particular, we show how a set of novel enhancements to the learning algorithm, based on new stochastic gradient descent approaches, generate significant classification improvement over previously published results on a standard dataset called MSTAR.

Keywords: ATR, SAR, DNN Training

1. INTRODUCTION

Recent computational and algorithmic advances have brought increased attention to a new class of signal processing algorithms referred to Deep Neural Networks (DNNs)[1]. A neural network consists of interconnected groups of nodes, akin to the vast network of neurons in a brain. Each group of nodes corresponds to a layer. A "deep" neural network is one that has a large number of layers and may contain hundreds or thousands of nodes. The application of neural networks to classification problems we discuss here uses a large number of labeled examples to have the network "learn" a function, which maps the input data to output classes. As such it is related to more conventional supervised learning approaches like support vector machines and regression[2]. The principal advantage over these techniques is its ability to learn arbitrarily complicated decision surfaces.

In general, more nodes allow for richer exploitation of the input data, while more layers allow for more intricate decision surfaces. Convolutional Neural Networks (CNNs)[3] are a type of neural network where the individual neurons are tiled in such a way that they respond to overlapping regions in the visual field. CNNs typically involve alternating convolutional layers and pooling layers, where convolutional layers extract features and pooling layers generalize features. Networks that are both deep and employ convolutional feature extraction, deep convolutional neural networks, currently produce state-of-the art results in a wide range of image processing applications[4], as this combination can both maximally exploit the input data and realize arbitrarily non-linear decision surfaces.

A natural application of such networks is to the synthetic aperture radar (SAR) automatic target recognition (ATR) problem. The SAR ATR problem is analogous to the image classification problem, wherein a large database of labeled images exists and the requirement is to label a new unknown sample. The SAR modality has a number of characteristics that distinguish it from natural imagery, most prominently the fact that the data includes both magnitude and phase, but also the large dynamic range and the fact that object signatures are not rotation invariant. These features may provide additional richness that the network can exploit to its advantage.

The Moving and Stationary Target Acquisition and Recognition (MSTAR) database[5] is a publically released collection of ten military vehicles taken from a number of aspect angles and provides a standard, common dataset that ATR researchers have used extensively for algorithm development. Most previous approaches to SAR-based ATR in open literature are based on template matching[6-8]. This approach involves creating a set of templates for each object type, then comparing the correlation between each class template and a test image. While other researchers have proposed using neural networks[8,9] or SVMs[10,11] for SAR ATR, Morgan[12] appears to be the first to publish results of a deep convolutional neural network for this application. In this paper, we adopt the network topology of [12] and describe new approaches to training that are drawn from the most modern techniques in deep learning and image processing literature[4,13,14]. We show through a combination of modern techniques, we can achieve significant classification performance improvement on the MSTAR dataset.

This paper presents the following two main contributions. First, Section 2 describes modern training approaches for CNNs applied to the SAR-ATR problem. Next, Section 3 shows that the deep learning methods we employ provide improved performance on the standard 10-class MSTAR SAR test set by comparing it to other published classification algorithms on this dataset. In particular, while [3] was remarkably able to achieve nearly 93% correct classification on the test set, we show training approaches that achieve nearly 98% correct classification.

2. MODERN TRAINING APPROACHES FOR DNNS

A deep neural network, as employed here, is a signal processing algorithm that learns a nonlinear classification function from a large collection of labeled input data. It is characterized as follows: First, it starts with an input layer corresponding to input data – here synthetic aperture radar data. Next, it has a number of hidden layers, which perform feature extraction and aggregation. In a typical convolutional neural network, the hidden layers are alternating sequences of convolutional and pooling layers. Finally, the network has an output layer, which acts as its classification mechanism. Each layer has neurons with weights that act upon the input features and pass the result to the next layer. An effective network learns weights for each neuron in each layer that produce correct classification outputs for input images.

A number of factors affect the performance of the network, most prominently the network topology (which we use here broadly to refer to the number and types of layers, the number of nodes in each layer, and the activation functions therein), and the approach used to train the network. In this work, we have elected to employ the network topology defined in [12] and focus on developing effective methods for training. With this as background, this section first reviews the network topology and then discusses our training approaches.

2.1 NETWORK TOPOLOGY

We employ an 8-layer (counting input and output) convolutional neural network, modeled after [12]. The first layer is an input layer. It is followed by 6 hidden layers, which alternate convolutional and pooling functions. The final layer is the output layer. This subsection specifies the components of this topology.

