Paired Training Framework for Time-Constrained Learning

Jung-Eun Kim
Computer Science
Yale University
New Haven, CT, USA
jung-eun.kim@yale.edu

Richard Bradford
Commercial Avionics Engineering
Collins Aerospace
Cedar Rapids, IA, USA
richard.bradford@collins.com

Max Del Giudice
Computer Science
Yale University
New Haven, CT, USA
max.delgiudice@yale.edu

Zhong Shao
Computer Science
Yale University
New Haven, CT, USA
zhong.shao@yale.edu

***Abstract*—This paper presents a design framework for machine learning applications that operate in systems such as cyber-physical systems where time is a scarce resource. We manage the tradeoff between processing time and solution quality by performing as much preprocessing of data as time will allow. This approach leads us to a design framework in which there are two separate learning networks: one for preprocessing and one for the core application functionality. We show how these networks can be trained together and how they can operate in an anytime fashion to optimize performance.**

I. Introduction

A Cyber-Physical System (CPS) interacts with the physical world and often faces resource – especially timing – constraints depending on the state of its dynamic environment. Moreover, in a CPS, software components' interactions are carefully controlled, therefore determinism and predictability are important requirements. Meanwhile, CPSes are expected to evolve by obtaining intelligence from emerging learning frameworks to achieve advanced autonomy [6]. In contrast with current CPS where determinism is of the utmost importance, learning technologies are intrinsically non-deterministic. This issue can hinder a CPS equipped with learning modules from managing its constituent learning components in a controllable manner.

This paper presents a framework for designing learning applications to operate in time-constrained systems. We consider applications in which the input data may be imperfect but may be amenable to refinement by *preprocessing*. We also consider environments in which the call for an inference may arise with little advance warning, with a response then needed no later than a given deadline. Given these assumptions, we consider a learning-enabled system that operates in an anytime fashion, where a usable answer is derived quickly but is improved by additional processing as time allows.

We assume that the core application runs to completion with relatively predictable performance. In contrast, at least over the ranges of interest, additional time spent preprocessing the input data will result in monotonic improvement of the quality of the system's output. As a result, we consider it useful to think of the system as being composed of two separate networks – a *preprocessor* and a *target network*. The preprocessor processes input data and feeds its output (processed data) to the target network which provides the functionality of interest. Additional processing may be performed as time allows.

Our framework of the system as including separate preprocessing and target networks raises interesting questions as to how best to train such a system. In particular, if we are concerned with the performance of the overall system, how should we train each of the two networks? Note that our goal lies in exploring the practicality of the paired training framework, not in achieving state-of-the-art performance of any individual or entire network.

This work is supported in part by NSF grants 1945541, 1763399, and 1521523. All views expressed here are those of the authors and not necessarily those of sponsors.

To illustrate the various design issues associated with our general framework, we consider two possible example applications for the autonomous vehicle domain. One application includes a classification neural network that classifies traffic sign images; the other predicts the best angle of the steering wheel based on images of the upcoming segment of road. Each application has a denoising (*i.e.,* noise-removing) network, so the target network can keep receiving (possibly) improved data from the denoising network as time allows. Removal of noise from the input data would increase the likelihood that the target network generates a high-quality output. When the inputs are observed in poor weather conditions, *e.g.*, fog, rain or snow, the benefit from a preprocessor (denoiser, in this case) should be especially significant.

We illustrate the implementation of our example two-network applications by describing each one's training and operational phases. We show how the preprocessing and target networks can be trained so as to optimize the performance of the overall system. We show that our proposed *Paired Training* approach results in better system performance than an alternative approach of training the two networks *separately*. For the operational phase, we illustrate the efficient use of processing time, so that the system achieves *anytime* performance, *i.e.* it provides a usable answer initially, with improved results when time allows for additional processing [12], [17], [23]. To the best of our knowledge, this is the first framework that provides anytime performance by employing two different networks working in tandem.

II. System Model

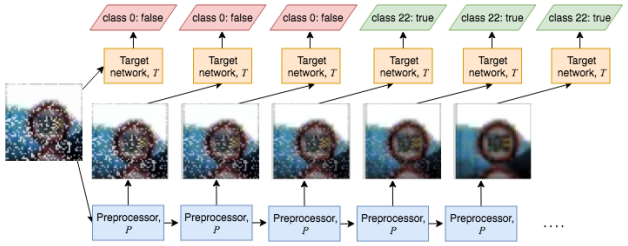
A. Overview

We propose a framework consisting of two learning networks – one network refines the input that is passed to the other network, which is ultimately responsible for executing the given target task. The network executing the target functionality is called the *target network*. A target network can provide one-time prediction from raw input. However, by employing an additional network called the *preprocessor*, the target network can take advantage of potentially improved inputs from the preprocessor as shown in Fig. 1.

