# Bridging a Gap in SAR-ATR: Training on Fully Synthetic and Testing on Measured Data

Nathan Inkawhich, Matthew J. Inkawhich, Eric K. Davis, Uttam K. Majumder, *Senior Member, IEEE*, Erin Tripp, Chris Capraro, and Yiran Chen, *Fellow, IEEE*

***Abstract*—Obtaining measured synthetic aperture radar (SAR) data for training automatic target recognition (ATR) models can be too expensive (in terms of time and money) and complex of a process in many situations. In response, researchers have developed methods for creating synthetic SAR data for targets using electro-magnetic prediction software, which is then used to enrich an existing measured training dataset. However, this approach relies on the availability of some amount of measured data. In this work, we focus on the case of having 100% synthetic training data, while testing on only measured data. We use the SAMPLE dataset public released by AFRL, and find significant challenges to learning generalizable representations from the synthetic data due to distributional differences between the two modalities and extremely limited training sample quantities. Using deep learning-based ATR models, we propose data augmentation, model construction, loss function choices, and ensembling techniques to enhance the representation learned from the synthetic data, and ultimately achieved over 95% accuracy on the SAMPLE dataset. We then analyze the functionality of our ATR models using saliency and feature-space investigations and find them to learn a more cohesive representation of the measured and synthetic data. Finally, we evaluate the out-of-library detection performance of our synthetic-only models and find that they are nearly 10% more effective than baseline methods at identifying measured test samples that do not belong to the training class set. Overall, our techniques and their compositions significantly enhance the feasibility of using ATR models trained exclusively on synthetic data.**

***Index Terms*—Automatic target recognition (ATR), deep learning, synthetic aperture radar (SAR).**

## I. Introduction

A KEY issue in the development of automatic target recognition (ATR) models for SAR data is the availability of realistic *measured* data for targets-of-interest, in situations-of-interest. The means of collecting measured SAR data for training an ATR model is costly, and involves several complex processes [1]. From the availability of the physical radar system, to the staging and imaging of realistic scenes, to the postprocessing and manual labeling of ground truth data, it is not feasible to invoke this pipeline for every ATR task of interest.

One promising way to reduce (or potentially eliminate) the need to collect measured training data is to simulate the collection process and create *synthetic* SAR data. Unlike the previously mentioned pipeline, the simulation starts with the formation of realistic computer-aided design (CAD) models of the targets, and the estimation of the reflective properties for each surface on the model and background. Then, radar signal returns are predicted at regular positions over a dome-shaped pattern surrounding the target using a ray-tracing based technique [2], [3]. The predicted returns can then be used to build simulated phase history records similar to what would be collected by a sensor under common operating conditions. Finally, the simulated phase history is processed by an SAR image formation algorithm, such as the FFT-based polar format algorithm [4], to form the complex image for a range of selected azimuth views at a desired resolution [2]. Albeit computationally expensive, creating synthetic data for training ATR models avoids many of the challenges related to measured data collection; the major downsides being the accuracy with which we can simulate the targets in CAD, and the complexities of the radar estimation.

*The central motivation of this article is to train DL-based SAR-ATR models entirely on synthetically generated data, and achieve high performance when testing on measured data.* Critically, we consider the performance in the context of both accuracy on known targets as well as the ability of the ATR system to detect and reject out-of-library confusers. This definition of performance addresses a more realistic "open-world" assumption for the deployment of SAR-ATR systems, where it can not be assumed that all targets encountered in the field have been trained on. The key challenge we must overcome lies in the differences between the synthetic (training) and measured (testing) data distributions [5]. Our models must not over-fit to the unique properties of the synthetic data, while also learning a robust and transferable representation of the target signatures. We introduce several techniques that are individually known to bolster generalization and reduce over-fitting, and whose composition we find to greatly improve the effectiveness of ATR systems trained on synthetic and tested on measured

Manuscript received November 2, 2020; revised January 21, 2021; accepted February 4, 2021. Date of publication February 17, 2021; date of current version March 17, 2021. This work was supported by AFRL under Contract FA8750-18-C-0148. *(Corresponding author: Nathan Inkawhich.)*

Nathan Inkawhich, Matthew J. Inkawhich, and Yiran Chen are with the Electrical and Computer Engineering Department, Duke University, Durham, NC 27708-0187 USA (e-mail: nai2@duke.edu; matthew.inkawhich@duke.edu; yiran.chen@duke.edu).

Eric K. Davis is with Machine Learning & AI, SRC Inc., North Syracuse, NY 13212-2509 USA (e-mail: davis.ek38@gmail.com).

Uttam K. Majumder is with Computing and Communications, Air Force Research Laboratory Information Directorate, Rome, NY 13441-4514 USA (e-mail: ukmccny@gmail.com).

Erin Tripp is with the High Performance Systems Branch, Air Force Research Laboratory Information Directorate, Rome, NY 13441-4514 USA (e-mail: erin.tripp.4@us.af.mil).

Chris Capraro is with Radars and Sensors, SRC Inc., North Syracuse, NY 13212-2509 USA (e-mail: ccapraro@srcinc.com).

This article has supplementary downloadable multimedia material available at http://ieeexplore.ieee.org, provided by the authors.

Digital Object Identifier 10.1109/JSTARS.2021.3059991

data. Specifically, our techniques fall in the categories of data augmentation, model construction, training function choice, and model ensembling.

To perform the experiments, we leverage the recent synthetic and measured paired labeled experiment (SAMPLE) dataset [2], which has (measured, synthetic) pairs of targets from the moving and stationary target acquisition and recognition (MSTAR) public dataset [6]. As a part of [2], four core experimental designs were proposed, of which we consider two (i.e., Experiments 4.1 and 4.3 of [2]). The first experiment is to maximize the accuracy of a classifier that is trained on a mixture of measured and synthetic data, and tested on exclusively measured data. Specifically, we focus on the case where 100% of the training data is synthetic. The second experiment is to achieve high accuracy on some in-distribution set of classes, while being capable of identifying out-of-library test samples. We follow the previous setup of training on synthetic and testing on measured data, however, we additionally hold-out a subset of classes from the training dataset and work to identify samples from these classes at test time as out-of-distribution (OOD). Throughout the experiments, emphasis is also placed on the analysis of our techniques, which provides intuition for how/why they work.

A summary of our contributions are as follows.

1) We identify and articulate several key problems for training ATR algorithms on 100% synthetic and testing on 100% measured SAR data.
2) We introduce four categories of modifications to the standard SAR-ATR training procedure: data augmentation, model construction, training function choice, and ensembling; and show that individually each category can boost the transferability of the learned representation, while being most effective when composed.
3) We analyze the underlying behaviors behind our proposed techniques and find that they cause the ATR models to focus on salient regions of the target object and to blend the representations of the synthetic and measured samples in feature space.
4) We show that models trained with a composition of our proposed modifications are significantly more proficient at identifying out-of-library confusers during testing.

The remainder of this work is organized as follows. Section II provides background information regarding the SAMPLE dataset, challenges of working with synthetic data, and several related works. Section III describes our training methodologies and experimental results focused on accuracy. Section IV describes methodologies and results for detecting out-of-library confusers at test time, and Section V provides conclusions and motivations for future works.

## II. Background

### A. SAMPLE Dataset

We use the SAMPLE dataset [2] to train and evaluate our ATR models, which contains (measured, synthetic) pairs for the ten public MSTAR [6] target classes. Measured[1] SAR data

[1]for the remainder of this document, we will use the terms "measured" and "real" data interchangeably.

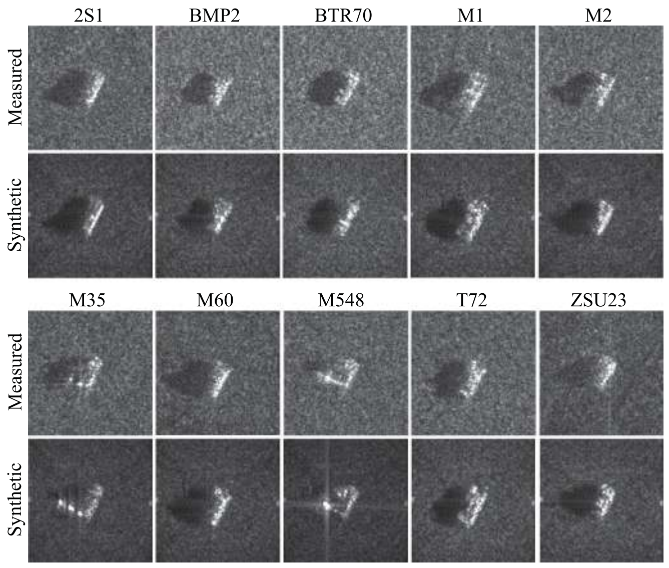

Fig. 1. SAMPLE dataset (measured, synthetic) pairs.

TABLE I
Details of the SAMPLE Dataset

| Class # | Class Name | # Train | # Test | Total |
|---|---|---|---|---|
| 0 | 2S1 | 116 | 58 | 174 |
| 1 | BMP2 | 55 | 52 | 107 |
| 2 | BTR70 | 43 | 49 | 92 |
| 3 | M1 | 78 | 51 | 129 |
| 4 | M2 | 75 | 53 | 128 |
| 5 | M35 | 76 | 53 | 129 |
| 6 | M548 | 75 | 53 | 128 |
| 7 | M60 | 116 | 60 | 176 |
| 8 | T72 | 56 | 52 | 108 |
| 9 | ZSU23 | 116 | 58 | 174 |
| | Totals | 806 | 539 | 1345 |

were taken directly from the MSTAR public release dataset. Corresponding synthetic samples for each of the MSTAR images were first recreated as CAD models in software with estimated reflectivity properties for each surface, then the electromagnetic signatures of each target were predicted using asymptotic ray-tracing techniques [2]. The synthetic phase history was then formed to images using a polar format algorithm [4] with parameters to mimic the MSTAR targets (i.e., 0.3 m range resolution, $128 \times 128$ px image size, *HH* polarization, $10^\circ - 80^\circ$ azimuth range). Fig. 1 shows one (measured, synthetic) pair from each class. Like the MSTAR standard operating condition (SOC) test, the training and test data vary in elevation [6], [7]. Under SOC for SAMPLE, the training data come from elevations $14^\circ - 16^\circ$ and the test data are all from $17^\circ$ elevation. Table I shows the number of training and test samples from each class.

