AnytimeNet: Controlling Time-Quality Tradeoffs in Deep Neural Network Architectures

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

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

***Abstract*—Deeper neural networks, especially those with extremely large numbers of internal parameters, impose a heavy computational burden in obtaining sufficiently high-quality results. These burdens are impeding the application of machine learning and related techniques to time-critical computing systems. To address this challenge, we are proposing an architectural approach for neural networks that adaptively trades off computation time and solution quality to achieve high-quality solutions with timeliness. We propose a novel and general framework, AnytimeNet, that gradually inserts additional layers, so users can expect monotonically increasing quality of solutions as more computation time is expended. The framework allows users to select on the fly when to retrieve a result during runtime. Extensive evaluation results on classification tasks demonstrate that our proposed architecture provides adaptive control of classification solution quality according to the available computation time.**

***Index Terms*—cyber-physical system, time-quality tradeoff, time-critical system, adaptive neural network, machine learning**

I. Introduction

Time-critical computing systems are often constrained by the state of the dynamic physical environment in which they operate. Moreover, in safety-critical systems such as automotive, avionic, or medical systems, software components interact in a carefully controlled way, therefore determinism and predictability are important requirements [1, 2]. In contrast, modern statistical machine learning is by nature non-deterministic. For certain machine learning tasks, data processing cannot be guaranteed to be complete within strict time limits.

This divergence has hindered learning modules from being exploited and incorporated in such time-critical systems, while time-critical computing systems are expected to evolve by gaining intelligence so as to become more autonomous [3]. The gap widens, especially, when large numbers of internal parameters cause the learning modules to impose a heavy computational burden in obtaining sufficiently high-quality results. In this paper, we address the gap by proposing an architectural framework for a computationally flexible learning network. Hence, as a system, the constituent learning components can be exploited in a more adaptive manner: instead of enforcing strict or predefined requirements for time and quality of results, we grant the user full control over the time-quality trade-off on the fly at runtime. For example, in some contexts, the user might prefer to live with 70% of the ideal quality (which was assumed to be potentially achievable), if doing so saves 50% of the execution cost.

This work is supported in part by grants from NSF 1521523 and 1715154 and GPU Grant by NVIDIA Corporation. All views expressed here are those of the authors and not necessarily those of sponsors.

Neural networks typically have very complex internal structures, making them difficult to analyze, and making the process of deriving early predictions practically impossible. Our proposed framework, *AnytimeNet*, breaks down this complexity into smaller transactions, so as to facilitate modularity in obtaining intermediate results and to support adaptive timeliness. AnytimeNet iteratively accumulates layers on top of the network of the previous iteration. The process provides the option of obtaining more refined results at later iterations.

Our proposed architecture, *AnytimeNet*, is built over iterations. At each iteration, AnytimeNet adds new blocks and can "cache" most of the previously-executed blocks. Through this iterative constructing/caching process, AnytimeNet achieves monotonically increasing quality of results and saves a substantial execution expense. At the end of every iteration, a new result is generated so the user can take the current and best-to-date result in quality. As reported in [4, 5], the performance gains achieved by ResNet are not solely due to network depth but rather by a combination of multiple networks or a gradual refinement of features from block to block. Those insights are formalized into our iterative architecture and training procedure.

Using well-known image classification benchmark data sets, we show how our framework achieves results efficiently in comparison with a baseline alternative. We also demonstrate the use of a confidence metric that can serve as an early indicator of the likely solution accuracy and the potential gains from further processing. The experiments show that our framework yields monotonically improving solutions, while providing the capability of retrieving intermediate results on the fly.