2.1.1 INPUT LAYER

Input data comes from the MSTAR database[5], which consists of radar images for 10 classes of military ground targets. Each class has hundreds of example chips (imagery centered around a target of interest) taken at different look angles. Each chip is a 128×128 array of complex data (both magnitude and phase). Let $x^{(j)}$, here a 128×128 image, denote the j^{th} input. $x^{(j)}$ comes from one of $N = 10$ classes. Let $T^{(j)}$ be the N dimensional vector describing the true classification of input j, i.e., if example j belongs to class 3, then $T^j = [0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0]$. The standard approach used in automatic target recognition in SAR imagery involves using only the magnitudes of the returns[5-12].

2.1.2 HIDDEN LAYERS

The hidden layers alternate between convolutional and pooling functions. The second layer takes as input the 128×128 image from the input layer. It applies convolutions using 9×9 kernels and generates a set of 18 feature maps of size 120×120 as output. The third layer takes these as input and performs pooling to generate a set of 18 feature maps of size 20×20. The network continues to alternate the convolutional and pooling layers until the output layer, which produces a 10×1 output. Table 1 summarizes this structure.

Table 1. The network structure adopted here and in [12].

Layer Type	Image Size	Feature Maps	Kernel Size
Input Layer	128 x 128	1	-
Convolutional	120 x 120	18	9 x 9
Pooling	20 x 20	18	6 x 6
Convolutional	16 x 16	36	5 x 5
Pooling	4 x 4	36	4 x 4
Convolutional	1 x 1	120	4 x 4
Fully Connected	1	120	-
Output	1	10	-

Each node employs an activation function, which is a function defining the node input-output relationship. Within the past few years, the rectified linear units (ReLU)[13] and close variations (e.g. leaky-ReLU[14], and parameterized-ReLU[15]) have become the activation functions of choice for many deep learning applications. Unlike the sigmoid and hyperbolic tangent functions, the ReLU activation function does not suffer from vanishing gradients, allowing learning to remain effective, even in deeply stacked layers. Here we have employed ReLU, modeled by the max function.

2.1.3 OBJECTIVE FUNCTION

Finally, the network has an objective function (also known as a "loss function") which defines the error between the correct set of outputs (here, classifications) and the network output with the current set of weights. Let θ denote the set of weights learned by the network. The network output corresponding to input example j is a function of the input and the weights and will be denoted $y^{(j)}(x^{(j)}, \theta)$. y will be an N-dimensional vector with the same meaning as the truth $T^{(j)}$.

With that as background, the cross-entropy error function E defines how well the network outputs match the truth labels. Let D be the number of examples. Then the error is given by

$$E(\theta, D) = \frac{1}{D} \sum_{j=0}^{D-1} \sum_{i=0}^{N-1} T_i^{(j)} \ \log y_i^{(j)}\left(x^{(j)}, \theta\right). \tag{1}$$

This objective function is what the training process is ultimately attempting to minimize (without overfitting).

We use thus cross-entropy error function in conjunction with a softmax output layer. Softmax has become a recent standard in the field of deep learning for classification applications. The softmax layer produces an output probability for each class, where all outputs sum to unity. Here, the output from the previous layer x' of size k is passed through the softmax function to yield the output y for the i^{th} class as

$$y_i = e^{x_i'} / \sum_{k=0}^{K-1} e^{x_k'}. \tag{2}$$

2.2 NETWORK TRAINING

We now describe our approach to training the network. As mentioned earlier, here the objective of the training procedure is to use a set of labeled examples to deduce a set of weights that provide the correct classification for future unknown input images. This optimization is performed by combining an overall network objective (loss) function with an iterative gradient descent algorithm. We employ the widely-used backpropagation algorithm as the means of computing the gradient of the loss function with respect to the weights. An important fact is that the objective function contains many local minima and the minimization is also susceptible to overfitting. Therefore, it is crucial to design a training approach which addresses both of these characteristics.