### B. Challenges Presented by Synthetic Data

The core challenge of training exclusively on synthetic data is the apparent *distribution gap* between the synthetic and measured distributions. Informally, from the samples in Fig. 1 we

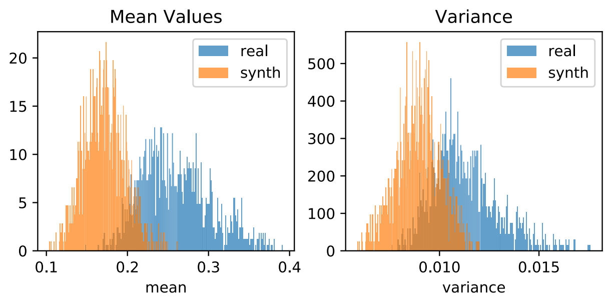

Fig. 2. Histograms of image means and variances.

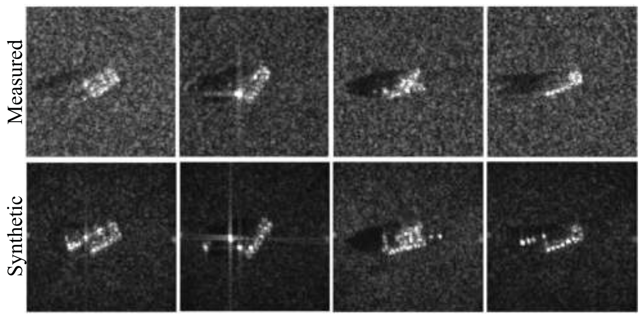

Fig. 3. Select (measured, synthetic) pairs with significant differences.

may describe the synthetic and measured data as "similar-but-different." Reasons for the differences include difficulties in accurately modeling intricate objects in CAD software, imperfect radar estimation algorithms, variability operating conditions, and complications in simulating realistic background clutter. One straightforward way to show the statistical differences between the synthetic and measured data is to plot histograms of the image means and variances in Fig. 2. Clearly, the synthetic data tend to have lower mean and variance than the measured. In part, this difference has to do with the lack of clutter in the synthetic chips [5], [8], as evident from the darker background regions in Fig. 1. In addition, Fig. 3 shows selected (measured, synthetic) data pairs with clear differences in the overall structure of the targets as well as fine differences in the target details. To succeed, our methods must prevent over-fitting to the unique properties of the synthetic data distribution, while simultaneously learning a robust representation of the target classes.

Another challenge more specific to the SAMPLE dataset is the *lack of training data*. From Table I, there are a total of 806 training images, yielding an average of 80 samples per class. This lack of data opposes the sample-complexity requirements of training DNNs and also implicates the "curse of dimensionality" because our training samples are sparsely distributed in the high dimensional input space. To overcome the lack of training samples, our methods must also focus on bolstering generalization with very little data.

### C. Learning With Synthetic Data

With the clear utility of leveraging synthetic SAR data in ATR tasks, there has emerged several different methodologies for learning with synthetic data, some of which also use the SAMPLE dataset.

*1) Learning a Transform Function:* One popular method is to learn a transform/preprocessing function between the synthetic and measured distributions. The goal is to reconcile any differences between the manifolds of the synthetic/measured data by transforming the synthetic data to "look" more like the measured data. This is motivated by the observation that using synthetic data in its unaltered form yields poor performing models in measured data applications [9]. Some methods in this category rely on training DL-based generative adversarial networks (GANs) and/or auto-encoders to learn the transform based on seeing many examples of both synthetic and measured data [9]–[11]. Further, Scarnati and Lewis [8] design a preprocessing function for synthetic SAMPLE data which perform a despecking, quantization, and clutter transfer between (measured, synthetic) pairs. Critically, the act of learning transform functions implicitly requires access to both synthetic and measured data. Thus, learned transforms do not fit within our 100% synthetic training data scheme.

*2) Transfer Learning:* Another popular technique for leveraging synthetic data is to perform transfer learning (TL) [12]–[14]. In most of these works, one trains an initial source model on synthetic SAR data to learn useful and generalizable features of the targets. Then, the model is fine-tuned by retraining to update the learned representation using a training set of measured data. Thus, the goal is to learn the most beneficial features from the synthetic data and to efficiently refine them in the retraining process. Note, the TL problem statement also assumes access to training sets of both measured and synthetic data, which makes it outside of our synthetic-only problem statement.

*3) Toward Fully Synthetic Training:* Finally, there are a few works that focus on using little to no measured training data, which are the most closely related to our work. Sellers *et al.* [15] design an augmentation method for applying measured-phase-error noise to the synthetic data as to account for differences in measurement conditions. To apply the noise, the method requires a small amount of measured training data from which it can then extrapolate the noise to all of the synthetic images. The method ultimately uses 1% measured and 99% synthetic training data, and is able to improve the classification performance of the SAMPLE problem to 95%, while incidentally increasing the training time by a factor of 10×. Although impressive, we remark that there is a critical distinction between having/needing 1% and 0% measured data available for training. Lewis *et al.* [16] consider the 100% synthetic training data case for the SAMPLE dataset, while using an atypical implementation of adversarial training (AT). The intuition is that the AT procedure would better align the decision boundaries for separating the measured data. However, they report marginal benefits. We also consider AT in this work, however, we use a standardized version [17] which has proven to learn more robust representations, and is distinct from the version used in the referenced work [16].

## III. Enhancing Generalization and Accuracy

Within our stated "open-world" operating assumption, our first performance target is to train accurate models for the in-distribution test samples using a purely synthetic training distribution. In this case, we can assume that the test data are

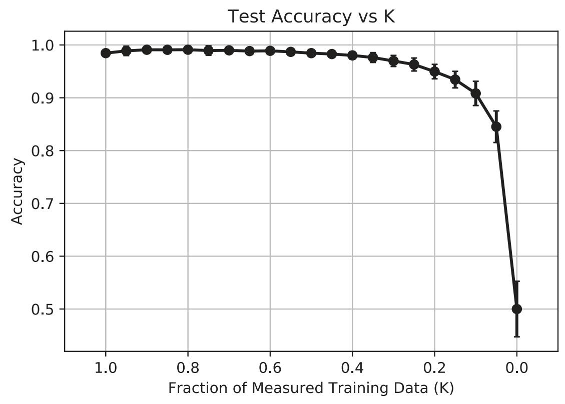

Fig. 4. Test accuracy versus percentage of measured training data.

from the set of training classes, so performance is measured as the generalization ability (i.e., accuracy) of the trained classifier to the measured distribution. Specifically, we operate within the rules described in *Experiment 4.1* of [2]. In the generic setup of Experiment 4.1, a parameter $K \in [0, 1]$ is introduced which defines the percentage of measured training data, while the test data is always 100% measured. For example, if $K = 0.75$ then 75% of the 806 (measured, synthetic) training data pairs use the measured component, while the remaining 25% of training samples are represented by their synthetic data component. Most related to our work, if $K = 0$, then 100% of the training samples are synthetic.

To show that $K = 0$ is a special case worth studying, consider Fig. 4 which shows the test accuracy as $K$ is swept from 0 to 1 (using the default model and training configuration from [2]). The general trend is that as $K \to 0$, the test accuracy decreases. This alone is not surprising because the model is presumably learning the dominant features of the synthetic data, which may not align perfectly with the dominant features of the measured test data. However, notice the rate of change in the test accuracy w.r.t. $K$. For $K \in [0.1, 1]$ the average test accuracy is consistently above 90%, for $K = 0.05$ accuracy drops to $\sim 85\%$, then at $K = 0$ accuracy drops sharply to $\sim 50\%$. Motivated by understanding and reconciling this drop in performance, the remainder of this section is dedicated to improving accuracy at $K = 0$.

## A. Methodology for Training At $K = 0$

To address the aforementioned problems with training on 100% synthetic data and testing on 100% measured data, we consider three methodological categories of modifications to the standard training setup: 1) data augmentations; 2) model construction; 3) training function selection. Each category includes two or more methods, on which we elaborate here.

*1) Data Augmentation:* The first category of improvements adds image-level augmentations to the training data, which reduce the model's propensity to over-fit any particular view of the data [18], [19]. Thus, the model learns more informative underlying features, as opposed to memorizing weak features such as orientation or exact pixel magnitudes. In this article, we consider two augmentations: Gaussian noising and rotation. To apply Gaussian noise to a training image $x$, we first set the standard deviation $\sigma$ (as a hyper-parameter), then create an augmented version as $\tilde{x} = x + \mathcal{N}(\text{mean} = 0, \text{stdev} = \sigma)$. The values of $\tilde{x}$ are then clipped to the image range of [0,1]. Since the synthetic chips are found to have a lack of background clutter, adding Gaussian noise to effectively estimate such clutter is of particular interest, and may directly address the found distribution gap. The second training augmentation we consider is a straightforward image rotation by a maximum of $\pm r^\circ$. This process is randomized, so for any given training image the amount of rotation applied is $\texttt{rot} \sim Uniform(-r, r)$. Rotation augmentations are of interest in this work because they have been shown to enhance the learned representations of DNNs and reduce over-fitting in several computer vision tasks [19], [20]. Importantly, data augmentations are only applied at training time.

*2) Model Construction:* The second category of improvements is to change the construction of the DNN model itself. This technique addresses the bias-variance tradeoff by controlling the complexity of the searched hypothesis space. First, we would like to avoid a situation where an unnecessarily high bias is induced through the choice of an overly simplistic model (i.e., we use a very small model to search a limited hypothesis space that cannot perform well on a complex task). We would also like to avoid extreme variance via over-fitting the model's decision boundaries to the training samples (i.e., we use too big of a model which memorizes the training data and has weak generalization to unseen samples).

The first method of changing model construction is to add dropout layers [21] between existing processing layers of a DNN (usually, dropout layers are inserted directly after the nonlinear activation unit, and may be added between convolutional and/or linear layers). The strength of a dropout layer is determined by a parameter `p_drop`, which represents the probability that a unit within the feature map will be randomly dropped. For dropout layers placed between convolutions, `p_drop` is the probability that any channel of the input feature map will be zeroed out. For dropout layers between linear layers, `p_drop` is the probability that the output of any node in the layer will be zeroed out. During training, the dropout layers are active and randomly select units of the feature maps to drop for each input sample. During testing, the dropout layers are inactive, and act as pass-throughs. Intuitively, each layer (and ultimately the whole model) learns a more robust and generalized representation of the data by simulating cases when not all features are present. In terms of the bias-variance tradeoff, dropout layers act to reduce variance by resisting a large model's ability to over-fit.

