SAND2021-3579C

Exploring Characteristics of Neural Network Architecture Computation for Enabling SAR ATR

Ryan Melzer[a], William M. Severa[a], Mark Plagge[a], and Craig M. Vineyard[a]

[a]Sandia National Laboratories, 1515 Eubank SE, Albuquerque, NM, United States

ABSTRACT

Neural network approaches have periodically been explored in the pursuit of high performing SAR ATR solutions. With deep neural networks (DNNs) now offering many state-of-the-art solutions to computer vision tasks, neural networks are once again being revisited for ATR processing. Here, we characterize and explore a suite of neural network architectural topologies. In doing so, we assess how different architectural approaches impact performance and consider the associated computational costs. This includes characterizing network depth, width, scale, connectivity patterns, as well as convolution layer optimizations. We have explored a suite of architectural topologies applied to both the canonical MSTAR dataset, as well as the more operationally realistic Synthetic and Measured Paired and Labeled Experiment (SAMPLE) dataset. The latter pairs high fidelity computational models of targets with actual measured SAR data. Effectively, this dataset offers the ability to train a DNN on simulated data and test the network performance on measured data. Not only does our in-depth architecture topology analysis offer insight into how different architectural approaches impact performance, but we also have trained DNNs attaining state-of-the-art performance on both datasets. Furthermore, beyond just accuracy, we also assess how efficiently an accelerator architecture executes these neural networks. Specifically, Using an analytical assessment tool, we forecast energy and latency for an edge TPU like architecture. Taken together, this tradespace exploration offers insight into the interplay of accuracy, energy, and latency for executing these networks.

Keywords: SAR, ATR, neural networks, CNN, MSTAR, SAMPLE, neuromorphic accelerators

1. INTRODUCTION

Synthetic aperture radar (SAR) automatic target recognition (ATR) systems attempt to localize and identify targets or other objects contained in SAR imagery. These systems are increasingly becoming key components in many military and civil applications.[1] Traditional SAR ATR methods generally consist of hand-designed pre-processing and feature extraction steps, statistical binning and quantization, and/or non-neural pattern recognition algorithms for feature classification, such as template matching.[2–4] With the rise of deep learning in computer vision over the last decade however, neural networks are again being visited for SAR ATR. Revisiting the use of neural network models for SAR ATR has already shown significant promise.[5–8] However, prominent and effective existing deep neural network algorithms and architectures have yet to be leveraged to the fullest extent possible for SAR ATR.

In this paper, we collect and assess the performance of a suite of high-performing neural network models from computer vision literature and apply them to SAR ATR. These models encompass various depths, widths, scales, input resolutions, connectivity patterns, and convolution layer optimizations. Accordingly, we explore the impact of these architectural approaches on SAR ATR classification performance. Furthermore, as SAR ATR systems are often run on resource constrained computational platforms, we also asses the computational cost of each model and how efficiently each may be executed on an Edge Tensor Processor Unit (TPU) like neural network accelerator architecture.[9] Figure 1 illustrates this overarching investigation in the work presented here.

Further author information: Send correspondence to Craig M. Vineyard (E-mail: cmviney@sandia.gov)

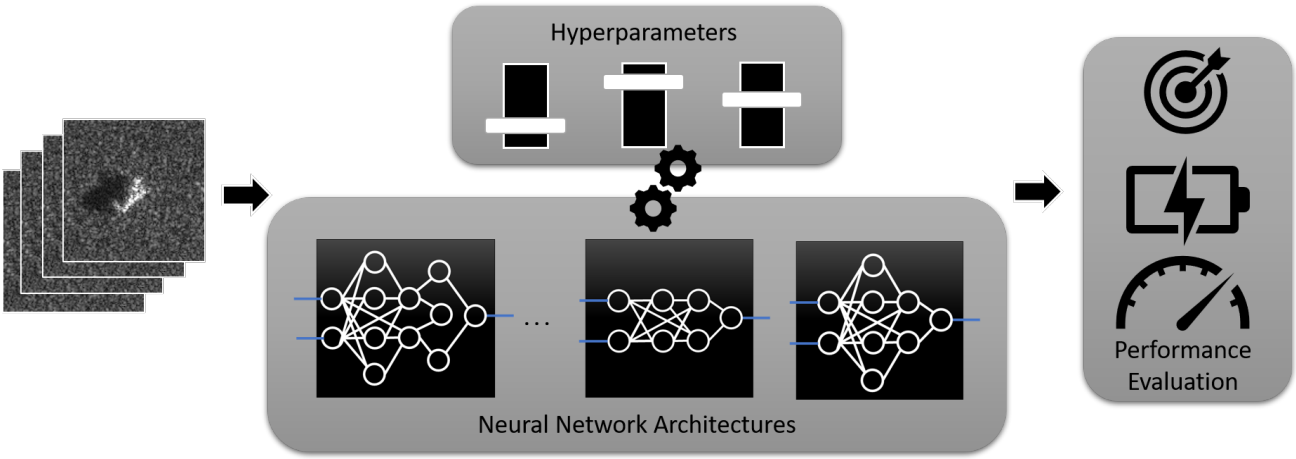

Figure 1. Overview of our exploration into characteristics of neural networks for enabling SAR ATR. This effort consists of assessing a breadth of different neural network architectures, tuning the parameters which maximize their impact, and then characterizing their accuracy, energy, and latency.

2. BACKGROUND

2.1 Synthetic Aperture Radar and Automated Target Recognition

SAR is a class of radar in which the motion of a mounted antenna on a moving vehicle is used to simulate a much larger aperture than what can be achieved with conventional radar, which is bound by the physical dimensions of the antenna. This "synthetic" aperture is used to create high-resolution imagery which can be captured not only at great distances, but also at any time of day and through atmospheric noise, fog, clouds, and storms. However, unlike natural optical imagery, the interpretation and understanding of SAR imagery is difficult and requires the use of specialists and algorithms to be understood and interpreted. Regardless, SAR is often used as a sensing modality for ATR, the task of using algorithms to locate and identify objects (targets) in sensor data. Historically, target recognition was done by human operators, but more recently computer vision algorithms have become effective replacements.

Prior to the great successes of deep learning for computer vision, much of the research on SAR ATR identification stage algorithms (what we refer to here as "traditional" methods) has consisted of building elaborate, hand-designed modular pipelines of components which include sub-stages of hand-designed image pre-processing steps, hand-designed feature extraction steps, potential further feature processing such as normalization and discrimination, and then finally classification. The purpose of this image pre-processing and subsequent feature extraction is to transform the input into a representation which has entries that are more separable, have less variation, and are more predictive of the input's class when fed to a classifier in place of the raw input. Dichotomies to differentiate approaches include broadly characterizing the representation and reasoning methods. This includes describing the traditional methods as model-based, feature-based, and knowledge-based.[2,10] There is some overlap in the dichotomies, but at large they differentiate approaches based upon how the methods transform an input.

Many approaches to feature extraction and pre-processing have been studied. Feature extraction approaches include techniques from computer vision such as corner and edge detection, smoothing, filtering, and masking, Bayesian methods such as probability distribution fitting and graphical models, modeling radar and radar properties, as well as signal processing techniques such as wavelet transforms. Pre-processing techniques include filtering and masking in an effort to separate vehicles from clutter and shadow, transforms to estimate target length and width, image and target translation, image and target scaling, as well as pose estimation algorithms and centering used to normalize input targets for pose. Classifiers such as support vector machines (SVMs), adaptive boosting algorithms, genetic algorithms, statistical classifiers, k-nearest-neighbors, and template matching have all been used with various properties and trade-offs associated with each.

2.2 Neural Networks and Deep Learning

Historically, neural network approaches have periodically been pursued for SAR ATR, however they have often been outperformed by traditional methods. Advances made by deep learning and deep neural networks (DNNs) in computer vision in recent years have made them a compelling approach for SAR ATR once again. With the discovery of DNN effectiveness, along with explosion of improved compute resources, open-source software, and subsequent renewed interest in the field, deep learning has seen an immense resurgence over the last decade with DNNs now currently providing the best existing solutions to many problems in a variety of data science and signal processing fields, including image classification and object recognition. Not only are models architecturally different, but mechanisms for training, inference, and data have been drastically improved and continue to improve at an unprecedented rate.

In this paper, we are focused on convolutional neural networks (CNNs). CNNs are a class of DNN architecture which make an explicit assumption that its inputs contain structured, spatially meaningful information. CNNs are made up of a hierarchy of operations (layers), most commonly convolution layers and pooling layers, potentially followed by fully connected layers similar to those found in a Multi-Layer Perceptron (MLP). In the case of a network being used as a classifier, a final fully connected layer is used as a linear classifier whose weights are trained jointly with the rest of the network, with a softmax function applied to its output units to compute class membership probabilities. Figure 2 depicts the general CNN computational structure.

A convolution layer consists of groups of n-dimensional volumes of neurons known as filters or kernels, rather than the one-dimensional vectors of neurons found in MLPs. A single (3D) kernel is a $K \times K \times C$ block of learnable weights (with an optional scalar bias) where K is the length and width of the block and C is the depth of the block, represented as the number of input channels to the kernel e.g. 3 in the case of an RGB image. A kernel is slid across an input with some stride length s and at each step, a 3D dot product is taken between the kernel weights and the $K \times K \times C$ section of the input that aligns with the kernels current position. Each dot product computes a scalar value that is aggregated into a 2D output channel whose length and width depends on s and K. Zero-padding on all sides of the input is often used to achieve desired resulting output dimensions. There is an output channel for each kernel, and these outputs are aggregated into one volume along the channel dimension. This output volume is then fed through a non-linear activation function and then into the next layer of the network, such as another convolution layer or a pooling layer. This aggregated 3D volume of output is generally referred to as a feature map. A single 2D slice of a feature map computed from one kernel is referred to as an activation map.