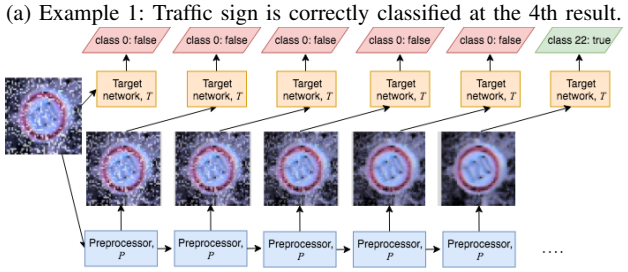
During training, the two networks interact so each network can learn the counterpart's behavior. The interactions are detailed in various training schemes that we propose in this paper, and the experimental results show that these interactions improve the overall system performance, when compared to the case of no interaction. Fig. 1 shows an abstract view of our proposed framework with a denoising and classifying network example.

B. System Description

- **Target Network**, $\mathbf{T}$: Take an input from the problem domain and execute the target functionality. Target functionality can be classification, regression, clustering, etc: $\mathbf{T} : X \rightarrow Y$, where X is a (general) problem domain, and Y is the output space, *e.g.*, classes, values between $[0, 1]$, etc. Because X and Y are generally



(a) Example 1: Traffic sign is correctly classified at the 4th result.



(b) Example 2: Traffic sign is correctly classified at the 6th result.

Fig. 1: Overview with denoised and classified image examples.

different, the output from the target network cannot serve as the input to another iteration of the target's execution.

- **Preprocessor**, $\mathbf{P}$: Refine the input data into a form that is optimized by the target network: $\mathbf{P} : X \rightarrow X$, where X is the problem domain. The specific form of the mapping $\mathbf{P}$ depends on the training data and the task at hand. Because the input and output spaces are the same, it is mathematically feasible for the preprocessor to operate iteratively; *i.e.*, the output from executing the preprocessor can serve as the input to another execution. We use a recurrent neural network (RNN) to achieve this functionality.
  While the data for image denoising is not sequential, the problem-space we are working in is; the structure of an RNN is fundamentally well-suited to the task of iterative refinement of a result. The input at each iteration can be a combination of the original low resolution image and the super-resolved image from the previous iteration.
  Our preprocessor network is trained to produce a prediction at each timestep. Let $\mathbf{s}_t$ represent the output of $\mathbf{P}$ after iteration t. The ultimate prediction of the network at iteration t is defined as the output, at time t, averaged with all of the available outputs, *i.e.*: $\mathbf{s}_t \leftarrow \mathbf{conv}(\frac{1}{t}\sum_{i=1}^{t}\mathbf{w}_i)$, where $\mathbf{conv}(\cdot)$ refers to convolution (for a moving average), and w_i is a full pass through the ith iteration of the network (the sum and convolution operation effectively define the output layer of an iteration through the preprocessor). This formulation ensures that the output of each iteration contributes to the final output. This strengthens gradient flow through the network and aids in training a deep recurrent network.
- **Unified network**, $\mathbf{T} \circ \mathbf{P}$: We are ultimately interested in the performance of the unified network where the two networks work in tandem, $\mathbf{T} \circ \mathbf{P} : X \rightarrow Y$. More specifically, we are interested in the set of predictions $\{T(s_0), ..., T(s_t)\}$ defined by the output s_i of $\mathbf{P}$ at each timestep.

III. Paired Training Techniques

We use two techniques to connect the preprocessor and target network during training: dependent loss protocols, and interleaved training schemes. The technical details are following.

A. Loss Protocols

When considered independently, the preprocessor and target network have individual goals (*e.g.*, denoising an image and classifying images). As parts of a larger system (*e.g.*, classifying noisy images), we associate their training processes to optimize system performance, by encoding this insight through their respective loss functions.