2.2.1 WEIGHT INITIALIZATION

The first step is weight initialization, which is the process of selecting an initial set of weights θ for the first iteration of training. We select the initial weights from a uniform random distribution. The bounds of this distribution incorporate both the number of neurons in the layer that acts as input (known as $fan\ in$) and outgoing neuron information (known as $fan\ out$, which is the equivalent to the $fan\ in$ of the next convolution layer) as described in[16]. That is, initial connection weights are uniformly distributed between

$$\left[-\sqrt{\frac{6}{fan\ in + fan\ out}}, \quad \sqrt{\frac{6}{fan\ in + fan\ out}}\right] \tag{3}$$

2.2.2 POOLING

Next, the alternating pooling levels employ a procedure which takes input values and constructs a pooled or reduced output. In recent years, the approach called "max" pooling[17] has proven to be more effective at preserving relevant information during the pooling process than the previously popular average pooling (or "mean pooling")[17]. Max pooling divides the image into multiple areas based on the size of the provided kernel (sometimes referred to as the "pooling area") and only propagates forward the most prominent value in each area.

Stochastic Pooling[18] is a new regularization method for CNN training aimed at preventing overfitting. It has shown to provide superior performance in image processing applications[18]. The key idea is to replace the standard deterministic pooling (Max Pooling or Average Pooling) after each convolution layer with a stochastic approach which selects the activation from a distribution corresponding to the activations in the pooling region. The probability for pixel i, p_i, in region R is denoted

$$p_i = a_i / \sum_{k \in R} a_k, \tag{4}$$

where a_i denotes a specific activation. The sampled activation of the j^{th} pooling region s_j, is then a function of the probabilities and activations in the region, given by

$$s_j = a_l \;\; where \;\; l \sim P\left(p_1, \ldots, p_{|R|}\right). \tag{5}$$

2.2.3 WEIGHT OPTIMIZATION

Given these approaches, stochastic gradient descent[19] (SGD) is our general approach for optimizing weight vectors. We report here on the performance of several variants of SGD. Our main contribution is to investigate three modifications to conventional SGD that improve the training speed, prevent the network from becoming trapped in local minima, and automate the process of selecting hyper parameters for these optimizations. The techniques we describe here are SGD with momentum and weight decay[20], AdaGrad[21] and AdaDelta[22].

Each of these techniques is used in conjunction with a stochastic mini-batches[23-25]. In this mini-batch, a large training set of T examples is broken into B "batches" of M elements (where $MB = T$). The examples in each batch are sampled without replacement from the initial dataset. This approach has a number of benefits, including accelerating the process[25], allowing parallelization[23], and preventing overfitting by allowing the optimization to exit shallow minima.

We first review conventional SGD and then describe modifications for advanced optimization of the effective learning rate. In conventional SGD, the only hyper parameter is a global learning rate, ϵ. The change in a network weight is determined only by the product of the learning rate and the weight error gradient, given as

$$\theta_{i+1} = \theta_i - \epsilon \frac{\partial E}{\partial \theta} \tag{6}$$

With that as background, SGD with momentum and weight decay adds a history of the last update made (momentum) and a small rate for slowly depleting inactive weights (known as weight decay). The change in weight at iteration $i + 1$, v_{i+1}, is given by

$$v_{i+1} = -\alpha v_i - \beta \epsilon \theta_i - \epsilon \frac{\partial E}{\partial \theta} \tag{7}$$

Where α specifies the momentum and β is the weight decay constant. The weight decay term is purposely scaled with the learning rate, as the learning rate is adjusted several times throughout training; the weight decay term needs to remain small relative to the learning term. The weight update is then

$$\theta_{i+1} = \theta_i + v_{i+1} \tag{8}$$

Most time learning is spent winding through narrow ravines. Momentum can help push learning along the direction of the ravine, helping avoid what would otherwise be near-indefinite rocking back and forth between ravine walls. Momentum also helps propel past local minima. Weight decay can minimize less useful weights; taking these weights out of the prediction equation can improve generalization.

Two of the main difficulties in efficiently implementing this approach are the selection of the learning rate parameters and the need to manually update the global learning rate. For that purpose, we have employed the AdaGrad and AdaDelta techniques, which aim to handle these selections automatically.

In AdaGrad, each weight gets an individualized effective learning rate. The effective learning rates start high and eventually decay to 0. Let g_i capture the squared gradient history at iteration i of a particular weight, i.e.,

$$g_{i+1} = g_i + \left(\frac{\partial E}{\partial \theta}\right)^2, \tag{9}$$

then the AdaGrad weight update is

$$\theta_{i+1} = \theta_i - \frac{\epsilon}{f + \sqrt{g_{i+1}}} \frac{\partial E}{\partial \theta}, \tag{10}$$

where f is a constant small value meant to prevent the effective learning rate from being infinite at the start of training.