The next method of changing model construction is the use of different DNN architectures. The first model we employ is a relatively small LeNet-style [22] architecture, called the `sample` model (SMPL). This model is first described in [2, Table 7] and is composed of four {convolution, max pool} blocks, followed by four linear layers. The next model is ResNet18 (RN18) [23] which is larger in parameter count and also incorporates *residual connections*. The inclusion of residual connections has shown to significantly improve the performance of ResNets on the

ImageNet task, which prompts its use here. The final model is Wide-ResNet18 (WRN18) [24] and is the largest in parameter count. This model has a similar structure to ResNet18, however, has 2× the number of convolutional kernels in each layer, and thus, is considerably larger than the other two models. By using these three architectures, we are testing the effect of using small, medium, and large architectures, which effectively represents three points on the bias-variance curve.

*3) Training Function Choice:* The third category of improvements is to change the primary training function for learning the model weights, to accomplish the SAMPLE data classification task. We consider a total of five variants, all of which stem from the generalized *risk minimization* objective for classifiers over the distribution of SAR images ($p_{true}$)

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \underset{(x,y)\sim p_{true}}{\mathbb{E}} \left[ L(x, y; \theta) \right]. \tag{1}$$

For a given model parameterized by $\theta$, we wish to learn the optimal set of parameters $\theta^*$ to minimize the expected classification loss $L$ for SAR chips $x$ and labels $y$ drawn from $p_{true}$. However, we do not have access to the infinite set of samples in $p_{true}$, rather, we assume to start with a $N$ sample training dataset $\mathcal{D} = \{(x_n, y_n)\}_{n=1 \ldots N} \sim p_{true}$. In our specific problem, $\mathcal{D}$ is the 806 synthetic training data samples from the 10-classes in the SAMPLE dataset.

The first three training function variants extend directly from the formulation of *empirical risk minimization* (ERM) over samples in $\mathcal{D}$, i.e.,

$$\min_{\theta} \frac{1}{N} \sum_{(x_i, y_i)\sim\mathcal{D}} L(x_i, y_i; \theta). \tag{2}$$

Critically, the variation between these techniques comes from modifications to the loss function $L$ and ground truth labels $y$, which encode how we measure risk.

*a) Cross-entropy with one-hot labels:* The first variant will serve as the baseline and is the most common configuration of loss in DNN image classifiers: cross-entropy loss with one-hot labels. Cross-entropy loss is the extension of log-loss for the binary classification case to a multi-class setting. For a $C$-class classification problem, cross-entropy loss is defined as

$$L_{xent}(x, y; \theta) = -\sum_{c\in C} y_c log(f(x; \theta)_c). \tag{3}$$

Here, $y_c$ is the $c$th element of the ground truth label vector $y$, and $f(x; \theta)_c$ is the predicted probability that input $x$ belongs to class $c$ (by our DNN model $f$ which is parameterized by $\theta$). Being a one-hot label, the $y$ vector is sparse so the log-loss is only calculated against the predicted probability for class $c$. The intuition behind this loss/label formulation is to learn a model that maximizes the output probability for the true class.

*b) Label smoothing:* The second variant of loss function choice we consider is label smoothing [25]. The key difference is the construction of the ground truth label, as cross-entropy loss is still used. Label smoothing introduces a single parameter $\alpha$ which defines how much weight is distributed across the incorrect classes in the label formulation. The smoothed label vector $y^{LS}$ is constructed from the original one-hot label vector $y$ as

$$y_c^{LS} = y_c(1-\alpha) + \alpha/C. \tag{4}$$

For example, if we have a 4-class classification problem with $\alpha = 0.1$ and one-hot ground truth label vector $y = [0,0,1,0]$, the smoothed label is $y^{LS} = [0.025, 0.025, 0.925, 0.025]$. Note, $y^{LS}$ still constitutes a valid probability distribution over the classes, and by training with this ground truth label the model learns to simultaneously maximize the predicted probability of the true class while uniformly minimizing the predicted probability of the false classes. In practice, label smoothing has shown to improve the generalization capability of DNNs through a regularization effect [18], while also improving the calibration of predictions [25], both of which are useful properties in our SAMPLE classification setting.

*c) Cosine loss:* Unlike the previous two variants, the cosine loss [26] does not use the cross-entropy function, however, it does have a direct interpretation from the ERM objective in (2) through defining $L$ as $L_{cos}$. The intuition for cosine loss is to encourage the model's output *vector* to have high cosine similarity with the ground truth one-hot label vector. This is as opposed to the cross-entropy loss which tries to predict a truth-encoded label distribution. The cosine loss we use is described as

$$L_{cos}(x, y; \theta) = 1 - \left\langle y \, , \frac{f(x; \theta)}{||f(x; \theta)||_2} \right\rangle. \tag{5}$$

Note, we consider the cosine loss because it has been empirically shown to improve generalization in extremely low data settings [26]. The authors believe this is because cosine-similarity as a loss metric is more informative than cross-entropy when there are very limited samples per class [26].

*d) Adversarial training:* The fourth loss function variant is called adversarial training (AT) [17]. Different from the previous three methods, AT reformulates the ERM objective in (2) into a saddle point problem through the introduction of a concept called *adversarial risk*. The AT objective is

$$\min_{\theta} \underset{(x,y)\sim\mathcal{D}}{\mathbb{E}} \left[ \max_{\delta\in\mathcal{S}} L_{xent}(x + \delta, y; \theta) \right]. \tag{6}$$

Note, the principal difference between (2) and (6) is the inclusion of an inner maximizer, which, given the current set of model parameters, maximizes the loss via perturbations to the input data $x$. Once the inner maximization is approximately solved, the network parameters are then updated to minimize this adversarially maximized loss (i.e., to minimize the adversarial risk). Here, $\delta$ is an image-domain perturbation and $\mathcal{S}$ is a set of allowable perturbations w.r.t. $x$. Commonly, $\mathcal{S}$ is defined as an $\ell_p$ norm-ball of radius $\epsilon$ centered at $x$ (so, $||\delta||_p \leq \epsilon$). This norm constraint ensures that the perturbed sample is sufficiently close to $x$ as to be adversarial. In practice, the inner maximization is approximately solved by an iterative projected gradient descent (PGD) adversarial attack [17].

Although AT is traditionally used for training models robust to adversarial attacks, in recent literature it has also been shown to prioritize the learning of "robust" features that are capable of maintaining correlation with the true label despite perturbation [27]. This is as opposed to standard models which may rely on "non-robust" features that are weakly correlated with the true label. In the context of SAR-ATR, [7] shows that performing AT on MSTAR models can improve the accuracy of the classifiers while also boosting robustness. In this work, we believe AT may improve performance at $K = 0$ because the synthetic and measured data share the "robust" features of the targets; whereas in contrast, non-AT models rely more heavily on nonrobust features such as clutter.

*e) Mixup:* The final training method is `mixup` [28], which defines a generic vicinal distribution on which to perform *empirical vicinal risk minimization* (VRM) [29] (as opposed to ERM). To define the vicinal distribution, `mixup` creates "virtual" feature-label samples. First, two feature-label pairs $(x_i, y_i)$ and $(x_j, y_j)$ are sampled from the training dataset $\mathcal{D}$. Then, `mixup` creates a virtual sample $(\tilde{x}, \tilde{y})$ as

$$\begin{aligned}\tilde{x} &= \lambda x_i + (1-\lambda)x_j \\ \tilde{y} &= \lambda y_i + (1-\lambda)y_j\end{aligned} \tag{7}$$

where $\lambda \sim \mathrm{Beta}(\alpha, \alpha)$ for the only introduced hyperparameter $\alpha \in (0, \infty)$ (which ensures that $\lambda \in [0, 1]$). Using these virtual samples, the network parameters are trained to minimize the expected cross-entropy loss between $f(\tilde{x}; \theta)$ and $\tilde{y}$ in a way very similar to (2). The key difference between VRM and ERM is that the risk is minimized using the virtual training samples as opposed to just using the samples in $\mathcal{D}$. The net effect is the model learns to have linear behavior between and around the training samples. In practice, `mixup` has shown to improve generalization performance on the ImageNet task and displays properties of a strong regularizer. As applied to the SAMPLE problem, the regularization effect may help to increase the bias (and thus reduce the variance) of a large DNN classifier applied to a relatively small dataset.

*Aggregation of Techniques:* As introduced, each of the aforementioned techniques alone has the potential to improve ATR performance in our setting, where there is a distribution gap between the training and test data. Data augmentations work to reduce over-fitting to noninformative image-level features of the training data. Model construction choices implicate the bias-variance-tradeoff through dictation of the hypothesis space to be searched. And, the definition of the training function directly effects the features learned by the model through redefinition of risk. However, we believe that aggregating techniques from across these methodological categories may yield the most substantial gains, as there is potential to attain *combined benefits*. To understand how each of the techniques fit into the training pipeline, see the pseudocode for the training loop in the supplemental materials.

As an example of combined benefits, consider that including data augmentation in conjunction with dropout layers may further reduce over-fitting to the synthetic data distribution by actively altering the image-level and intermediate-level representations during training. Similarly, using data augmentations with modified training objectives lends the potential to learn higher quality features of the underlying SAR targets than just including one of the techniques in isolation. Also, recognize the intimate relationship between model construction and training function -where changing the model assuredly changes the solution to the risk minimization, and changing the definition of risk certainly changes the final solution in terms of model weights. Finally, we posit that using a combination of techniques from all three categories may incur the greatest benefits (assuming proper configuration).

## B. Experimental Setup

Throughout the following experiments, we are primarily focused on the effect of each training modification individually, as well as their composition. To obtain a more true expectation of performance, for each of the tested configurations we perform 100 training runs, each starting from a random initialization of the 32-b model parameters. Performance is reported as the minimum, maximum, and average accuracy (with standard deviation) over the 100 iterations. Since the training data is 100% synthetic, there is no way to effectively perform early-stopping with a validation spit. To get an estimate of the impact early-stopping may have if a measured validation set were available, we also report a "Perfect Knowledge" accuracy (Perf), which represents a near upper-limit on the expected accuracy of each method if the trainer knew exactly when to stop to avoid over-fitting. For the shared training parameters over all configurations, we closely follow the setup from [2]. Each model is trained for 60 epochs, with a random initialization of parameters, fixed learning rate of 0.001, batch size of 128 and uses the ADAM optimizer. All SAR chips are in magnitude format and are distributed as 8-b JPEGs. Before input into the models, we first center crop to $64 \times 64$ pixels, then normalize to the range of $[-1, 1]$.