The preprocessor can either try to solely optimize its goal (train independently), or it can receive feedback from the target network, by running the target on the preprocessed outputs and checking the target loss (the preprocessor loss is now dependent on the target loss). Likewise, the classifier can be trained on non-processed clean data, or it can be trained on the preprocessor's output data. We detail such dependent/independent loss functions below. Henceforth, x indicates noisy inputs, $\hat{x}$ indicates clean (non-noisy) labels as a reference, and c indicates the (task)-label for x.

- **Training preprocessor**: When training preprocessor independently or dependently (to target network's loss) the following protocols are used, respectively:
  - **Independent training**: Train the preprocessor with the standard mean squared error (MSE) loss which counts the difference between the objective (reference) and a predicted value:

$$L = L_{MSE}(\mathbf{P}(x), \hat{x}) = (\mathbf{P}(x) - \hat{x})^2 \tag{1}$$

  - **Dependent training**: Feed the prediction $\tilde{x}$ into a target network and use the resulting performance loss L_p (dependent upon the target task) to inform the preprocessor training. We call this protocol as *Pair Target Feedback (PTF)*. Assuming x has appropriate task-label c, we define L as:

$$L = L_{MSE}(\mathbf{P}(x), \hat{x}) + \lambda_p L_p((\mathbf{T} \circ \mathbf{P})(x), c) \tag{2}$$

    where λ_p is the weight of the classifier loss relative to the MSE loss. The precise formulation of L_p is contingent upon the goal of the target network.
- **Training target network**: We consider two protocols:
  - **Independent training**: Train the target network with clean input in isolation from the preprocessor:

$$L = L_p(\mathbf{T}(\hat{x}), c) \tag{3}$$

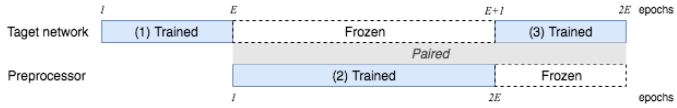
  - **Dependent training**: Use a preprocessor to process the noisy input, and then run the target network on the processed output:

$$L = L_p((\mathbf{T} \circ \mathbf{P})(x), c) \tag{4}$$

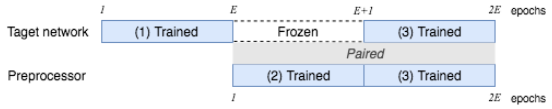
B. Interleaved Training Framework

Given the proposed system where the target performance is dependent on the preprocessor output, we discovered that training the preprocessor and target network by turns, which we call *Interleaved Training* (see Fig. 2), and pairing one another's loss results in improved system performance. It stems from the fact that even though in the course of one's training the counterpart's behavior (performance loss) is taken into account, if the counterpart is already pre-trained (independently too mature) or is not trained at all (too immature), the performance is not promising. As a result, we have found that the following training cases do not show competitive performance: (a) Preprocessor and target network are *separately* trained; (b) Target network has been separately trained although it is involved in preprocessor's training procedure. With this motivation in mind, we devised the following interleaved training techniques. The following training schemes connect the training processes of the preprocessor and target network in different ways. Let $2E$ be the number of epochs for an entire training procedure.

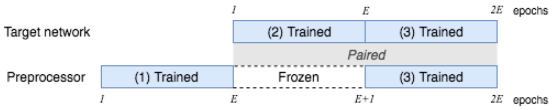
- Alternating: (i) Train target network independently for E epochs, (ii) Train preprocessor for $2E$ epochs dependently with "frozen" (*i.e.*, keeping the weights) target network and PTF loss (see (2)),



(a) Interleaved training ALTERNATING.



(b) Interleaved training MERGE1.



(c) Interleaved training MERGE2.

Fig. 2: Visual representation of *Interleaved Training*.

and then (iii) Train target network for E epochs dependently with preprocessor output. See Fig. 2(a).

- MERGE1: (i) Train target network independently for E epochs, (ii) Train preprocessor for E epochs dependently with frozen target network and PTF loss, and then (iii) Train both networks, using the preprocessor output to train the target network and PTF loss to train the preprocessor. See Fig. 2(b).
- MERGE2: (i) Train preprocessor for E epochs, (ii) Train target network for E epochs with frozen preprocessor, and then (iii) Train both networks, using the preprocessor output to train the target network and PTF loss to train the preprocessor. See Fig. 2(c).

By training the target network directly with the preprocessor output, and using feedback from the target network, we optimize the whole system rather than each network individually.