The AdaDelta algorithm incorporates the features of SGD with momentum with AdaGrad. Let

$$v_{i+1} = \left(\frac{-\sqrt{x_i + \epsilon}}{\sqrt{g_{i+1} + \epsilon}}\right)\frac{\partial E}{\partial \theta} \tag{11}$$

where x_{i+1} is an approximation to Hessian, given by

$$x_{i+1} = \alpha x_i + (1-\alpha) v_{i+1}^2 \tag{12}$$

and g_{i+1} is the approximation to the squared gradient, given by

$$g_{i+1} = \alpha g_i + (1-\alpha)\left(\frac{\partial E}{\partial \theta}\right)^2. \tag{13}$$

Then the weight update is given by

$$\theta_{i+1} = \theta_i + v_{i+1}. \tag{14}$$

The following section shows the performance of each of these approaches.

3. EXPERIMENTAL RESULTS

This section describes our experimental results using the 10-class MSTAR database[5]. The database consists of complex, 128×128 pixel SAR chips with vehicles centered. The database is segregated into training and testing sets. The training set has between 195 and 587 examples of each class, totaling 3671 examples. The testing set has 3203 examples with similar properties. The training set is collected at 17° elevation and the test set is collected at 15° elevation.

Here we describe a set of experiments based on detected SAR imagery. Input data was preprocessed by taking the magnitudes of each pixel to create imagery superficially similar to standard optical imagery as illustrated in Figure 1. In addition, all pixel magnitudes are normalized between the range of [-0.5, 0.5].

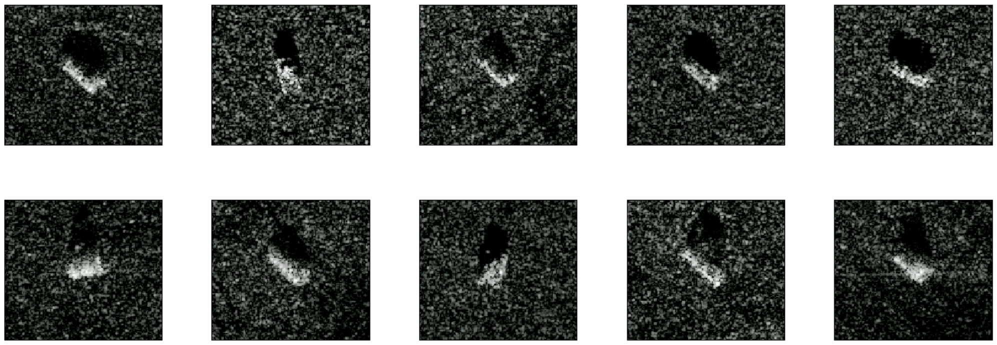

Figure 1. The absolute value (detected) imagery for one element in each of ten MSTAR classes.

We now describe the classification performance on the sequestered test set using each of the approaches of Section 2. As discussed there, training is an iterative process. We refer to each iteration, wherein each element of the training dataset is considered, as an epoch. Each epoch includes a number of batches, which are random subsets of the training data of a predefined batch size. The network is trained for a number of epochs using the training set (only). The trained network is then applied to the sequestered testing set to determine a correct classification rate. For each algorithm, we report the parameter choices, the final classification performance on the sequestered test set, a plot of performance versus training epoch, a confusion matrix which describes the relationship between the true and predicted classes, and the training loss.

3.1 SGD WITH MOMENTUM AND WEIGHT DECAY

We employed SGD with momentum and weight decay as described by equations (7) and (8). We selected a batch size of 49, so there are 75 batches per epoch to include all 3671 training examples. Some training examples were repeated so the training set is an integer multiple of the batch size. The global learning rate was initialized with learning parameter $\epsilon = 0.05$, and scaled by 0.5 at the manually selected batch numbers $1352,\ 1877,\ 2252,\ 2552,\ 2927$ and 3152, which is where the learning was found to plateau. The momentum was set at $\alpha = 0.9$ and scaled by 1.05 at the same batches. The weight decay parameter was selected as $\beta = .0005$.

The network was trained for 100 epochs, and we find the performance and instantaneous loss stabilize after about 50 epochs, as Figure 2 illustrates. The left panel shows the correct classification rate of the classifier as a function of training epoch. The right panel gives the instantaneous loss (i.e., the value of the objective function for a particular batch) as a function of the training epoch. Since the network is trained using batches much smaller than the entire dataset, the instantaneous loss, calculated for each batch inherently has a large variance. The epochs where the learning rate was (manually) adjusted are highlighted with dashed lines.