Pooling layers are non-parametric downsampling operations frequently applied after convolution layers. A pooling layer will slide a small window across an input and apply a downsampling function such as computing a max or average. This shrinks the dimensionality, improving computational burden. Additionally, these pooling operations may help CNNs to be more robust to small feature variations and translations.

Each learnable filter in a convolution layer creates a small receptive field which is applied over the entire volume of the input, the results of which are aggregated into a single activation map. Each entry in an activation map can be interpreted as the output of a single artificial neuron which looks at a small region of the input, where the neurons across an activation map share weights. This allows a given filter (or group of shared weight neurons under this interpretation) to respond to the same local feature across the entire input and aggregate the results into a 2D activation map. Repeating this for many filters and aggregating the results into a feature map volume allows for a large number of learned local features which are detected across the entire input. When these small receptive fields are applied across adjacent convolution layers, they form progressively larger and more complex features. This allows a CNN to learn representations of small regions of the input and use them to build representations of larger regions, e.g. from edges and corners in early layers, to shapes, to small objects, to larger objects, to entire scenes in late layers.

This feature learning and resulting classification affords a streamlined learning of every step in the traditional SAR ATR classification pipeline. This is a pattern seen in computer vision more broadly. That is, CNNs replacing pipelines of pre-processing, hand-designed features, and classifiers with end-to-end learned models. These end-to-end learned classifiers offer substantial benefits over traditional methods. In place of hand-designed stages of pre-processing and feature extraction followed by classification (the choice of which is usually disjoint

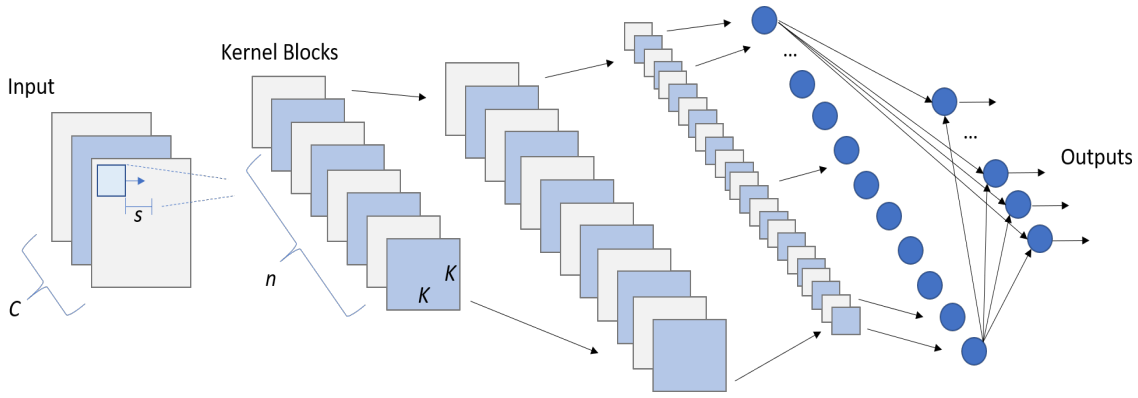

Figure 2. General depiction of the primary components of a CNN architecture. Inputs may comprise multiple channels and pass through a series of computational blocks which consists of convolution kernels and pooling operations leading to final fully connected layers yielding the network output. Variations in the structure of these components, as well as their connectivity and computational execution, are what yield architectural differences.

from feature design); an end-to-end CNN model can be learned that requires little to no pre-processing, can generalize to an arbitrary number of classes, and can be easily transferred to different problems and datasets.

CNNs have already matched state-of-the-art performances achieved by traditional methods, however current works have several shortcomings. First, many works utilize simple, hand crafted network architectures without much justification for the specific architecture choices made. These networks do not leverage state-of-the-art CNN topologies and techniques. Systematic methods of choosing maximally performing architecture configurations and hyperparameters are also not leveraged. Compute cost and efficiency are also not considered to the extent necessary for SAR ATR systems. Prior works often compare details such as optimizer choice or weight initialization choices rather than architecture topology choice, compute cost and efficiency, or ability to generalize from few examples or from synthetic to measured data. Since we are looking for the best models for SAR ATR systems, a promising alternative to current work is drawing from existing state-of-the-art work in image classification and object recognition to choose architectures, and optimizing both their topologies, parameters, and hyperparameters in a more systematic way, then comparing these architectures across axes of performance which are important for SAR ATR.

3. APPROACH

In this work, we characterize an array of CNN architectural approaches found in recent computer vision literature and apply them to SAR ATR. CNN research has extensively focused on maximizing performance in optical band imagery across several dimensions. These performance dimensions include generalization capability, model size, computational cost, parallelizability, training and inference speed, ease of implementation, and amounts of training data required for good generalization. Accordingly, we leverage those investigations to explore their impact when applied to the SAR ATR domain. We provide an overview summary of prominent features of the CNNs explored here, identifying why these CNN architectures are explored, and then discuss how their performance is assessed.

3.1 CNN Architectures

In our assessment, we have collected a suite of models and techniques from the computer vision literature which we believe to be illustrative of general trends and improvements in the field. The naming conventions are borrowed from the standard naming conventions found in the neural network literature. That is, the name of the architecture (e.g. DenseNet) followed by a size and/or version component (e.g. DenseNet264, MobileNet-V2, ShuffleNet-V2_2.0x). Though it is not chronological, we have attempted to order the descriptions in such a way that illustrates trends in CNN model improvements for computer vision over the last decade. For a more comprehensive description, as well as further architectural details of a model, consult the original papers. Figure 3 notionally illustrates many of these architectural explorations which we describe.

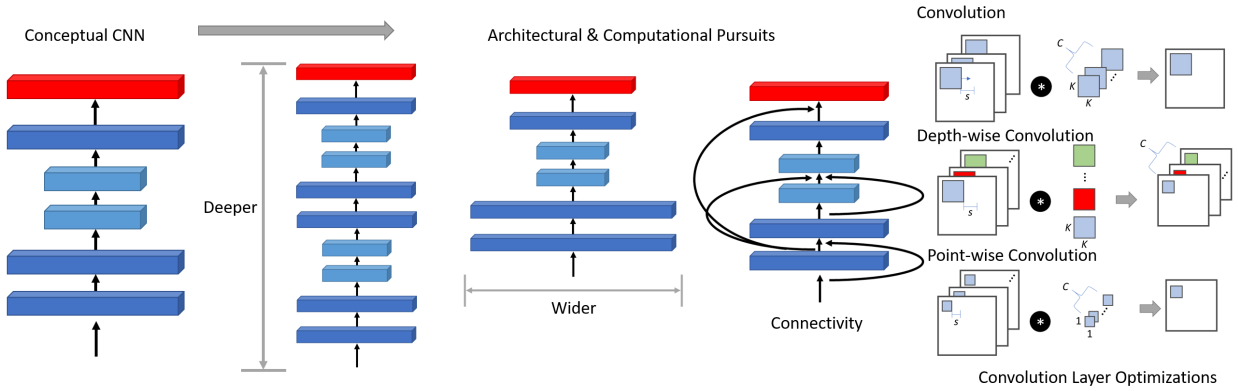

Figure 3. Visual illustration of CNN computational components and trends in architectural variations.

3.1.1 AlexNet

AlexNet[11] is the original record-breaking ImageNet CNN architecture which kick-started the deep learning for computer vision revolution. AlexNet is an architecture with one primary distinguishing feature from its predecessors: it is an order of magnitude larger than any other CNN trained at the time of its publication. This increased size was enabled by the recent GPU developments, as well as a highly optimized GPU implementation of the convolution operation. Prior to AlexNet, CNNs of that size trained on that quantity of data were unfeasible due to compute restrictions at the time. AlexNet features roughly 60mm parameters in eight layers, five convolution and three fully-connected.

3.1.2 Batch Normalization

As AlexNet and many ensuing models employ depth and width for increased performance, advances in the learning process have been necessary. In particular, batch normalization is a step inserted after a hidden layer's linear function (in our case a convolution operation) prior to a layer's activation function.[12] Input normalization during both training and inference is a common technique used to make learning problems easier. Batch normalization extends this idea and applies a normalization step to each hidden layer output of a DNN. Batch normalization makes models more robust to different hyperparameter choices, speeds up training, enables the training of deeper models, and is used in every model discussed hereafter.

3.1.3 VGG

VGG[13] is an architecture proposed as a refinement to the ideas found in AlexNet. Its primary contributions are simple but effective: make the network deeper, wider, and use 3×3 kernel convolution layers in stages delineated by max-pooling and increasing number of filters. VGG network layers are homogenous - the number of filters in each stage increase by powers of two, each use batch normalization in every layer, each use max-pooling after every stage of convolution, each use only 3×3 kernels with stride 1, and each use the ReLU activation. There are four architectures proposed, employing this topology over 11, 13, 16, and 19 layers. Each has five stages of convolution, with 64, 128, 256, 512, and 512 filters being used in each convolution per stage respectively. VGG shows that making CNNs deeper (with more layers) and wider (with more filters) using homogeneous layers and 3×3 filters can be effective.

3.1.4 ResNet and WideResNet

Networks have been shown to perform better as they get deeper, however information gets harder to propagate when the model is forced to learn too many sequential non-linear transformations. In order to push network depth further, deep residual networks (ResNets)[14] use residual connections to help propagate information and ease the training of substantially deeper networks.