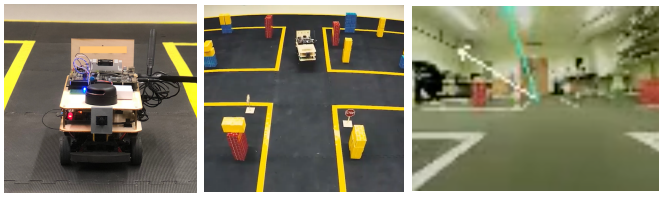
Our aim does not lie in developing an optimal paired training scheme, rather we intend to show the impact of paired training approaches by comparing them with non-paired approaches. To achieve the goal, through the experiments, we answer the following questions: Is the entire system performance aligned with the individual performance of the component networks, preprocessor and target network?; How do we train and optimize the component networks to perform well *in tandem*?; How do we optimize anytime performance?

IV. EXPERIMENTS

A. Evaluation Setup

1) Applications: While the training scheme with preprocessor and target network generalizes to multiple applications, our experiments examine the case where the preprocessor is a denoising network and the target network is either a classification or regression network:

- **Preprocessor (Denoising)**: *removes different types of noise from an image.* The denoising network shares some architectural similarities with one in [8]. Our denoiser begins with a learning rate of $1e{-}4$, reduced to $1e{-}5$ after 100 ($= E$) epochs, as the entire training lasts 200 epochs. Depending on the training scheme, the loss function is either represented by (1) or (2). In (2), we empirically determine 0.25 to be a good value for λ_p. The denoiser is trained for 6 iterations since improvements level off after this point.
- **Target network (Classification)**: *contains 43 traffic sign classes.* The classification target network is based on an 18-layer ResNet [9]. Our ResNet implementation begins with a learning rate of $1e{-}4$, reduced to $1e{-}5$ after 100 epochs of training. In the loss function, L_p is cross entropy with logits. The standard cross entropy loss is augmented with a L2-regularization term with weight $\lambda = 1e-4$.



(a) Miniaturized 4-wheel motor vehicle (b) Indoor track (c) Example frame of STEERINGPATTERN.

Fig. 3: Experiment environment

- **Target network (Regression)**: *its training model is learning driving patterns of a human driver (controller) of a ground vehicle from the mapping of a steering angle and the image taken from the frontal camera. In inference time, when an image of track is received, the trained model infers the most plausible steering angle by regression. The model learns to emulate a human driver's steering pattern from recorded steering angle and frontal images* (i.e., *behavior cloning; refer to Fig. 3(c) - white arrow is a reference (ground truth) and green arrow is a predicted angle).* The architecture of the target network, RegressionNet, loosely follows the model detailed in [2]. Our RegressionNet implementation begins with a learning rate of $1e{-}4$, reduced to $1e{-}6$ after 100 epochs of training. The dropout layers have probability $p = 0.5$. L_p is mean squared error.

2) Data Sets: We use the German Traffic Sign Recognition Benchmark Dataset GTSRB [19] and CIFAR-10 [13] for classification and the STEERINGPATTERN dataset (self-collected) for regression.

- GTSRB: This dataset consists of a pre-divided training and test dataset, covering 43 classes of traffic signs. The training and test datasets contain 39209 and 12630 images, respectively. [19] We perform several preprocessing steps on the raw images:
  1) Convert the image to HSV, and perform histogram equalization on the value component. This increases the contrast of the image and potentially improves classification performance. We convert the image back to RGB after this operation.
  2) Center the images, cropping unnecessary background pixels.
  3) Resize each image to 48×48 resolution and scale the pixel values between $[0, 1]$.
- CIFAR-10: The CIFAR-10 dataset consists of 60,000 32×32 color images, divided into a training set of 50,000 images and a test set of 10,000 images. There are ten classes. [13]
- STEERINGPATTERN: We collected this dataset for regression to correctly predict steering angle given an input image. The data is collected from a frontal camera of a miniaturized 4-wheel motor vehicle on indoor tracks - refer to Fig. 3. The dataset is composed of 17923 images. The label consists of the steering angle, as similar to the data collected in [2]. The dataset is augmented by performing flips along the vertical axis of each image, resulting in an opposite turn angle, resulting in 35486 images in total. Each image is resized to dimension 60×80.

3) Data Preparation - Manual Noising Procedure: the denoiser + classification/regression pipeline was inspired by real-world noise, such as weather conditions, that may affect the performance of a neural network embedded in a self-driving car (*e.g.*, fog, rain, snow, etc). However, for experimental purposes, we use two noising procedures:

- GAUSSIAN noise: We add GAUSSIAN noise with $\mu = 0$ and $\sigma = 50$ to our images as shown in Fig. 4(b).
- SYNTHETIC noise: We randomly white-out (change the pixel-value to 1) 25% of the pixels of a given input image, resulting in a static-like effect (see Fig. 4(c)).