*Hyperparameter Selection:* Most of the techniques we introduce involve a single hyper-parameter that may be set or tuned. During testing, for each parameter we perform a sweep over a range of reasonable values and report the top performing result. For the Gaussian noise augmentation, we search in range $\sigma = [0.1, 0.2, 0.3, 0.4, 0.5]$, which covers the spectrum of "barely visible" to "obvious and potentially destructive." For the Dropout layers, we consider `p_drop` $=$ $[0.1, 0.2, 0.3, 0.4, 0.5]$, which mirrors the primary range and interval of test values used in [21]. For label smoothing, we test $\alpha = [0.06, 0.08, 0.10, 0.12]$, which represents a small range around the suggested value of $\alpha = 0.1$ that was used exclusively in [25]. For mixup, we search in $\alpha = [0.1, 0.4, 1, 4, 16]$, all of which are used in [28] (interestingly, we find that our results are not extremely sensitive to this parameter). For AT, we search in $\epsilon = [2/255, 4/255, 8/255]$ which is the suggested range found in [7] that leads to minimal performance degradation in standard settings and has significant robustness benefits. Uniquely, for the rotation augmentation, we use a fixed value of $r = 5$, which is considered a "safe" transform as the semantic label will be preserved [19] (note, rather than sweeping values of $r$, our primary interest is simply: does random rotation help or not?). Finally, when two or more techniques are combined, we perform a grid search over hyperparameter values to find the most productive setting of each. It is worth noting that in most cases the settings do not change when techniques are combined (e.g.,

TABLE II
SINGLE-STEP CHANGES TO TRAINING ALGORITHM ($K = 0$)

| Model | Method | Min | Max | Avg $\pm$ std | Perf |
|---|---|---|---|---|---|
| SMPL | baseline | 35.25 | 69.94 | 49.95 $\pm$ 5.59 | 77.55 |
| | rotation=5 | 36.73 | 67.71 | 49.44 $\pm$ 6.63 | 79.03 |
| | gaus=0.3 | 63.63 | 86.08 | 77.44 $\pm$ 4.58 | 87.94 |
| | dropout=0.4 | 45.08 | 84.41 | 65.61 $\pm$ 9.16 | 87.38 |
| | lblsm=0.1 | 41.37 | 79.59 | 52.14 $\pm$ 5.15 | 83.11 |
| | mixup=1 | 39.88 | 76.80 | 59.77 $\pm$ 8.18 | 78.66 |
| | cosine_loss | 41.00 | 76.80 | 54.95 $\pm$ 7.29 | 79.40 |
| | AT ($\epsilon = 8$) | 67.71 | 86.08 | **78.14** $\pm$ 3.74 | 88.31 |
| RN18 | baseline | 39.70 | 81.44 | 66.45 $\pm$ 7.26 | 81.81 |
| | rotation=5 | 42.67 | 82.93 | 66.15 $\pm$ 7.68 | 84.78 |
| | gaus=0.4 | 72.35 | 94.43 | **85.15** $\pm$ 4.81 | 97.03 |
| | dropout=0.4 | 62.52 | 89.42 | 81.10 $\pm$ 5.07 | 90.53 |
| | lblsm=0.08 | 74.39 | 86.82 | 81.22 $\pm$ 2.55 | 87.94 |
| | mixup=16 | 64.00 | 87.01 | 79.93 $\pm$ 4.09 | 87.57 |
| | cosine_loss | 69.38 | 87.19 | 78.15 $\pm$ 3.66 | 87.57 |
| | AT ($\epsilon = 8$) | 73.65 | 88.86 | 84.63 $\pm$ 2.65 | 93.50 |
| WRN18 | baseline | 44.89 | 78.29 | 65.10 $\pm$ 7.69 | 83.67 |
| | rotation=5 | 48.05 | 79.22 | 63.90 $\pm$ 7.25 | 83.85 |
| | gaus=0.3 | 58.62 | 91.46 | 82.42 $\pm$ 5.59 | 94.62 |
| | dropout=0.4 | 28.94 | 87.19 | 79.56 $\pm$ 7.06 | 89.23 |
| | lblsm=0.08 | 75.51 | 87.75 | 82.65 $\pm$ 2.44 | 89.05 |
| | mixup=16 | 69.94 | 86.82 | 80.20 $\pm$ 3.40 | 89.79 |
| | cosine_loss | 47.49 | 87.19 | 79.28 $\pm$ 5.11 | 87.19 |
| | AT ($\epsilon = 8$) | 76.06 | 88.49 | **83.49** $\pm$ 2.61 | 92.20 |

The bold entries are the top performing algorithm configurations for each model type.

TABLE III
TWO-STEP CHANGES: GAUSSIAN NOISE + OTHER ($K = 0$)

| Model | Method | Min | Max | Avg $\pm$ std | Perf |
|---|---|---|---|---|---|
| SMPL | gaus=0.3 | 63.63 | 86.08 | 77.44 $\pm$ 4.58 | 87.94 |
| | gaus=0.3, drop=0.3 | 70.87 | 90.72 | 82.07 $\pm$ 4.14 | 92.02 |
| | gaus=0.3, lblsm=0.08 | 74.95 | 90.53 | 83.47 $\pm$ 3.38 | 92.02 |
| | gaus=0.3, mixup=0.1 | 68.46 | 91.28 | 80.57 $\pm$ 5.02 | 91.28 |
| | gaus=0.3, cosine_loss | 68.08 | 92.02 | 83.15 $\pm$ 4.25 | 92.76 |
| | gaus=0.2, AT ($\epsilon = 8$) | 80.89 | 92.76 | **87.28** $\pm$ 2.74 | 93.87 |
| RN18 | gaus=0.4 | 72.35 | 94.43 | 85.15 $\pm$ 4.81 | 97.03 |
| | gaus=0.4, drop=0.4 | 81.26 | 95.36 | **89.83** $\pm$ 3.00 | 96.28 |
| | gaus=0.4, lblsm=0.08 | 67.16 | 94.06 | 86.23 $\pm$ 4.76 | 96.84 |
| | gaus=0.4, mixup=0.1 | 74.58 | 92.02 | 85.29 $\pm$ 3.96 | 95.73 |
| | gaus=0.4, cosine_loss | 66.97 | 95.17 | 84.97 $\pm$ 4.59 | 95.36 |
| | gaus=0.3, AT ($\epsilon = 4$) | 82.93 | 93.13 | 88.90 $\pm$ 2.54 | 95.54 |
| WRN18 | gaus=0.3 | 58.62 | 91.46 | 82.42 $\pm$ 5.59 | 94.62 |
| | gaus=0.4, drop=0.4 | 80.51 | 94.61 | **88.79** $\pm$ 2.85 | 96.28 |
| | gaus=0.4, lblsm=0.08 | 76.06 | 94.99 | 86.11 $\pm$ 3.71 | 95.54 |
| | gaus=0.3, mixup=0.1 | 54.73 | 89.79 | 79.36 $\pm$ 7.04 | 93.87 |
| | gaus=0.3, cosine_loss | 68.27 | 92.02 | 83.13 $\pm$ 4.78 | 93.50 |
| | gaus=0.3, AT ($\epsilon = 4$) | 79.03 | 93.50 | 87.94 $\pm$ 3.14 | 96.84 |

The bold entries are the top performing algorithm configurations for each model type.

Gaussian noise and dropout levels are fairly consistent across experiments). One notable case in which the settings do change is when using AT with a Gaussian noise augmentation. This is likely because these techniques have a direct interaction with each-other, as both additively "perturb" the input data. For reproducibility, the code used to run these experiments can be found at: https://github.com/inkawhich/synthetic-to-measured-sar.

### C. Experimental Results

*1) Single-Step:* The first result of interest is to measure the impact of each of the proposed methodological modifications, individually (single-step). The *baseline* configuration uses no data augmentations, the SMPL model architecture with no dropout, and standard cross-entropy with one-hot labels objective. The results of the "single-step" experiments are reported in Table II. Notice the average accuracy of the baseline configuration is about 50%, which matches the $K = 0$ value in Fig. 4 and serves as the number to compare against when considering the other methods. Now, consider the impact of the data augmentation methods, i.e., rotation and Gaussian noising (gaus), on average accuracy. Rotation by $\pm 5^\circ$ harms performance by about 1%. However, Gaussian noising has a tremendously positive impact on generalization, and boosts the average accuracy by over 21% across the three model types.

Next, consider the implications of changing the model construction. Recall, such modifications add dropout layers and/or change the whole DNN architecture. The effect of adding dropout layers appears quite positive, and when configured with `p_drop`$= 40\%$, increases average accuracy by about 15% across the three models. We also notice a sizable benefit to using a more complex model architecture than the SMPL model. With no augmentation and the standard loss function, using RN18 and WRN18 improves accuracy by 16.5% and 15.1%, respectively. Notice, however, that the RN18 outperforms WRN18, which suggests that increasing model complexity does *not* always improve performance.

Finally, we acknowledge the benefits of training function choice, between: cross-entropy with one-hot labels (baseline), label smoothing (lblsm), cosine loss, mixup, and AT. Importantly, in all cases a modified training function advances the performance over the baseline. On average, there is an improvement of 11.5% with label smoothing, 12.8% with mixup, 10.2% with cosine loss, and 21.5% with AT. Overall, the best average accuracy reported in Table II is 85.15% from the RN18 model trained with Gaussian noise, which is a 35% boost over the previous baseline. Our other takeaways are that RN18 is generally the top performing architecture, and Gaussian noise augmentation and AT are the most effective single step techniques given a fixed architecture.

*2) Two-Step:* Next, we incrementally compose methods from two different methodological categories (two-step). Given the effectiveness of Gaussian noise augmentation alone, in Table III we provide results for combining it with individual methods from the model construction and training function categories. We also remark that there is no increase in training time when using Gaussian noise augmentation. In fact, of the methods we consider the only two which yield a discernible training time increase is using more complex DNN architectures (i.e., RN18 and WRN18) and performing AT.