This performance figure was constructed by recording the weights after each epoch (pass through the entire training set) and using those weights to compute performance on the sequestered testing set. Training set performance was also noted. Notice that due to the batching and stochastic gradient, network weights continue to exhibit small fluctuations even after many epochs. This leads to a corresponding fluctuation in testing performance. We find the classification performance on the sequestered test set is between 96.6% and 97.7%, the minimum and maximum performances after epoch 50.

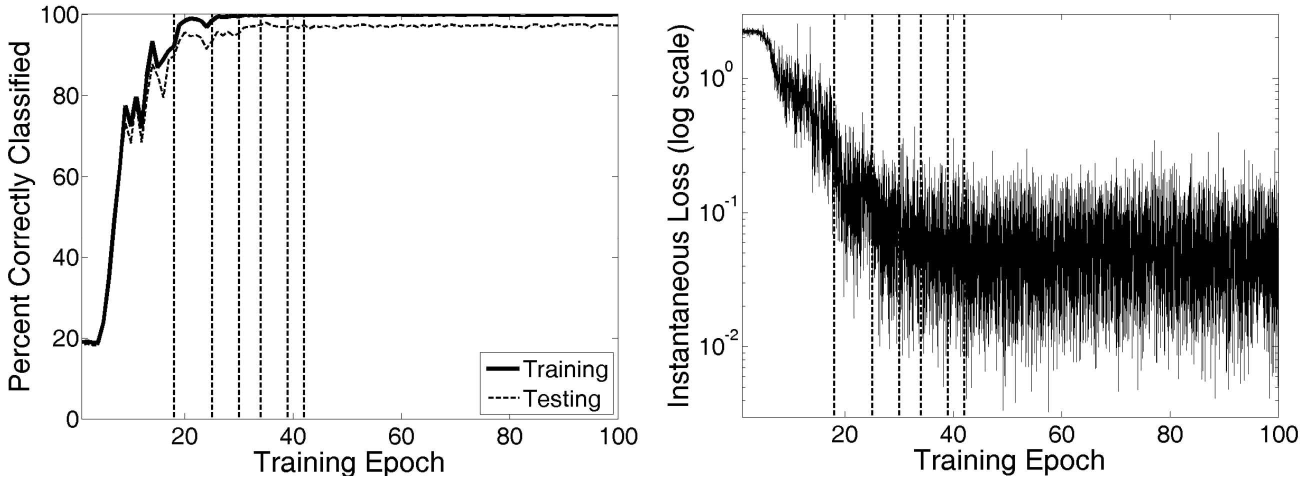

Figure 2. Left: The classification ability versus training epoch, and Right: the instantaneous loss versus training epoch.

Table 2 is a confusion matrix illustrating how the classifier labeled the ten classes. We have elected to show the results at epoch 55, which is the lowest testing performance after epoch 50.

Table 2. The confusion matrix corresponding to 96.6% correct classification on the 10-class MSTAR set using SGD with momentum and weight decay.

		PREDICTED										
		2S1	**BMP2**	**BRDM2**	**BTR60**	**BTR70**	**D7**	**T62**	**T72**	**ZIL131**	**ZSU23**	**% CORR**
ACTUAL	**2S1**	270	0	0	1	0	0	0	2	1	0	98.54
	BMP2	1	527	0	2	3	0	0	54	0	0	89.78
	BRDM2	0	3	269	0	0	0	1	1	0	0	98.18
	BTR60	0	0	1	183	1	0	1	5	0	4	93.85
	BTR70	0	0	0	3	193	0	0	0	0	0	98.47
	D7	0	0	0	0	0	272	1	0	1	0	99.27
	T62	0	0	0	0	0	2	261	7	2	1	95.6
	T72	1	3	0	0	0	0	1	576	0	1	98.97
	ZIL131	0	0	0	0	0	0	0	0	274	0	100
	ZSU23	0	0	0	0	0	5	0	0	0	269	98.18

3.2 ADAGRAD

We applied the AdaGrad technique defined by equations (9)-(10) using learning rate $\epsilon = .01$, regularization term $F = 0.1$, and a 49 element batch as before. This approach, which is not manually tuned other than choosing the learning rate and regularization, can be viewed as an intermediate step between the carefully tuned learning shown in Section 3.1 and the alternative automated AdaDelta approach shown in Section 3.3.