(a) Unaltered images (from GTSRB) (b) From (a) w/ GAUSSIAN noise ($\mu = 0; \sigma = 50$) (c) From (a) w/ SYNTHETIC noise

Fig. 4: Examples of different data noising types.

4) Performance Measures:

- System performance: For the GTSRB and CIFAR-10 datasets, we use Top-1 classification accuracy. For the STEERINGPATTERN dataset, we observe the mean-square error. We evaluate the performance with respect to each iteration of the preprocessor.
- Preprocessor performance: We independently evaluate the preprocessor's performance by computing the average mean squared error over noisy test images. We evaluate the performance with respect to each iteration of the preprocessor.
- Target network performance: For the classification network, we evaluate performance by computing average Top-1 accuracy over clean (non-noisy) test images. For the regression network, we compute average mean squared error over clean test images.

5) Approaches: In addition to the paired training approaches proposed in Section III-B, ALTERNATING, MERGE1, and MERGE2, we evaluate the following baseline approaches for comparison:

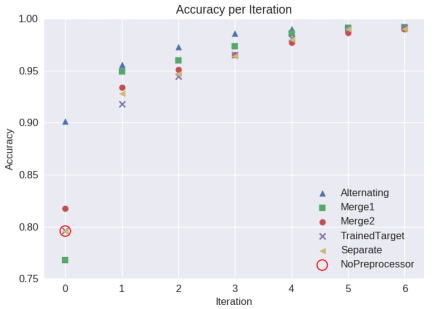
- NOPREPROCESSOR: target network runs without preprocessor support. Since Anytime performance is realized by the support from a preprocessor, any further performance improvement is not expected - only a single result is expected.
- SEPARATE: target network and preprocessor are independently trained with no knowledge about each other. Preprocessor conforms to loss function (1) and target network does to (3). For the classification network, L_p is cross entropy with logits; for the regression network, L_p is mean squared error.
- TRAINEDTARGET: preprocessor is trained with a pre-trained target network (which has been independently trained) according to loss function as shown in (3). Target network's weights stay the same.

B. System Performance

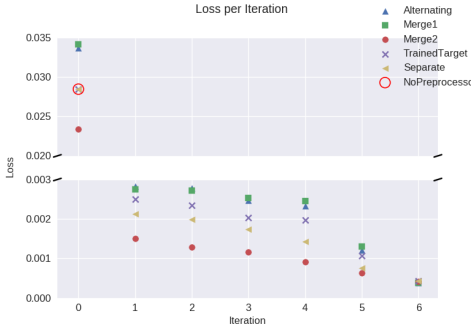
To begin, in Fig. 5, the impact of the preprocessor is seen by comparing NOPREPROCESSOR with the later iteration outputs of the other approaches. By having a preprocessor we can expect further improved accuracy. With no support from any preprocessor, the result of NOPREPROCESSOR is the only output that we can expect.

We notice the impact of employing *paired training* schemes, which is shown by comparing SEPARATE and ALTERNATING, MERGE1, and MERGE2. As explained in Sec. IV-A, in ALTERNATING, MERGE1, and MERGE2, either the preprocessor or target network considers the other in its training procedure, while in SEPARATE, the two networks are trained independently. The distinction between SEPARATE and the other schemes shows the positive contribution of paired training to overall system performance. That is, generally and independently

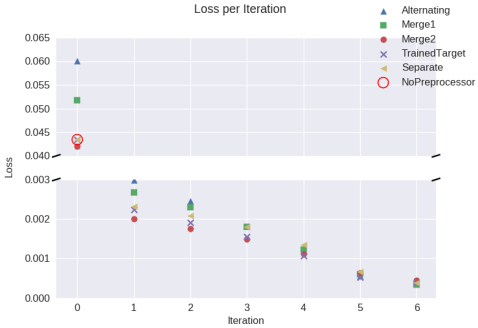
(a) On GTSRB with GAUSSIAN noise.