With two exceptions (cosine loss on RN18 and mixup on WRN18), composing Gaussian noise augmentation with either a model construction or training function change is consistently beneficial. The average accuracy improvements across the model architectures are 5.2% with dropout, 3.6% with label smoothing, 0.1% with mixup, 2.1% with cosine loss, and 6.4% with AT. The best overall configuration is the RN18 model with Gaussian noise augmentation and dropout, which has an average test

TABLE IV
THREE-STEP CHANGES: GAUSSIAN NOISE + DROPOUT + OTHER ($K = 0$)

| Model | Method | Min | Max | Avg ± std | Perf |
|---|---|---|---|---|---|
| SMPL | gaus=0.3, drop=0.3 | 70.87 | 90.72 | 82.07 ± 4.14 | 92.02 |
| | gaus=0.3, drop=0.3, lblsm=0.08 | 76.43 | 93.50 | 86.69 ± 3.41 | 94.62 |
| | gaus=0.3, drop=0.3, mixup=0.1 | 65.86 | 91.83 | 84.16 ± 4.31 | 92.57 |
| | gaus=0.3, drop=0.3, cosine_loss | 61.03 | 92.20 | 83.34 ± 5.10 | 94.62 |
| | gaus=0.2, drop=0.2, AT ($\epsilon = 8$) | 81.07 | 93.69 | **88.51** ± 2.68 | 94.80 |
| RN18 | gaus=0.4, drop=0.4 | 81.26 | 95.36 | 89.83 ± 3.00 | 96.28 |
| | gaus=0.4, drop=0.4, lblsm=0.1 | 84.97 | 95.54 | **91.87** ± 2.17 | 96.47 |
| | gaus=0.4, drop=0.4, mixup=0.1 | 84.41 | 96.84 | 90.79 ± 2.64 | 96.84 |
| | gaus=0.4, drop=0.4, cosine_loss | 82.93 | 95.73 | 90.41 ± 2.66 | 96.10 |
| | gaus=0.3, drop=0.3, AT ($\epsilon = 2$) | 83.11 | 94.80 | 90.22 ± 2.63 | 96.66 |
| WRN18 | gaus=0.4, drop=0.4 | 80.51 | 94.61 | 88.79 ± 2.85 | 96.28 |
| | gaus=0.4, drop=0.4, lblsm=0.08 | 84.97 | 95.91 | **91.05** ± 2.41 | 97.03 |
| | gaus=0.4, drop=0.4, mixup=0.1 | 79.40 | 97.21 | 90.14 ± 3.21 | 97.40 |
| | gaus=0.4, drop=0.4, cosine_loss | 79.77 | 95.36 | 90.01 ± 2.92 | 96.84 |
| | gaus=0.3, drop=0.3, AT ($\epsilon = 2$) | 81.44 | 95.36 | 90.43 ± 2.81 | 96.28 |

The bold entries are the top performing algorithm configurations for each model type.

accuracy of 89.83%. If we compare to the original baseline accuracy of 50%, this two-step change produces an improvement of nearly 40%.

*3) Three-Step:* Now, we combine methods from all three methodological categories (three-step). From Table III, Gaussian noising used with dropout layers is the most productive two-step change. In Table IV, we show models with Gaussian noising, dropout layers, and a training function modification.

Similar to previous results, modifying the training function always improves the average accuracy over the model trained with cross-entropy with one-hot labels. Across model architectures, the accuracy improvements from each of the different training methods are: 3% with label smoothing, 1.5% with mixup, 1% with cosine loss, and 2.8% with AT. The best performing configuration is the RN18 trained with Gaussian noise, dropout, and label smoothing, which has an average test accuracy 91.87%.

Though we have mainly focused on average accuracy, also notice the evolution of Min and Max values across Tables II–IV. These values have increased significantly, to the point where the worst performing three-step models are very competitive with the best performing baseline and one-step models. Also, the standard deviation of accuracy has decreased to about 3%, which indicates that the random parameter initialization is less influential on final accuracy when using three-step changes. Interestingly, even with perfect knowledge (Perf) of when to do early-stopping, the best model in Table IV is only 97.4% accurate, showing that perfect accuracy on the test set has not been attained even with our methods. Finally, we do not ignore the fact that AT has always been a top competing method. Throughout these experiments, it has consistently produced highly accurate models and is the best method when using the SMPL architecture.

*4) Ensembles:* To this point, we have focused on training highly accurate individual models. One way to boost accuracy is to create an *ensemble*. We leverage a technique called model "bagging" to achieve less variance (recall the bias-variance tradeoff discussion) by deriving a prediction from a collection of independently trained models [18]. To produce the final prediction, we use a soft-voting scheme which averages the predicted probability distributions from each of the ensemble components, then chooses the class with the highest average confidence. In this way, not all models in the ensemble have to be correct all the time in order for the aggregate prediction to be correct. To evaluate effectiveness, we incrementally build up to a five model ensemble through greedy selection of which model to include next (candidates for selection are all three-step models from Table IV). We start from the top performing individual model, and end with a five model ensemble composed of: [RN18-lblsm, RN18-mixup, RN18-AT2, SMPL-AT8, RN18-AT8].

Results of the ensembling experiment are shown in Table V, where the min/max/avg/std statistics are computed over 50 trials. We see a consistent benefit in both average accuracy and minimum accuracy as we add up to five models. The standard deviation also decreases to below 1%, indicating a reduced reliance on the random initialization of parameters (compare this to the error bars shown for $K = 0$ in Fig. 4). Finally, notice that with a four model ensemble we cross the 95% accuracy threshold on the measured data test set, even though we are still using exclusively synthetic training data. We view this as a significant achievement, considering the baseline accuracy is only 50% under these same conditions.

Fig. 5 shows the average confusion matrix on the (measured) test dataset for the five model ensemble. Note, the patterns of

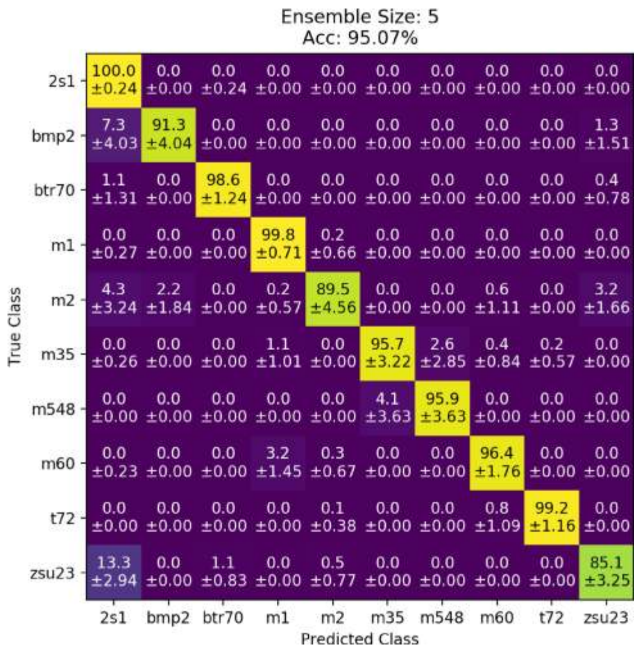

Fig. 5. Confusion matrix of best five model ensemble.

TABLE V
PERFORMANCE OF ENSEMBLING $K = 0$ MODELS. ALL MODELS ALSO INCLUDE GAUSSIAN NOISE + DROPOUT

| # Models | Ensemble Components | Min | Max | Avg ± std |
|---|---|---|---|---|
| 1 | RN18-lblsm=0.1 | 83.48 | 95.91 | 91.74 ± 2.23 |
| 2 | + RN18-mixup=0.1 | 88.68 | 96.66 | 93.26 ± 1.71 |
| 3 | + RN18-AT ($\epsilon = 2$) | 91.09 | 96.28 | 94.02 ± 1.34 |
| 4 | + SMPL-AT ($\epsilon = 8$) | 93.32 | 96.66 | 95.04 ± 0.93 |
| 5 | + RN18-AT ($\epsilon = 8$) | 93.32 | 96.66 | **95.06** ± 0.87 |

The bold entries are the top performing algorithm configurations for each model type.

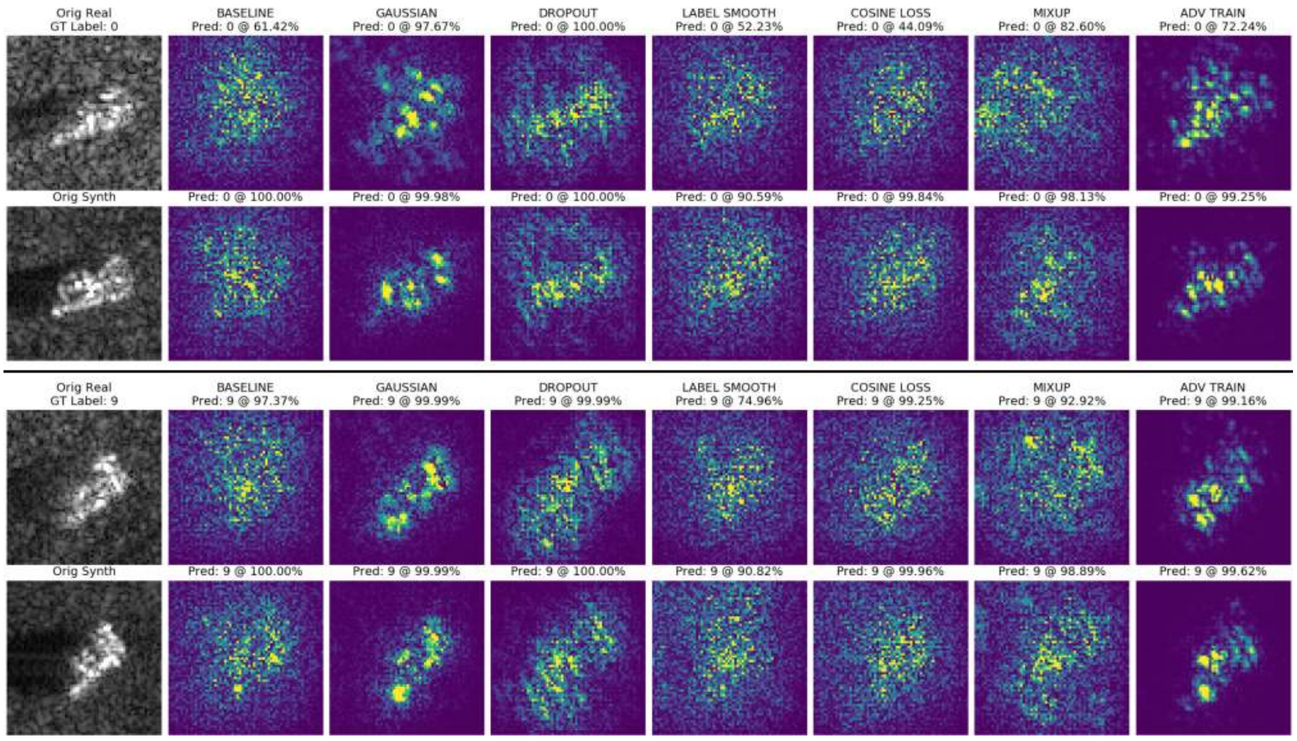

Fig. 6. Saliency maps of models trained with different techniques.

confusion are a natural consequence of the training procedure, and not hand-designed in any way. There are a few significant patterns to note: the BMP2 class is commonly confused with 2S1, the M2 class has confusions with several other classes, and the ZSU23 class data are often mispredicted as 2S1. A future work is to design models for the ensemble that compensate for these weaknesses.