We find the training process is much slower and that the trained network is not as good as the manually tuned version of Section 3.1. In particular, we find there are several classes that are very poorly classified. These features are illustrated in Figure 3, where the asymptotic classification rate on the test set is between 89 and 92%.

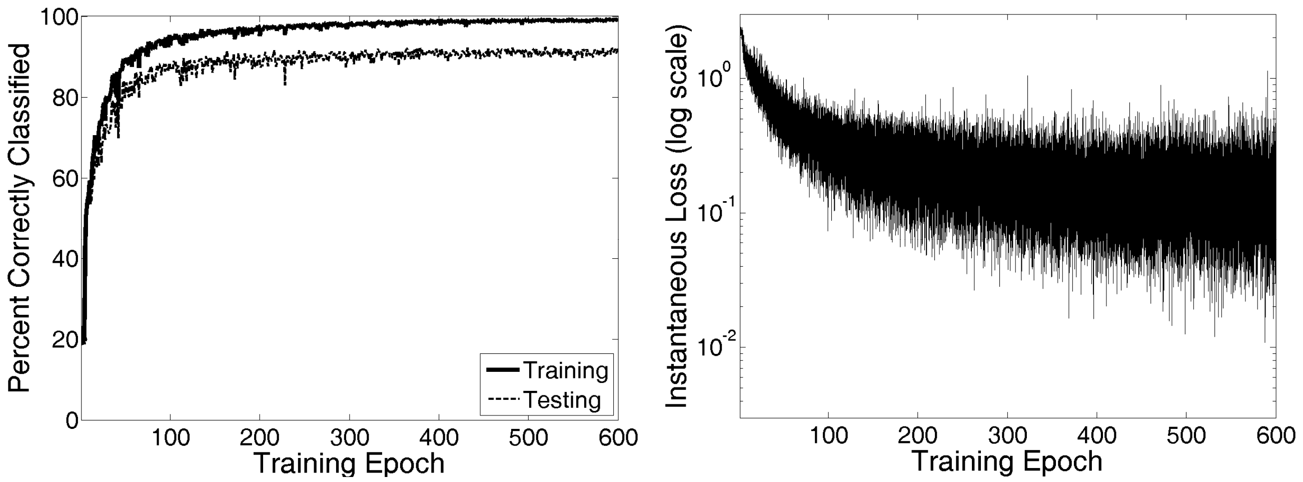

Figure 3. Left: The classification ability versus training epoch, and Right: the instantaneous loss versus training epoch.

The confusion matrix corresponding to the lowest of these classification rates is shown below.

Table 3. The confusion matrix corresponding to training with the AdaGrad method.

		PREDICTED										
		2S1	**BMP2**	**BRDM2**	**BTR60**	**BTR70**	**D7**	**T62**	**T72**	**ZIL131**	**ZSU23**	**% CORR**
ACTUAL	**2S1**	165	20	0	3	0	0	1	85	0	0	60.2
	BMP2	4	523	4	7	3	0	0	42	4	0	89.1
	BRDM2	1	10	261	0	0	1	1	0	0	0	95.2
	BTR60	0	30	0	144	6	0	0	7	0	8	73.9
	BTR70	1	36	1	7	151	0	0	0	0	0	77.0
	D7	0	1	0	0	0	269	2	0	0	2	98.2
	T62	0	0	0	0	0	1	260	8	3	1	95.2
	T72	0	8	0	2	0	0	0	570	0	2	97.9
	ZIL131	1	0	1	0	1	0	8	4	258	1	94.1
	ZSU23	0	0	0	0	0	0	0	0	0	274	100.0

3.3 ADADELTA

We next illustrate the results of the AdaDelta technique, which was defined in equations (11) - (14). As described in Section 2, AdaDelta provides a method for performing SGD with momentum which continually and automatically updates the various rates that dictate the learning. As such, it removes the manual intervention from the training process described in Section 3.1.

We find that the AdaDelta algorithm gives performance as good or better than the hand-tuned approach. Furthermore, we also find that this learner is significantly more robust to parameter selection.

We employed an AdaDelta learning rate of $\epsilon = 1e - 6$ and momentum constant of $\alpha = .99$. As alluded to above, we find results similar to that described here when we selected the parameters $\epsilon = 1e - 7$ and $\alpha = .9$. The batch size remained at 49 as in the other experiments. The network was trained for 100 epochs and found to stabilize after about 50 epochs. It achieved an accuracy on the testing set of between 97.5% and 98.5%. The confusion matrix for the lower of these two is given in Table 4.