The simplest residual connection is an additional connection added between two adjacent convolution layers that adds the input to layer $(l-1)$, referred to as the residual, to the output of layer l. There are numerous

configurations of where and how to add the residual, e.g. before or after activation, with or without learned parameter weighting, etc..., however the only configuration that tends to be used is adding the residual to layer l's intermediate output, i.e. before its activation, with no transformation applied other than non-parametric downsampling when necessary.

ResNet architectures, like VGGs, have 5 stages of convolution, with multiple blocks of layers per stage, and increase the number of filters in each convolution layer by powers of two per stage. Downsampling between stages is achieved by using a 1×1 identity kernel with a stride length of 2 after each stage in place of max pooling. Five architectures are proposed, employing 18, 34, 50, 101, 152 layers. These architectures are made up of residual blocks, i.e. blocks of two or three convolution layers connected by a residual. A two layer block, used by ResNet18 and ResNet34, is referred to as a basic residual block and implements the simple residual connection described above. The three layer block, used by ResNet50, 101, and 152, is referred to as a residual bottleneck block. In this configuration, residuals are propagated from two layers back instead of one layer back in a $(1 \times 1, 3 \times 3, 1 \times 1)$ convolution configuration with many more filters contained in the 1×1 kernels, giving the block a bottleneck shape. The increased depth of ResNets enables them to use fewer filters and in doing so, have fewer parameters than both AlexNet and VGGs. Every model discussed hereafter uses these residual connections in some form.

The base ResNet architecture, as well as VGG and AlexNet, favor depth for improved performance. To explore scaling properties, the WideResNet model adds a tuneable parameter for scaling how wide (parameter k) the ResNet block is.[15] Parameter k is a multiplier on the number of filters per convolution layer in each basic residual block, or the number of filters in the 3×3 convolution layers of each residual bottleneck block. Width increases explored include values of k ranging from 2 up to 12. WideResNets show that trading increased depth for increased width in a ResNet may significantly improve both accuracy and latency.

3.1.5 DenseNet

DenseNet[16] extends the residual connection idea by forwarding each layer's output to every subsequent layer. For each convolution layer, feature maps from all previous layers are used as input. Architecturally, this idea is achieved through the use of a stack of (dense block, transition layer) pairs. A dense block consists of a sequence of convolution layers, where each layer receives its input from all previous convolution layer outputs in that block concatenated into a single tensor. To keep matching dimensions over many dense connections while still downsampling input length and width throughout the network, a transition layer is used outside of each dense block for downsampling and further convolution prior to the next dense block. A given instance of a DenseNet is described by the number of (dense block, transition layer) pairs, the number of layers in each dense block, and a growth rate k which determines the increase in the number of filters per convolution layer. The original paper defines architectures that are constructed with both a single (dense block, transition layer) pair, as well as multiple pairs. Single pair models include networks with 40, 100, 190 layers with growth rates 12, 24, and 40 respectively. Multiple pair models include models with 121, 169, 201, and 264 layers all with growth rates of 32. The larger amount of information propagation in DenseNets allows them to be even deeper and narrower than ResNets.

3.1.6 SqueezeNet

The development of the CNN models discussed up until this point have been primarily focused on improving accuracy and generalization without giving much consideration to model size and computational cost. SqueezeNet[17] is an architecture which is representative of first attempts at a push to develop smaller, faster, and more efficient models for mobile and embedded deployment while retaining the accuracy of larger models. SqueezeNet achieves AlexNet level accuracy on the ImageNet dataset with roughly $730,000$ parameters, over 50x fewer than AlexNet. To do so, the SqueezeNet architecture is composed of a sequence of Fire modules. A fire module consists of a 1×1 convolution layer follow by parallel 1×1 and 3×3 convolution layers, the results of which are concatenated. Fire modules are stacked with a limited increase in the number of filters per module and with pooling occurring only in later layers. Replacing most 3×3 convolutions with 1×1 convolutions drastically reduces the number of parameters in the model. Additionally, a Fire module limits the number of input channels to each 3×3 filter to further reduce the number of parameters. To attempt to help retain accuracy with fewer parameters, SqueezeNet downsamples with max-pooling only late in the network. The intuition being that larger activation

maps throughout the network will lead to higher accuracy. SqueezeNet also includes optional residual connections, the use of which denotes the architecture as ResidualSqueezeNet.

3.1.7 ResNeXt

Continuing to pursue computational efficiencies and further accuracy improvements, ResNeXt[18] architectures combine ResNets with grouped convolutions. The grouped convolution is an effective modification to regular convolutions for reducing computational cost and model size, and is demonstrated by ResNeXt to also improve accuracy. In models discussed so far, each convolution layer has been composed of N filter's which each have size $K \times K \times C$. A grouped convolution splits N filters each of size $K \times K \times C$ into N filters grouped into G groups each having size $K \times K \times C/G$. Each of these filters have the same receptive field size as the original convolution, but only see a C/G size contiguous portion of the input along the channel dimension. The filters are applied to their respective input portion and afterwards concatenated to form the same size output as would be computed by a normal $K \times K \times C$ convolution with N filters. The ResNeXt architectures themselves replace all 3×3 convolutions in the bottleneck blocks of ResNet50 and ResNet101 with grouped convolutions of $G = 32$ and $G = 64$ groups, denoted as for example, ResNeXt50_32, and contain roughly half as many parameters as their ResNet counterparts with a greater than half reduction in computational complexity.

ResNext shows that increasing G versus further increasing depth or width can be more a effective way of improving accuracy beyond certain depth/width limits. Experimental evidence suggests that groups are more effective for two reasons. First, their sparsity offers a form of implicit regularization which allows them to generalize better through less over-fitting. Second, there is empirical evidence to show that grouped convolutions form stronger representations versus regular convolutions.[18]

3.1.8 MobileNet-V2

As denoted by the architecture name, MobileNet-V2 is an architecture which is targeted at embedded, mobile, and/or resource constrained platforms.[19] The primary contribution of MobileNet-V2 is a configuration of stacked convolution layers called an inverted residual bottleneck (IRB) block. An IRB block consists of three components: a depth-wise separable convolution, a linear bottleneck, and an inverted residual.

Depth-wise separable convolutions[20] are drop-in replacements for standard convolution layers that use a factorization to split a $K \times K \times C$ convolution into a $K \times K \times C$ convolution with C groups, i.e. one group per input channel, referred to as a depth-wise convolution, followed by a heavier $1 \times 1 \times C$ point-wise convolution. Depth-wise separable convolutions are nearly as effective as their regular counterparts with an associated complexity that is roughly a factor of $\frac{1}{K^2}$ less (see the original paper for a full complexity analysis). The linear bottleneck component is an additional $1 \times 1 \times C$ linear convolution layer that is inserted between the two layers of the depth-wise separable convolution to prevent the non-linearities from destroying too much information. Finally, the inverted residual is an inversion of the residual connections found in deep ResNets. In the residual bottleneck blocks found in deeper ResNets, as well as ResNeXts, the residual connections connect non-bottleneck convolutions. MobileNet-V2 inverts these and connects the bottleneck convolutions with residual connections. The structure of a single IRB block consists of a 1×1 point-wise convolution, followed by a 3×3 depth-wise convolution, and finally a 1×1 linear bottleneck. The block is structured in this way in order achieve the residual connection inversion. MobileNet-V2 is constructed with by using a stack of these IRB blocks.

3.1.9 ShuffleNet-V2

Grouped convolution operations decrease total time complexity and FLOPS necessary for convolution, however FLOPS is an indirect latency metric. It does not include either level of model parallelism or memory access cost. Models with fewer FLOPS than others still may execute with greater latency on certain platforms. The ShuffleNet-V2[21] architecture is designed to jointly optimize accuracy with latency directly.

ShuffleNet-V2 employs four principles to decrease latency. First, it uses equal channel width convolutions to decrease memory access cost. Second, it uses carefully selected group numbers in depth-wise separable convolutions to further decrease memory access cost. Third, it reduces network fragmentation to increase parallelism. And fourth, it reduces element-wise operations to again further decrease memory access cost.

The ShuffleNet-V2 architecture is made up of stacks of depth-wise separable blocks similar to MobileNet-V2. The primary mechanisms used to implement the latency reducing principles while simultaneously maintaining or improving accuracy are the channel shuffle and channel split operations found in each block. In stacked grouped convolutions, outputs from filters will only be computed from a fraction of the input channels. Channel shuffle reshapes outputs of grouped convolutions with a transpose and flatten in order to share information across groups in stacked grouped convolutions. The channel split operation simply splits an input in half along the channel dimension. In a ShuffleNet-V2 block, the channel split operation splits an input in half channel-wise such that the convolution layers in a block are only applied to half of the input channels. The remaining half are propagated without convolution, the two halves are concatenated, and the channel shuffle operation is then applied.

3.1.10 Squeeze and Excitation

Convolution operations in CNNs excel at encoding spacial relationships, however they only implicitly encode channel-wise relationships. A Squeeze and Excitation[22] (SE) block is an architectural component added to existing models which directly encodes channel-wise relationships and dependencies. SE blocks re-calibrate channels using a learned weighting from global channel information. An SE block receives a $H \times W \times C$ convolution layer output, average pools each channel to compute a $1 \times 1 \times C$ vector of channel descriptors (squeeze), applies a linear layer with a ReLU activation to the channel descriptors, followed by another linear layer with a sigmoid activation to compute channel weightings. Each channel in the $H \times W \times C$ convolution layer output is then scaled by the corresponding element in the resulting $1 \times 1 \times C$ vector of channel weightings (excite).