(b) On GTSRB with SYNTHETIC noise

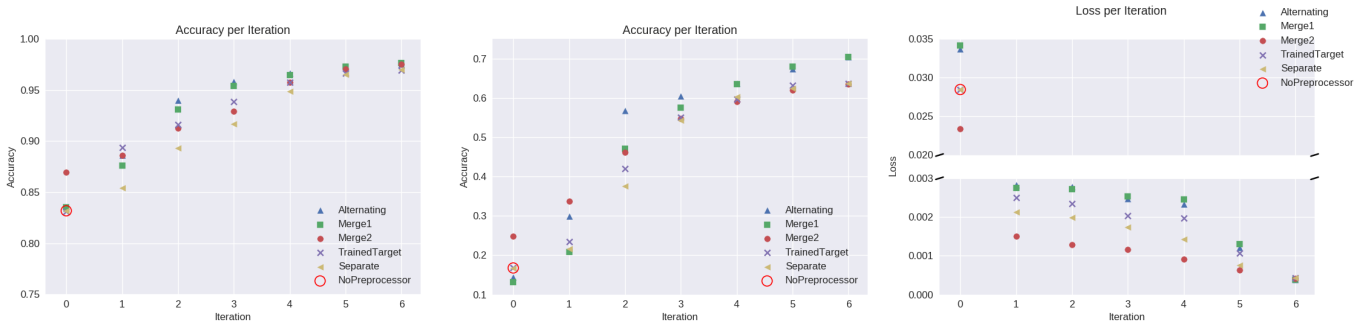


(c) On STEERINGPATTERN with GAUSSIAN noise



(d) On STEERINGPATTERN with SYNTHETIC noise

Fig. 5: System performance evaluations on GTSRB/STEERINGPATTERN with GAUSSIAN/SYNTHETIC noise types.



(a) On GTSRB. (b) On CIFAR-10. (c) On STEERINGPATTERN.

Fig. 6: Comparison of training approaches on different data sets with GAUSSIAN noise.

pre-trained networks are not expected to provide optimal performance if they are supposed to interact with each other in inference time.

1) Impact of Target Task: In order to determine the generalizability of this framework over various potential target networks, we trained a regression network in addition to the classification network. Our RegressionNet takes as input an image from the front-facing camera of a vehicle, and then performs steering angle prediction. The model is trained on the STEERINGPATTERN dataset. Results are shown in Fig. 5(c) and 5(d). Note that the first iteration (no iterations of the denoiser) is omitted, as it is an order of magnitude higher than the following iterations, and it compresses the results shown in the figures.

We notice ALTERNATING and MERGE1 perform worse than SEPARATE in earlier iterations, but excel by the final iteration. Likewise, MERGE2 displays more steady performance, outperforming SEPARATE for each iteration, and roughly matching SEPARATE by the final iteration. This is particularly notable in the Gaussian noise case. While this trend is much more visible in the regression context, it is present in the classification context as well – that is, we see MERGE2 outperform MERGE1 and ALTERNATING for the first iteration or two, after which MERGE1 and ALTERNATING tend to surpass MERGE2.

For ALTERNATING and MERGE1, the results are showing that training the target network first, and using the trained target network to provide feedback across the rest of the training process, directs the gradient descent process towards the ultimate goal of classification/regression rather than the ancillary (denoiser) goal of reducing mean squared error. This leads to ultimately superior performance by the final iteration. However, leading with training the denoiser, as in MERGE2, translates to a more stable and quickly converging training process, at the cost of final performance.

The result indicates that there are tradeoffs implicit in the proposed interleaved approaches with respect to anytime performance. Namely, better performing earlier iterations corresponds with a less optimal final iteration, whereas better performing later iterations corresponds with a worse-performing earlier iterations. This can be more broadly related with the convergence of an "anytime system" such as the one proposed in the paper. Our experimental setup remained largely unchanged between classification and regression - with a longer, more careful training procedure, ALTERNATING and MERGE1 could excel over MERGE2 in most iterations, similar to the classification case.

2) Impact of Noise Complexity: Comparing the accuracy curves in Fig. 5, the noise type impacts the potential benefits of paired training. The more complex the noise type, the more significant the improvement in performance. GAUSSIAN noise is relatively complex, given its randomized nature and application to each pixel, whereas SYNTHETIC noise is simpler. In GAUSSIAN noise curves, all paired training schemes exceed SEPARATE, both in final performance and in earlier iterations. For SYNTHETIC noise, the paired training schemes also outperform SEPARATE, but the improvement is slightly less significant. The complexity of denoising is responsible for this variance. For more complex noise, the denoiser is more likely introducing artifacts or impacting the performance of the classifier. Then, training the classifier with output from the denoiser or the denoiser with classifier feedback allows the system to optimize with knowledge of the artifacts, resulting in a substantial performance boost.