II. AnytimeNet

A. Base Architecture

The base architecture is a simplified version of a modern ResNet architecture [6], named BaseResNet. ResNets are deep convolutional networks [6, 7] employing residual connections - letting $\mathbf{h}_i$ represent hidden layer i and F be a transformation,

$$\mathbf{h}_{i+1} = \mathbf{h}_i + F^{(i)}(\mathbf{h}_i) \tag{1}$$

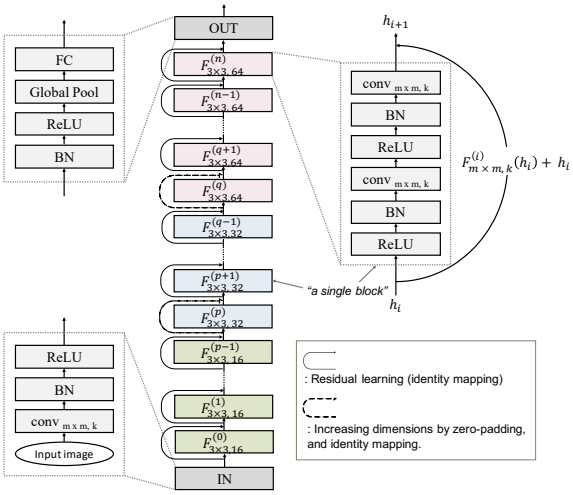


Fig. 1: Overall view of feed forward version of BaseResNet. Same-color boxes represent a matching feature size section except for IN and OUT.

Each residual block $F^{(i)}$ is primarily characterized through the specification ($m \times m$ kernel size and number of filters k) of its convolution operation. When we wish to emphasize this distinction, we include the additional notation, $F^{(i)}_{m\times m,k}$. The layer configuration of a single block $F^{(i)}$ is shown in Fig. 1.

As shown in Fig. 1, a feed forward version of BaseResNet contains 3 different feature size sections, rather than 4 (or more). As a result, it is architecturally smaller and simpler (the final layer consists of 64 filters) than modern ResNet architectures. Having said that, the techniques discussed in the following section are *generalizable* and still apply to any ResNet architecture. BaseResNet uses the full pre-activation schema developed in [8]. There is a specialized input block IN responsible for transforming an image into the appropriate feature space, and a specialized output block OUT responsible for transforming the features into an appropriate classification. Given an input image x and corresponding output classification o, we can use the notation developed above to formalize a full pass through BaseResNet:

$$o = \text{OUT}\left(\bigcirc_{i=0}^{r} F_{+}^{(i)}(IN(x))\right) \tag{2}$$

where $F_{+}^{(i)}(x) = F^{(i)}(x) + x$ encapsulates the entire (layer operations and identity connection) residual operation and the large $\bigcirc$ represents repeated function composition. Note that the special cases $F^{(p)}$ and $F^{(q)}$ (in Fig. 1 where the dimensions change) are slightly different than the other $F^{(i)}$, as they must increase the dimensions of the input connection. Hence, in those layers, the input is zero-padded before the addition operation.

B. Constructing AnytimeNet

The performance gains achieved by ResNet are not solely due to network depth. Indeed, the authors in [4] stated that an ensemble effect (*i.e.*, the ResNet behaves as a combination of multiple smaller networks rather than one large network) is primarily responsible for ResNet's performance. More recently, [5] showed that the performance may also partially be due to progressive feature refinement, or the tendency of the network to gradually refine features from block to block.

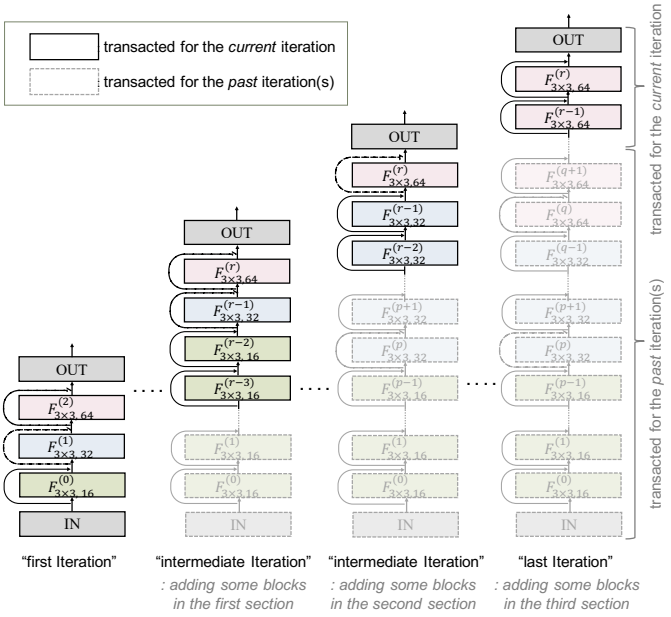


Fig. 2: AnytimeNet: iterative construction with BaseResNet blocks. Same-color boxes represent a matching feature size section except for IN and OUT. (The superscript of F^i only matters with respect to the current iteration.)