Although different configurations can be used, inserting the SE block after convolution and batch normalization prior to activation is empirically shown to be most effective. This concept led to the development of the SENet model, which is a ResNeXt154_64 model with an SE block inserted into the third convolution layer of each residual bottleneck block. Additionally, SE blocks have been applied to several other architectures such as ResNet and MobileNet, denoted by SEResNet and SEMobileNet respectively, and are used in every subsequent architecture discussed.

3.1.11 Neural Architecture Search, MNASNet, and MobileNet-V3

Neural architecture search (NAS) is the algorithmic exploration of neural network configurations. Various search methodologies including reinforcement learning (RL), evolutionary search (ES), gradient based, and combinatorial optimization seek high performing network configurations by exploring how to combine the computational blocks which comprise the network architecture itself.[23–25] The different search methodologies offer trade-offs in potential search space definitions, how the search space is traversed, how coarse or fine the search space can be, as well as what performance objectives are considered. NAS has been utilized to discover several high-performing, compact architectures.

One of those architectures is MnasNet,[26] an architecture that notably factorizes a CNN into a sequence of blocks and applies RL NAS to find a unique architecture within each block. The approach constrains the search space while jointly optimizing for accuracy and speed by receiving feedback from both a models accuracy, as well as simulated real-world resource constrained deployment (mobile phone inference latency) at each search iteration. The base MnasNet architecture found (MnasNet-A1) also includes a depth multiplier hyperparamater allowing for further user-chosen latency/accuracy trade-offs.

MobileNet-V3[27] improves upon the MobileNet-V2 architecture by employing RL NAS in a similar manner, as well as integrating SE blocks, and utilizing the *swish* activation function. MnasNet's RL NAS algorithm is first used to search for a coarse architecture by optimizing block level structure. NetAdapt,[28] a tool for iterative latency optimization of a trained model for mobile deployment, is then used to fine tune individual layers of the model, identifying under-utilized convolution channels and pruning them. Two architectures, MobileNet-V3_Small and MobileNet-V3_Large are defined, targeted at higher and lower resource constraints respectively.

3.1.12 EfficientNet

The EfficientNet[29] family of architectures leverage both NAS and a method of tunable, compound model scaling. A lightweight baseline model, EfficientNet-B0, is first found using the same RL NAS algorithm used for MnasNet but with a larger FLOPS target in place of the mobile device latency target. Compound scaling is used to

uniformly scale the network's depth, width, and input resolution together based on a factor ϕ coupled with a heuristic scaling formula. EfficientNet-B1 to B7 are then obtained by scaling up the baseline network using this formula with increasing ϕ. This compound scaling is shown to be more effective over scaling dimensions individually and independently, such as is seen in scaling the depth of a DenseNet or the width of a WideResNet. EfficientNet-B0 incorporates every architectural innovation discussed so far, as well as stochastic depth,[30] a regularization method which, for each mini-batch processed during training, randomly bypasses a different subset of layers in a deep network with the identity function.

3.2 Hyperparameter Optimization

The performance of the above CNN models is dependent not only upon the model structure, but also dependent upon choice of hyperparameters. Even with a fixed architecture topology, there are numerous hyperparameter choices which need to be made when training one of the above CNNs on a given dataset. These include batch size, learning rate, optimizer choice, weight decay rate, weight initialization, learning rate schedule, etc. With current models, the right choice of hyperparameters, as we will show, can make a significant impact on accuracy.

There are several challenges in searching for and choosing an optimal set of hyperparameters for a DNN. First, hyperparameter optimization is a non-convex problem and there is no gradient information to direct a hyperparameter search. Second, each chosen set of candidate parameters is expensive to evaluate. Although early stopping is an oft-used technique, a model must be trained partly, if not to completion, in order to evaluate a candidate hyperparameter set. This may take minutes, hours, or days. Third, the resulting evaluation is noisy, as stochastic gradient descent methods and random weight initializations inject stochasticity into the training process. This stochasticity is beneficial for generalization and speed of convergence, however it makes searching for optimal hyperparameters more difficult.

Principled algorithmic methods enable a more systematic and automated exploration of hyperparameter choice over hand-tuning or uniform exploration. The characteristics of the hyperparameter optimization problem makes Bayesian optimization a natural choice with several effective existing approaches.[31–33] In particular, we have used the Bayesian hyperparameter optimization tool Optuna,[34] a distributed asynchronous Bayesian hyperparameter optimization framework written in Python. Optuna implements a hybrid of several state-of-the-art hyperparameter search algorithms and can be configured to search over any hyperparameter space.

3.2.1 Hyperparameter Architecture Search

We have further leveraged Optuna to blend Bayesian hyperparameter optimization with NAS and have employed a hyperparameter architecture search (HAS) technique. By using CNN architectures with intrinsically tunable features we have employed Bayesian hyperparameter optimization to explore configuration variations of a few models to tailor them for SAR ATR. Specifically, we have explored the VGG, ResNet, and EfficientNet family of architectures in this manner. We define a generic, configurable model from each of these families of CNN architectures and with this configurable model definition, we leverage Optuna's automated Bayesian hyperparameter optimization to search for the highest performing architecture within our defined search space of models in that architectural family while jointly searching for the best set of hyperparameters to train said model.

3.3 Datasets

To evaluate the efficacy of the neural network approaches illustrated here, we have assessed their accuracy on two SAR ATR datasets. Analogous to the prevalent datasets such as MNIST, CIFAR, and ImageNet which have been the foundations of DNN research, these SAR datasets serve as the basis for benchmarking SAR ATR performance. The Moving and Stationary Target Acquisition and Recognition (MSTAR) dataset is the standard for SAR ATR.[35,36] It consists of 128×128 HH-polarization X-band SAR image chips with range and cross-range resolution of 0.3 m. The dataset consists of the following ten targets: BMP-2 tank, BRDM-2 truck, BTR-60 transport, BTR-70 transport, D-7 bulldozer, T-62 tank, T-72 tank, ZIL-131 truck, ZSU-23-4 gun, and 2s1 gun. The MSTAR dataset is split between 17 degree and 15 degree collection angles which serve as the training and validation datasets respectively.

Unlike the large scale optical image datasets found in DNN research, ATR tasks cannot assume an abundance of training data is available, if any at all. To address this challenge, electromagnetic (EM) computational tools

have been pursued as a means of simulating radar returns. Given a CAD geometry, this offers a technique where SAR data may be simulated using a computer model of a target. The Synthetic and Measured Paired Labeled Experiment (SAMPLE) dataset employs this concept while coupling the simulated SAR data with measured collections.[37] The simulated data serves as the training set in the SAMPLE dataset. Then, the performance of a model may be tested against corresponding measured data. The SAMPLE dataset consists 128×128 image chips of the following ten targets: BMP-2 tank, BTR-70 transport, T-72 tank, ZSU-23-4 gun, 2s1 gun, M1 tank, M2 tank, M35 truck, M548 truck, and M60 tank. For the evaluations conducted here, we have used the public versions of each dataset.

3.4 Computational Cost Analysis

In addition to classification accuracy, we have also analyzed the computational cost of performing inference using these CNNs. To do so, we have extended the MAESTRO analytical tool which, given a CNN configuration, a dataflow, and a hardware configuration, calculates how many operations are executed and the amount of data movement necessary to run the network on the target architecture.[38] Our extensions include incorporating the cost of pooling layers, accounting for ReLU activation functions, and a semi-automated PyTorch model importing system.

CNN acceleration hardware, in general, implements activation functions and pooling layer operations in dedicated hardware components on-chip. Our extensions account for the number of bit-wise comparisons needed for computing a maximum function in the underlying hardware to account for ReLU activations and max pooling layers. Our tool computes the number of bits output by the underlying network layer for activation functions and the number of bit-wise comparisons needed to process pooling layers. We assume that the word-size of the underlying hardware represents the number of bits that can be compared at once, and that the data is sent to a buffer within the comparison circuit on-chip. The energy use and performance estimate data are based on information compiled in,[39] and applied to the total count of max-operations produced by our tool. Average pooling layers are computed as a specialized version of a convolution layer, where the average function is computed by setting the weights and kernel size to re-create a moving average against the input convolution layers. The energy and latency estimates that our tool generates are not dependent on the actual data moved through the network, rather the tool estimates energy based on all input, filter, kernel, and other dimensions of the layer, which affords flexibility when building the models in this tool.

The performance values computed by our tool are computed as a set of MAC operations, latency in clock-cycles, and on-chip global and local memory energy use. The memory energies are based on a 32nm process, from values given in the MAESTRO tool.[38] We also convert the number of MAC operations to energy used in pJ using values integrated with the MAESTRO tool. By using this estimated energy and latency cost, based on a common process node, effective comparisons between the performance of different CNNs are possible.

Some of the models that we investigate use non-ReLU activations, which we cannot model. Removing the ReLU energy costs allows relative comparisons across model architectures. We then separately examine the ReLU energy cost data against our tool's energy use estimate, additionally providing insights into the energy contribution of ReLU activation.

4. EXPERIMENTS

To assess whether neural-inspired algorithms and architectures offer a computationally advantageous solution for SAR ATR, we consider both accuracy and computational costs. First we present the resulting accuracies of the CNNs characterized here, followed by an exploration into computational costs including estimated energy usage and latency. We conclude with a discussion of trade-offs and trends between architectures seen in our results that are applicable to SAR ATR.