## D. Analysis

*1) Saliency:* The first way to analyze how/why our proposed modifications work well for the SAMPLE $K = 0$ case is to examine which parts of the input data each model relies on to make its prediction. For DL models, we can produce a saliency map for a given input image using the gradient of the classification loss w.r.t. the input. Intuitively, the magnitude of the gradient describes which parts of the input are most influential in the classification decision [30]. To measure saliency for a given model and input SAR chip, we use the SmoothGrad [31] method.

Fig. 6 shows saliency for two (measured, synthetic) test pairs (4 total input images), for models trained with single-step improvements using the RN18 architecture. The top pair corresponds to a 2S1 target chip (label=0), and the bottom pair corresponds to a ZSU23 target chip (label=9). All models produced correct predictions and the gradients are computed w.r.t. the true class labels. First, observe the difference in saliency across any of the four rows (i.e., across the single-step improvement methods). The baseline, dropout, label smoothed, cosine loss, and mixup models all yield scattered nonlocalized salient regions that include background and target information. On the other hand, the Gaussian noise and adversarially trained models have salient regions focused on the actual target signatures, and very little on the background clutter regions. Intuitively, we believe that the ability of the Gaussian noised and adversarially trained models to predict based almost exclusively on features of the target signatures is a large contributing factor to why they are the top performing methods.

Next, consider the difference in saliency between each (measured, synthetic) pair. The models used to generate these saliency maps have all been trained on synthetic data only, so saliency w.r.t. the real (measured) data are a generalization test of learned features. Of all the methods, the baseline and mixup models appear to have the most significant difference in saliency across the pairs. Differently, the Gaussian and adversarially trained models use similarly localized salient regions for both versions of the data. This indicates that these models in particular have learned a transferable representation of the data, and can recognize the same salient features in either the measured or synthetic version of the chips. Saliency analysis should be a very important consideration when training SAMPLE models (especially at $K = 0$), because it elucidates if the models are leveraging critical features of the target signatures, or have memorized characteristics of the background clutter regions.

*2) Feature Space:* Another way to analyze the impacts of our proposed training modifications is to monitor how the measured and synthetic data is treated in the feature space of the DNNs. Intuitively, if the model has learned a robust and transferable representation of the data, the intermediate representations should be homogeneous and there will not be large differences in how the representations are processed. To analyze the feature

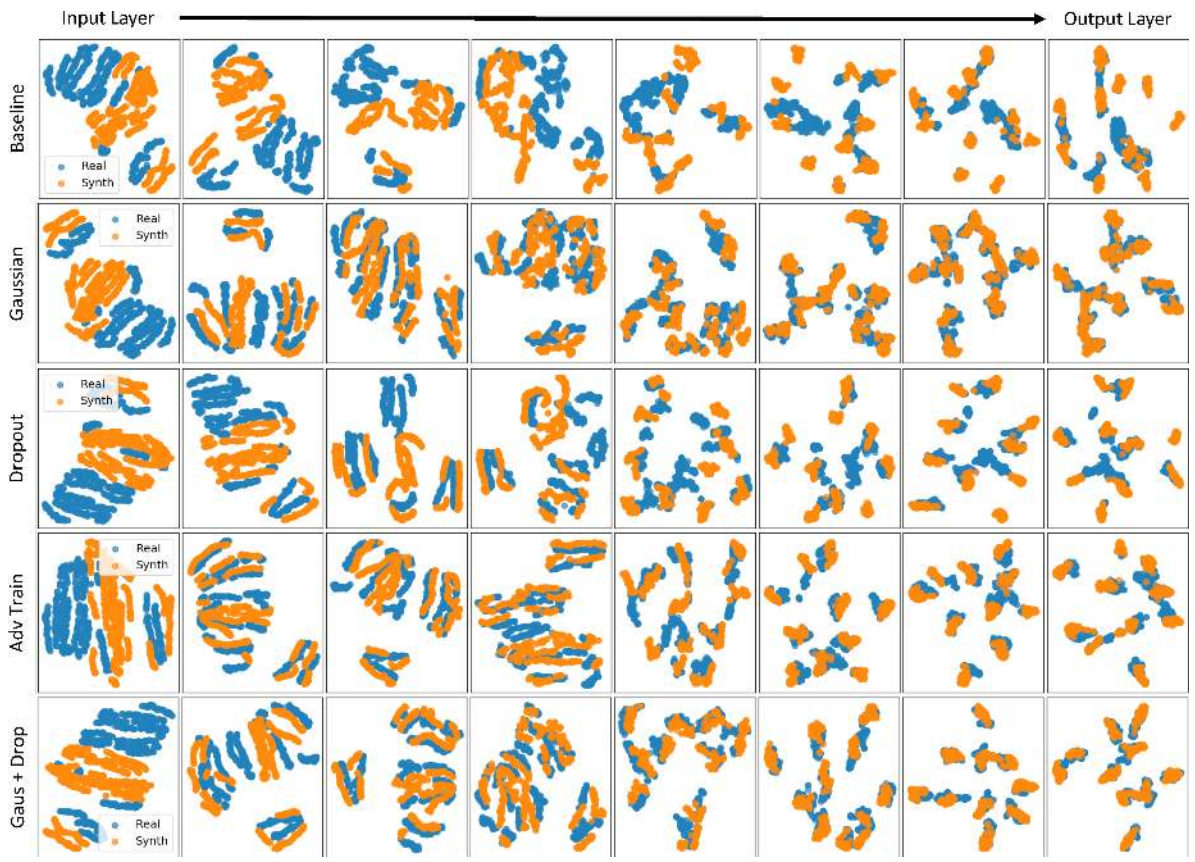

Fig. 7. Low dimensional analysis of `sample` model's feature space using t-SNE embeddings.

space, we start with a pretrained SMPL model. Recall, this architecture uses a total of eight layers, four convolutional, and four fully connected. We insert probes into the model to capture the output feature maps from each of these eight layers. We then input both the real and synthetic versions of the test dataset through the probed model, save the intermediate representations for each SAR chip, then perform a t-SNE low-dimensional embedding [32] of the feature maps to visualize the structure of each layer's feature space in 2-D.

Fig. 7 shows the t-SNE plots for five SMPL models trained with: baseline, Gaussian noise augmentation, dropout, AT, and Gaussian noise augmentation with dropout. From Tables II and III, the baseline and dropout models are the lowest performing in terms of average accuracy, and the others are all quite high performing in comparison. The left-most column shows the input layer's feature space (i.e., the first convolutional layer); moving left to right indicates increasing in depth by following the forward propagation of the data through the model; and the rightmost column shows the output layer's feature space (i.e., the final fully connected layer from which the predictions are derived). Notice the (approximately) ten clusters in the output layer feature space, which correspond to the ten classes in the SAMPLE dataset.

The first core observation from Fig. 7 is that the worst performing models (baseline and dropout) keep largely separated and nonoverlapping representations of the real and synthetic data throughout the feature space. This is an intuitive reason for why the accuracy is low, because at the output layer the measured test data does not respect the ten class clusters that the model's decision boundaries separate. The second core observation is that the high performing models (Gaussian, Adv Train, Gaus+Drop) integrate the feature space of the real and synthetic data which explains the accuracy improvement. In these models, the real and synthetic data share the same class clusters at the output layer, so the decision boundaries that were formed for the synthetic data are also valid for the measured data. The integration of features also alludes to the learning of a transferable and nonoverfit representation of the synthetic data, which means that the impact of the distribution gap discussed in Section II-B has been minimized. An interesting note is how in all models, the feature space at the input layer is largely separate, which may be due to the differing image-level statistics of the datasets.

## IV. Detecting Out-of-Library Confusers

The second key performance target within our "open-world" assumption is to reliably detect and reject test samples that are not from any of the training classes. This constitutes an OOD detection problem, and is critical because the model's output on such OOD data are inherently misleading. In the first experiment, we showed methodologies for training highly accurate models in the $K = 0$ case, but detecting OOD test samples may be just as important, especially if the consequences of making erroneous predictions are severe.

Historically, OOD detection is a very challenging task for DNN models, as they are wired to indiscriminately provide outputs for any data that are formatted as input. In our case, the models will produce a predicted probability distribution over the SAMPLE classes for any $64 \times 64$ pixel input image with values in [0,1]. In this work, we follow Experiment 4.3 from [2], which considers evaluating SAMPLE models under an open-world assumption while varying the ratio of synthetic and measured training data with $K$ (note, we are still focused on $K = 0$ in this context). To manufacture an OOD detection problem, [2] considers holding out a random subset of $J$ classes from the 10 classes worth of training data. The remaining $10 - J$ classes are used to train the model, and test data from these classes are considered in-distribution (ID). During testing, data from the selected $J$ classes are treated as OOD, and rather than producing a correct classification, the goal is to detect and reject them.

## A. Detection Methodology

A primary hindrance in OOD detection is that we cannot simply train the model with an additional class to represent OOD data, as we do not assume access to such data for training. Rather, we only have training data from the ID set, so we must intelligently leverage any signals available from the ID-only trained classifier to predict if a sample is OOD. Specifically, the process we consider for detecting OOD samples at inference-time is as follows. First, a classifier is trained on the $10 - J$ ID classes. Then, the measured test data from all 10 SAMPLE classes is input into the model producing tentative predictions. Leveraging some internal signal from the classifier, an "OOD score" is produced for each input sample. If the score is above some threshold, the data are deemed OOD and the model abstains from outputting any prediction. If the score is below the threshold, the sample is deemed ID and the model outputs its prediction.

We consider two primary methods for producing an "OOD score." The first is called the Softmax Thresholding Baseline (Softmax-Thresh) and derives an OOD prediction based on the confidence level for the predicted class [33]. The key observation is that OOD samples tend to have lower confidence predictions than ID test samples. The second method is called the Mahalanobis-distance detector (Mahalanobis) [34], which is slightly more complex and works in two steps. First, after the ID classifier is trained, the Mahalanobis detector models the feature space at the penultimate layer of the classifier with class-conditional Gaussian distributions, estimated over the training dataset. Then, at test time the OOD score is computed as the proximity of the test sample's representation to the nearest class-conditional distribution, as measured by Mahalanobis distance. ID test samples tend to be much closer to these modeled distributions than OOD samples, which is the primary intuition for this style of detection. Note, both detectors produce a real-valued OOD score which is then compared to some threshold. In operation, one would set this threshold based on the system's tolerance for error.