3) Dataset Impact: We examined the performance of paired training schemes across the CIFAR-10 and GTSRB datasets on Gaussian noise (see Fig. 6). We found similar trends across both datasets: ALTERNATING and MERGE1 ultimately outperform MERGE2, but MERGE2 displays stronger performance in the earlier iterations.

We do note that the gap between the best approach and SEPARATE for the final iteration of CIFAR-10 and GTSRB are of different orders of magnitude. Indeed, there is a accuracy gap of 0.01 between ALTERNATING and SEPARATE on the GTSRB dataset, while there is a 0.08 accuracy gap between ALTERNATING and SEPARATE. We attribute this difference to complexity of the target task – the CIFAR-10 images are more complex than those in GTSRB, and they have lower resolution. The different active ranges on the Y-axes of each case reflect the fact. Consequently, the application of Gaussian noise impacts the performance of the classifier more severely, making the performance of the classifier much more tied to the performance of the denoiser. In this situation, a form of interleaving is highly beneficial.

C. Independent Performance

The proposed framework trains the two networks to boost the system performance through paired training approaches and loss protocols. In the proposed system, each individual network does not necessarily display optimal performance respectively, since the paired loss protocol and training approaches optimize for system performance rather than individual network performance. We show the cost paid for overall performance by looking at the individual performance of the two component networks: classification performance of the target network on clean inputs, and mean squared error of the denoiser.

TABLE I: Performance (on clean and ideal inputs) of independent target network trained under different schemes/noise types on GTSRB.

	Gaussian	Synthetic
SEPARATE	0.991	0.991
TRAINEDTARGET	0.990	0.991
ALTERNATING	0.989	0.990
MERGE1	0.989	0.990
MERGE2	0.989	0.989

1) Target Performance: The impact of paired training on each classifier's independent performance, i.e., performance on clean inputs. Table I shows how the paired training and separate training approaches train the target networks over GAUSSIAN and SYNTHETIC noise. We can see that the individual target networks produced by the four paired training approaches perform worse than the SEPARATE and TRAINEDTARGET target networks. This result corresponds to the training processes for each paired approach. TRAINEDTARGET trains the target network independently, so the performance should be roughly equivalent to SEPARATE. Conversely, ALTERNATING, MERGE1, and MERGE2 all use preprocessor output to train the target at some point. Thus, the target networks for these approaches become optimized with respect to the corresponding denoising network, resulting in a small negative impact on their performance over clean data.

2) Preprocessor Performance: The independent performance of the denoiser is also affected by the paired training scheme. Contrasting with the independent performance of the target network, paired training approaches improve the independent performance of the denoiser. The feedback provided by the target helps the denoiser learn high-level information about the images being denoised. Looking at the specific approaches, those which begin by training the target and use that target to train the denoiser (ALTERNATING, MERGE1, and SEPARATE) have the most improved performance. MERGE2 only training the denoiser with paired loss at the end of training has the weakest performance.

V. RELATED WORK

For problems involving temporal sequences of data, a model must be able to encode dependencies across multiple timesteps. RNNs can model these sequential-type data: *e.g.*, text-prediction, speech recognition, or handwriting recognition. What differentiates RNNs from typical feed-forward neural networks is the existence of internal *states* that encode sequences of inputs that have occurred earlier. This is a form of memory that enables RNNs to process sequential data. Critical to the understanding of RNNs is that the learnable parameters, input, state, and output at a given timestep, are *shared* between iterations. Were parameters not shared between iterations, the number of parameters in the model would explode for any sequence of appreciable length, making training intractable.

Many recent architectures have combined RNN and convolutional neural network (CNN) structures: basic LSTM components with convolutions [11], [14], [16]; more complex LSTMs [18]; dilated convolutions [4], and quasi-recurrence [3]. The authors in [16] designed recurrent convolutional layers for classification. The work in [11] developed a deeply-recursive convolutional network for image super-resolution. A single RCL to learn complex feature maps is used not to introduce large numbers of parameters.