We formalize this insight into our training procedure by constructing the architecture in an iterative manner. We conceptualize the feed forward version as being composed of number of blocks plus an input and output block. We iteratively build *AnytimeNet* from these blocks, as shown in Fig. 2, by adding additional blocks at each iteration.

In detail, an iteration is formed by obtaining a(n intermediate) result from an OUT block as shown in Fig. 2. The first iteration starts with one block from each section. We must have at least one block from each section initially, in order to transform the input to the appropriate dimensionality for the output block. The minimal first iteration therefore consists of one block from each section, and an input and output block. At each iteration and for each feature size section, some number of blocks are newly inserted above the previously transacted blocks in the same section (dotted grey-outlined). The model can be used in an anytime fashion by executing each iteration consecutively. Although certain blocks need to be re-executed, significant time savings can be incurred by *caching* the results of previously executed blocks. The transparent boxes (dotted gray-outlined) in Fig. 2 represent blocks that are not executed for the current iteration. As a result, only the black-outlined boxes expend execution time at each iteration. Note that, even if a block needs to be executed for the current iteration, if it has been executed in past iteration(s), its parameters do not need to be derived again. That is, its parameters are shared iteration to iteration. Only the newly introduced blocks use new parameters.

AnytimeNet iteratively provides results iteration to iteration. Hence, as computation progresses, when a new (intermediate) result is generated, the current prediction value is posted and updated. That is, whenever a user accesses a result, it can take the up-to-date and optimal level of quality.

Aside from the main architectural conception of AnytimeNet, the process of iterative construction (number of blocks to add and how many times to add them) must also be considered. These choices affect the total number of iterations the network is capable of, and also determine the amount of overhead (repeated blocks) necessary during computation. We fix a single iteration scheme, specified in Sec. III, as a full investigation of potential iteration schemes is outside the scope of this paper. We also note that one could elect to not share parameters across repeated execution blocks (*e.g.*, the OUT layers). Preliminary experiments with non-shared output blocks did not show any performance improvement, so we elected to use the simpler model with shared repeated blocks. The shared parameter scheme also makes the comparison between AnytimeNet and BaseResNet more straightforward, as both models then share the same number of parameters, resulting in a more equitable experimental evaluation.

C. Training and Loss Function

The loss at each iteration, L_j, is the cross-entropy loss of predicted label against the ground truth label. Then, we define the total loss L as follows:

$$L = \sum_{i=j}^{n} f(n, j) \cdot L_j \tag{3}$$

where f is a function of n (total number of iterations) and j (each iteration) that determines the weights for each L_j. In particular, by using $f(n, j) = 1/(n - j + 1)^2$ as the weight scheme, we have the following total loss:

$$L = \sum_{j=1}^{n} \frac{L_j}{(n - j + 1)^2} \tag{4}$$

Connecting each iteration with the loss layer promotes gradient flow throughout the entirety of the network. The losses from earlier iterations are weighted less than the final iteration, in order to promote optimal final performance. We investigated several different weighting schemes and empirically found the one above provides the best balance between overall performance of each iteration, and performance of the final iteration. We find that the precise weighting scheme chosen has the following impacts: too much weight on earlier iterations results in an underperforming final iteration, while very low weights on earlier iterations results in an iterative model where the earlier iterations provide little value.

III. Experiment

A. Experimental Setup

1) Data Sets:

- CIFAR-10: The CIFAR-10 [9] consists of 60,000 32×32 color images - 50,000 training and 10,000 testing images. The images cover 10 classes.
- GTSRB: The GTSRB (German Traffic Sign Recognition Benchmark Dataset) [10] contains 39,209 training and 12,630 testing images, covering 43 classes of traffic signs.
- SFCars: The SFCars (Stanford Cars) [11] is divided into 8,144 training and 8,041 testing images. The images are classified into one of 196 classes at the level of make, model, and year. We use the provided bounding boxes to tightly crop the images around the car data, and we resize each image to 48×48.