4.1 Training and Evaluation Setup

We include a total of 53 CNN models in our evaluation suite and have implemented each using the open source deep learning library PyTorch.[40] An instance of each model is trained on MSTAR 17 degree data and evaluated on

MSTAR 15 degree data. A second instance of each model is trained on SAMPLE synthetic data, then evaluated on SAMPLE measured data. Models are trained on NVIDIA GPUs for 120 epochs each.

Prior to hyperparameter optimization, we hand-tune a set of hyperparameters for training each model. For the set of hand-tuned hyperparameters, a batch size of 4 is fixed for each model to narrow the search space, as we found through manual exploration that small, single digit batch sizes afford the best performances across most models. The optimizer is chosen between Adam[41] and stochastic gradient descent (SGD) with a momentum value of 0.9. Weight decay is set to either 0 or 1×10^{-5}. The learning rate is manually tuned, though we settled on 3×10^{-4} for many models. We use a cosine annealing with warm restarts learning rate schedule[42] with values $T_0 = 10$ and $T_{mult} = 2$. For SAMPLE data, we optionally either add speckle noise sampled from $N(0, 0.1)$ to each pixel, or a apply a gaussian smoothing filter with a σ of 0.75. Model weights are initialized with PyTorch default options. For convolution weights, this is the Kaiming Uniform intialialization.[43]

Importantly, we note that there are some deficiencies regarding data separation both inherent in these datasets as well as in our methodology. In data science, it is important to ensure that there is no 'information leak' between the training and testing set. The intent being to learn the salient features of the learning problem, not overfit and memorize a problem's data. A fully isolated test set helps assess an algorithm's ability to generalize, forecasting its behavior for future unseen samples. Unfortunately, separation deficiencies are difficult to avoid due to the rarity of SAR ATR data. The same vehicle instances appear in both the training and test sets and is a byproduct of the data collection/synthesis. Furthermore, although not ideal, we choose to use the test set for validation during hyperparameter optimization. We believe that this workflow best approximates a deployment scenario and note that it is difficult to further subdivide these datasets without creating strong correlations between samples (e.g. two random subsets of MSTAR/15 degree will have highly correlated samples due to repetitions of the same vehicle instance). We recognize this a shortcoming and caution that overfitting is likely. Nevertheless, the results presented here are intended to convey the strong performance capabilities neural networks can enable for SAR ATR and if anything, we hope this work motivates further research for sample efficiency, the ability to utilize synthetic data, as well as larger, more realistic and more comprehensive SAR ATR datasets.

4.2 Optuna Hyperparameter Optimization

We perform our automated hyperparameter optimization using the Optuna Bayesian hyperparameter optimization Python library. In the Optuna workflow, a search space of hyperparameters is defined dynamically in Python code and searched over during optimization. During one iteration of hyperparameter optimization, a new set of hyperparameters is sampled from the current search space and a new model instance is trained with this sampled set of hyperparameters for a number of epochs. The model's performance under this set of hyperparameters is then returned to Optuna and used to further refine the search space.

Optuna can be used in a distributed, asynchronous fashion. We perform our hyperparameter search for each model asynchronously across up to 32 GPUs, depending on internal availability and GPU hours needed for model training time. We assign each GPU to a Python process which repeatedly samples a set of hyperparameters from Optuna, trains a new instance of the current model on the GPU assigned to that process using the sampled hyperparameter set, and reports the corresponding results back to Optuna.

Our hyperparameter search space can be seen in Table 1. We run 1000 to 1200 iterations of hyperparameter optimization for each model using Optuna's default parameters, as returns diminished for further iterations beyond that. For each iteration, we train a model for 120 epochs and report the best results seen across those epochs. Note that we only use a single GPU per model instance when training. This limits the maximum batch size for some models to a smaller value than others when searching.

4.3 Optuna Hyperparameter Architecture Search

Models included in our suite were not designed with domain adaptation in mind. That is, a shifting data distribution between training data and validation data such as is seen in the SAMPLE dataset. We extend our Optuna hyperparameter search to include searching for architectural configurations of EfficientNet, ResNet, and VGG which are tailored to the domain adaptation necessitated by the SAMPLE dataset and SAR ATR more broadly. To achieve this, we use PyTorch to define a configurable model from each family. Using the same workflow described in the previous section, we then jointly search for the hyperparameters in Table 1 and the

Table 1. Training Hyperparameter Search Space

Hyperparameter	Range
Batch size	$\{1, 2, ..., *\}$
Learning rate	$[1 \times 10^{-5},\ 1.0]$; log uniform sampling
Weight decay	$[1 \times 10^{-6},\ 5 \times 10^{-4}]$; log uniform sampling
Weight initialization	$\{N(0, 1),\ U[-1, 1]$, kaiming normal, kaiming uniform, xavier normal, xavier uniform$\}$
Optimizer	{adam, sgd, sgd with momentum, rmsprop, rmsprop with momentum}
Learning rate scheduler	{stepwise, exponential, cosine annealing}
Data pre-processing	{none, gaussian noise, gaussian smoothing}

Table 2. HAS Search Space & Results

VGG		**ResNet**		**EfficientNet**	
Hyperparameter & Range	**Result**	**Hyperparameter & Range**	**Result**	**Hyperparameter & Range**	**Result**
Conv64 layers: $\{0, 1, ..., 10\}$	0	Block type: {basic, bottleneck}	basic	Depth multiplier: $[0.1, 4.0]$, $step = 0.1$	1.2
Conv128 layers: $\{0, 1, ..., 10\}$	1	Conv. stage 2 layers: $\{1, 2, ..., 36\}$	10	Width multiplier: $[0.1, 2.2]$, $step = 0.1$	1.7
Conv256 layers: $\{0, 1, ..., 10\}$	1	Conv. stage 3 layers: $\{1, 2, ..., 36\}$	9	Input resolution: $\{32, 33, ..., 250\}$	57
Conv512 stage 1 layers: $\{0, 1, ..., 10\}$	6	Conv. stage 4 layers: $\{1, 2, ..., 36\}$	27	Dropout rate: $[0.0, 0.6]$, $step = 0.01$	0.51
Conv512 stage 2 layers: $\{0, 1, ..., 10\}$	6	Conv. stage 5 layers: $\{1, 2, ..., 36\}$	1		
Width multiplier: $\{1, 2, 3, 4\}$	4	Width multiplier: $\{1, 2, 3, 4, 5\}$	4		
Batch Normalization: $\{true, false\}$	$true$	Groups: $\{1, 4, 8, 16, 32, 64\}$	64		
Dropout rate: $[0.0, 0.5]$, $step = 0.1$	0.0	SE: $\{true, false\}$	$false$		
Fully connected units: $\{2^{\{7,8,...13\}}\}$	2^{10}	SE reduction ratio: $\{4, 8, 16\}$	NA		
Average pool size: $\{1, 2, ..., 10\}$	1				

architectural hyperparameters we define in our configurable models. All architecture hyperparameters searched over for each model family, their ranges, as well as the search results can be seen in Table 2.

This idea is most naturally implemented with EfficientNet, as the architecture definition already contains depth multiplier, width multiplier, input resolution, and dropout rate hyperparameters whose variations define the EfficientNet instances B0-B7. For the ResNet family, we search over the number of convolutions to include in each of the 5 stages of convolution found in every ResNet, the residual block type (basic or bottleneck), the width multiplier k seen in WideResNet, the number of groups G, whether to include SE blocks, and if SE blocks are included, what reduction ratio to use. For the VGG family, we similarly search over the number of convolutions to include in each of the 5 stages, with an option to skip a stage using a chosen value of 0 for that stage. We also include the dropout rate, number of units in fully connected layers, average pool size after convolution stages, whether or not to use batch normalization, and a small width multiplier.

4.4 Performance

Table 3 provides the accuracies and sizes of the networks we have included in our evaluation suite on MSTAR and SAMPLE validation sets. To demonstrate the efficacy of systematic hyperparameter optimization, we include both accuracies obtained with hand-tuned hyperparameter choice and hyperparameter optimized accuracies. For each model that was hyperparameter optimized, we report the performance of the highest performing model found across all iterations of hyperparameter search. Asterisk columns denote hyperparameter optimized models.

Both the VGG and ResNet architecture implementations follow their original papers. WideResNet18 and WideResNet34 are the same as their ResNet counterparts but include a width multiplier of $k = 5$ applied to every convolution layer. WideResNet50 and WideResNet101 use a width multiplier of $k = 2$ applied to the 3×3 convolutions in each residual bottleneck block. The ResNeXt models include a ResNet50 with 32 groups and a ResNet101 with 32 and 64 groups, denoted as ResNeXt50_32, ResNeXt101_32, ResNeXt101_64 respectively. The DenseNet architecture implementations also follow the original paper and include the number of layers in the name, e.g. DenseNet_L40_k12 contains 40 layers with a growth rate of 12 in a single (dense block, transition layer) pair, and DenseNet121 contains 121 layers across multiple (dense block, transition layer) pairs. The SqueezeNets, SENet154, MobileNets, EfficientNets, ShuffleNet-V2s, and MnasNet-A1s are all as they appear in the original papers.

Due to both time and available compute constraints, we have not optimized hyperparameters for all networks. We have prioritized optimizing networks trained on SAMPLE due to several factors. First, networks trained on MSTAR yielded high performance with manual hyperparameter tuning. Second, the challenge that domain adaptation introduces makes SAMPLE a more difficult and operationally relevant dataset, warranting optimization.