Image restoration has long been a central problem in low-level image processing. Procedures such as K-SVD [1], BM3D [5], the LRA-SVD method [7], and the LPG-PCA method [20] exploit the structured nature of image data, deriving information from groups of image patches to fill-in missing or obscured data. More recently, neural network based approaches have begun to surpass the more traditional approaches discussed above. Initially, autoencoders [22] and convolutional neural networks [10] showed promising results. More recently, deeper convolutional networks such as ResNets [21], GANs [15], and RNNs [8] have all broken numerous existing benchmarks for denoising and super-resolution.

VI. CONCLUSION

We proposed a framework employing two networks, a preprocessor and a target network, working in tandem; a raw input goes into a preprocessor, and the target network generates the final output. Several paired and interleaved training approaches, along with a joint loss mechanism, are compared to a baseline of independently trained networks. Pairing the training processes using these mechanisms results in better system performance than training each network in isolation. Further explorations into confidence levels at each iteration to obtain a strict Anytime curve, hyperparameters on paired loss functions, and different variations of interleaving mechanisms for paired training approaches will be interesting future directions.

REFERENCES

[1] M. Aharon, M. Elad, and A. Bruckstein. rmk-svd: An algorithm for designing overcomplete dictionaries for sparse representation. *IEEE Transactions on Signal Processing*, 54(11):4311–4322, Nov 2006.

[2] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba. End to end learning for self-driving cars. *CoRR*, abs/1604.07316, 2016.

[3] J. Bradbury, S. Merity, C. Xiong, and R. Socher. Quasi-recurrent neural networks. *CoRR*, abs/1611.01576, 2016.

[4] S. Chang, Y. Zhang, W. Han, M. Yu, X. Guo, W. Tan, X. Cui, M. Witbrock, M. A. Hasegawa-Johnson, and T. S. Huang. Dilated recurrent neural networks. In *NIPS*. 2017.

[5] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian. Image denoising by sparse 3-d transform-domain collaborative filtering. *IEEE Transactions on Image Processing*, 16(8):2080–2095, Aug 2007.

[6] Defense Science Board. Report of the defense science board on autonomy. Jun. 2016.

[7] Q. Guo, C. Zhang, Y. Zhang, and H. Liu. An efficient svd-based method for image denoising. *IEEE Transactions on Circuits and Systems for Video Technology*, 26(5):868–880, May 2016.

[8] W. Han, S. Chang, D. Liu, M. Yu, M. J. Witbrock, and T. S. Huang. Image super-resolution via dual-state recurrent networks. *CoRR*, abs/1805.02704, 2018.

[9] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[10] V. Jain and S. Seung. Natural image denoising with convolutional networks. In *NIPS*. 2009.

[11] J. Kim, J. K. Lee, and K. M. Lee. Deeply-recursive convolutional network for image super-resolution. In *CVPR*, 2016.

[12] J.-E. Kim, R. Bradford, and Z. Shao. Anytimenet: Controlling time-quality tradeoffs in deep neural network architectures. In *the 23rd Conference on Design, Automation and Test in Europe*, DATE, page 945–950, 2020.

[13] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[14] S. Lai, L. Xu, K. Liu, and J. Zhao. Recurrent convolutional neural networks for text classification. In *AAAI*, 2015.

[15] C. Ledig, L. Theis, F. Huszar, J. Caballero, A. P. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi. Photo-realistic single image super-resolution using a generative adversarial network. *CoRR*, abs/1609.04802, 2016.

[16] M. Liang and X. Hu. Recurrent convolutional neural network for object recognition. In *CVPR*, 2015.

[17] J. W. S. Liu, K. J. Lin, W. K. Shih, A. C. Yu, J. Y. Chung, and W. Zhao. *Algorithms for Scheduling Imprecise Computations*, pages 203–249. Springer US, Boston, MA, 1991.

[18] X. Shi, Z. Chen, H. Wang, D. Yeung, W. Wong, and W. Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. *CoRR*, abs/1506.04214, 2015.

[19] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks*, 32(0):323–332, 2012.

[20] M. Vijay and S. V. Subha. Spatially adaptive image restoration method using lpg-pca and jbf. In *MVIP*, 2012.

[21] T. Wang, M. Sun, and K. Hu. Dilated residual network for image denoising. *CoRR*, abs/1708.05473, 2017.

[22] J. Xie, L. Xu, and E. Chen. Image denoising and inpainting with deep neural networks. In *NIPS*. 2012.

[23] S. Zilberstein and S. Russell. *Approximate Reasoning Using Anytime Algorithms*, pages 43–62. Springer US, Boston, MA, 1995.