All datasets with bounding boxes are centered/cropped appropriately, and pixel-values are normalized between $[0, 1]$.

2) Implementations:

- AnytimeNet: For experiments of our proposed AnytimeNet, we run 7 iterations. The first iteration starts with one block from each section. And then as the iterations proceed, 4 more blocks in the first section are stacked and then 3 more. This same process is applied to all sections, consecutively. The final iteration ends with 24 inner blocks and an IN and OUT block. The block count in each section, and the iteration count are all design parameters.
- Baseline: As a counterpart to compare with, we implemented Baseline which is a non-iterative single feed forward BaseResNet (Fig. 1). For comparison purposes, it contains 24 inner blocks and an IN and OUT block, architecturally equivalent to the final iteration of AnytimeNet.

3) Performance Metrics and Time Measurement: While we define accuracy as the number of correctly classified test images over the total numbers, for confidence we use the outputs of the final softmax layer as an approximation for model prediction confidence. Let $S(x)$ represent the model's softmax output of the correct label for an input image x. Then confidence overall is measured as:

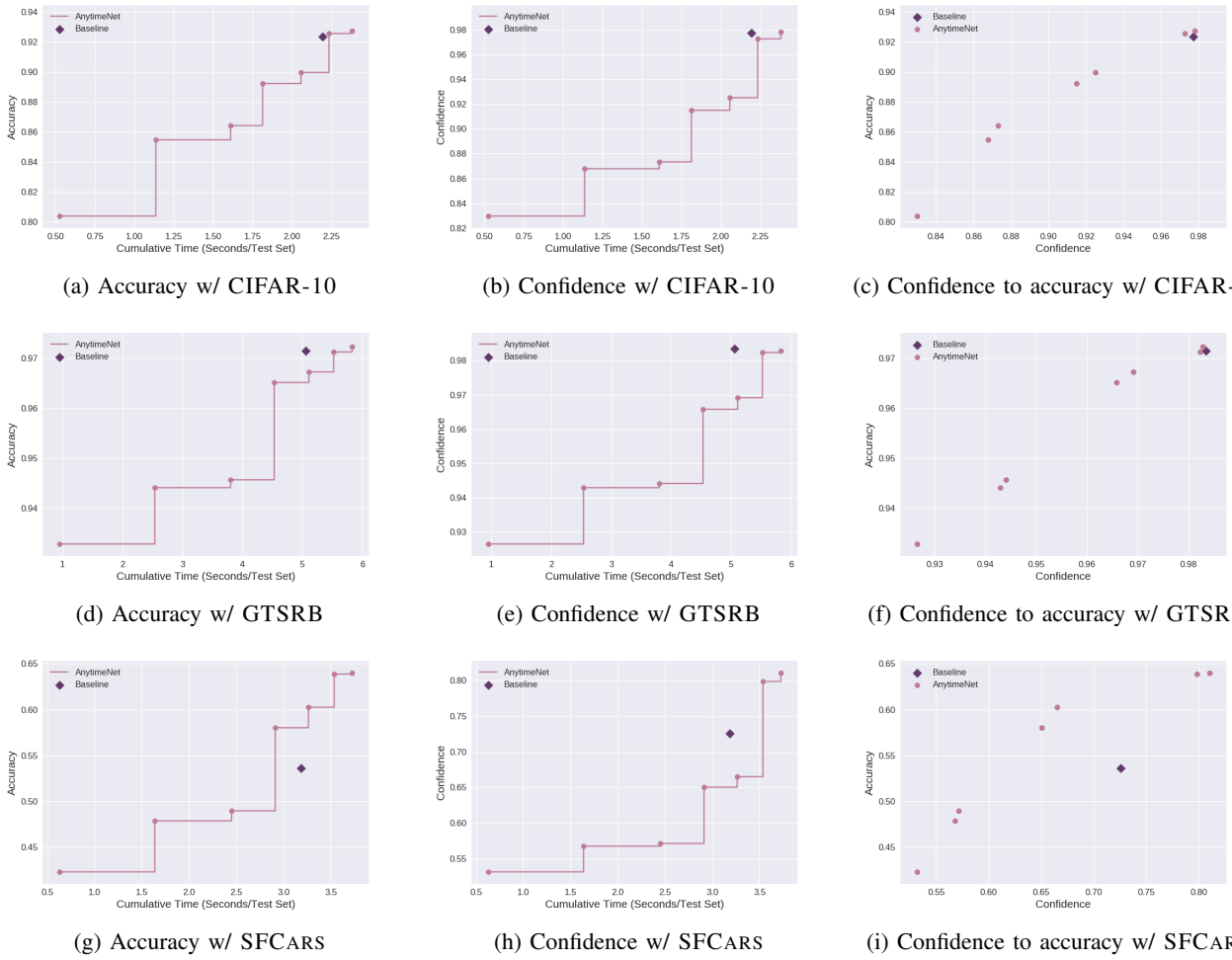
$$\frac{\sum\limits_{x \in \text{test images}} S(x)}{\#\{\text{test images}\}} \tag{5}$$