Table 4. The confusion matrix corresponding to training with the AdaDelta method.

		PREDICTED										
		2S1	**BMP2**	**BRDM2**	**BTR60**	**BTR70**	**D7**	**T62**	**T72**	**ZIL131**	**ZSU23**	**% CORR**
ACTUAL	**2S1**	266	2	0	2	1	0	0	3	0	0	97.08
	BMP2	1	547	0	4	5	0	0	30	0	0	93.19
	BRDM2	0	0	271	0	0	0	0	0	3	0	98.91
	BTR60	0	0	3	177	7	0	0	4	0	4	90.77
	BTR70	0	0	0	2	193	0	0	1	0	0	98.47
	D7	0	0	0	0	0	273	0	0	1	0	99.64
	T62	1	0	0	0	0	2	268	2	0	0	98.17
	T72	1	0	0	0	0	0	0	581	0	0	99.83
	ZIL131	1	0	0	0	0	1	0	0	272	0	99.27
	ZSU23	0	0	0	0	0	3	0	0	0	271	98.91

Figure 4 again further illuminates the network training procedure by showing both the classification performance as a function of training epoch and the instantaneous loss. Note that the scale on the instantaneous loss is set two orders of magnitude lower than those of Figure 2 and Figure 3, indicating the optimization has found a minimum significantly lower than those found with the other optimization approaches.

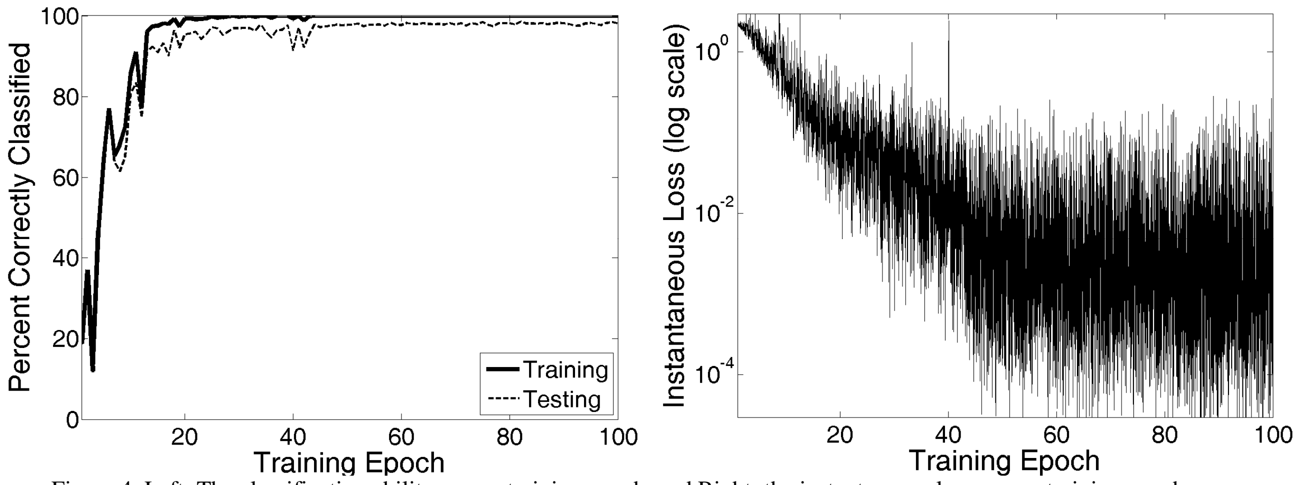

Figure 4. Left: The classification ability versus training epoch, and Right: the instantaneous loss versus training epoch.

4. CONCLUSION

This paper has described modern training approaches for deep neural networks applied to the SAR ATR problem. We have illustrated that with a combination of good initialization and effective stochastic gradient decent modifications, we can achieve nearly 98% correct classification on the MSTAR test set using only the training set for learning. Exploitation of complex features is an additional rich area for research, as the phase information has shown to be quite informative in other applications. Additional future work includes developing mechanisms for reporting confidence on classification calls and automatically detecting test chips that are not present the training database.

REFERENCES

[1] Bengio, S., Deng, L., Larochelle, H., Lee, H., and Salakhutdinov, R., Special Issue on Learning Deep Architectures, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 8, no. 35 (2013).