## B. Detection Results

To quantify the OOD detection performance of our ATR models, we measure the ability of the detectors to efficiently separate the "OOD scores" for the ID and OOD test splits as we sweep $J$ from 1 to 8. We use the Area Under the Receiver Operating Characteristic Curve (AUROC) and the True Negative Rate at a threshold computed to achieve a 95% True Positive Rate (TNR@95TPR) [34] as metrics of performance. AUROC is a threshold independent performance metric which quantifies the detectors ability to distinguish ID from OOD data by assessing the tradeoff between true positive rate and false positive rates across threshold values. TNR@95TPR measures a situation where a fixed OOD score threshold is selected to achieve a 95% true positive rate to meet some stated performance criteria, and we are thus interested in the true negative rate at this threshold value. Note, the possible values of both metrics lie in the range [0,1] where the higher the value, the better the detection ability.

We consider six total classifier models for use with the detectors, all of which use the SMPL architecture and whose naming convention is carried over from Section III. The first is the baseline model (i.e., no augmentation and cross-entropy with one-hot label loss function) trained at $K = 1$ (`baseline (k=1)`), which represents an upper-level of performance because there is no distribution gap between the training and test sets. The second model is a baseline configuration trained at $K = 0$ (`baseline (k=0)`), which represents the lower-level of performance starting point. The remaining four models, `gaus (k=0)`, `gaus+drop (k=0)`, `gaus+drop+lblsm (k=0)`, `gaus+drop+AT (k=0)`, incrementally add Gaussian noise (gaus), dropout layers (drop), label smoothing (lblsm), and AT, which we previously showed to improve expected accuracy. For each model, at each value of $J$, we perform 100 experiments with randomly chosen samples of $J$ classes and report average detection performance.

Fig. 8 shows the detection performance of both the Softmax-Thresh and Mahalanobis detectors versus the number of hold-out classes. As a general rule, the `baseline (k=1)` models have the highest detection performance, which is not surprising given that they do not have to contend with the distribution gap between the synthetic and measured data. These models have inherently learned high-quality features of the measured data during training, which evidently makes them better equipped to detect measured OOD samples. On the other hand, the `baseline (k=0)` models tend to be the lowest performing, which is also intuitive as these models have quite low accuracy performance in relation to the others.

Consider the left two subplots, which show the AUROC vs. $J$ for both detection methods. For all models, the Mahalanobis detector is superior to the Softmax-Thresh detector in terms of AUROC. Our top performing `gaus+drop+AT (k=0)` model achieves nearly 0.7 AUROC, which is a $\sim 10\%$ absolute improvement over the `baseline (k=0)` model across all values of $J$. However, this same AT model under-performs the `baseline (k=1)` model by nearly 20% AUROC. Next, consider the right two subplots, which show the TNR@95TPR vs. $J$ for both detection methods. A similar trend appears where the

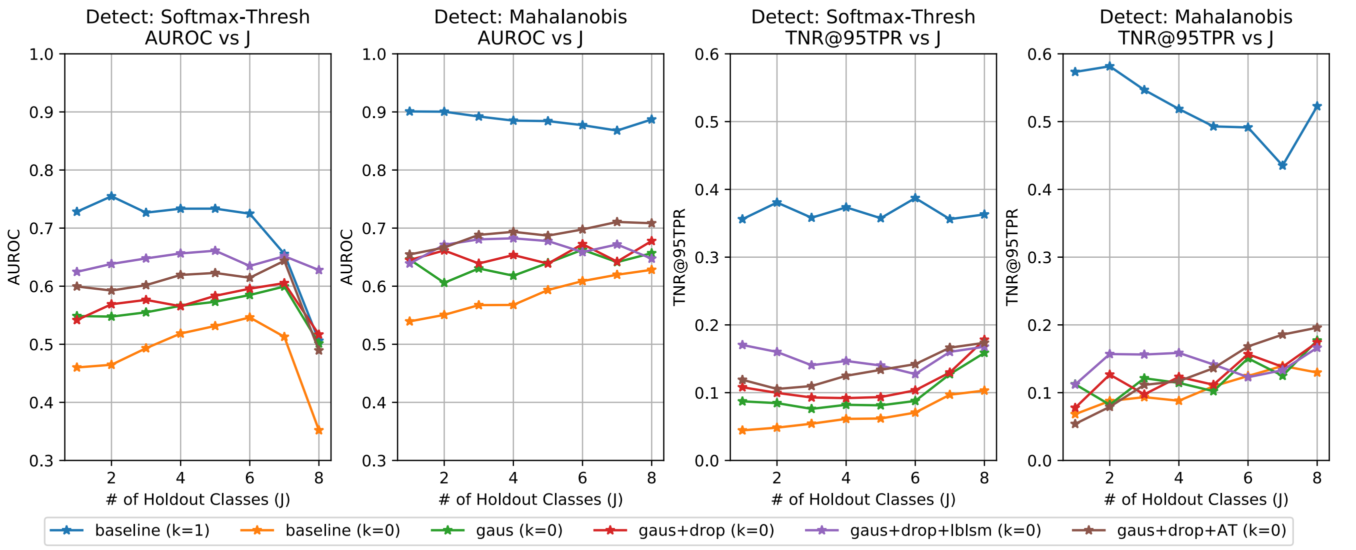

Fig. 8. OOD detection performance.

Mahalanobis detector outperforms the Softmax-Thresh detector in general. In this metric, our top performing `gaus+drop+AT (k=0)` and `gaus+drop+lblsm (k=0)` models achieve between 0.15 and 0.2 TNR@95TPR, which is a gain of less than 10% over the `baseline (k=0)` model, yet a drop of over 30% from the `baseline (k=1)` model. Even more so than in the AUROC tests, the `baseline (k=1)` model vastly outperforms all of the $K = 0$ models in terms of TNR@95TPR.

Overall, we pose two main take-aways from these OOD experiments. First, our training enhancements, including data augmentation, model construction, and training function choice, do in-fact improve the OOD detection performance of $K = 0$ trained models. We attribute this gain to the sizeable accuracy improvement yielded by our methods, which is indicative of a more transferable and robust learned representation of the ID classes. However, our second take-away is that despite such a large gain in accuracy, our improved $K = 0$ models still fall short of models trained with $K = 1$. This motivates future work to co-design classification and OOD detection systems for SAR-ATR.

## V. Conclusion

The cost and complexity of collecting large training datasets of real/measured SAR data for every ATR task-of-interest may be prohibitive in many cases, especially if rapid development is required. Thus, the goal of training ATR models on purely simulated SAR data, for use in measured data deployment environments, is both an important and significant challenge. To this point, achieving high-performing models with only synthetic training data has proven difficult, as there exists a distribution gap between the training (synthetic) and testing (measured) data distributions due to difficulties in perfectly simulating SAR targets. Such distributional differences have caused DL-based ATR models to over-fit to the unique properties of the synthetic distribution, yielding minimal generalization to the measured distribution.

In this work, we develop advanced training procedures for DL-based SAR-ATR models using 100% synthetic training datasets, for use in an “open-world” operating environment. To boost generalization and improve the quality of learned features, we consider composing techniques from the categories of data augmentation, model construction, training function choice, and model ensembling. Our models ultimately achieve over 95% accuracy on within-library test data, which is a near 45% improvement over baseline methods. For the problem of detecting out-of-library confusers, our more accurate models also lead to a near 10% improvement in detection ability. To explain these improvements, through analysis we find that models trained with our methods learn representations that rely almost exclusively on salient features of the targets, while also creating a homogeneous mixture of the measured and synthetic data in feature space. Overall, our methods have achieved state-of-the-art performance when only using synthetic training data, while also motivating a codesign for accuracy and out-of-library detection in future works.

*Discussion:* As a final discussion, we would like to note that a key assumption made throughout this work is the availability of high-quality CAD models for the targets of interest (as used in the generation of the SAMPLE dataset). Such models have been developed through significant effort, to reflect the physical and electrical properties of the corresponding targets in the MSTAR collection. The SAR response of these targets has then been simulated with very high quality electromagnetic modeling. This high degree of knowledge about the targets to be simulated, and quality of the simulation, increases the complexity and cost of producing simulated copies of future targets, but critically enables training on targets that might otherwise be completely unavailable. As a future work, it would be prudent to better understand how the overall quality of the simulation impacts the effectiveness of our proposed training framework. Along this track, one may consider a variety of onerous training conditions, where it is specifically measured how inaccuracies in the synthetic data generation process affects the ATR results on the measured data. One way to do this may be to purposely remove/modify details of the targets (e.g., remove the barrel of the T-72 tank) in the CAD modeling software and retrain the ATR models on such a tainted dataset [35]. Another interesting future work is to purposely modify details of the synthetic targets as a form of domain-relevant data augmentation, to potentially improve the performance of the ATR models through creation of a more diverse training dataset. Finally, we specify that our intention is for this work to outline a framework of techniques that can be used when synthetic training data are available, and we do not intend that our final parameter settings are universal to all ATR tasks.

## Acknowledgment

The views expressed in this article are those of the authors and do not reflect official policy of the United States Air Force, Department of Defense or the U.S. Government. Public release number: 88ABW-2020-3313.

## References

[1] U. K. Majumder, E. P. Blasch, and D. A. Garren, *Deep Learning for Radar and Communications Automatic Target Recognition*. Norwood, MA, USA: Artech House, 2020.

[2] B. Lewis, T. Scarnati, E. Sudkamp, J. Nehrbass, S. Rosencrantz, and E. Zelnio, "A SAR dataset for ATR development: The synthetic and measured paired labeled experiment (SAMPLE)," *Proc. SPIE*, vol. 10987, pp. 39–54, 2019.

[3] K. E. Dungan, C. Austin, J. Nehrbass, and L. C. Potter, "Civilian vehicle radar data domes," *Proc. SPIE*, vol. 7699, pp. 242–253, 2010.

[4] R. Deming, M. Best, and S. Farrell, "Polar format algorithm for SAR imaging with Matlab," *Proc. SPIE*, vol. 9093, pp. 47–66, 2014.

[5] J. Godwin, M. Moore, D. Waagen, D. Hulsey, and R. Conner, "Statistical analysis of SAR signature domains," *Proc. SPIE*, vol. 11393, pp. 125–138, 2020.