We use the softmax output as a proxy for confidence, though we acknowledge the caveats with treating the softmax output layer as a sensible probability distribution. Bayesian methods for achieving principled uncertainty estimates are superior, but require more overhead and are outside the scope of this paper.

We measure the execution time of each iteration, caching the outputs of previous blocks to save time as described in Fig. 2. Training was done on both an NVIDIA Quadro P6000 and Tesla P100. All (inference) time evaluations were performed on the NVIDIA Quadro P6000 (Intel(R) Xeon(R) CPU E5-2687W, 12 cores, 3.00 GHz CPU frequency, and 64 GB of main memory) to preserve consistency of results.

4) Hyperparameters: We use the following training parameters for all datasets, and for both AnytimeNet and Baseline:

- **Number of epochs**: 200 epochs for each network.
- **Learning rate and decay**: We initialize training with a learning rate of 0.1 After 100 epochs, we decrease the learning rate by a factor of 10. After another 50 epochs, we again decrease the learning rate by a factor of 10.
- **Batch norm,** ε: We set $\varepsilon = 0.001$.



(a) Accuracy w/ CIFAR-10 (b) Confidence w/ CIFAR-10 (c) Confidence to accuracy w/ CIFAR-10

(d) Accuracy w/ GTSRB (e) Confidence w/ GTSRB (f) Confidence to accuracy w/ GTSRB

(g) Accuracy w/ SFCARS (h) Confidence w/ SFCARS (i) Confidence to accuracy w/ SFCARS

Fig. 3: Accuracy and confidence across the three datasets. For ANYTIMENET, the data points represent results from the first to last iteration in order from left to right. The x-axis is cumulative time it takes to run through the entire test set. (Because accuracy, confidence, and computation time vary across the three datasets, the axes ranges are not aligned.)

Since it is not our goal to create an optimal network, we do not do rigorous performance tuning of the hyperparameters.