[2] Hastie, T., Tibshirani, and R., Friedman, J., The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Springer (2009).

[3] LeCun, Y., Bottou, L, Bengio, Y., and Haffner, P., “Gradient-based learning applied to document recognition”, The Proceedings of the IEEE 86, vol. 11, pp. 2278-2324 (1998).

[4] Krizhevsky, A., Sutskever, I., and Hinton, G. E., “Imagenet classification with deep convolutional neural networks," in Advances in Neural Information Processing Systems (NIPS), (2012).

[5] Ross, T., Worrell, S., Velten, V., Mossing, J., and Bryant, M., “Standard SAR ATR evaluation experiments using the MSTAR public release data set”, Proc. SPIE 3370 (1998).

[6] O'Sullivan, J., DeVore, M., Kedia, V., and Miller, M., “SAR ATR Performance Using a Conditionally Gaussian Model,” IEEE Transactions on Aerospace and Electronic Systems, vol. 37, no. 91 (2001).

[7] Srinivas, U., Monga, V., and Raj, R., “SAR automatic target recognition using discriminative graphical models,” IEEE Transaction on Aerospace and Electronic Systems, vol. 50, (2014)..

[8] Sun, Y., Liu, Z., Todorovic, S., and Li, J., “Adaptive boosting for SAR automatic target recognition,” IEEE Transactions on Aerospace and Electronic Systems, vol. 53 (2007).

[9] Principe, J., Kim, M., and Fisher III, J., “Target Discrimination in Synthetic Aperture Radar Using Artificial Neural Networks”, IEEE Transactions on Image Processing, vol. 7, no. 8, pp. 1136-1149 (1998).

[10] Zhao, Q. and Principe, J., “Support vector machines for SAR automatic target recognition,” IEEE Transactions on Aerospace and Electronic Systems, vol. 37 (2001).

[11] Bryant, M. and Gaber, F., “SVM Classifier Applied to the MSTAR Public Data Set”, Proc. SPIE 3721 (1999).

[12] Morgan, D., "Deep convolutional neural networks for ATR from SAR imagery ", Proc. SPIE 9475, (2015).

[13] Jarrett, K., Kavukcuoglu, K., Ranzato, M., and LeCun, Y. “What is the best multi-stage architecture for object recognition?” In Proc. IEEE International Conference on Computer Vision pp. 2146-2153 (2009).

[14] Maas, A., Hannun, A., and Ng, A., “Rectifier Nonlinearities Improve Neural Network Acoustic Models,” In Proceedings of the 30th International Conference on Machine Learning (2013).

[15] He, K., Zhang, x., Ren, S., and Sun, J., “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification” in arXiv:1502.01852v1 (6 February 2015)

[16] Bengio, Y., “Practical Recommendations for Gradient-Based Training of Deep Architectures” in arXiv:1206.5533v2 (2012).

[17] Scherer, D., Müller, A., and Behnke, S., “Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition”, 20th International Conference on Artificial Neural Networks, (2010).

[18] Zeiler, M., and Fergus, R., “Stochastic Pooling for Regularization of Deep Convolutional Neural Networks”, International Conference on Learning Representations, (2013)

[19] Bottou L., Bousquet, O. “The Tradeoffs of Large Scale Learning”, *Advances in Neural Information Processing Systems*, pp. 161–168 (2008).

[20] Krogh, A., and Hertz., J., “A Simple Weight Decay Can Improve Generalization,” In Advances in Neural Information Processing Systems, pp. 950–957, (1992).

[21] Duchi, J., Hazan, E., and Singer, Y., “Adaptive Subgradient Methods for Online Learning and Stochastic Optimization”, (2010).

[22] Zeiler, M., “AdaDelta: An Adaptive Learning Rate Method”, arXiv:1212.5701v1 (22 December 2012).

[23] Li, M., Zhang, T., Chen, Y., Smola, A., "Efficient Mini-batch Training for Stochastic Optimization", Proceedings of ACM SIGKDD Conference on Knowledge Discovery and Data Mining, (2014).

[24] Cotter, A., Shamir, O., Srebro, N., and Sridharan, K., "Better Mini-Batch Algorithms via Accelerated Gradient Methods", In Advances in Neural Information Processing Systems (2011).

[25] Bengio, Y., "Speeding up Gradient Descent", In NIPS Workshop on Efficient Machine Learning (2007).