[6] T. Ross, S. Worrell, V. Velten, J. Mossing, and M. Bryant, "Standard SAR ATR evaluation experiments using the MSTAR public release data set," *Proc. SPIE*, vol. 3370, pp. 566–573, 1998.

[7] N. Inkawhich, E. Davis, U. Majumder, C. Capraro, and Y. Chen, "Advanced techniques for robust SAR ATR: Mitigating noise and phase errors," in *Proc. IEEE Int. Radar Conf.*, 2020, pp. 844–849.

[8] T. Scarnati and B. Lewis, "A deep learning approach to the synthetic and measured paired and labeled experiment (SAMPLE) challenge problem," *Proc. SPIE*, vol. 10987, pp. 29–38, 2019.

[9] M. Cha, A. Majumdar, H. T. Kung, and J. Barber, "Improving SAR automatic target recognition using simulated images under deep residual refinements," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, 2018, pp. 2606–2610.

[10] B. Lewis, J. Liu, and A. Wong, "Generative adversarial networks for SAR image realism," *Proc. SPIE*, vol. 10647, pp. 37–47, 2018.

[11] B. Lewis, O. DeGuchy, J. Sebastian, and J. Kaminski, "Realistic SAR data augmentation using machine learning techniques," *Proc. SPIE*, vol. 10987, pp. 12– 28, 2019.

[12] D. Malmgren-Hansen, A. Kusk, J. Dall, A. A. Nielsen, R. Engholm, and H. Skriver, "Improving SAR automatic target recognition models with transfer learning from simulated data," *IEEE Geosci. Remote Sens. Lett.*, vol. 14, no. 9, pp. 1484–1488, Sep. 2017.

[13] J. M. Arnold, L. J. Moore, and E. G. Zelnio, "Blending synthetic and measured data using transfer learning for synthetic aperture radar (SAR) target classification," *Proc. SPIE*, vol. 10647, pp. 48–57, 2018.

[14] C. Clum, D. G. Mixon, and T. Scarnati, "Matching component analysis for transfer learning," *SIAM J. Math. Data Sci.*, vol. 2, no. 2, pp. 309–334, 2020.

[15] S. R. Sellers, P. J. Collins, and J. A. Jackson, "Augmenting simulations for SAR ATR neural network training," in *Proc. IEEE Int. Radar Conf.*, 2020, pp. 309–314.

[16] B. Lewis, K. Cai, and C. Bullard, "Adversarial training on SAR images," *Proc. SPIE*, vol. 11394, pp. 83–90, 2020.

[17] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," in *Proc. Int. Conf. Learn. Representations*, 2018.

[18] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org.

[19] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *J. Big Data*, vol. 6, 2019, Art. no. 60.

[20] S. Gidaris, P. Singh, and N. Komodakis, "Unsupervised representation learning by predicting image rotations," in *Proc. Int. Conf. Learn. Representations*, 2018.

[21] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.

[22] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[23] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit*.Comput. Soc., 2016, pp. 770–778.

[24] S. Zagoruyko and N. Komodakis, "Wide residual networks," in *Proc. Brit. Mach. Vis. Conf.*, 2016, pp. 87.1–87.12.

[25] R. Möller, S. Kornblith, and G. E. Hinton, "When does label smoothing help?," in *Proc. Adv. Neural Inf. Process. Syst*, 2019, pp. 4696–4705.

[26] B. Barz and J. Denzler, "Deep learning on small datasets without pre-training using cosine loss," in *Proc. IEEE Winter Conf. Appl. Comput. Vis.*, 2020, pp. 1360–1369.

[27] D. Tsipras, S. Santurkar, L. Engstrom, A. Turner, and A. Madry, "Robustness may be at odds with accuracy," in *Proc. Int. Conf. Learn. Representations*, 2019.

[28] H. Zhang, M. Cissé, Y. N. Dauphin, and D. Lopez-Paz, "Mixup: Beyond empirical risk minimization," in *Proc. Int. Conf. Learn. Representations*, 2018.

[29] O. Chapelle, J. Weston, L. Bottou, and V. Vapnik, "Vicinal risk minimization," in *Proc. Adv. Neural Inf. Process. Syst.*, 2000, pp. 368–374.

[30] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," *Int. Conf. Learn. Representation*, 2014.

[31] D. Smilkov, N. Thorat, B. Kim, F. B. Viégas, and M. Wattenberg, "SmoothGrad: Removing noise by adding noise," 2017, *arXiv:1706.03825*.

[32] L. van der Maaten and G. Hinton, "Visualizing data using t-SNE," *J. Mach. Learn. Res.*, vol. 9, pp. 2579–2605, 2008.

[33] D. Hendrycks and K. Gimpel, "A baseline for detecting misclassified and out-of-distribution examples in neural networks," in *Proc. Int. Conf. Learn. Representations*, 2017.

[34] K. Lee, K. Lee, H. Lee, and J. Shin, "A simple unified framework for detecting out-of-distribution samples and adversarial attacks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018., pp. 7167–7177.

[35] C. Paulson, A. Nolan, S. Goley, S. Nehrbass, and E. Zelnio, "Articulation study for SAR ATR baseline algorithm," *Proc. SPIE*, vol. 10987, pp. 73–89, 2019.

**Nathan A. Inkawhich** received the B.S. degree in computer engineering from Clarkson University, Potsdam, NY, USA, in 2016. He is currently working toward the Ph.D. degree with the Electrical and Computer Engineering Department, Duke University, Durham, NC, USA, advised by Dr. Yiran Chen of the Computational Evolutionary Intelligence (CEI) Lab.

He was with the Air Force Research Laboratory Information Directorate (AFRL/RI) in Rome, NY, USA. His research interests include machine learning algorithms and security, deep learning, anomaly detection, and robust automatic target recognition.

**Matthew J. Inkawhich** received the B.S. degree in software engineering from Clarkson University, Potsdam, NY, USA, in 2017. He is currently working toward the Ph.D. degree in electrical and computer engineering with Duke University, Durham, NC, USA.

His research interests include building more robust and accurate convolutional network backbones for object detection models.

**Eric K. Davis** received the B.S. degree in electrical engineering from Northeastern University, Boston, MA, USA, in 2013 and the M.S. degree in electrical and computer engineering from the Georgia Institute of Technology, Atlanta, GA, USA, in 2017.

Since 2017, he has been with SRC, Inc., North Syracuse, NY, USA, where he is currently a Lead Systems Engineer on the Machine Learning and Artificial Intelligence Team. Prior to SRC, he was with the General Electric Company, Boston, MA, USA, developing embedded sensor systems for scientific and commercial applications. His current projects involve applications in machine intelligence, signal processing, and high-performance embedded computing.

**Uttam K. Majumder** (Senior Member, IEEE) received the B.S. degree (summa cum laude) in computer science from the City College of New York (CCNY), New York, NY, USA, in 2003, the M.S. degree in electrical engineering from Air Force Institute of Technology, Rome, OH, USA, in 2007, the MBA degree from Wright State University, Dayton, Ohio, in 2009, and the Ph.D. degree in electrical engineering from Purdue University, West Lafayette, IN, USA, in 2014.

He is a Senior Electronics Engineer with U.S. Air Force Research Laboratory (AFRL), Rome, OH, USA. He recently published a book on "*Deep Learning for Radar and Communications Automatic Target Recognition*." His research interests include artificial intelligence/machine learning (AI/ML), synthetic aperture radar (SAR) algorithms development for surveillance applications, radar waveforms design, and high performance computing for SAR-based automatic target recognition (ATR).

Among various awards, Dr. Majumder received AFOSR "STAR Team" Award, Air Force Distinguished Civilian Award, and AFRL Science and Technology Achievement Award for Radar Systems Development.

**Erin Tripp** received the B.S. degree in mathematics from the University of California, Santa Barbara, Santa Barbara, CA, USA, in 2013, the M.S. and Ph.D. degrees in mathematics from Syracuse University, Syracuse, NY, USA, in 2017 and 2019, respectively.

She is currently a Research Mathematician with the Air Force Research Laboratory Information Directorate, Rome, NY, USA, working in optimization theory with applications to signal and image processing and machine learning.

**Chris Capraro** received the B.S. degree in engineering physics from Rensselaer Polytechnic Institute in Hartford, CT, USA, in 1993, and the M.E. degree in electrical engineering in 2008 from the University of Illinois at Chicago in Chicago, IL, USA, in 2008.

He has over 25 years of experience in radar signal processing research and software development, and over 15 years of extensive experience in the program management of Government and commercial contracts. His experience includes neuromorphic computing, synthetic aperture radar (SAR), space-time adaptive processing (STAP), waveform design and selection, waveform diversity, high-performance computing, and software development. He is currently involved in developing a pod-based high-performance distributed embedded computer, developing convolutional and deep learning neural networks, and implementing topological processing algorithms.

**Yiran Chen** (Fellow, IEEE) received the B.S. and M.S. degrees in electrical engineering from Tsinghua University, Beijing, China, in 1998 and 2001, respectively, and the Ph.D. degree in electrical and computer engineering from Purdue University, West Lafayette, IN, USA, in 2005.

After five years in industry, he joined the University of Pittsburgh, Pittsburgh, PA, USA, in 2010 as an Assistant Professor and then promoted to an Associate Professor with tenure in 2014, held Bicentennial Alumni Faculty Fellow. He now is the Professor with the Department of Electrical and Computer Engineering, Duke University, Durham, NC, USA, and serving as the Director of NSF Industry-University Cooperative Research Center (IUCRC) for Alternative Sustainable and Intelligent Computing (ASIC) and the Co-Director of Duke University Center for Computational Evolutionary Intelligence (CEI), focusing on the research of new memory and storage systems, machine learning and neuromorphic computing, and mobile computing systems. He has published one book and more than 400 technical publications and has been granted 96 U.S. patents.

Dr. Chen serves or served as the Associate Editor of several IEEE and ACM transactions/journals and served on the technical and organization committees of more than 50 international conferences. He is now serving as the Editor-in-Chief of *IEEE Circuits and Systems Magazine*. He received seven best paper awards, one best poster award, and 15 best paper nominations from international conferences and workshops. He was the recipient of NSF CAREER Award, ACM SIGDA Outstanding New Faculty Award, the Humboldt Research Fellowship for Experienced Researchers, and the IEEE SYSC/CEDA TCCPS Mid-Career Award. He is the distinguished member of ACM and a distinguished Lecturer of IEEE CEDA.