B. Experimental Results

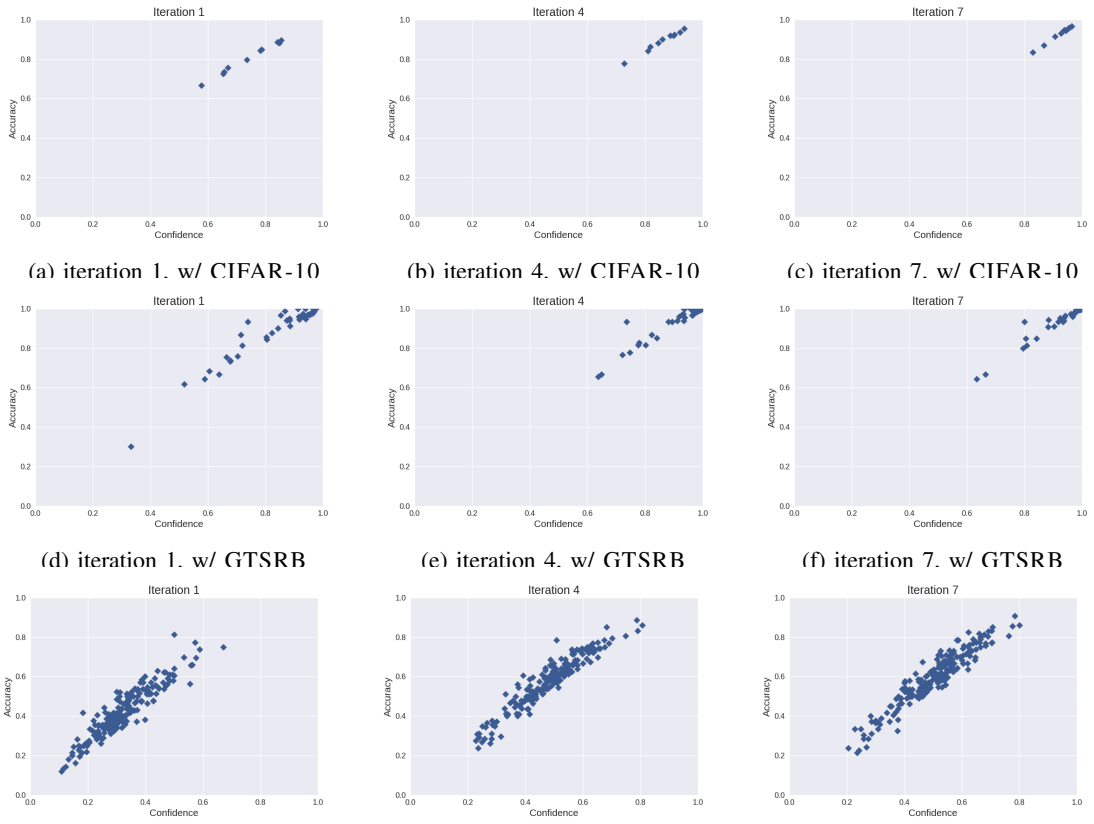
1) ANYTIMENET *and Performance Enhancement:* In Fig. 3, one point represents a result from each iteration – from the first iteration to the last from left to right. BASELINE provides one graph point. The x-axis shows ANYTIMENET's per-iteration *cumulative* time to run the entire test set (no distinction in BASELINE). First, it should be noticed that most of the intermediate results of ANYTIMENET are obtained before BASELINE can provide a result. In addition, we can see that the quality of results are monotonically improving in accuracy and confidence as time proceeds.

Also significant is that ANYTIMENET exceeds BASELINE's performance by the final iteration on all three datasets. Although the primary purpose of developing ANYTIMENET is to obtain quick and ballpark intermediate results, the fact that in most cases ANYTIMENET eventually outperforms BASELINE also suggests a direction on how to architect and train a high-performing ResNet. This stems from directly connecting earlier blocks to the loss layer (refer to (3)), incorporating the iterative refinement of features directly into the training procedure.

Aside from the main results, the shape of the "steps" depends on how many or in which feature section blocks are added. For the first iteration, there is one block of each type. For subsequent iterations, we consecutively fill out each section (starting from the bottom and moving upward) by adding 4 blocks and then 3 blocks. This results in a total of 7 iterations to build an 8 block ANYTIMENET.

2) Correlation between Accuracy and Confidence: Unlike the training stage, during the inference (runtime) there is no reference to compare with the currently obtained result. Fig. 3 (comparing Fig. 3(a) & 3(b), 3(d) & 3(e), and 3(g) & 3(h)) shows how confidence - as an indirect observable indicator of accuracy during inference - align with accuracy.

In Fig. 4, we visualize the average per-class confidence and accuracy. Each dot represents the average confidence and accuracy value for a single class. As the model progresses through iterations, the distribution of classes shifts upwards,



(a) iteration 1, w/ CIFAR-10 (b) iteration 4, w/ CIFAR-10 (c) iteration 7, w/ CIFAR-10

(d) iteration 1, w/ GTSRB (e) iteration 4, w/ GTSRB (f) iteration 7, w/ GTSRB

(g) iteration 1, w/ SFCARS (h) iteration 4, w/ SFCARS (i) iteration 7, w/ SFCARS

Fig. 4: Correlation of confidence and accuracy in results of ANYTIMENET. Accuracy and confidence are averaged *per* class.

indicating an improvement both in accuracy and confidence. Moreover, the overall relationship between confidence and accuracy is linear, further indicating that confidence can be a potential proxy for output quality during runtime.