Using our enhanced MAESTRO, we examine performance on a Google Coral TPU based hardware design. Based on data from,[9,44] we created a hardware/data-flow specification similar to the Google Coral Edge TPU. The analyzed hardware consists of a systolic-grid array of processing elements (PEs), with groups of processing elements clustered into logical "Cores". Each core contains multiple compute lanes, which share a scratchpad memory and a global "PE" memory space. We recreated this hardware with 2 MB of per-PE memory and 8 KB of per core group memory. The system has a total of 16 PEs, which expand to a grid of 256×256 compute elements. We also varied where data was stored on the chip's memory. This variation encompasses storing weights, inputs, or outputs on the scratchpad or global buffer memory. Latency and energy usage of each model can be seen in Figure 4. The tradespace of accuracy, latency, and energy usage is illustrated in Figure 5.

EfficientNets, MobileNets, SENets, and MNasNets, employ different activation functions in place of or in addition to the ReLU function currently implemented in our enhanced MAESTRO tool. Therefore, to ensure that energy comparisons between networks are compatible, the results presented in Figure 4 do not contain the cost of ReLU activation functions. We found that the effects on energy of the ReLU circuit were less than $4.72 \times 10^{-5}\%$. In this work, we present the relative performance in terms of clock-cycles as latency, and energy estimates based on on-chip memory, compute, and network access. While our customized Maestro tool provides absolute measurement results, validating and comparing absolute energy and latency values between MAESTRO's reported values and the reported Google Coral TPU hardware benchmarks would require substantial investigation and is beyond the scope of this paper. For some on-hardware benchmarks on the Coral TPU, see https://coral.ai/docs/edgetpu/benchmarks/.

Table 3: Neural Network Accuracy Results

Model	MSTAR	MSTAR*	SAMPLE	SAMPLE*	# Model Parameters
AlexNet	89.64	94.10	77.17	82.60	58,299,082
VGG11	94.77	99.54	82.82	95.54	128,811,658
VGG13	97.59	99.74	78.95	94.05	128,996,554
VGG16	98.31	99.64	79.25	93.61	134,308,810
VGG19	96.92	99.80	85.57	90.33	139,621,066
VGG HAS	N/A	N/A	N/A	**96.88**	442,070,026
ResNet18	98.82	99.59	86.98	92.49	11,178,378
ResNet34	97.44	99.13	90.70	92.27	21,290,250
ResNet50	96.10	98.67	86.54	89.52	23,541,130
ResNet101	94.72	–	72.93	88.77	42,559,370
ResNet152	97.80	–	73.82	91.45	58,226,058
WideResNet18	98.10	–	89.00	93.61	279,041,930
WideResNet34	98.00	–	79.77	–	531,616,010
WideResNet50	95.54	–	77.02	–	66,874,890
WideResNet101	96.00	–	78.73	–	124,913,162
SEResNet18	97.49	–	80.00	91.67	11,267,458
SEResNet34	96.87	–	79.18	89.44	21,451,446
ResNet HAS	N/A	N/A	N/A	94.05	327,303,690
ResNeXt50_32	95.80	–	68.02	91.52	12,577,546
ResNeXt101_32	94.87	–	78.51	89.29	21,882,122
ResNeXt101_64	94.82	–	74.12	91.30	21,548,618
DenseNet_L40_k12	94.21	–	44.16	–	862,150
DenseNet_L100_k24	75.38	–	59.48	–	17,237,266

Continued on next page

Table 3 – *Continued from previous page*

Model	MSTAR	MSTAR*	SAMPLE	SAMPLE*	# Model Parameters
DenseNet121	96.41	–	74.49	90.11	7,488,778
DenseNet169	90.46	–	71.97	90.63	13,887,690
DenseNet201	88.10	–	76.80	–	19,959,370
DenseNet264	90.67	–	81.85	–	34,294,986
MobileNet-V2	95.69	99.03	83.19	88.48	5,531,082
MobileNet-V2_1.4x	92.31	98.51	77.24	89.74	10,755,092
SEMobileNet-V2	96.26	–	60.96	90.78	5,561,095
SEMobileNet-V2_1.4x	97.23	–	51.67	–	10,812,047
MobileNet-V3_Small	95.64	99.03	84.60	83.64	1,695,168
MobileNet-V3_Large	95.54	99.33	84.90	83.20	4,225,224
SqueezeNet	89.69	97.64	61.04	79.55	731,146
ResidualSqueezeNet	50.26	97.80	19.85	81.19	731,146
SENet154	98.26	–	17.17	91.97	51,480,954
EfficientNet-B0	97.28	**99.95**	80.07	90.71	7,166,938
EfficientNet-B1	95.85	**99.95**	77.32	89.74	12,084,990
EfficientNet-B2	95.49	**99.95**	67.50	–	14,302,002
EfficientNet-B3	92.31	–	69.29	–	20,032,858
EfficientNet-B4	91.44	–	78.28	–	33,187,304
EfficientNet-B5	89.64	–	49.07	–	54,166,898
EfficientNet-B6	96.56	–	62.00	–	78,226,124
EfficientNet-B7	97.13	–	71.38	–	123,397,762
EfficientNet HAS	N/A	N/A	N/A	91.97	34,782,636
ShuffleNet-V2_0.5x	96.41	98.62	87.28	91.38	352,666
ShuffleNet-V2_1.0x	96.67	99.03	81.26	90.33	1,265,906
ShuffleNet-V2_1.5x	95.33	99.64	87.36	89.00	2,492,186
ShuffleNet-V2_2.0x	96.87	99.08	86.32	89.96	4,358,514
MnasNet-A1_0.35x	95.28	99.08	76.28	88.77	836,150
MnasNet-A1_0.5x	97.69	99.33	80.89	88.03	1,388,772
MnasNet-A1_1.0x	96.05	99.18	82.37	88.03	3,879,200
MnasNet-A1_1.4x	95.69	99.13	81.33	87.51	7,333,302

4.5 Discussion

4.5.1 Hyperparameters and Dataset Performance

Examining Table 3, we observe exceedingly high validation accuracies on the MSTAR dataset across a majority of the models, with EfficientNet-B0, B1, and B2 achieving a 99.95% MSTAR validation accuracy. There is a clear degradation in accuracy when going from MSTAR to SAMPLE for each model, with a maximum validation accuracy of 96.88% achieved by our VGG HAS model.

From these results, we observe that well chosen hyperparameter sets for training on these SAR ATR datasets allows for more maximal generalization performance, particularly for the domain adaptation required by SAMPLE. Percentage improvements are up to 6% on MSTAR and well over 20% on SAMPLE between hand-tuned hyperparameters and Bayesian optimized hyperparameters. These results suggest several conclusions. First, existing Bayesian optimization methods for hyperparameter selection are effective at achieving high performance for SAR ATR. Second, our HAS model performance, in addition to demonstrating state-of-the-art accuracy on SAMPLE, indicates that NAS and domain specific NAS methods for achieving better generalization from synthetic to measured data is a promising avenue of further work. Third, the larger gap between hand-tuned and hyperparameter optimized performances on SAMPLE suggests that hyperparameter selection may be more

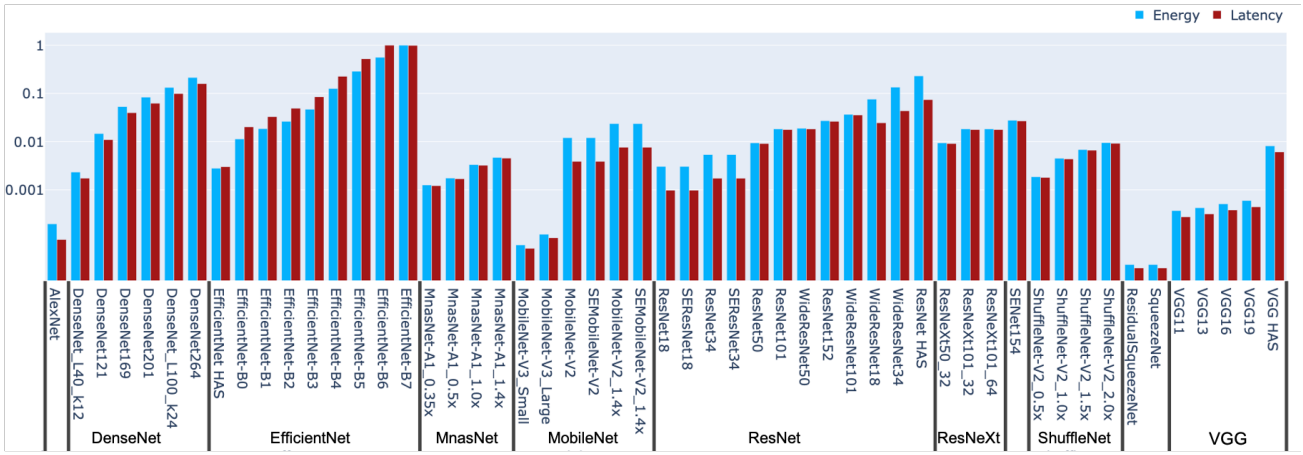

Figure 4. Relative Energy and Latency of CNN architectures. Energy and latency forecast for a TPU-like architecture are scaled relative to the maximum of the respective values.

sensitive for achieving good domain adaption between synthetic and measured data than when only training on measured data.

4.5.2 Architecture Trends and Comparisons

Primary axes in which these models vary is their depth (# of layers), width (# of channels in convolution layers), size (# of parameters), connectivity (e.g. residual connections), and convolution layer optimizations (e.g. depth-wise separable convolutions). From our assessment, the impact of varying these dimensions for SAR ATR does not exactly mimic performances seen in CNN literature on optical band imagery, though there are some similar trends.