3) Overhead of Re-executed Blocks: As Fig. 2 presents, unlike BASELINE, ANYTIMENET expends time repeating certain calculations – the OUT block is repeated with each iteration, along with several blocks in the second and third sections. Because first section blocks are never repeated and those expend the most computational energy, the total overhead is relatively low, but there is "no free lunch", so to speak. Enabling the capacity for early iterations necessarily adds some repeated blocks, resulting in increased computation time. We found that the scheme we chose minimized this overhead without negatively impacting ultimate classification performance. Table I shows the total number of blocks executed *by* (*i.e.*, up to and including) each iteration. In practice, ANYTIMENET expends roughly the same amount of time as BASELINE by iteration 5. From the dataset information and time measurements from Table I, we can provide Table II, showing the average block computational cost per iteration. We can see that the average cost decreases as the iterations progress. This is because of the asymmetrical nature of the added blocks, *i.e.*, a block from the 1st section is more expensive than one from the 3rd section, and one from the 2nd section is in between. Additionally, in ANYTIMENET, more low-overhead blocks (*e.g.*, OUT) are re-executed than in BASELINE.

4) Comparison Depending on Datasets: Both the overall performance and the performance of ANYTIMENET vs. BASELINE differs on each of the three datasets. The overall performance (of both networks) is best on GTSRB, second best on CIFAR-10, and worst on SFCARS. The reasons for these overall performance differences are twofold:

1) **Amount of training data**: The datasets have widely differing amounts of training data per class. On average, CIFAR-10 contains 5,000 training images per class, GTSRB contains about 911 training images per class, and SFCARS contains about 41 training images per class. That is, SFCARS is trained with fewer samples but needs to classify the images more precisely, and CIFAR-10 is vice versa and GTSRB is in between.
2) **Task complexity**: We can see that CIFAR-10 has a higher number of training data per class than GTSRB, yet the model performs better on the GTSRB dataset. Indeed, the GTSRB task is easier – the objects being classified are largely presented in a front-facing manner, whereas in CIFAR-10, we must learn to recognize multiple representative angles for each of the given classes. Likewise, the objects in GTSRB are always centered within the image, while for CIFAR-10 images the object to be classified may be relatively small or off-center.

Likewise, there are differences between relative performance

of ANYTIMENET and BASELINE with respect to the same dataset. In general, the final iteration of ANYTIMENET exceeds the accuracy of BASELINE by a small amount on each of the datasets. The exception to this is SFCARS, where ANYTIMENET far exceeds the performance of BASELINE. We suspect that the unusually low amount of training data per class accounts for the highly variable performance.

IV. RELATED WORK

Developed in [6], residual learning was proposed as a means of effectively training very deep neural networks. ResNets have provided state of the art performance on many image-processing tasks. Initially, the performance was assumed to be a result of the depth of ResNet – previously, very deep neural networks were difficult to train, and the ResNet architecture was able to circumvent some of these training issues. However, later work [4, 5, 12] uncovered that ResNet performance can be partially attributed to the network behaving like an ensemble of shallower networks, and the network's tendency to iteratively refine features. Due to the modularity of ResNets, blocks can be dropped without totally destroying the model's performance. This attribute is used in [13] to speed up ResNet computation, pruning blocks that contribute less to prediction accuracy.

Attention [14, 15] provides a way for a discriminative model to focus on parts of an input image that are most important to the final classification. While related to our goals, attention falls short of solving the problem of granular visual understanding. Conceptually, attention should decrease computational costs, as the network now has some way of telling which portions of the input data are most important to the given task. But implementing an attention mechanism involves introducing many more trainable parameters, resulting in longer execution times. Because our focus is on anytime behavior, a way to produce ballpark estimates without the overhead of many additional parameters is preferred.

In multiple papers including [16, 17], the concept of neural networks with early exits has been explored. However, these early-exit strategies were developed with the goal of improving computational efficiency by skipping layers automatically if the result exceeds certain confidence thresholds during the computation. They do not provide user-control of the tradeoff. These strategies can save computation resources to some extent in predefined conditions, whereas our proposed framework grants a user full control over preferred result quality, which may change depending on the problem domain.

TABLE I: Cumulative block count executed *by* each iteration.

	IN	SEC 1	SEC 2	SEC 3	OUT	Total
Iteration 1	1	1	1	1	1	5
Iteration 2	1	5	2	2	2	12
Iteration 3	1	8	3	3	3	18
Iteration 4	1	8	7	4	4	24
Iteration 5	1	8	10	5	5	29
Iteration 6	1	8	10	9	6	34
Iteration 7	1	8	10	12	7	38

TABLE II: Per-block avg time taken per iteration (in sec.)

	CIFAR-10	GTSRB	SFCARS
Avg. for all iterations	0.063	0.153	0.098
BASELINE	0.084	0.212	0.123

V. CONCLUSION

In this work, we proposed a new architectural framework that can be generalized in a number of DNN architectures. The primary aim lies in granting users control over the time-quality trade-offs, allowing them to employ adaptive timeliness for their contexts and resource limitations. The experimental results with different datasets exhibit that the proposed framework can provide monotonically increasing quality of results, providing several predictions before a non-iterative counterpart model did, and ultimately exceeding such a model by the final iteration. In addition, the results demonstrate that confidence is a potential indicator of accuracy. Promising areas for future work include further exploring the potential benefits of using non-shared parameters across repeated execution blocks.

REFERENCES

[1] Edward A Lee. Cyber physical systems: Design challenges. In *IEEE ISORC*, pages 363–369. IEEE, 2008.

[2] Ragunathan Rajkumar, Insup Lee, Lui Sha, and John Stankovic. Cyber-physical systems: the next computing revolution. In *Design Automation Conference (DAC), 2010 47th ACM/IEEE*, pages 731–736. IEEE, 2010.

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

[4] Andreas Veit, Michael Wilber, and Serge Belongie. Residual networks behave like ensembles of relatively shallow networks. In *NIPS*, pages 550–558, 2016.

[5] Stanislaw Jastrzebski, Devansh Arpit, Nicolas Ballas, Vikas Verma, Tong Che, and Yoshua Bengio. Residual connections encourage iterative inference. In *ICLR*, 2018.

[6] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *IEEE CVPR*, pages 770–778, June 2016.

[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, pages 1097–1105. 2012.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *ECCV*, 2016.

[9] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[10] 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.

[11] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In *Int'l IEEE Workshop on 3D Representation and Recognition*, Sydney, Australia, 2013.

[12] Klaus Greff, Rupesh Kumar Srivastava, and Jürgen Schmidhuber. Highway and residual networks learn unrolled iterative estimation. In *Proc. of the International Conference on Learning Representations*, 2017.

[13] Zuxuan Wu, Tushar Nagarajan, Abhishek Kumar, Steven Rennie, Larry S. Davis, Kristen Grauman, and Rogério Schmidt Feris. Blockdrop: Dynamic inference paths in residual networks. *IEEE CVPR*, 2018.

[14] Hugo Larochelle and Geoffrey E Hinton. Learning to combine foveal glimpses with a third-order Boltzmann machine. In *NIPS*. 2010.

[15] Misha Denil, Loris Bazzani, Hugo Larochelle, and Nando de Freitas. Learning where to attend with deep architectures for image tracking. *Neural Computation*, 24(8):2151–2184, 2012.

[16] Surat Teerapittayanon, Bradley McDanel, and H. T. Kung. Branchynet: Fast inference via early exiting from deep neural networks. In *the 23rd International Conference on Pattern Recognition (ICPR)*, 2016.

[17] Tolga Bolukbasi, Joseph Wang, Ofer Dekel, and Venkatesh Saligrama. Adaptive neural networks for efficient inference. In *ICML*, Aug 2017.