A distinction from optical band imagery performance is that making models larger within a model family does not necessarily improve performance, whereas in CNN literature there is generally an improvement in performance when making models larger within a model family.[13,14,16,21,26,29] This distinction can be seen in all of the model families demonstrated here. Potential reasons for this include the far smaller sizes of SAR ATR datasets, differing complexities, and noise profile, all of which may benefit from a less complex model.

We notice that favoring increased width over increased depth, after reaching a depth-benefit-ceiling, affords better generalization both in MSTAR and in SAMPLE. This is somewhat in line with performances on optical imagery.[15] Fixing network depth or increasing width versus depth for domain adaptation in SAMPLE looks to be particularly effective. A hypothesis for this phenomena is that progressively deeper networks may learn features which are progressively more specific to the synthetic data, whereas shallower but wider networks may learn many more simple features which generalize better across synthetic/measured domains. This is demonstrated by models such as WideResNet18 having better generalization than deeper ResNets, our EfficientNet HAS preferring a larger width multiplier to depth multiplier ratio than in the EfficientNets B0-B7, and our VGG and ResNet HAS models performing with top accuracies using large width multipliers. We observe these same heuristics are also an effective strategy for MSTAR.

Our assessment shows that increasing the complexity of connectivity patterns may be unnecessary for SAR ATR when model size is sufficiently large. This is illustrated by VGG model performance across both MSTAR and SAMPLE being top or near top accuracies and SqueezeNet being near bottom. Both VGG models and SqueezeNet only make use of basic, one-to-one layer connections. Residual connections or dense residual connections are not used. The primary distinguishing factor is that VGG models are substantially larger than SqueezeNet. However, other models with residuals often perform as good or better than VGGs, despite a smaller network size; this indicates that introducing more complex connectivity may allow reductions in model size while retaining or improving accuracy. This is additionally illustrated by the added residual connections in ResidualSqueezeNet

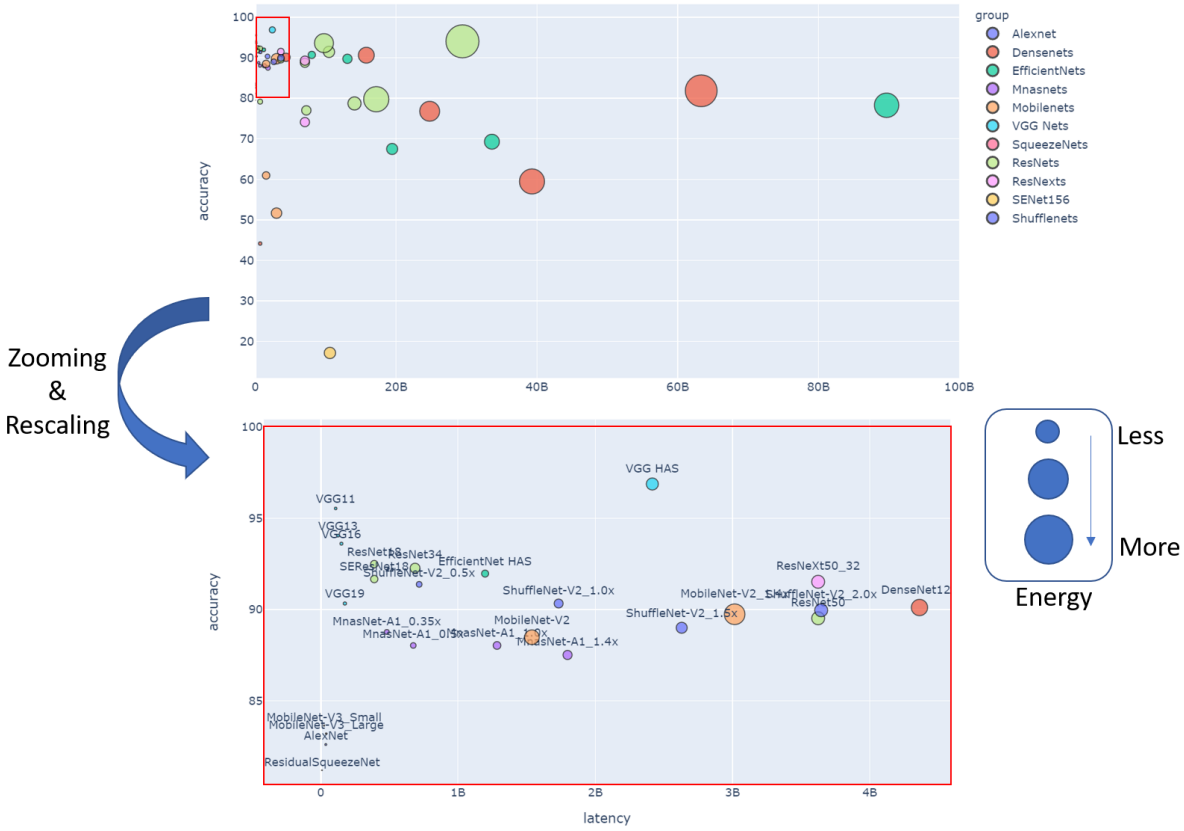

Figure 5. Tradespace of accuracy, latency, and energy for suite of explored CNNs forecasted on Google Coral edge TPU like neural network architecture. For visual clarity we have excluded the longer latency EfficientNet-b5, EfficientNet-b6, and EfficientNet-b7 models which skew the axis dimensions in the upper figure. The lower figure is a zoom and re-scale highlighting the region of highest accuracy and lowest latency. Marker size denotes the energy consumption of each model and the color denotes model families showing trends.

improving SqueezeNet accuracy. A potential explanation for this is that the feature re-use and information propagation achieved with residual connections affords a fewer number of learned features in later layers.

We make several observations examining performance trends across convolution layer optimizations. First, as illustrated by ResNet/ResNeXt comparisons, adding groups appears to be effective in marginally improving accuracy on both MSTAR and SAMPLE, which mimics optical band imagery performance trends. We hypothesize that this is due to the regularization afforded by the increased sparsity of grouped convolutions. Bottleneck convolutions seem to hurt performance on SAR ATR datasets. This is exemplified by the deeper ResNets performing worse than shallower and by the ResNet HAS choice of using basic blocks. This matches our observations above that wider and shallower over deeper and narrower is a better choice for these SAR ATR datasets. Finally, while depth-wise separable convolutions improve compute cost, outside of EfficientNet on MSTAR they seem to perform worse than regular convolutions when only considering accuracy. Not requiring a lot of extra depth for improved performance may alleviate the necessity of replacing regular convolutions with depth-wise separable convolutions. Additionally, full-depth 3×3 filters may capture more robust features for SAMPLE domain adaptation.

4.5.3 Computational Cost Trade-offs

Examining the performance seen in Figure 4 exemplifies the importance of minimal memory and network traffic for reduced latency and energy consumption. The group of VGG networks showed a much lower cost of energy

and latency when compared with models that are less demanding computationally. VGG networks, while large, are extremely operation dense. The structure of the network enables hardware acceleration devices to better re-use data than comparable networks. Networks designed for embedded systems, such as the MobileNet family of networks, also perform well. These networks have layers that can be mapped to maximize re-use when running on a grid-based hardware device. Additionally, shown in Figure 5, the relationship between energy/latency and accuracy may be counter-intuitive. Some of the highest accuracy models have both low energy usage and latency which makes these approaches yet more promising solutions for SAR ATR.

4.5.4 Ensembles

While the use of ensemble methods are beyond the scope of this paper, it is well known that ensembles can often improve performance compared to a single model.[45,46] As an exploration, we evaluated an ensemble of the hyperparameter optimized EfficientNet-B0 and B1 on MSTAR. The ensemble class probabilities were computed with an unweighted average over both model's outputs. **This two-network ensemble attains a perfect 100% accuracy on MSTAR**. Given this success, we believe that further investigations of ensembles and more sophisticated ensembling methods are a critical path forward.

4.5.5 Dataset Shortcomings

Many works, including ours, demonstrate exceedingly high accuracies on MSTAR targets, however caution should be taken when considering MSTAR performance a reflection of true operational performance. All MSTAR examples are centered, scaled evenly, and have an even distribution of vehicle pose. There is also little background noise, no clutter, no occlusion, and no confusers. A potentially more significant issue is a lack of sample variation described in Section 4.1. The collection consisting of a small number of vehicle instances, in similar locations and relative positioning, is a critical limitation. Ultimately, MSTAR should be a starting point and good performance on MSTAR should warrant further analysis on more difficult and realistic datasets. And, while SAMPLE attempts to address the lack of measured data available in a deployment scenario, SAMPLE's synthetic samples still suffer from many of the same issues.

5. CONCLUSION AND FUTURE WORK

We conduct an extensive exploration in how contemporary CNN architectures can be applied to SAR ATR. To do so, we investigate and characterize how a multitude of different CNN architectures and architectural components can impact SAR ATR performance. We demonstrate models that achieve 100% accuracy on MSTAR validation and 96.88% on SAMPLE validation while using no data augmentation and only making use of trivial data pre-processing operations such as re-scaling. To our knowledge, these performances are state-of-the-art as of this writing. Additionally, we asses the compute costs of executing these networks to gain insight not only into accuracy but latency and energy as well. In particular, we simulate the cost of executing these CNNs on a neuromorphic accelerator architecture modeled after the Edge TPU. This offers insight into the tradespace of performance for enabling SAR ATR on deployed platforms. We then discuss trade-offs between architectures that are relevant to SAR ATR given their performances on SAR ATR datasets and associated compute costs.

SAR ATR performance across the architectures we have explored does not completely mimic performances on optical band imagery seen in CNN literature. Our work provides a basis for showing the efficacy of contemporary and emerging CNN architectures while comparing what aspects of their computation show promise for SAR ATR. We believe that the results presented here are encouraging, but certainly not an end-solution. Important questions remain regarding the impact of confusers, clutter, occlusion, and open classification. Additionally, the role of reliability and explainability must be thoroughly explored.

We have only scratched the surface of contemporary deep learning approaches and recognize that the field of neural networks is an active research topic with many emerging contributions needing continued exploration for use in SAR ATR.

ACKNOWLEDGMENTS

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

REFERENCES

[1] Ball, J. E., Anderson, D. T., and Chan Sr, C. S., "Comprehensive survey of deep learning in remote sensing: theories, tools, and challenges for the community," *Journal of Applied Remote Sensing* **11**(4), 042609 (2017).

[2] El-Darymli, K., Gill, E. W., McGuire, P., Power, D., and Moloney, C., "Automatic target recognition in synthetic aperture radar imagery: A state-of-the-art review," *IEEE access* **4**, 6014–6058 (2016).

[3] Koudelka, M. L., Richards, J. A., and Koch, M. W., "Multinomial pattern matching for high range resolution radar profiles," in [*Algorithms for Synthetic Aperture Radar Imagery XIV*], **6568**, 65680V, International Society for Optics and Photonics (2007).

[4] Horvath, M. S. and Rigling, B. D., "Multinomial pattern matching revisited," in [*Algorithms for Synthetic Aperture Radar Imagery XXII*], **9475**, 94750H, International Society for Optics and Photonics (2015).

[5] Morgan, D. A., "Deep convolutional neural networks for atr from sar imagery," in [*Algorithms for Synthetic Aperture Radar Imagery XXII*], **9475**, 94750F, International Society for Optics and Photonics (2015).

[6] Wagner, S. A., "Sar atr by a combination of convolutional neural network and support vector machines," *IEEE transactions on Aerospace and Electronic Systems* **52**(6), 2861–2872 (2016).

[7] Gao, F., Huang, T., Sun, J., Wang, J., Hussain, A., and Yang, E., "A new algorithm for sar image target recognition based on an improved deep convolutional neural network," *Cognitive Computation* **11**(6), 809–824 (2019).

[8] Ding, J., Chen, B., Liu, H., and Huang, M., "Convolutional neural network with data augmentation for sar target recognition," *IEEE Geoscience and remote sensing letters* **13**(3), 364–368 (2016).

[9] Yazdanbakhsh, A., Seshadri, K., Akin, B., Laudon, J., and Narayanaswami, R., "An evaluation of edge tpu accelerators for convolutional neural networks," *arXiv preprint arXiv:2102.10423* (2021).

[10] Blasch, E., Majumder, U., Zelnio, E., and Velten, V., "Review of recent advances in ai/ml using the mstar data," in [*Algorithms for Synthetic Aperture Radar Imagery XXVII*], **11393**, 113930C, International Society for Optics and Photonics (2020).

[11] Krizhevsky, A., Sutskever, I., and Hinton, G. E., "Imagenet classification with deep convolutional neural networks," in [*Advances in Neural Information Processing Systems*], Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., eds., **25**, Curran Associates, Inc. (2012).

[12] Ioffe, S. and Szegedy, C., "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in [*International conference on machine learning*], 448–456, PMLR (2015).

[13] Simonyan, K. and Zisserman, A., "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556* (2014).

[14] He, K., Zhang, X., Ren, S., and Sun, J., "Deep residual learning for image recognition," in [*Proceedings of the IEEE conference on computer vision and pattern recognition*], 770–778 (2016).

[15] Zagoruyko, S. and Komodakis, N., "Wide residual networks," *arXiv preprint arXiv:1605.07146* (2016).

[16] Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q., "Densely connected convolutional networks," in [*Proceedings of the IEEE conference on computer vision and pattern recognition*], 4700–4708 (2017).

[17] Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., and Keutzer, K., "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size," *arXiv preprint arXiv:1602.07360* (2016).

[18] Xie, S., Girshick, R., Dollár, P., Tu, Z., and He, K., "Aggregated residual transformations for deep neural networks," in [*Proceedings of the IEEE conference on computer vision and pattern recognition*], 1492–1500 (2017).

[19] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C., "Mobilenetv2: Inverted residuals and linear bottlenecks," in [*Proceedings of the IEEE conference on computer vision and pattern recognition*], 4510–4520 (2018).

[20] Sifre, L. and Mallat, S., "Rigid-motion scattering for texture classification," *arXiv preprint arXiv:1403.1687* (2014).

[21] Ma, N., Zhang, X., Zheng, H.-T., and Sun, J., "Shufflenet v2: Practical guidelines for efficient cnn architecture design," in [*Proceedings of the European conference on computer vision (ECCV)*], 116–131 (2018).

[22] Hu, J., Shen, L., and Sun, G., "Squeeze-and-excitation networks," in [*Proceedings of the IEEE conference on computer vision and pattern recognition*], 7132–7141 (2018).

[23] Elsken, T., Metzen, J. H., Hutter, F., et al., "Neural architecture search: A survey.," *J. Mach. Learn. Res.* **20**(55), 1–21 (2019).

[24] Wistuba, M., Rawat, A., and Pedapati, T., "A survey on neural architecture search," *arXiv preprint arXiv:1905.01392* (2019).

[25] Ren, P., Xiao, Y., Chang, X., Huang, P.-Y., Li, Z., Chen, X., and Wang, X., "A comprehensive survey of neural architecture search: Challenges and solutions," *arXiv preprint arXiv:2006.02903* (2020).

[26] Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., and Le, Q. V., "Mnasnet: Platform-aware neural architecture search for mobile," in [*Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*], 2820–2828 (2019).

[27] Howard, A., Sandler, M., Chu, G., Chen, L.-C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., et al., "Searching for mobilenetv3," in [*Proceedings of the IEEE/CVF International Conference on Computer Vision*], 1314–1324 (2019).

[28] Yang, T.-J., Howard, A., Chen, B., Zhang, X., Go, A., Sandler, M., Sze, V., and Adam, H., "Netadapt: Platform-aware neural network adaptation for mobile applications," in [*Proceedings of the European Conference on Computer Vision (ECCV)*], 285–300 (2018).

[29] Tan, M. and Le, Q., "Efficientnet: Rethinking model scaling for convolutional neural networks," in [*International Conference on Machine Learning*], 6105–6114, PMLR (2019).

[30] Huang, G., Sun, Y., Liu, Z., Sedra, D., and Weinberger, K. Q., "Deep networks with stochastic depth," in [*European conference on computer vision*], 646–661, Springer (2016).

[31] Bergstra, J., Bardenet, R., Bengio, Y., and Kégl, B., "Algorithms for hyper-parameter optimization," in [*25th annual conference on neural information processing systems (NIPS 2011)*], **24**, Neural Information Processing Systems Foundation (2011).

[32] Snoek, J., Larochelle, H., and Adams, R. P., "Practical bayesian optimization of machine learning algorithms," *arXiv preprint arXiv:1206.2944* (2012).

[33] Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A., "Hyperband: A novel bandit-based approach to hyperparameter optimization," *The Journal of Machine Learning Research* **18**(1), 6765–6816 (2017).

[34] Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M., "Optuna: A next-generation hyperparameter optimization framework," in [*Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*], 2623–2631 (2019).

[35] Sample, B. D. and Problem, V.-D. C., "Air force research laboratory, sensor data management system," (2014).

[36] Force, U., "Mstar overview," (2013).

[37] Lewis, B., Scarnati, T., Sudkamp, E., Nehrbass, J., Rosencrantz, S., and Zelnio, E., "A sar dataset for atr development: the synthetic and measured paired labeled experiment (sample)," in [*Algorithms for Synthetic Aperture Radar Imagery XXVI*], **10987**, 109870H, International Society for Optics and Photonics (2019).

[38] Kwon, H., Chatarasi, P., Sarkar, V., Krishna, T., Pellauer, M., and Parashar, A., "Maestro: A data-centric approach to understand reuse, performance, and hardware cost of dnn mappings," *IEEE micro* **40**(3), 20–29 (2020).

[39] Bennett, C. H., Parmar, V., Calvet, L. E., Klein, J.-O., Suri, M., Marinella, M. J., and Querlioz, D., "Contrasting advantages of learning with random weights and backpropagation in non-volatile memory neural networks," *IEEE Access* **7**, 73938–73953 (2019).

[40] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S., "Pytorch: An imperative style, high-performance deep learning library," in [*Advances in Neural Information Processing Systems 32*], 8024–8035, Curran Associates, Inc. (2019).

[41] Kingma, D. P. and Ba, J., "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980* (2014).

[42] Loshchilov, I. and Hutter, F., "Sgdr: Stochastic gradient descent with warm restarts," *arXiv preprint arXiv:1608.03983* (2016).

[43] He, K., Zhang, X., Ren, S., and Sun, J., "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in [*Proceedings of the IEEE international conference on computer vision*], 1026–1034 (2015).

[44] Samajdar, A., Zhu, Y., Whatmough, P., Mattina, M., and Krishna, T., "Scale-sim: Systolic cnn accelerator simulator," *arXiv preprint arXiv:1811.02883* (2018).

[45] Dong, X., Yu, Z., Cao, W., Shi, Y., and Ma, Q., "A survey on ensemble learning," *Frontiers of Computer Science* **14**(2), 241–258 (2020).

[46] Fort, S., Hu, H., and Lakshminarayanan, B., "Deep ensembles: A loss landscape perspective," *arXiv preprint arXiv:1912.02757* (2019).